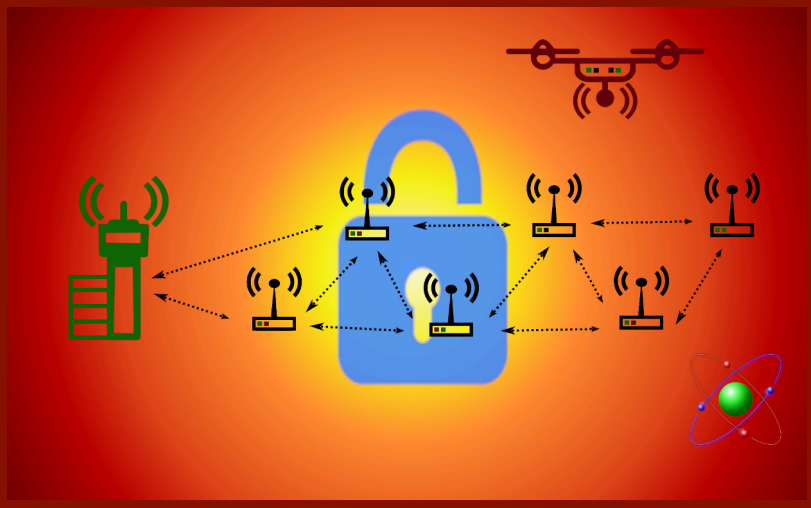
Tesis Doctoral
Doctorado en Ingeniería Automática, Electrónica y de Telecomunicación

Arquitectura de Seguridad para la Cooperación entre Robots Aéreos y Redes de Sensores



Autor: Francisco José Fernández Jiménez
Director: José Ramiro Martínez de Dios

Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Tesis Doctoral
Doctorado en Ingeniería Automática, Electrónica y de Telecomunicación

Arquitectura de Seguridad para la Cooperación entre Robots Aéreos y Redes de Sensores

Autor:

Francisco José Fernández Jiménez

Director:

José Ramiro Martínez de Dios

Catedrático de Universidad

Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

2023

A mis padres

Agradecimientos

Me gustaría dar las gracias a todas aquellas personas que han estado a mi lado todos los años que ha costado hacer esta Tesis Doctoral, que han sufrido conmigo y me han animado. Si tú has sido una de ellas... *¡Muchas gracias!*

Particularmente, quiero agradecer a mi amigo y director de Tesis Doctoral su dedicación, consejos, ideas, gestiones, correcciones, colaboración en artículos, ...Esta Tesis Doctoral no hubiera sido posible sin su ayuda y colaboración, y parte de ella le pertenece.

Me gustaría dar las gracias también a todos aquellos que han ayudado en la realización de los experimentos: a Alejandro Rosa Neupavert por realizar los ataques de *Red Team*, a Rafael Salmoral por pilotar los robos aéreos, a todo el "GRVC Robotics Lab" por facilitar los dispositivos, accesorios y software complementario necesario, y a la empresa LafargeHolcim por permitir utilizar su cementera de Jerez de la Frontera.

Por último, doy las gracias a mis compañeros del Departamento de Ingeniería Telemática por su preocupación, apoyo y compañerismo, por sus enseñanzas y colaboraciones, por ayudarme con la burocracia y por facilitarme la realización de esta Tesis Doctoral.

Resumen

En esta Tesis Doctoral se ha diseñado, desarrollado, implementado, experimentado y validado SNSR (*Sensor Network Security using Robots*), una arquitectura novedosa, abierta y flexible que mejora la seguridad en las redes inalámbricas de sensores estáticas al beneficiarse de la cooperación de la red de sensores con los robots. En SNSR, los robots realizan la autenticación de nodos sensores y la localización basada en radio (lo que permite el cálculo de topología centralizado y el establecimiento de rutas) e interactúan directamente con los nodos para enviarles configuraciones o recibir informes de estado y anomalías sin intermediarios. El funcionamiento de SNSR se divide en etapas configuradas en una estructura iterativa de retroalimentación, que permite repetir la ejecución de etapas para adaptarse a cambios, responder a ataques o detectar y corregir errores. El uso de robots brinda a SNSR capacidades mejoradas de detección y mitigación de ataques, y mejora la prevención al reemplazar tareas que en las arquitecturas tradicionales son propensas a ataques o requieren información preestablecida. Esto proporciona altos niveles de seguridad y adaptabilidad sin requerir mecanismos complejos.

Para implementar SNSR se ha creado un nuevo sistema operativo de red simple e independiente de la red de sensores que proporciona flexibilidad para probar diferentes configuraciones, algoritmos y protocolos, pero sin llegar a los niveles físico y de enlace. Se ha desarrollado e implementado una nueva pila de protocolos, ubicada sobre una capa de enlace existente. Esta pila de protocolos se compone de un nivel de aplicación, un nivel de transporte, un nivel de red y un nivel de adaptación al nivel de enlace. El nivel de adaptación hace que los niveles superiores sean independientes del nivel de enlace real. El nivel de aplicación permite ejecutar las aplicaciones de gestión de cada entidad siguiendo las especificaciones de SNSR.

Se ha realizado un análisis cualitativo de seguridad frente a 33 ataques de seguridad diferentes. Además, se ha creado una plataforma de pruebas que permita la experimentación repetible y rigorosa de nuevas tecnologías, protocolos y métodos de seguridad basados en SNSR tanto en escenarios reales como en virtualizados. Con ella se han realizado miles de experimentos con distintos escenarios virtuales y reales con robots terrestres y aéreos. Los resultados han demostrado la seguridad, eficiencia y escalabilidad de SNSR.

Abstract

In this Ph.D. Thesis, SNSR (*Sensor Network Security using Robots*), a novel, open, and flexible architecture that improves security in static wireless sensor networks by benefiting from robot-sensor network cooperation, has been designed, developed, implemented, experimented, and validated. In SNSR, the robots perform sensor node authentication and radio-based localization (enabling centralized topology computation and route establishment) and directly interact with nodes to send them configurations or receive status and anomaly reports without intermediaries. SNSR operation is divided into stages set in a feedback iterative structure, which enables repeating the execution of stages to adapt to changes, respond to attacks, or detect and correct errors. The use of robots provides SNSR with enhanced attack detection and mitigation capabilities, and improves prevention by replacing tasks that in traditional architectures are prone to attacks or require preset information. This provides high security levels and adaptability without requiring complex mechanisms.

To implement SNSR, a new simple network operating system has been created, independent of the sensor network, which provides flexibility for developing different configurations, algorithms and protocols, but without reaching the physical and link layers. A new protocol stack has been developed and implemented, sitting on top of an existing link layer. This protocol stack is composed of an application layer, a transport layer, a network layer, and a link layer adaptation layer. The adaptation layer makes the upper layers independent of the actual link layer. The application layer allows running the management applications of each entity following the SNSR specifications.

A qualitative security analysis has been carried out against 33 different security attacks. In addition, a test platform has been created that allows repeatable and rigorous experimentation of new technologies, protocols and security methods based on SNSR in both real and virtualized scenarios. With it, thousands of experiments have been carried out with different virtual and real scenarios with terrestrial and aerial robots. The results have demonstrated the security, efficiency and scalability of SNSR.

Índice

Notación, acrónimos y convenciones utilizadas

Notación

$izda.$	Izquierda
$dcha.$	Derecha
$\|$	Concatenación de cadenas
$\mathbf{PR}_x$	Clave privada de la entidad x
$\mathbf{PU}_x$	Clave pública de la entidad x
$\mathbf{Cert}_x^y$	Certificado de la entidad x firmado por la entidad y
$\mathbf{ID}_x$	Identidad de la entidad x
$E_K(P)$	Encriptación del mensaje P usando la clave K
$E_K^{-1}(P)$	Desencriptación del mensaje P usando la clave K
$E_{\mathbf{PR}_x}(P)$	Firma digital del mensaje P realizado por la entidad x
$E_{\mathbf{PU}_x}^{-1}(S)$	Verificación de que la firma digital S ha sido realizada por la entidad x
$\mathbb{R}$	Cuerpo de los números reales
$\mathbb{C}$	Cuerpo de los números complejos
$A \backslash B$	Diferencia entre el conjunto A y el B
$\|A\|$	Cardinalidad del conjunto A
$\forall$	Cuantificador universal
$\|\mathbf{v}\|$	Norma del vector $\mathbf{v}$
$\mod$	Módulo, resto de una división entera
$\stackrel{\text{def}}{=}$	Igual por definición
O	Orden

Acrónimos

6LoWPAN *IPv6 over Low-Power Wireless Personal Area Networks*. 116

AEAD *Authenticated Encryption with Associated Data* (encriptación autenticada con datos asociados). 56

AES *Advanced Encryption Standard* (Estándar de Encriptación Avanzado). 113, 114

AL *Application Layer* (nivel de aplicación). 50, 53, 62, 65, 84, 101, 114, 124, 125, 130, 132, 133, 147–150, 155, 201, 202, 204–206

AODV *Ad hoc On-Demand Distance Vector*. 33, 195, 198

APS *Application Support Sublayer* (subnivel de soporte de aplicaciones). 33–35, 37–39, 64

ASN.1 *Abstract Syntax Notation One* (Notación Sintáctica Abstracta 1). 115

BFS *Breadth First Search* (búsqueda en anchura). 40, 250

BGP *Border Gateway Protocol* (protocolo de puerta de enlace de frontera). 120

BS *Base Station* (estación base). 1, 2, 4, 7, 10–13, 24–26, 29–31, 33, 39–41, 43–51, 54, 55, 59, 60, 65, 67, 68, 71, 73, 75–83, 87, 89, 91–98, 100–107, 111–113, 115, 116, 118–124, 127–131, 133, 148–150, 156, 157, 159, 160, 166, 170–172, 176, 189, 192, 194, 195, 197–199, 201–206, 208–217, 219–225, 228, 229, 231–233, 237, 240, 241, 245, 248, 250, 251, 253–256

CA *Collision Avoidance* (Prevención de Colisiones). 32, 57

CA *Certificate Authority* (Autoridad de Certificación). 21, 22, 38, 47–49, 55, 69, 75, 78, 104, 160, 189, 191, 231, 235, 250–252, 254

CBKE *Certificate-Based Key Establishment* (establecimiento de claves basado en certificados). 37–39, 191

CH *Cluster Head* (cabeza de clúster). 13, 25–27, 30, 253

CLI *Command-Line Interface* (interfaz de línea de comandos). 170, 172–174

CN *Common Name* (Nombre Común). 48, 50, 54, 235, 239

CoAP *Constrained Application Protocol* (Protocolo de Aplicación Restringida). 22

CR *Certificate Repository* (Repositorio de Certificados). 47

CRC *Cyclic Redundancy Check* (Verificación de Redundancia Cíclica). 46

CRL *Certificate Revocation List* (Lista de Revocación de Certificados). 39, 47, 48, 54, 55, 73, 77–81, 83, 84, 86, 89, 91, 93, 94, 96, 100–102, 104, 105, 111, 115, 116, 119, 123, 155, 156, 158, 160, 169, 180, 186, 209, 213, 221, 225, 234, 251, 252, 259

CSMA *Carrier Sense Multiple Access* (Acceso Múltiple con Escucha de Señal Portadora). 32, 57

CTP *Collection Tree Protocol*. 25

DFS *Depth First Search* (búsqueda en profundidad). 120

DH Diffie-Hellman. 69

DoS *Denial of Service* (Denegación de Servicio). 17, 24, 26, 29, 38, 46, 68, 88, 233

DTLS *Datagram Transport Layer Security* (Seguridad de la Capa de Transporte de Datagramas). 22, 30, 69, 113, 132, 137, 186, 189, 205, 207, 229, 231, 235, 236, 248, 253

ECC *Elliptic Curve Cryptography* (Criptografía de Curva Elíptica). 21, 22, 28–30, 38, 52, 113, 116, 137, 186, 190, 191, 248, 253, 254

ECDHE *Elliptic-Curve Diffie–Hellman Ephemeral* (Diffie-Hellman de Curva Elíptica Efímero). 114, 123

ECDSA *Elliptic-Curve Digital Signature Algorithm* (Algoritmo de Firma Digital de Curva Elíptica). 39, 114, 190

ECQV *Elliptic Curve Qu-Vanstone*. 191

FCS *Frame Check Sequence* (secuencia de comprobación de trama). 57

FFD *Full-Function Device* (dispositivo de función completa). 32

GCM *Galois/Counter Mode*. 114

GNSS *Global Navigation Satellite System* (sistema global de navegación por satélite). 26

GPL *GNU General Public License* (Licencia Pública General de GNU). 113, 133

GTS *Guaranteed Time Slot*. 196

HMAC *Hash-based Message Authentication Code* (Código de Autenticación de Mensaje basado en *Hash*). 35

HTB *Hierarchical Token Bucket*. 161

HTTP *Hypertext Transfer Protocol* (Protocolo de Transferencia de Hipertexto). 180–182

HTTPS *Hypertext Transfer Protocol Secure* (Protocolo Seguro de Transferencia de Hipertexto). 180

IANA *Internet Assigned Numbers Authority*. 113, 133

IDS *Intrusion Detection Systems* (Sistemas de Detección de Intrusos). 2, 19, 23–25, 30, 97, 250, 251

IKE *Internet Key Exchange*. 67, 69

IoT *Internet of Things* (Internet de las Cosas). 1, 13, 15, 21, 29, 114

IP *Internet Protocol* (Protocolo de Internet). 116, 134, 181, 183, 185

IPS *Intrusion Prevention System* (Sistema de Prevención de Intrusos). 97

IPsec *Internet Protocol security* (Seguridad del Protocolo de Internet). 22

ITU *International Telecommunication Union* (Unión Internacional de Telecomunicaciones). 163

IV *Initialization Vector* (vector de inicialización). 56

IWSAN *Industrial Wireless Sensor and Actuator Network* (red de sensores y actuadores inalámbrica industrial). 15

IWSN *Industrial Wireless Sensor Network* (red de sensores inalámbrica industrial). 14

Java EE *Java Enterprise Edition*. 180

JSON *JavaScript Object Notation*. 114, 124, 160, 166, 170, 174, 177

JSP *JavaServer Pages*. 180

LL *Link Layer* (nivel de enlace). 46, 50–52, 57, 60, 62, 67, 73, 74, 84, 88, 103, 109, 112–115, 123, 124, 130, 133, 134, 136, 137, 140, 142, 158, 159, 161, 163, 169, 189, 193, 195–197, 203–205, 207, 208, 210, 211, 214, 215, 222, 225, 229, 234, 256, XXI

LLAL *Link Layer Adaptation Layer* (nivel de adaptación al nivel de enlace). 130, 133, 134, 138, 143, 186, 248, 252, 253

LLC *Logical Link Control* (Control de Enlace Lógico). 32, 57

LPWAN *Low Power Wide Area Network* (red de área amplia de baja potencia). 15

LR-WPAN *Low-rate Wireless Personal Area Network* (redes inalámbricas de área personal con tasas bajas de transmisión de datos). 15, 31

MAC *Media Access Control* (Control de Acceso al Medio). 15, 31, 32, 34, 35, 57–59, 127, 181, 182, 209, 212, 225

MAC *Message Authentication Code* (Código de Autenticación de Mensaje). 19, 39–41, 46, 54, 56, 66, 69, 84, 105, 133–135, 159, 204, 205, 207

MANET *Mobile Ad hoc Network* (red ad hoc móvil). 11

MIC *Message Integrity Code* (Código de Integridad del Mensaje). 35, 56

MITM *Man in the Middle attack*. 69

MQV Menezes–Qu–Vanstone. 69, 191

MS *Mobile Station* (estación móvil). 44

MTU *Maximum Transmission Unit* (unidad de transmisión máxima). 57–59, 101, 107, 124, 132, 135, 140, 142, 146, 189

NL *Network Layer* (nivel de red). 52, 57, 60–66, 80, 95, 100, 101, 125, 130, 134, 137, 138, 140, 144, 149, 150, 193, 198, 202, 203, 206, 207, XXI

NOS *Network Operating System* (Sistema Operativo de Red). 4, 5, 7, 106–108, 131, 158, 159, 185, 186, 248, 249, 252

OHC *One-way Hash Chain* (cadena *hash* unidireccional). 27, 39, 40

OSI *Open Systems Interconnection* (Interconexión de Sistemas Abiertos). 17, 32, XXI

OTA *Over-the-Air* (por el aire). 179, 250

PAN *Personal Area Network* (red de área personal). 33, 34

PDU *Protocol Data Unit* (unidad de datos de protocolo). 53, 59, 62, 65, 66, 75, 124

PFS *Perfect Forward Secrecy* (secreto perfecto hacia adelante). 38, 66, 71

PHY *Physical layer* (nivel físico). 56, 109, 193, 194, 196, 207

PKG *Private Key Generator* (generador de claves privadas). 22

PKI *Public Key Infrastructure* (Infraestructura de Clave Pública). 21, 22, 28–30, 47–49, 55, 160, 186, 248

RA *Registration Authority* (Autoridad de Registro). 47

RFD *Reduced-Function Device* (dispositivo de función reducida). 32

RSA Rivest–Shamir–Adleman. 21, 113

RSSI *Received Signal Strength Indicator* (indicador de fuerza de la señal recibida). 19, 26

RTT *Round-Trip Time* (tiempo de ida y vuelta). 125–127, 129, 142, 144, 236

SCTP *Stream Control Transmission Protocol* (Protocolo de Transmisión de Control de Flujo). 141

SDN *Software Defined Networking* (Redes Definidas por Software). 25, 30, 31, 51, 59, 116, 253

SDT *Sleep Deprivation Torture*. 196, 197

SDU *Service Data Unit* (unidad de datos de servicio). 57, 58, 61, 65, 124, 137

SHA *Secure Hash Algorithm* (Algoritmo de Hash Seguro). 114, 116, 181–183, 185, 190

SNSR *Sensor Network Security using Robots*. 3–7, 9, 28, 30, 31, 43, 44, 47–51, 62, 70, 72, 73, 76, 83, 84, 86, 96, 97, 103, 106–108, 112–115, 125, 127, 129, 130, 137, 149, 150, 155, 160, 164, 185–188, 191–208, 212, 229, 231, 232, 234, 240, 243–245, 247–253, 255, 259

SSH *Secure SHell*. 183

SSP *Security Service Provider* (Proveedor de Servicios de Seguridad). 35

SVG *Scalable Vector Graphics* (Gráficos Vectoriales Escalables). 171, 179

SVM *Support Vector Machines* (máquinas de vectores de soporte). 24

TC *Trust Center*. 34–39, 191, 195, 202–204, 206

TCP *Transmission Control Protocol* (Protocolo de Control de Transmisión). 62, 64, 66, 141, 142

TL *Transport Layer* (nivel de transporte). 50–52, 65–67, 73, 74, 82–84, 90, 94, 100, 101, 112–114, 124, 133, 136, 139, 141, 142, 149, 193, 198, 203–205, 207, 216, 252, XXI

TLS *Transport Layer Security* (Seguridad de la Capa de Transporte). 22, 62, 69, 137

TLV tipo, longitud y valor. 114, XXI

TSP *Traveling Salesman Problem* (problema del vendedor viajero). 27

TTL *Time To Live* (tiempo de vida). 60

UAV *Unmanned Aerial Vehicle* (vehículo aéreo no tripulado). 26–28, 30, 31, 44

UDP *User Datagram Protocol* (Protocolo de Datagramas de Usuario). 133, 134, 137, 141, 142, 161, 164, 169, 185, 189, 220

UML *Unified Modeling Language* (Lenguaje Unificado de Modelado). 78, 134, 138, 142, 147

URL *Uniform Resource Locator* (localizador de recursos uniforme). 181

UWB *Ultra-WideBand* (banda ultraancha). 209

VANET *Vehicular Ad hoc Networks* (red ad hoc vehicular). 11

WSN *Wireless Sensor Network* (red inalámbrica de sensores). 1–5, 7, 9–31, 39, 41, 43, 44, 46, 47, 49–51, 59, 71, 75–78, 92, 97, 100, 101, 106–108, 114, 116, 129, 131, 186, 188, 189, 243, 245, 247–249, 253, 254

ZAP *ZigBee Application Profile* (perfil de aplicación ZigBee). 34

ZC *ZigBee Coordinator* (coordinador de ZigBee). 32–36, 38

ZDO *ZigBee Device Object* (objeto de dispositivo ZigBee). 34

ZED *ZigBee End Device* (dispositivo final de ZigBee). 32–34, 37

ZR *ZigBee Router* (enrutador de ZigBee). 32–34, 36, 37

ZSE *ZigBee Smart Energy*. 37–39, 191, 193, 194, 207, 208, 244, 248

Convenciones utilizadas

Cadenas y operaciones de cadenas

Se denominará cadena a una secuencia de símbolos elegidos de un conjunto específico. Se consideran solo dos conjuntos: alfabeto binario 0,1 y octetos 0,1,...,255, dando lugar a cadenas binarias y cadenas de octetos. La longitud de una cadena será el número de símbolos que contiene. Si la longitud es 0, a la cadena se la denomina cadena vacía.

Se define la operación ***concatenación*** de dos cadenas *a* y *b* de longitudes *m* y *n* respectivamente, como la generación de una nueva cadena *c* de longitud $m+n$ en la que los m símbolos más a la izquierda coincide con los de *a* y, los n más a la derecha, con los de *b*. Se representará como: $c = a \parallel b$

Mensajes transmitidos

Cuando se quiera mostrar el contenido de un mensaje transmitido desde una entidad $ORIGEN$ a otra entidad $DESTINO$, se utilizará la siguiente representación:

$$ORIGEN \rightarrow DESTINO : campo_1 \parallel campo_2 \parallel \ldots$$

Si el mensaje es de difusión, se utilizará $*$ como $DESTINO$. El termino mensaje se utiliza aquí de manera genérica como información transmitida por la red, pero recibe distintos nombres según el nivel del modelo de Interconexión de Sistemas Abiertos (OSI, *Open Systems Interconnection*) que lo trata: trama en el nivel de enlace (LL, *Link Layer*), paquete en el nivel de red (NL, *Network Layer*), segmento o datagrama en el nivel de transporte (TL, *Transport Layer*), etc.

A menos que se especifique, los campos deben verse como cadenas de octetos opacos con una codificación dependiente de la implementación. Es decir, si no se especifica, la codificación de los campos es libre siempre que el receptor y emisor se pongan de acuerdo previamente en la implementación a usar. La arquitectura definida aquí podría implementarse utilizando codificaciones habituales:

- Utilizando campos de tamaño predefinidos o añadiendo un subcampo previo que indique la longitud.
- Utilizando determinadas cadenas de octetos (podría ser un único octeto) como separadores de campos.
- Dividiendo cada campo en tres subcampos: tipo, longitud y valor (TLV).
- Utilizando lenguajes de marcado, etc.

Orden de transmisión

El orden de transmisión se hará utilizando el formato *little-endian* donde se transmiten primero los bits menos significativos de cada campo, siendo el último en transmitirse el bit más significativo. Si un campo está compuesto por varios octetos (un octeto son 8 bits), será enviado en orden desde el primer octeto en transmitirse, que es aquel que contiene el bit menos significativo, hasta el último octeto que contenga el bit más significativo.

Las representaciones gráficas de tramas en este documento mostrarán los bits en el mismo orden en el que serán transmitidos por el nivel físico, siendo por tanto el bit situado más a la izquierda del gráfico el primero en ser transmitido.

Figuras con topologías de red

La mayoría de las figuras con topologías de red se representarán como grafos etiquetados, donde los nodos son óvalos cuyo color de relleno depende de la distancia del nodo a otro especial, que siempre se muestra en rojo. Además del rojo, se usan 16 colores, con 4 tonalidades (verde, azul, morado y amarillo) que se van alternando con diferente intensidad. Si existieran distancias superiores a 16, los colores volverían a repetirse. En la Figura 1 se muestra un ejemplo de topología lineal, donde el número de cada nodo indica la distancia al nodo especial 0. En este documento el nodo especial será siempre una estación base.

Figura 1 Ejemplo de topología lineal.

Cita en género femenino

Las referencias a personas y colectivos figuran en el presente documento en género masculino como género gramatical no marcado. Cuando proceda, será válida la cita correspondiente en género femenino.

1 Introducción

Si no sabes hacia qué puerto zarpa tu barco, ningún viento te será favorable.

SÉNECA

En este capítulo se presentará la motivación y el contexto en el que surge esta tesis, los objetivos buscados, las contribuciones de esta Tesis Doctoral, las publicaciones realizadas y una breve descripción del resto de capítulos.

1.1 Motivación y contexto

Una red inalámbrica de sensores (WSN, *Wireless Sensor Network*) es un conjunto de dispositivos espacialmente dispersos equipados con capacidades de detección, computacionales y de comunicación inalámbrica que pueden autoorganizarse y colaborar para realizar tareas. Las WSN se utilizan en muchas aplicaciones diferentes [78]: militares [144], sanitarias [89], medioambientales y de monitorización de fauna y flora [95], agrícolas [63], industriales [37], etc. Las WSN dependen en gran medida de la aplicación y pueden formar parte integrante de sistemas Internet de las Cosas (IoT, *Internet of Things*). La diversidad de tipos de WSN es muy amplia, con diferentes requisitos y características que incluyen capacidades de dispositivos (homogéneas o heterogéneas), entidades estáticas o dinámicas, diferentes topologías, entre muchas otras.

Esta Tesis Doctoral se enfoca en las WSN utilizadas en la monitorización en entornos industriales o infraestructuras civiles, donde los nodos sensores recopilan mediciones y colaboran mediante comunicaciones ad hoc para entregarlas a un nodo sumidero, también conocido como estación base (BS, *Base Station*). Una vez desplegados, los nodos permanecen estáticos (WSN estática), excepto por cambios de ubicación ocasionales o la adición/eliminación de nodos.

La seguridad es crítica en muchas aplicaciones de monitorización. Las WSN tiene una gran superficie de ataque debido a una variedad de vulnerabilidades [167, 29] originadas

principalmente por: a) nodos con energía y recursos computacionales restringidos, que limitan sus funcionalidades; b) despliegue en entornos hostiles a menudo en posiciones desconocidas y de difícil acceso, lo que dificulta la reparación de los nodos; c) la operación desatendida y la gestión remota de la red que dificultan la verificación física del estado de los nodos y simplifican la realización de ataques físicos; d) alta exposición de los nodos a la desconexión, destrucción o modificación; e) fácil acceso al canal inalámbrico para espiar o inyectar paquetes; f) uso de una infraestructura de red ad hoc, donde los nodos confían entre sí; g) uso de una topología de red con frecuentes intercambios de información de enrutamiento que podría ser falsa; h) comunicación poco fiable, que dificulta distinguir entre ataques y errores; y e) uso de algoritmos distribuidos que involucran muchos nodos, que son propensos a ataques internos. Existen muchos ataques que se aprovechan de estas vulnerabilidades, como los descritos en [29, 150, 167]. Estos ataques pueden ser activos o pasivos, internos o externos, y se diferencian según su objetivo y nivel de comunicación empleado.

Entre los principales requisitos de seguridad que debe garantizar una WSN se encuentran la disponibilidad del servicio, la confidencialidad, integridad, autenticidad y frescura de los datos, autenticación y autorización de nodos, secreto hacia adelante y hacia atrás y localización segura de nodos críticos [167, 29]. Existen distintas medidas de seguridad, o contramedidas, para garantizarlos. Las principales contramedidas de seguridad pueden clasificarse como preventivas, Sistemas de Detección de Intrusos (IDS, *Intrusion Detection Systems*) y reactivas [150]. Las contramedidas preventivas fortalecen la WSN contra ciertos ataques antes de que ocurran. Un IDS identifica los ataques cuando ocurren. Por último, las contramedidas reactivas suelen complementar un IDS y se activan después del ataque para recuperarse y evitar nuevos intentos.

La mayoría de los trabajos existentes se centran en aportar soluciones a ataques específicos [29, 150]. Sin embargo, la superficie de ataque es amplia y se necesitan muchos mecanismos y protocolos para proteger una WSN. El uso de estos mecanismos en la misma WSN aumenta la complejidad y la demanda de recursos, lo que limita la viabilidad práctica. Muchos métodos adoptan enfoques centralizados [26, 3, 99, 168] en los que las entidades con más recursos, generalmente las BS, realizan las tareas más complejas, lo que también reduce la influencia de posibles nodos comprometidos en la toma de decisiones. Sin embargo, estos enfoques aumentan la dependencia del nodo central y requieren rutas para llegar a la BS, cuyo establecimiento también puede ser atacado.

Un robot es una entidad móvil dotada de capacidades computacionales, de detección y de comunicación. Recientemente, los robots aéreos se han convertido en herramientas comunes en tareas de monitorización como inspección, detección de fallos o mantenimiento predictivo, y esta tendencia está en auge [135, 111, 118]. Los robots pueden complementar las WSN aportando beneficios como su heterogeneidad, en el sentido de tener más recursos que los nodos, su movilidad y su capacidad de comunicarse directamente con los nodos. Por eso, los robots han obtenido resultados notables al ayudar a resolver una amplia variedad de problemas en WSN [138, 136], incluido el despliegue de nodos, la reparación de WSN, la recarga de baterías, la localización de nodos o la recopilación de datos. Sin embargo, la cooperación entre redes de sensores y robots para mejorar la seguridad de WSN aún no se ha investigado suficientemente [118].

En esta Tesis Doctoral se propone una arquitectura, métodos, protocolos y algoritmos para incrementar la seguridad en las WSN mediante la cooperación de robots en aplicaciones de inspección y monitorización y se ha aplicado para la monitorización de infraestructuras e instalaciones industriales.

Esta Tesis Doctoral se ha realizado en el contexto de los siguientes proyectos de investigación:

- H2020 RESIST[1] (ID 769066). Este proyecto financiado por la Unión Europea utiliza análisis predictivo e inspecciones con robots aéreos para aumentar la resiliencia física de puentes y túneles, permitir restaurar el tráfico rápidamente ante eventos extremos, y comunicar estos a usuarios y servicios de emergencia.
- H2020 PILOTING[2] (ID 871542). Este proyecto financiado por la Unión Europea desarrolla una plataforma integrada que, mediante soluciones robóticas, busca aumentar la eficiencia y calidad de las actividades de inspección y mantenimiento en infraestructuras como refinerías, puentes y túneles para mantener unos niveles de seguridad adecuados en ellas.
- H2020 AERIAL-CORE[3] (ID 871479). Este proyecto financiado por la Unión Europea desarrolla un sistema robótico cognitivo aéreo para ayudar a los operarios en las actividades de inspección y mantenimiento. En concreto, este proyecto lleva a cabo la integración de robots aéreos para la inspección muy precisa de grandes infraestructuras lineales.
- ROBMIND (Robots aéreos inteligentes para inspección y mantenimiento de instalaciones industriales) (PDC2021-121524-I00). Este proyecto financiado por el Ministerio de Ciencia e Innovación de España desarrolla nuevos prototipos robóticos tanto para inspecciones con y sin contacto como para tareas de mantenimiento sencillas. Esto permitirá a las empresas industriales adoptar un mantenimiento predictivo más ágil y frecuente.

1.2 Objetivos

El objetivo principal de esta tesis es explotar la cooperación entre robots y redes de sensores para mejorar el nivel de seguridad de estas y mantener la eficiencia en el consumo de recursos.

Se propone diseñar una nueva arquitectura de seguridad para WSN, que se denominará *Sensor Network Security using Robots* (SNSR), con una estructura iterativa de observación y reconfiguración que, mediante el uso de robots, mejore la prevención, detección y mitigación de ataques. Esta debe tener una alta flexibilidad y complementar las medidas de seguridad existentes, sin imponer protocolos, criterios de optimización o algoritmos. Debe aprovechar las capacidades de los robots, como la de localización basada en radio, la de comunicarse a corta distancia sin intermediarios y la de realizar observaciones directas.

[1] *https://resistproject.eu/*
[2] *https://piloting-project.eu/*
[3] *https://aerial-core.eu/*

La arquitectura SNSR requerirá la creación de nuevos protocolos y mecanismos de seguridad que sustituyan a otros que en las arquitecturas tradicionales suelen realizarse de forma distribuida o utilizan información preestablecida. Se pretende así reducir en gran medida la superficie de ataque y el consumo de recursos de los nodos.

Se debe concretar SNSR para el caso con un robot y una BS, pero con vistas a considerar en un futuro casos con más robots o BS. Por tanto, la flexibilidad y extensibilidad deben ser un requisito principal.

La implementación SNSR se hará usando tecnologías existentes bien probadas, sin depender de WSN estándares o privadas existentes, pero que pueda adaptarse a cualquiera. La evaluación y validación experimental se hará tanto teóricamente como en escenarios al aire libre realistas.

Se busca crear una plataforma o banco de pruebas (*testbed*) que permita la experimentación repetible y rigorosa de nuevas tecnologías, protocolos y métodos de seguridad basados en SNSR y que sirva como base para futuras investigaciones.

1.3 Contribuciones

Esta tesis ha generado las siguientes contribuciones:

Contribución 1: *Diseño de una nueva arquitectura de seguridad combinando robots y WSN.*

Esta contribución consiste en el diseño de una arquitectura de seguridad novedosa, genérica, abierta y flexible que combinando robots y WSN explota las sinergias entre el procesamiento heterogéneo y las capacidades de comunicación de los robots y los nodos de la WSN con el fin de proporcionar soluciones computacionalmente eficientes y seguras para los ataques más comunes. Esta arquitectura se ha llamado *Sensor Network Security using Robots* (SNSR) y contiene importantes novedades sobre los esquemas de seguridad de WSN tradicionales. Los robots implementan funcionalidades de seguridad que en los esquemas tradicionales son asumidas por la BS o por los propios nodos, permitiendo la comunicación en un solo salto entre el robot y cada nodo sensor, lo que mejora significativamente la seguridad.

Contribución 2: *Arquitectura SNSR con un robot y una estación base.*

Esta contribución consiste en adaptar la arquitectura genérica SNSR para el caso más básico en el que existe una única BS y como mucho un robot en funcionamiento. Este caso está pensado para redes de sensores pequeñas o medias con unos centenares de nodos como máximo. Se detallan y concretan todas las funcionalidades de cada una de las entidades de la red, las operaciones y su secuenciación, y los protocolos utilizados. Se mantiene la flexibilidad de todos aquellos aspectos en los que no afecta el número de robots o BS.

Contribución 3: *Sistema operativo de red genérico.*

En esta contribución se desarrolla un nuevo Sistema Operativo de Red (NOS, *Network Operating System*), que es el software que va a implementar una pila de red genérica

adecuada a la arquitectura SNSR. Utiliza el nivel de enlace proporcionado por el sistema operativo y proporciona el resto de los niveles: un nivel de adaptación, nivel de red, nivel de transporte y nivel de aplicación. Todos los niveles son originales. Además, proporciona servicios de seguridad, de gestión y la capacidad de ejecutar distintas aplicaciones independientemente del dispositivo. Este NOS es independiente del uso de SNSR, pero se ha diseñado teniendo en cuenta exclusivamente sus requisitos. Es básico, no depende de WSN estándares o privadas existentes, por lo que prescinde de mecanismos innecesarios en SNSR, pero es fácilmente ampliable. Destacan la posibilidad de usarlo tanto en equipos reales como virtualizados, el uso reducido de recursos conforme a las limitaciones de los nodos sensores y la posibilidad de adaptarlo a otros sistemas operativos gracias al uso de niveles de adaptación. El NOS permite probar la arquitectura SNSR en equipos reales. Además, es lo suficientemente flexible para permitir probar distintas configuraciones, algoritmos, protocolos, etc.

Contribución 4: *Aplicaciones de gestión de robots, estaciones base y nodos sensores para SNSR.*

En esta contribución se han implementado las aplicaciones de gestión de los tres tipos de entidad de SNSR: robots, estaciones base y nodos sensores. Estas aplicaciones de gestión cumplen las especificaciones, el protocolo y el comportamiento descrito en la arquitectura SNSR. Cada aplicación de gestión también implementa una lista de comandos administrativos, dependientes del tipo, para poder controlar su funcionamiento y obtener información de estado durante los experimentos. Estas aplicaciones se ejecutan sobre el NOS de la *Contribución 3*.

Contribución 5: *Compositor/orquestador de escenarios y sistema de gestión remota de dispositivos.*

Debido a la novedad de SNSR, para realizar experimentos en escenarios reales y virtualizados/simulados se ha creado una plataforma de pruebas nueva. Un escenario es una instalación industrial real o simulada con una WSN que debe tener un nivel de seguridad concreto utilizando al menos un robot. En un escenario también pueden participar agentes maliciosos. Esta contribución consiste en la creación de dos componentes, uno para escenarios reales y otro para escenarios virtuales. Para los escenarios reales se ha creado un sistema de gestión remota de dispositivos que permite monitorizar las versiones instaladas del software en cada dispositivo, así como realizar actualizaciones de éste. También permite cargar distintas configuraciones para poder hacer pruebas en distintos escenarios. Para los escenarios virtuales se ha creado un compositor/orquestador de escenarios, que se ha denominado *lanzador*, que permite probar exactamente el mismo software que en escenarios reales. Existen dos versiones distintas del *lanzador* con dos formas diferentes de simular los canales de comunicación. Las propiedades de estos se pueden modificar durante el experimento. Con el *lanzador* se puede controlar la ejecución de cada entidad, planificar eventos, simular movimientos y capturar tráfico de la red. En una de las versiones del *lanzador* también se ha creado un módulo de movimiento simulado que es capaz de calcular las trayectorias del robot y moverlo según su estado. Con todo esto, se han podido hacer miles de experimentos con repeticiones de manera sencilla.

Contribución 6: *Experimentación y validación de todas las contribuciones anteriores.*

Para validar la implementación y la plataforma de pruebas, y caracterizar el comportamiento de SNSR se han realizado miles de experimentos con distintos escenarios virtuales y reales con robots terrestres y aéreos. En estos experimentos se han probado diferentes medidas de seguridad, configuraciones, número de nodos y topologías en condiciones normales y con ataques. Los resultados han demostrado la seguridad, eficiencia y escalabilidad de SNSR.

1.4 Publicaciones

Las contribuciones de esta Tesis Doctoral han dado como resultado las publicaciones científicas que se enumeran a continuación. Destaca el artículo de 17 páginas publicado en la revista *IEEE Internet of Things Journal* cuyo factor de impacto en 2021 es de **10,238** y es **Q1** en todas sus categorías.

También se ha publicado un conjunto de datos experimentales (*dataset*). El código desarrollado en esta Tesis Doctoral está pendiente de ser publicado.

1.4.1 Artículos en revistas indexadas

1. Francisco Jose Fernandez-Jimenez and Jose Ramiro Martinez De Dios, *A Robot-Sensor Network Security Architecture for Monitoring Applications*, IEEE Internet of Things Journal 9 (2022), no. 8, 6288–6304. [41]
   Indicios de calidad del año 2021 (*Web of Science*): *Journal Impact Factor* = **10,238**; Categoría *COMPUTER SCIENCE, INFORMATION SYSTEMS* = 9/164, Q1, Decil 1, Percentil 94,82; Categoría *ENGINEERING, ELECTRICAL & ELECTRONIC* = 18/276, Q1, Decil 1, Percentil 93,66; Categoría *TELECOMMUNICATIONS* = 6/93, Q1, Decil 1, Percentil 94,09.
2. José Ramiro Martínez-de Dios, Anibal Ollero, Francisco José Fernández, and Carolina Regoli, *On-Line RSSI-Range Model Learning for Target Localization and Tracking*, Journal of Sensor and Actuator Networks 6 (2017), no. 3. [122]
   Indicios de calidad del año 2021 (*Web of Science*): *Journal Impact Factor* = (no disponible); Categoría *TELECOMMUNICATIONS* = 48/115, Q2, Percentil 58,70.
3. Javier Munoz-Calle, Francisco J. Fernandez-Jimenez, Teresa Ariza, Antonio J. Sierra, and Juan M. Vozmediano, *Computing Labs on Virtual Environments: A Flexible, Portable, and Multidisciplinary Model*, IEEE Revista Iberoamericana de Tecnologias del Aprendizaje 11 (2016), no. 4, 235–341. [102]
   Indicios de calidad del año 2021 (*Web of Science*): *Journal Impact Factor* = (no disponible); Categoría *COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS* = 132/156, Q4, Percentil 15,71.

1.4.2 Aportaciones a congresos internacionales

1. Francisco J Fernández-Jiménez and J Ramiro Martínez-de Dios, *Design of a Robot-Sensor Network Security Architecture for Monitoring Applications*, ROBOT 2017:

Third Iberian Robotics Conference, Springer International Publishing, 2018, pp. 200–212. [40]

1.4.3 *Datasets*

1. Material de varios experimentos virtuales y reales realizados, como versiones de información de topología, resultados obtenidos y ejemplos de los certificados utilizados. También se incluye la definición de la información de topología y los mensajes de la aplicación de gestión utilizando el formato *Protocol Buffers*. [42]

1.5 Estructura del documento

El resto del documento está estructurado como se indica a continuación.

El **Capítulo 2** resume el estado del arte de la seguridad en las WSN y la colaboración entre robots móviles y las WSN. Se muestra la taxonomía de los principales trabajos descritos y se hace una comparación con SNSR. También se describen con más detalle el protocolo INSENS [26] y el estándar ZigBee con el perfil de aplicación ZigBee Smart Energy [8, 113] cuya seguridad será analizada en el Capítulo 5.

El **Capítulo 3** presenta el diseño de la nueva arquitectura de seguridad SNSR que combina robots y redes de sensores. Se analizan los requisitos que debe cumplir la torre de protocolos de la WSN para poder implementarla. Se describen las fases de funcionamiento, las relaciones entre tipos de entidades, protocolos de comunicación, información almacenada y parámetros de configuración para el caso de usar un robot y una BS. Este capítulo desarrolla la *Contribución 1* y la *Contribución 2* de esta Tesis Doctoral.

El **Capítulo 4** explica el NOS desarrollado capaz de ejecutarse en distintos equipos (robot, BS y nodos sensores) que implementa las funcionalidades descritas en la arquitectura SNSR y que no depende de una WSN concreta. Dada la flexibilidad de SNSR, también se detallan las decisiones tomadas a la hora de implementar la arquitectura. Se describe el compositor/orquestador de escenarios virtuales para probar estas redes sin necesidad de hacerlo en un entorno real, y el sistema de gestión remota que permite controlar los dispositivos empleados en escenarios reales. También se realiza una descripción de herramientas utilizadas para simular ataques. Este capítulo desarrolla la *Contribución 3*, la *Contribución 4* y la *Contribución 5* de esta Tesis Doctoral.

El **Capítulo 5** muestra un análisis comparativo de seguridad y detalla los experimentos llevados a cabo para validar la arquitectura SNSR y la implementación desarrollada junto con los resultados obtenidos. Se han realizado pruebas en laboratorio así como experimentos con robots terrestres y aéreos en entornos de monitorización de infraestructuras y de industrias. Este capítulo desarrolla la *Contribución 6* de esta Tesis Doctoral.

El **Capítulo 6** resume las principales conclusiones de esta Tesis Doctoral y propone líneas futuras de trabajo e investigación.

2 Estado del arte

Grandes descubrimientos y mejoras implican invariablemente la cooperación de muchas mentes.

Alexander Graham Bell

En este capítulo se hace una revisión de trabajos cuyo objetivo es mejorar la seguridad de una WSN y de aquellos que han empleado robots en colaboración con ellas, que es la idea principal de esta Tesis Doctoral. Este estado del arte permite tener una panorámica de la situación de la que parte esta Tesis Doctoral y las novedades que aporta respecto a trabajos previos.

La Sección 2.1 comienza con una visión general de la diversidad de las WSN existentes y las características de estas que dificultan encontrar una arquitectura de seguridad genérica. A continuación, en la Sección 2.2, se describen las vulnerabilidades habituales y los ataques conocidos. Después, en la Sección 2.3, se muestran las medidas de seguridad desarrolladas en otros trabajos.

La segunda parte del capítulo comienza en la Sección 2.4, que revisa la colaboración entre robots móviles y las WSN. Se muestran las tareas que han realizado los robots en este contexto y se analizan trabajos donde contribuyen a mejorar la seguridad de la WSN.

En la tercera parte, comenzando en la Sección 2.5.1, se muestra una taxonomía de los principales trabajos descritos en las partes anteriores, para dar una visión general al lector de las características de esos trabajos. Después, en la Sección 2.5.2, se hace una comparación de SNSR respecto a los trabajos anteriores.

Por último, en la Sección 2.6 y la Sección 2.7, se explican con más detalles aquellos trabajos y especificaciones cuya seguridad será analizada cualitativamente en el Capítulo 5 y que será comparada con la seguridad de SNSR. Puede consultar en la Sección 5.2 este análisis, así como la justificación de la elección.

Además de la literatura comentada en este capítulo, para la realización de esta Tesis Doctoral se han tenido que consultar otros trabajos relativos a problemas secundarios que han surgido en la implementación y experimentación de SNSR. Se comentarán brevemente estos trabajos en las secciones correspondientes de capítulos posteriores.

2.1 Redes inalámbricas de sensores

Genéricamente, una WSN es un conjunto de dispositivos o nodos sensores equipados con sensores que monitorizan el entorno y que colaboran para hacer llegar la información obtenida a un nodo sumidero, también conocido como nodo pasarela o BS, usando comunicaciones inalámbricas. Es un tipo de red inalámbrica ad hoc especializada para las condiciones anteriores. Las WSN han sido usadas en muchos campos distintos [78] y son muy dependientes de la aplicación para la que se van a utilizar. A partir del concepto general surgen una gran diversidad de tipos de WSN, con requisitos y características diferentes. Se pueden destacar las siguientes características de una WSN que influyen en las tecnologías necesarias para su implantación [78, 29, 120, 56]:

- **Diversidad en las capacidades de los dispositivos**. Los dispositivos se clasifican en distintos tipos según las capacidades que afectan a la aplicación concreta, como por ejemplo el consumo de energía y acceso a esta (usando la red eléctrica, con baterías, captando energía del medio, ...), la potencia computacional, el almacenamiento disponible, etc. [78]. Los nodos sumideros suelen ser considerados diferentes a los nodos sensores. En cuanto a los nodos sensores, estos podrían ser considerados todos similares (red homogénea) o pertenecer a distintos tipos (red heterogénea). En una red heterogénea, los nodos sensores suelen realizan distintas funciones según su tipo. En redes homogéneas, los nodos también podrían realizar funciones especiales asignadas por turnos a distintos nodos. Habitualmente se buscan nodos sensores de bajo costo, de reducidas dimensiones y eficientes en consumo de energía. Esto es crítico en aplicaciones donde se usan nodos sensores que no se pueden recuperar una vez que agotan su batería. Lo anterior implica que normalmente los nodos sensores van a tener recursos muy limitados. Con estos dispositivos de bajo costo, la robustez se consigue mediante estrategias donde se emplean muchos nodos y su operación se adapta para conseguir mantener el funcionamiento esperado de la red. Gran parte del esfuerzo investigador en los últimos años se ha centrado en el objetivo de diseñar WSN eficientes cuyo tiempo de funcionamiento sin tener que intervenir sea el mayor posible [51].
- **Redundancia de funcionalidad**. Los nodos sensores pueden estar desplegados de tal manera que ninguno de ellos sea imprescindible, ya que su misma funcionalidad la puede proporcionar otro nodo. Por ejemplo, varios nodos pueden tomar medidas de la misma variable, y cualquiera de ellas sería suficiente. Esto a priori es deseable al hacer la red más robusta, pero a veces puede ser imposible realizarlo o entrar en conflicto con otros requisitos como el coste en dispositivos, despliegue, transmisión y procesamiento de la información y de mantenimiento. La redundancia disponible afecta a la posibilidad de filtrar y detectar medidas erróneas y el esfuerzo necesario para garantizar la fiabilidad de las comunicaciones, proteger nodos críticos, detectar nodos caídos y reemplazarlos.
- **Política de recuperación ante fallos en nodos**. Los nodos pueden tener fallos parciales o totales, fortuitos o intencionados. Por ejemplo, un nodo podría agotar su batería o haber sido comprometido por un atacante y comportarse de manera

incorrecta. Si la redundancia es alta en la red, se puede ignorar o excluir el nodo sin mayor inconveniente. Pero si la red no tiene redundancia o esta cae por debajo de una calidad de servicio admisible, los nodos deben ser sustituidos o reparados. El coste de un nodo, el coste del despliegue inicial de un nuevo nodo y el de reparación suelen decidir la mejor acción a tomar. Por ejemplo, las WSN usadas para monitorizar el medio ambiente suelen usar nodos de muy bajo costo que se despliegan densamente en el área objetivo. En estos casos, cuando el número de nodos con fallos es excesivo lo normal es volver a repoblar el área desplegando nuevos nodos. En otros casos, con nodos sensores accesibles, es más eficiente intentar realizar la reparación evitando tener que hacer reconfiguraciones adicionales.

- **Número de sumideros o BS**. Las WSN más simples suelen contar con una única BS. Pero también existen otras con múltiples BS para aumentar la robustez de la red, disminuir la distancia que deben recorrer los datos y reducir la congestión, entre otros beneficios. Los nodos pueden enviar las medidas a una BS seleccionada o a varias de manera simultánea o alternativa. Sin embargo, tener varias BS puede traer consigo inconvenientes, como la necesidad de agregar la información recopilada en todas las BS y de crear mecanismos de selección de BS en los nodos sensores y el posible aumento en los mensajes transmitidos.

- **Dinamismo en el número de nodos**. Esto hace referencia a la frecuencia con la que se añaden o se eliminan nodos. Influye en aspectos tales como los protocolos de enrutamiento usados, el descubrimiento mutuo entre los nodos y la red, el proceso de reconfiguración y la autenticación de nuevos nodos. Un dinamismo alto implica WSN más complejas y dedicar más energía a mantener la conectividad y seguridad de la red.

- **Estáticas o con elementos móviles**. En las WSN más comunes todas las entidades (nodos sensores y sumideros) que pertenecen a ella son estáticas, permaneciendo en la misma posición desde que son desplegadas. En estos casos se dice que la WSN es estática. En otros casos, es posible que sumideros, nodos sensores o ambos sean móviles. Si hay muchos elementos móviles, se habla de WSN móviles y se aplican técnicas similares a las redes ad hoc móviles (MANETs, *Mobile Ad hoc Network*s) y redes ad hoc vehiculares (VANETs, *Vehicular Ad hoc Networks*s) adaptándolas a dispositivos con menores recursos.

- **Tipos de interacciones entre entidades de la red**. Los elementos que se pueden relacionar entre sí y si la comunicación es bidireccional, unidireccional, por difusión o multidifusión es otra característica que puede cambiar de una WSN a otra. En las WSN más simples los sensores envían sus medidas a la BS en mensajes unidireccionales y la BS anuncia su disponibilidad mediante difusión por inundación. Para la transmisión fiable de los datos se requiere comunicación bidireccional. En otras, los nodos sensores tienen que colaborar entre sí para formar y mantener la red. Por último, existen redes cuya actividad va más allá de sensorizar, donde existen nodos actuadores y se permite la comunicación bidireccional y grupal entre todas las entidades. Un ejemplo serían las WSN industriales donde los nodos participan en el control de procesos.

- **Topología de la red**. En algunas WSN se busca una determinada topología de la red. El alcance radio y características de los dispositivos, la eficiencia energética o la necesidad de redundancia en la red son factores que influyen. Además, la topología tiene implicaciones en el direccionamiento y el enrutamiento usado. Las topologías más habituales son las siguientes:
  - **Topología en estrella**. Los nodos se conectan a un nodo central que actúa como conmutador entre ellos. En redes simples, la BS es el nodo central y los nodos sensores no necesitan implementar nivel de red. Se conocen también como redes de un solo salto debido a que los sensores están conectados a la BS, a diferencia de las redes multisalto. Son fáciles de desplegar y ampliar. Sin embargo, tiene algunos inconvenientes: solo son posibles si el alcance radio de los sensores lo permite, el nodo central se convierte en un único punto de fallo y cuello de botella, la probabilidad de colisiones durante las transmisiones aumenta y los nodos alejados gastan mucha más energía al transmitir debido a la atenuación (el consumo de la comunicación vía radio aumenta en función del cuadrado de la distancia). A veces, aunque esta topología sea posible, muchas veces se prefieren topologías multisalto ya que transmitir en n saltos pequeños controlando la potencia de transmisión consume aproximadamente n veces menos que en un único salto.
  - **Topología en árbol**. Existe un nodo raíz (habitualmente la BS) al que se conectan otros nodos que forman el siguiente nivel de la jerarquía y así sucesivamente. El consumo energético es mayor en los niveles más próximos al nodo raíz debido al procesado y reenvío de mensajes de otros nodos. La formación del árbol no suele venir dada por las condiciones físicas de la red, sino siguiendo reglas preestablecidas cuando un nodo descubre a su vecindad. El direccionamiento y enrutamiento suelen ser sencillos y de bajo coste. Los nodos hojas pueden ser más simples que el resto ya que no retransmiten mensajes de otros nodos. Sin embargo, existe un único punto de fallos global y cada nodo es crítico para los que están debajo de él, por lo que cuando un nodo falla suele ser necesario regenerar el árbol. Un caso límite es la topología lineal, utilizadas en algunas WSN destinadas a monitorizar infraestructuras lineales como oleoductos y líneas eléctricas.
  - **Topología mallada**. Los nodos se comunican con los vecinos que están a su alcance como iguales y todos funcionan como encaminadores retransmitiendo mensajes. El número máximo de vecinos con los que se relaciona cada nodo puede estar limitado. La malla se dice que es completa cuando cada nodo está conectado con todos los demás, y en caso contrario se denomina malla parcial. En redes cableadas esta topología suele ser costosa debido al tendido de cableado adicional que involucra, pero en redes inalámbricas no existe ese problema. Sin embargo, requiere que todos los nodos sean similares y aumenta el consumo energético destinado a mantener las conexiones con vecinos. Esta topología es más robusta que las anteriores debido a que puede existir más de un camino posible entre dos nodos. Los criterios para elegir el camino que debe seguir un paquete pueden diferencia a una WSN de otra.

Una misma red puede tener distintas topologías simultáneamente, sobre todo en redes heterogéneas.

- **Agrupación de nodos en clústeres o no**. Existen WSN que internamente se subdividen en grupos de nodos o clústeres [110]. En cada clúster se elije un nodo cabeza de clúster (CH, *Cluster Head*). Solo el CH se comunica con el resto de la red y los otros nodos del clúster solo se comunican entre ellos y el CH. Al CH también se le suelen asignar otras funciones especiales si las hubiera, destacando la agregación y fusión de información antes de enviarla a la BS. Se hace que los nodos CH tengan un mayor gasto energético para reducir el consumo del resto de nodos. El resultado es una red jerárquica que busca mejorar la escalabilidad y la eficiencia energética. La elección del CH es un problema adicional que deben resolver estas redes, ya sea con algoritmos distribuidos o centralizados. En redes heterogéneas es habitual escoger a los nodos con más recursos. En redes homogéneas es posible asignar esta tarea por turnos. Los nodos CH forman una red inter-clúster entre ellos que no tiene que tener las mismas características que la subred intra-clúster. Por ejemplo, pueden tener distinta topología, encaminamiento, etc. A las redes que no contemplan la agrupación en clústeres se les suele denominar redes planas.
- **Diversidad en el enrutamiento**. Normalmente el enrutamiento y el direccionamiento usado es consecuencia del resto de características y requisitos de la WSN, pero a veces viene impuesto por compatibilidad con los dispositivos usados, para poder hacer comparativas entre distintos protocolos de enrutamiento o para permitir el acceso desde el exterior cuando la WSN se integra en arquitecturas de IoT. Existen multitud de protocolos de enrutamiento y distintos criterios de clasificación [6, 131]. Tienen gran influencia en el consumo energético global y la seguridad de la red.
- **WSN determinista o no determinista**. Una WSN se considera determinista cuando es posible determinar con anterioridad al despliegue la posición que ocupará cada sensor. Con lo anterior es posible calcular y configurar todos los nodos sensores antes de su despliegue. Esto solo es aplicable a muy pocas redes, y normalmente la posición exacta de los sensores será desconocida, puede ser variable e incluso pueden añadirse nuevos nodos más adelante. En estos casos, la red se denomina no determinista y requiere sistemas más complejos y adaptativos.
- **Cuándo y cómo se envían las medidas**. Las medidas pueden enviarse de manera periódica, al producirse un cambio significativo en el valor monitorizado o por petición expresa. Los requisitos de retardo y fiabilidad de transmisión difieren de una aplicación a otra [37].
- **Autonomía de los nodos sensores**. El grado de autosuficiencia de los nodos y si la red es autoconfigurable también puede ser diferente según la WSN. Los nodos pueden ser activos e iniciar la formación de la red si no detectan ninguna a la que unirse o esperar a detectarla. Pueden anunciar la disponibilidad de datos y descubrir al sumidero o esperar que este se anuncie o pida los datos. Aunque las WSN como redes ad hoc se pueden autoconfigurar y autoorganizar sin necesidad de infraestructura previa, esto a veces implica un consumo energético no adecuado para determinadas aplicaciones y además puede conllevar problemas de seguridad. Por

consiguiente, en algunas arquitecturas se utilizan elementos centralizados o nodos con funcionalidades especiales encargados de la configuración y organización de la red.

- **Dominios administrativos, federación y compartición de la red**. La propiedad y gestión de una red puede involucrar a una única organización o a varias. La red puede tener un uso exclusivamente privado o ser pública y permitir su uso por otras organizaciones. La participación de múltiples organizaciones trae consigo problemas de incompatibilidad, seguridad y confianza que pueden requerir una negociación previa antes del despliegue. Se dice que las redes tienen un dominio administrativo único cuando todos los participantes pertenecen a una misma organización o comparten un punto de confianza común, que permite negociar con antelación las credenciales y mecanismos de seguridad que se emplearán. Si la red tiene dominios administrativos múltiples, los participantes no tienen que haber tenido contacto previo y pertenecen a distintas organizaciones y dominios de seguridad. Esto imposibilita la negociación previa y requiere el establecimiento ad hoc de parámetros compatibles. Por otro lado, la federación consiste en la colaboración de múltiples redes con distintos dominios administrativos para un fin común, y la compartición de la red implica que una misma red se utilizar para proporcionar varios servicios independientes a la vez [64]. Tanto la federación y la compartición implica que la red debe estar preparada para aplicar distintos parámetros de seguridad simultáneamente.
- **Diversidad en la seguridad exigida**. La seguridad tiene un coste de computación, de almacenamiento volátil y no volátil y de transmisión. Además, puede ser incompatible con algunas funcionalidades de la red, especialmente con la agregación y fusión de datos si no se desea que el nodo que realiza esta tarea tenga acceso a los datos en claro. En este último caso las técnicas de cifrado estándar no lo permitirían y se necesitaría emplear cifrado homomórfico [7], que permite realizar operaciones algebraicas sobre los datos cifrados. No todas las aplicaciones tienen los mismos requisitos de seguridad. Estos requisitos pueden ser de disponibilidad del servicio, integridad de los datos, confidencialidad de la información, privacidad de los usuarios, autorización, autenticación, no repudio y frescura de los mensajes [29]. Por ejemplo, una posible aplicación puede permitir que los datos sean accesibles públicamente, por lo que no le puede interesar que vayan cifrados, y aún así quiera que se garantice la autenticidad de ellos. Los administradores de una WSN deben evaluar la probabilidad de que se produzcan ataques, el valor y privacidad de los datos transmitidos, el coste de aceptar información falsa, la facilidad de acceder a los dispositivos por terceras personas y la confianza de los participantes entre otros criterios para definir el nivel de seguridad requerido.

En definitiva, existe una gran variedad de tipos de WSN [78, 29, 120, 56]. Muchas veces las distintas capacidades requeridas compiten entre sí y puede que una funcionalidad que se introduce para optimizar unas de las características implica el empeoramiento de otra, por lo que hay que adoptar criterios que optimicen el funcionamiento global.

Esta variedad también se refleja en el número de estándares y tecnologías comerciales existentes que muchas veces compiten entre sí y otras veces se complementan. En [37] se describen muchas de ellas aplicables a una red de sensores inalámbrica industrial (IWSN,

Industrial Wireless Sensor Network) o una red de sensores y actuadores inalámbrica industrial (IWSAN, *Industrial Wireless Sensor and Actuator Network*), destacando el IEEE 802.15.4 [1]. Este estándar define los niveles físicos y de Control de Acceso al Medio (MAC, *Media Access Control*) para su uso en redes inalámbricas de área personal con tasas bajas de transmisión de datos (LR-WPAN, *Low-rate Wireless Personal Area Network*). Su versión inicial fue publicada en el 2003 y desde entonces se ha ampliado mediante modificaciones y revisiones (que consolidan las modificaciones previas). En la actualidad incorpora multitud de niveles físicos y MAC lo que le proporciona mucha flexibilidad y es la base de otros protocolos de alto nivel como ZigBee (que se detallará en la Sección 2.6), ISA100.11a (también conocido como IEC 62734 o ISA100), 6LowPAN (del IETF, detallado en los RFC 4919 y RFC 4944), 6TiSCH (RFC 9030 del IETF), WirelessHART o WIA-PA. Entre los posibles competidores destacan Bluetooth (IEEE 802.15.1), sobre todo la versión *Low Energy*, Wireless IO-Link (IEC 61139-3), Wi-Fi (la familia de estándares IEEE 802.11, especialmente el IEEE 802.11ah o Wi-Fi HaLow desarrollado pensando en IoT o la última versión, IEEE 802.11ax o Wi-Fi 6, que reduce el consumo y aumenta la robustez), y estándares de red de área amplia de baja potencia (LPWAN, *Low Power Wide Area Network*) como pueden ser LoRaWAN, Sigfox, NB-IoT o LTE-M. Dado que en esta Tesis Doctoral no nos centraremos en los niveles bajos, no se detallarán las características de las tecnologías antes enunciadas, pero sí se han tenido en cuenta las diferencias a la hora de buscar una solución de seguridad que pueda ser adaptable a distintos niveles físicos y de enlace.

Las WSN pueden formar parte de sistemas IoT. En estos sistemas se puede habilitar la comunicación con los sensores y actuadores de una WSN desde el exterior de esta, a través de pasarelas. En IoT por consiguiente aumentan los riesgos relativos a la privacidad y seguridad y hay que resolver problemas relativos al control de acceso externo que antes no surgían. No es el objetivo de esta Tesis Doctoral analizar la seguridad de las aplicaciones concretas que se ejecutan en la WSN, incluyendo las relacionadas con IoT.

El resto del estado del arte se focalizará en las WSN usadas para la monitorización de instalaciones industriales y grandes infraestructuras civiles, donde los nodos, una vez desplegados, permanecen estáticos a excepción de cambios de posición ocasionales y donde pueden también añadirse o quitarse nodos a lo largo del tiempo. En muchas de estas aplicaciones de WSN para la monitorización, la seguridad es un aspecto crítico.

2.2 Vulnerabilidades y ataques en WSN

Las WSN tienen una gran superficie de ataque, entendida como el conjunto de vulnerabilidades y técnicas, también conocidas como vectores de ataques, que puede utilizar un usuario no autorizado (atacante) para lograr que un ataque tenga éxito. Estas vulnerabilidades se originan principalmente por las siguientes características de las WSN [167, 29, 51, 150]:

- Los nodos sensores son frecuentemente de bajo costo y tienen limitados sus recursos computacionales y energéticos. Esto restringe las operaciones que pueden realizar. Es habitual desplegar los nodos sensores en entornos donde no hay disponible una

fuente de energía constante por lo que tienen que operar con baterías. Opcionalmente pueden usar captación de energía de fuentes renovables que no garantizan el suministro totalmente. Muchas aplicaciones requieren el empleo de un gran número de sensores y solo son viables económicamente si el coste de estos es bajo.

- El uso de distintos tipos de nodos con diferentes capacidades y mecanismos de seguridad, lo que implica un compromiso entre la compatibilidad y la seguridad. No sirve de nada tener nodos que admitan un alto grado de seguridad si no pueden hacer uso de ella para comunicarse con otros que no lo soportan.
- El despliegue en entornos hostiles y a menudo en posiciones desconocidas y difíciles de acceder. Esto dificulta la reparación o recuperación de los nodos.
- Funcionamiento desatendido y gestión de red remota que complica la verificación física del estado de los nodos y facilita la realización de ataques físicos o la planificación de ataques complejos.
- Alta exposición de los nodos a la desconexión, destrucción o modificación. Muchas veces es imposible prevenir que terceras personas puedan acercarse a los nodos.
- El uso de transmisión inalámbrica hace que sea sencillo el acceso al canal de comunicaciones, por ejemplo, para crear interferencias, espiar los datos transmitidos o inyectar paquetes.
- Una topología de red dinámica que requiere intercambios de información de encaminamiento frecuentes, que además podría ser falsa.
- La comunicación no fiable, que dificulta distinguir entre ataques y errores del canal no malintencionados.
- La utilización de una infraestructura de red ad hoc, donde los nodos deben confiar en otros para poder realizar su función.
- El empleo de algoritmos distribuidos que involucran muchos nodos, que son propensos a ampliar los efectos de ataques internos (con nodos infiltrados o mediante la modificación maliciosa de nodos existentes).

Una superficie de ataque tan extensa hace que asegurar una WSN sea una tarea compleja. Multitud de ataques se aprovechan de las anteriores vulnerabilidades. En la literatura existen revisiones detalladas de ataques genéricos en WSN, como las realizadas en [29] (incluye 23 ataques), [150] (describe 21 ataques) y [167] (identifica 24 ataques). También existen revisiones especializadas en componentes concretos de la red, por ejemplo, la que se realiza en [155] de los posibles ataques al protocolo RPL (*IPv6 Routing Protocol for Low-Power and Lossy Networks*) usado en 6LowPAN. Estas revisiones suelen además describir los requisitos de seguridad según el tipo de red, ordenar los ataques por categorías y describir las técnicas de defensa y contramedidas que se puede adoptar contra cada uno de ellos, y ejemplos de trabajos que las implementan. Más adelante, en la Sección 5.2, se hará un análisis cualitativo de seguridad del trabajo realizado respecto a otros existentes, donde se explicarán 33 posibles ataques con sus consecuencias. En este documento se han usado los nombres de los ataques en inglés para facilitar su consulta en la literatura y por homogeneidad.

Los ataques pueden ser clasificados como [29, 150, 167]:

- **Activos o pasivos**. La diferencia está entre si el ataque implica interactuar con la red o tener acceso físico a ella, o no. Los ataques de *eavesdropping* (ver Sección 5.2.27) y algunas variantes de *traffic analysis* (ver Sección 5.2.28) son ejemplos de ataques pasivos. La mayoría del resto de ataques son activos, por ejemplo, los de *jamming* (ver Sección 5.2.2). Los ataques pasivos son muy difíciles de detectar al no dejar ningún rastro. Los ataques activos, son a su vez clasificados por [30], en 4 subcategorías: los que introducen paquetes en la red, los que introducen ruido, los que introducen paquetes y ruidos, y los que modifican el software o firmware de los nodos.
- **Externos o internos**. En los primeros, los atacantes no pertenecen a la red que se quiere atacar, mientras que, en los segundos, el atacante es una de las entidades que ya forman parte de la red y que está actuando maliciosamente. El ataque *selective forwarding* (ver Sección 5.2.20) es un claro ejemplo de ataque interno, mientras que el de *tampering* (ver Sección 5.2.3) es uno externo. Normalmente un ataque interno requiere que, previamente, un ataque externo haya comprometido una entidad de la red o que la red esté abierta a cualquier participante.
- **Según el nivel OSI donde se produce el ataque**. Se pueden producir ataques en cualquiera de los niveles de comunicaciones. Hay ataques cuya mecánica puede ser utilizada en varios niveles dependiendo de la torre de protocolos que use la red, debido a que algunos niveles realizan operaciones similares. Por ejemplo, los ataques de *acknowledgment spoofing* (ver Sección 5.2.12), *rushing attack* (ver Sección 5.2.16) o *service request power attack* (ver Sección 5.2.9) podrían usarse en distintos niveles, aunque se suelen clasificar por el nivel que suele ser más vulnerable a esos ataques.
- **Según el objetivo del ataque**. Pueden ser ataques a: a) la integridad y disponibilidad del servicio, lo que se conoce como de Denegación de Servicio (DoS, *Denial of Service*); b) a la privacidad y confidencialidad de la información; o c) la integridad de los datos. Los ataques *node destruction* (ver Sección 5.2.4) y *black hole* (ver Sección 5.2.18) son ejemplos del primer grupo, los ataques *traffic analysis* (ver Sección 5.2.28) e *information disclosure* (ver Sección 5.2.26) del segundo, y *packet injection* (ver Sección 5.2.32) y *node replication* (ver Sección 5.2.31) del último.

Múltiples atacantes pueden coordinarse para realizar ataques como en el *byzantine attack* (ver Sección 5.2.23) y pueden realizarse múltiples ataques simultánea o secuencialmente. Un atacante también puede actuar en instantes aleatorios o muy lentamente para evitar ser detectado.

Por último, hay que destacar que existen muchos ataques de tipo *jamming*. En esta Tesis Doctoral no se tratará la seguridad de los niveles inferiores. También hay muchos relacionados con los protocolos de encaminamiento. El motivo es que, en las WSN como ocurre en otras redes ad hoc, los protocolos de encaminamiento suelen ser distribuidos y pueden afectar mucho al consumo energético de la red. Esto hace que haya que buscar un equilibrio entre el gasto requerido para mantener el enrutamiento y la tarea principal de sensorización, y elegir entre protocolos sencillos que no sobrecarguen a los nodos y

otros eficientes y seguros. Muchas veces la información de enrutamiento es intercambiada sin verificación debido al dinamismo de las redes. La mayoría de los protocolos de encaminamiento no contemplan medidas de seguridad (o son muy simples), aunque también se han desarrollado bastantes protocolos seguros, como los descritos en [164, 141]. Relacionado con lo anterior, también está el problema de hacer la difusión de mensajes segura [117, 166].

2.3 Medidas de seguridad para WSN

Entre los principales requisitos de seguridad que debe cumplir una WSN (muchos de ellos aplicables a cualquier tipo de red) se encuentran [167, 29]:

- **Disponibilidad del servicio**. Ante perturbaciones, la red debe mantenerse en funcionamiento y los nodos deben continuar realizando su tarea.
- **Confidencialidad**. El acceso a los datos solo debe estar permitido a entidades autorizadas.
- **Integridad**. Se debe evitar las modificaciones de los datos no autorizadas.
- **Autenticidad**. Es posible verificar la identidad del emisor de cualquier mensaje.
- **Frescura de los datos**. Se puede comprobar que los datos son recientes y no se han procesado previamente, de tal manera que la información duplicada o la repetición de solicitudes pasadas sean descartadas.
- **Autenticación y autorización de nodos**. Los nodos deben autenticarse antes de solicitar servicios o datos que requieran de ciertos privilegios.
- **Secreto hacia delante y atrás**. Las claves usadas para el cifrado se renuevan cada cierto tiempo y son independientes entre sí. El conocer la clave usada en un determinado instante no permite averiguar las claves anteriores o futuras.
- **No repudio**. Un emisor no puede negar la autoría de un mensaje o datos que ha generado, lo que facilita la detección de nodos que se comportan mal.
- **Localización segura de nodos críticos**. En ocasiones no se desea revelar cuáles son los nodos críticos de la red y su localización para evitar que sean atacados. En otras ocasiones se desea evitar la falsificación o modificación de la información de localización de los nodos porque se utilizan mecanismos basados en ella, como el encaminamiento.

La seguridad es un problema grave en muchas WSN por lo que se han desarrollado una amplia variedad de enfoques, protocolos, arquitecturas y métodos de seguridad, muchos de ellos derivados de las redes tradicionales, pero adaptados a la implementación con recursos limitados [99, 150, 29].

La medida de seguridad más básica es conseguir hacer que la superficie de ataque sea lo más pequeña posible. Si no se puede reducir, una WSN debe implementar medidas de seguridad que garanticen que los requisitos se cumplen, aunque se realicen ataques. Los requisitos de seguridad cada vez más exigentes han dado como resultado métodos

altamente sofisticados y que demandan recursos con una implementación difícil o compleja en problemas prácticos [15, 164].

Las medidas de seguridad pueden incluir mecanismos de prevención, detección y mitigación. Por consiguiente, las principales contramedidas de seguridad pueden ser clasificadas como [150]:

- Preventivas.
- Sistemas de Detección de Intrusos (IDS, *Intrusion Detection Systems*).
- Reactivas.

Las contramedidas preventivas fortalecen la WSN contra ciertos ataques antes de que ocurran. Ejemplos típicos son [29, 167]:

- El cifrado para proteger la confidencialidad.
- Las firmas digitales para la autenticación de las entidades.
- Añadir un Código de Autenticación de Mensaje (MAC, *Message Authentication Code*) a los mensajes para autenticar y garantizar la integridad.
- Las marcas de tiempo para verificar la frescura de los datos.
- Realizar cambios de canal y usar espectro ensanchado por salto de frecuencia para evitar interferencias.
- Localización de los nodos para evitar intrusiones e información falsa.
- Aleatorizar las comunicaciones para evitar el análisis del tráfico.

También es habitual utilizar protocolos seguros que ya integran varias de las anteriores técnicas. La criptografía es la base de muchas contramedidas proactivas, por lo que se verá con más detalle en la Sección 2.3.1.

Los IDS identifican los ataques cuando ocurren usando técnicas basadas en el control del tráfico de la red, observando el indicador de fuerza de la señal recibida (RSSI, *Received Signal Strength Indicator*) de los paquetes intercambiados, detectando comportamientos anormales o patrones sospechosos, etc. y analizando lo anterior con diferentes métodos, por ejemplo, con la teoría de Dempster-Shafer o con aprendizaje automático (*machine learning*) [16, 120, 2]. En la Sección 2.3.2 se amplía más el tema de la detección de ataques.

Por último, las soluciones reactivas normalmente complementan a los IDS y se activan después de un ataque para recuperarse del mismo y evitar nuevos intentos. Suelen requerir cambiar la configuración de los nodos, revocar/regenerar claves o expulsar nodos maliciosos de la red.

Una arquitectura de seguridad completa debe incluir contramedidas de todos los tipos. De hecho, la mayoría de las arquitecturas de seguridad adoptan estructuras de configuración-observación en las que las anomalías observadas por las entidades se utilizan para detectar ataques y determinar reconfiguraciones adecuadas para mitigar los efectos del ataque.

La mayoría de los trabajos existentes se centran en aportar soluciones a ataques específicos [3, 29, 155]. Sin embargo, la superficie de ataque es muy grande y se necesitan

muchos mecanismos y protocolos para proteger una WSN. El uso de muchos mecanismos en la misma WSN aumenta la complejidad y la demanda de recursos, lo que limita la viabilidad práctica.

2.3.1 Criptografía

La autenticación, el cifrado y la verificación de la integridad basados en la criptografía son unas de las medidas de seguridad preventivas más investigadas y utilizadas [29, 167].

Existen dos tipos de sistemas criptográficos principales, los simétricos y los asimétricos [56]. A partir de ambos también se han creado sistemas criptográficos híbridos que combinan los dos tipos [51].

En la criptografía simétrica, también conocida como criptografía de clave secreta, se usa una misma clave para cifrar y descifrar que debe ser conocida por los emisores y receptores de los mensajes cifrados. Su principal ventaja es su eficiencia energética en comparación con la criptografía asimétrica y su principal inconveniente es el reto que supone la gestión de las claves sobre todo en lo relativo a su intercambio y almacenamiento. Este problema es habitual en una WSN, donde un nodo puede tener que comunicarse con un gran número de pares (vecinos, sumideros, etc.) que puede no conocer inicialmente, tanto mediante mensajes dirigidos a un único destinatario o a múltiples (por difusión o multidifusión) [8]. La posesión de la clave simétrica se interpreta como autenticación implícita en muchos sistemas.

En [51] y [56] se describen diferentes esquemas que hacen uso de criptografía simétrica donde se observan deficiencias como:

- La necesidad de almacenar un número elevado de claves para que un nodo pueda comunicarse con cualquiera de los otros. Es posible que muchas no se lleguen a utilizar nunca.
- El uso de distribuciones aleatorias de claves que no garantizan la conectividad completa. Estos esquemas buscan minimizar el número de claves que debe almacenar un nodo maximizando las probabilidades de que cualquier comunicación pueda establecerse, aunque no se utilicen los caminos óptimos.
- El uso de entidades intermediarias para obtener las claves necesarias en la comunicación con otro. El intermediario debe ser una entidad conocida (puede ser otro nodo con el cual se comparte una clave secreta), que genera y conoce las claves que utilizan otros nodos y se presupone que es de confianza, pero esto es difícil de garantizar.
- El uso de claves compartidas, globalmente entre todos los participantes de la red o entre un grupo de ellos. La captura o compromiso de uno de los conocedores de la clave afecta al resto, ampliándose los efectos del ataque o posibilitando otros.
- Renovación de claves complejas o inexistentes. Si las claves no se renuevan tras un uso prolongado, se facilita la posibilidad de que un atacante pueda obtenerlas mediante criptoanálisis.

- La imposibilidad de revocar claves. De manera similar al caso anterior, la revocación de claves no suele estar contemplada y si un atacante se hace con una clave no hay forma de evitar que otros la acepten como válida.
- El requisito de que los nodos tengan los relojes sincronizados. Si la WSN no requiere la sincronización para el resto de sus funciones, esto puede suponer un consumo extra. Además, la red será más sensible a interferencias o errores de comunicación y es otro vector de ataque que hay que considerar.

La criptografía asimétrica, también conocida como criptografía de clave pública, utiliza un par de claves interdependientes, una clave pública (su revelación y distribución está permitida) y una privada (clave secreta que debe protegerse). Un mensaje cifrado con una de las claves solo puede ser descifrado con la otra. Si se cifra un mensaje con la clave pública, solo el propietario de la clave privada puede descifrarlo, proporcionando confidencialidad. Si el propietario cifra un mensaje con la clave privada (firma digital), cualquiera con la clave pública puede descifrarlo, proporcionando autenticación. Usando la criptografía asimétrica la distribución de claves es más simple y segura, ya que la clave pública no necesita ser mantenida en secreto, pero su principal desventaja es que suele ser más costosa computacionalmente. Esto hizo que al principio algunos autores la descartaran para su uso en WSN [29, 117].

Existen varios tipos de sistemas criptográficos asimétricos, siendo Criptografía de Curva Elíptica (ECC, *Elliptic Curve Cryptography*) y Rivest–Shamir–Adleman (RSA) los más estudiados y usados. ECC [84] es más eficiente que RSA y requiere claves más pequeñas para lograr el mismo nivel de seguridad. En [83] se muestra que ECC puede ser implementado en nodos sensores de bajos recursos computacionales y [152] ha realizado una implementación hardware consiguiendo una drástica reducción de tiempos y consumos. En el trabajo [121] se proporciona una revisión de la literatura de otros estudios de comunicaciones seguras en WSN utilizando ECC y presenta un nuevo protocolo de seguridad mejorado para WSN heterogéneas donde ECC se utiliza en la autenticación de los mensajes. El trabajo [63] emplea ECC como base de un acuerdo de claves autenticado dentro de un protocolo utilizado para la gestión distribuida de claves grupales en sistemas IoT en el sector agrícola inteligente (*Smart Agro*). Otro trabajo reciente es [89], que propone firmar digitalmente con ECC los mensajes enviados por sensores médicos en una WSN de atención médica y luego agregar las firmas de diferentes mensajes en una sola firma que sea más fácil de verificar, reduciendo los costos computacionales y de comunicación.

Un reto adicional que existe en la criptografía asimétrica es cómo autenticar la asociación entre una identidad con una clave pública y su posible revocación posterior [168]. Las siguientes alternativas se han empleado en WSN:

- Una Infraestructura de Clave Pública (PKI, *Public Key Infrastructure*) es el sistema más usado. Se emplean certificados que contienen la anterior asociación y un número de serie y que están firmados digitalmente por una Autoridad de Certificación (CA, *Certificate Authority*) (entidad en la que se confía plenamente). El principal inconveniente es que los certificados deben ser almacenados, transmitidos y verificados, consumiendo recursos. La revocación de certificados puede realizarse distribuyendo listas de los números de serie de los certificados revocados o haciendo peticiones

del estado de los certificados a la CA u otra autoridad de confianza. El trabajo [62] discute el uso de certificados en Seguridad de la Capa de Transporte de Datagramas (DTLS, *Datagram Transport Layer Security*) [125], y muestra pautas sobre cómo reducir el costo de su uso, por ejemplo, mediante el uso de la verificación previa, reanudación de sesiones, etc.

- La criptografía basada en identidad es un sistema alternativo donde la clave pública es la identidad en sí, por lo que no es necesario transmitirla. Sin embargo, tiene varias desventajas. Para empezar, requiere una tercera entidad de confianza, conocida como generador de claves privadas (PKG, *Private Key Generator*), que genera y conoce todas las claves privadas (en PKI la CA no conoce las claves privadas). Si el PKG se ve comprometido, toda la red estará comprometida. Además, la revocación es más compleja ya que no se pueden reutilizar identificadores. Por último, es más costosa desde el punto de vista computacional debido al proceso de emparejamiento (asociación bilinear de una identidad a un punto en la curva ECC). Recientemente, [168] propuso una adaptación de esta alternativa para WSN donde se evitaba realizar el proceso de emparejamiento, aunque limitando su empleo a la operación de firma digital.

- La criptografía sin certificados (*certificateless*) es una variante de la anterior. Esta alternativa resuelve el problema de la custodia de claves que presentaba la anterior, pero requiere de nuevo el intercambio de claves públicas, ya que no se pueden derivar directamente de la identidad. Además, sigue siendo muy exigente computacionalmente. Un ejemplo de un trabajo que adapta esta alternativa a WSN es [89], que también evita el costoso proceso de emparejamiento a costa de limitar su uso a la firma digital.

En los sistemas híbridos se busca combinar las ventajas de la criptografía simétrica y la asimétrica. Muchos de los protocolos seguros en funcionamiento son híbridos. La criptografía asimétrica se usa principalmente para la autenticación inicial y durante la negociación de la clave simétrica de sesión. Diffie-Hellman [32] es un protocolo muy usado para el intercambio de claves. Tras intercambiar las claves de sesión, se usa criptografía simétrica para cifrar y autenticar el resto de los mensajes porque es significativamente más eficiente. De este modo el mayor costo de la criptografía asimétrica se amortiza con el tiempo. Esta manera de operar se utiliza en los principales protocolos de seguridad de Internet tales como Seguridad del Protocolo de Internet (IPsec, *Internet Protocol security*) [81], Seguridad de la Capa de Transporte (TLS, *Transport Layer Security*) [31] y DTLS [125]. Algunos autores han realizado esfuerzos para implementar los protocolos anteriores en WSN: [124] es un ejemplo de uso de IPsec, y [115] es un ejemplo de uso de TLS/DTLS. DTLS también se usa en el Protocolo de Aplicación Restringida (CoAP, *Constrained Application Protocol*) [134], un protocolo de transferencia web especializado para redes y nodos restringidos con pocos recursos. Esta solución también ha sido adoptada por *ZigBee Smart Energy* [113], aunque usando un protocolo basado en *Standards for Efficient Cryptography 1* (SEC 1) [19].

2.3.2 Detección de ataques

Muchos sistemas de seguridad se basan en la detección de los ataques, y la identificación y localización de los atacantes ya sean externos o internos. Este es el caso de los IDS. El trabajo [76] presenta un estudio comparativo de varios IDS usados en WSN y divide su operación en 3 tareas: recogida de información, creación de un modelo de detección y generación de respuesta ante un ataque.

La monitorización de la red y la detección de anomalías pueden ser realizadas por todos los nodos como en [153], en un conjunto de nodos seleccionados como en [11], o de manera centralizada mediante el análisis del tráfico de la red como en [58]. No todos los nodos tienen que realizar las mismas tareas, sino que pueden realizarse de manera heterogénea, dependiendo de los recursos de cada entidad, como propone [158]. También existen IDS basados en agentes móviles tales como el descrito en [48].

La recolección de anomalías (o eventos de interés) para su posterior procesado puede realizarse de manera distribuida, como hacen por ejemplo [88, 103, 99], o centralizada como en [76, 132]. La principal desventaja de los métodos descentralizados es que los nodos solo tienen un conocimiento local y parcial de la red, y su tasa de detección es menor.

Muchos autores se centran en la capacidad de detectar diferentes ataques mediante el análisis de la información recopilada. Se pueden distinguir las siguientes categorías principales de IDS [120]:

- **Basados en anomalías (*anomaly-based*)**. En este caso, primero se extraen patrones asociados al comportamiento normal del sistema de la observación del funcionamiento de la red sin ataques. Posteriormente, las desviaciones respecto a los anteriores patrones se clasifican como ataques potenciales [76]. Si la red cambia o se introducen nuevas funcionalidades hay que repetir el proceso anterior. Estos IDS son dependientes del método usado para extraer patrones normales, pero permiten incluso la detección de ataques desconocidos.

- **Basados en el uso indebido (*misuse-based*)**. En estos IDS se analizan los ataques conocidos y de ellos se saca la firma o huella característica que los identifica (secuencia de acciones que realizan, mensajes que se producen durante su ejecución, patrones de comportamiento de los atacantes, etc.). Si se vuelven a producir eventos que coincidan con una huella de ataque conocida, se identificarán como posible ataque. Son capaces de detectar con precisión y eficiencia los ataques conocidos, pero la desventaja es que no son capaces de detectar ataques desconocidos.

- **Basados en especificaciones (*specification-based*)**. Aquí, se desarrollan manualmente un conjunto de especificaciones y restricciones. Un evento que no cumpla lo anterior se identifica como un comportamiento anormal y se evalúa como posible ataque. Trata de combinar las ventajas de las dos anteriores categorías, permitiendo la detección eficiente incluso de ataques desconocidos. Sin embargo, el desarrollo manual de las especificaciones y restricciones es un proceso laborioso y suele requerir invertir mucho tiempo, sobre todo en las redes más complejas. Si se consigue simplificar el sistema y reducir su superficie de ataque, este tipo de IDS son viables.

Se han adoptado muchos métodos para el análisis de eventos e inferir ataques. Ejemplos de estos métodos son los siguientes:

- Coincidencia de patrones (*pattern matching*) [77].
- Desviación del comportamiento normal del tráfico [16].
- Aprendizaje automático (*machine learning*) usando por ejemplo redes neuronales y máquinas de vectores de soporte (SVM, *Support Vector Machines*) [76].
- Macrodatos (*big data*) [88].
- Modelos de predicción de energía [99].
- Confianza y reputación [147].

Idealmente un IDS debería detectar cualquier tipo de ataque. Sin embargo, la detección de algunos ataques es difícil usando técnicas genéricas. Por este motivo algunos trabajos desarrollan detectores para ataques específicos. Por ejemplo, en [108] se hace una revisión de los detectores especializados en la detección de nodos clonados en redes estáticas y [59] presenta un detector específico para ataques de DoS de baja tasa basado en el análisis de señales y en macrodatos.

De los trabajos previos con IDS, destacamos algunas conclusiones. Para empezar, la funcionalidad de detección en los propios nodos y la transmisión de anomalías puede suponer un consumo energético considerable. Un atacante puede aprovechar esto para provocar el agotamiento prematuro de la batería de los nodos. La detección no es perfecta y se deben considerar los efectos de los falsos positivos/negativos además de la probabilidad de que estos se produzcan. Gran parte de los trabajos previos se centran en el procesamiento de la información de anomalías, pero también se deben considerar la capacidad de observar anomalías y la transmisión de anomalías y acciones correctivas. Los atacantes pueden interferir con la transmisión de anomalías, generar anomalías falsas e incluso comprometer a los miembros de un IDS distribuido.

2.3.3 Medidas de seguridad centralizadas

Los trabajos existentes presentan soluciones distribuidas y centralizadas a los problemas de seguridad.

Las ventajas que los esquemas distribuidos aportan a una WSN son el aumento de la independencia y autonomía de los nodos, autoorganización y la ausencia de un punto de fallo único (*single point of failure*) [108]. Un punto de fallo único es un componente del sistema cuyo fallo provoca el fallo del sistema completo. Las desventajas incluyen la necesidad de incrementar el intercambio de mensajes y de realizar cálculos adicionales en nodos con recursos limitados (provocando un mayor consumo energético), el uso de información local e incompleta y la mayor probabilidad de que un nodo malicioso influencie al sistema completo.

Los esquemas centralizados aprovechan la heterogeneidad de las diferentes entidades y tratan de que el trabajo más exigente se realice en las entidades más fiables y con más recursos [25]. Esa entidad normalmente suele ser la BS, pero también pueden ser nodos que cumplan determinados requisitos, como que estén conectados a una fuente de alimentación

o que tengan la mayor carga de batería, que estén situados en posiciones estratégicas, etc. La elección de las entidades encargadas puede ser permanente o por turnos. Si la elección no está predeterminada, se convierte en un problema que hay que resolver con protocolos de elección auxiliares, que, de nuevo, pueden ser distribuidos o centralizados. Estas soluciones requieren la transmisión de mensajes a mayores distancias y presentan un cuello de botella y un punto único de fallo en el elemento centralizado. Además, es necesario que existan rutas establecidas para comunicarse con él y el establecimiento de estas rutas puede ser atacado.

El cálculo de rutas es una tarea que puede ser centralizada, evitando que los atacantes puedan influir en el proceso. En la versión básica de INSENS (*Intrusion-Tolerant Routing in Wireless Sensor Networks*) [25, 26], la BS calcula las rutas a ella desde todos los nodos y selecciona los vecinos con los que un nodo debe comunicarse usando la información topológica local que estos le envían. Se hará una descripción más detalla de este trabajo en la Sección 2.7 dada sus similitudes con el trabajo desarrollado en esta Tesis Doctoral. Más reciente es el protocolo descrito en [3], donde el cálculo de rutas también se realiza en la BS buscando la detección del *wormhole attack* (en la Sección 5.2.21 se describe este ataque) aunque sin ofrecer protección contra otros tipos de ataques.

Otro ejemplo de enrutamiento centralizado es el paradigma Redes Definidas por Software (SDN, *Software Defined Networking*), que desacopla el plano de control, que es el plano de red que toma las decisiones de cómo enrutar paquetes, del plano de datos, que es el plano que realiza el reenvío de paquetes. Esto ha sido aplicado en las WSN haciendo que los nodos reciban instrucciones de un controlador centralizado de cómo deben procesar los paquetes que reciben. El trabajo [104] ha hecho un extenso estudio de implementaciones de SDN en WSN. SDN hace que la gestión de la red sea más flexible y simplificada, pero requiere que la comunicación con el controlador sea fiable. Al principio, los nodos deben descubrir una ruta al controlador por otros medios (por ejemplo, usando protocolos de encaminamiento sencillos como *Collection Tree Protocol* (CTP)). Posteriormente, la conexión con el controlador no debe interrumpirse. Estos requisitos permiten la aparición de nuevos ataques contra los que hay que protegerse, como muestran [132, 34].

Otros ejemplos de centralización son el uso de un centro de confianza en ZigBee para generar la clave de red y autenticar los nodos que se unen a la red, el uso de un centro de generación de claves para generar claves privadas parciales de nodos en [89], la elección centralizada de CH en redes con clústeres como se hace en [157] y la implementación de IDS con tareas centralizadas como se indicó previamente en [76, 132, 58].

2.4 Robots en redes de sensores

Desde el punto de vista de esta Tesis Doctoral, un robot es una entidad móvil dotada de capacidades computacionales, de detección y de comunicación. Recientemente, los robots aéreos se han convertido en herramientas comunes en tareas de monitorización tales como inspección, detección de fallos o mantenimiento predictivo, y esta tendencia parece que se incrementará en el futuro reciente [135, 111, 118].

A diferencia de los nodos sensores, los robots tienen menos restricciones en computación, consumo de energía y ancho de banda. Usados en las WSN, los robots han obtenido

resultados notables al ayudar a resolver una amplia variedad de problemas [138]. En [138, 161, 118] se describen trabajos y sistemas donde robots móviles colaboran con las WSN. Por ejemplo, los robots se han empleado para:

- **Localización de nodos**. El robot puede ubicarse a sí mismo mediante técnicas de localización con dispositivos que empleen un sistema global de navegación por satélite (GNSS, *Global Navigation Satellite System*) o sin ellos y a partir de su posición usar triangulación con medidas RSSI para localizar a los nodos [116].
- **Despliegue, reparación y sustitución de nodos**. El robot teleoperado o de manera autónoma puede transportar y situar nodos sensores (nuevos o de reemplazo) en las posiciones deseadas, sobre todo en lugares de difícil acceso. También se puede utilizar para reparar o reposicionar los nodos [95].
- **Reparación temporal del funcionamiento de la WSN**. Un robot puede sustituir temporalmente un fallo de un nodo que afecte a la conectividad de la red [23].
- **Recargar baterías**. Algunos investigadores han usado el robot para que periódicamente vaya recargando la batería de los distintos nodos.
- **Agregación y recolección de datos**. El robot hace rondas visitando a todos los nodos y ayuda a transportar la información obtenida por los sensores al sumidero. También puede hacer las veces de sumidero intermitente [94].
- **Extensión de cobertura**. El robot conecta zonas de la WSN aisladas del resto, como si fuera un router.

Sin embargo, a pesar de sus ventajas potenciales, la cooperación entre redes de sensores y robots para mejorar la seguridad de WSN aún está poco investigada [118].

En muchos trabajos que integran robots móviles con WSN, la seguridad suele ser un objetivo accesorio o un efecto secundario. Por ejemplo, si un robot se emplea para mejorar la robustez ante fallos de conectividad, también aumenta la protección contra ataques DoS. A continuación, se describen dos trabajos que sirven como ejemplo de esto.

En [157] se usa un vehículo aéreo no tripulado (UAV, *Unmanned Aerial Vehicle*) para seleccionar los nodos que funcionarán como CH de una WSN con clústeres. El UAV da vueltas constantemente por el área ocupada por la WSN recopilando la energía disponible en cada nodo y los datos generados por el clúster desde la última vez. Con los datos de energía disponible, se elige a los nodos que tengan más energía como CH. Se parte de la hipótesis de que un atacante siempre quiere convertirse en CH, lo que le permitiría falsificar todos los datos del clúster, pero los nodos no pueden inventarse su energía residual durante mucho tiempo porque se está monitorizando periódicamente y se detectarían valores incoherentes con facilidad. Demuestra, mediante simulaciones, que este tipo de elección centralizada reduce el consumo y la probabilidad de que un nodo malicioso sea elegido como CH respecto a otros protocolos de elección de CH. Presupone que el robot y la BS conocen la posición exacta de todos los nodos y pueden calcular en qué momento pasará el robot por cada zona. La BS notifica a todos los nodos la hora de la próxima visita del robot. El resto de medidas de seguridad son simples: cada nodo tiene claves simétricas preestablecidas con el robot, la BS y una lista de vecinos que se emplean para establecer al inicio otras claves de comunicación entre ellos y una clave de grupo; el robot emplea

la clave de grupo (que todos los nodos de un clúster conocen y podrían utilizar si son maliciosos) para enviar por difusión la identidad del CH elegido y una lista negra de nodos que se deben excluir; simula su propuesta considerando que no se pueden producir ataques durante la fase de establecimiento de claves; y no considera ningún ataque diferente al previamente comentado.

El trabajo [75] propone usar un UAV para la recolección de datos directamente desde una serie de nodos que denomina anclaje de datos (*data anchors*), que pueden ser los CH u otros nodos, y, para este propósito, desarrolla un algoritmo de selección de ruta. El objetivo principal es maximizar el tiempo de vida de la red, optimizando la elección de los nodos de anclajes de datos y una ruta que pase por todos ellos. La elección de la ruta la modela como un caso de problema del vendedor viajero (TSP, *Traveling Salesman Problem*) en vez de un problema de planificación de ruta de cobertura [20], sin tener en cuenta el alcance radio y obligando a que el robot pase exactamente por la posición de cada nodo de anclaje de datos. Adicionalmente a ese objetivo, se añade la posibilidad teórica de que el UAV sea capaz de tomar las mismas medidas que realizan los nodos sensores para comprobar si un nodo está comportándose correctamente y calcular un índice de confianza. No se indica cómo los nodos saben cuál es su anclaje de datos más cercano y si ellos lo son ni ninguna otra medida de seguridad.

Desde el punto de vista de la seguridad general de la WSN, el número de trabajos es muy pequeño y los robots se han propuesto principalmente para la tarea de distribuir claves, como por ejemplo [146, 130] que se describen a continuación.

En [146] se usa un elemento móvil para distribuir claves simétricas compartidas a un grupo de nodos situados en una zona alrededor de él de pequeño radio, para sustituir los esquemas de predistribución de claves simétricas aleatorias. Esto se realiza tratando de minimizar las zonas de diseminación y el número de claves que debe almacenar cada nodo, pero garantizando la conectividad de la red, tanto si se conoce la posición de los nodos como si no. Todas las claves son simétricas y para la autenticación utiliza una adaptación de μTesla [117] que se basa en el uso de una cadena *hash* unidireccional (OHC, *One-way Hash Chain*) (ver la Sección 2.7). Todos los nodos llevan preinstalada una clave global y el último valor de la OHC. En cada zona, el robot envía dos mensajes, uno con la clave que se quiere difundir cifrada con la clave global y otro para autenticar el mensaje anterior. Este sistema presenta varias vulnerabilidades. Un atacante puede obtener las claves transmitidas si se hace con la clave global comprometiendo uno de los nodos. El uso de OHC es susceptible de ataques *packet duplication* (ver descripción en Sección 5.2.33) y *rushing attack*. Por último, solo se consideran ataques pasivos a nivel local y el autor menciona que el robot podría monitorizar la red para detectar claves comprometidas y revocarlas, aunque no explica cómo llevarlo a cabo.

En [130] se usa un UAV como centro de coordinación y distribución de claves públicas. Un nodo que quiere comunicarse con otro nodo vecino, que previamente ha anunciado su identidad por difusión, debe solicitar la clave pública del otro extremo al UAV. El robot conoce la clave pública autorizada para cada identidad y los nodos solo necesitan conocer a priori la clave pública del robot. Todas las comunicaciones con el UAV van cifradas con criptografía asimétrica. Los nodos confían en el UAV y se supone que el robot no puede capturarse, así que la clave pública devuelta por el robot se supone auténtica. El vecino debe también realizar el mismo proceso. Ambos nodos, usando las claves públicas, acuerdan

una clave simétrica que utilizan para cifrar el resto de sus comunicaciones (mediante el algoritmo RC4 que en la actualidad está desaconsejado). El autor no menciona ninguna medida para evitar la duplicación de mensajes o que impidan la falsificación de identidades ni cómo se comunican nodos que no sean vecinos, limitándose a la tarea descrita. Además, parece ineficiente el empleo exclusivo de la criptografía asimétrica en las comunicaciones con el UAV (a diferencia de lo que se hace entre nodos).

Estos trabajos restringen el rol del robot a la distribución de claves y no aprovechan completamente la cooperación entre los robots y las WSN o las capacidades del robot en otras tareas de seguridad.

2.5 Comparación del estado del arte

A continuación, se muestra una taxonomía de los principales trabajos descritos en las secciones anteriores, para dar una visión general al lector de las características de esos trabajos, y, después, se hace una comparación de SNSR respecto a los trabajos anteriores.

2.5.1 Taxonomía del estado del arte

La Tabla 2.1 muestra la taxonomía de los trabajos previamente presentados más recientes o significativos, de acuerdo al tema de seguridad tratado, sus mecanismos de defensa, el uso de mecanismos centralizados y el empleo de robots móviles. En esta tabla se han usado las siguientes abreviaturas y símbolos:

Ref. Referencia bibliográfica del trabajo

✓ Sí

× No

P Mecanismos de prevención

D Mecanismos de detección

M Mecanismos de mitigación

C Mecanismos centralizados

R Uso de robots móviles

Tabla 2.1 Taxonomía del estado del arte.

Ref.	Tema	P	D	M	C	R	Características destacadas
[83]	Criptografía	✓	×	×	×	×	PKI distribuido implementado con ECC en nodos sensores de bajos recursos
[152]	Criptografía	✓	×	×	×	×	Implementación hardware de algoritmos criptográficos en sensores, incluyendo ECC

Continúa en la siguiente página »

Tabla 2.1 (Continúa de la página anterior)

Ref.	Tema	P	D	M	C	R	Características destacadas
[121]	Criptografía	✓	×	×	×	×	Revisión de protocolos de WSN que usan ECC, nuevo protocolo que almacena claves públicas prevalidadas en los nodos
[63]	Criptografía	✓	×	×	×	×	Se usa ECC para la gestión distribuida de claves grupales
[89]	Criptografía	✓	×	×	✓	×	Agregación de firmas ECC, adaptación de criptografía asimétrica sin certificados a WSN
[113]	Estándar	✓	×	×	✓	×	Arquitectura comercial estándar con protocolos que usan tanto criptografía simétrica como asimétrica
[62]	Criptografía	✓	×	×	✓	×	Pautas para reducir la sobrecarga del uso de PKI en IoT
[168]	Criptografía	✓	×	✓	✓	×	Adaptación de criptografía asimétrica basada en identidad a WSN
[76]	IDS	×	✓	×	✓	×	Recopilación centralizada de anomalías, IDS basado en anomalías
[153]	IDS	×	✓	×	✓	×	Detección de anomalías realizada en todos los nodos
[11]	IDS, Enrutamiento	✓	✓	✓	×	×	Detección y análisis de anomalías realizados solo en nodos seleccionados
[58]	IDS	×	✓	×	✓	×	Solo la BS detecta ataques analizando el tráfico de red, IDS basado en uso indebido
[158]	IDS	×	✓	×	✓	×	La detección se realiza de forma heterogénea en función de los recursos de cada entidad
[88]	IDS	×	✓	×	×	×	Recopilación y detección distribuidas y procesamiento de *big data*
[99]	IDS	×	✓	✓	✓	×	Recopilación de anomalías distribuidas, detección de ataques mediante modelos de predicción de energía
[59]	IDS	×	✓	×	✓	×	IDS específico para ataques DoS de baja tasa en protocolos de enrutamiento
[147]	IDS	×	✓	×	×	×	IDS basado en confianza y reputación con parámetros ambientales
[26]	Enrutamiento	✓	×	×	✓	×	Protocolo de enrutamiento seguro donde la BS calcula todas las rutas y vecinos de todos los nodos
[3]	Enrutamiento	✓	✓	✓	✓	×	Protocolo de enrutamiento reactivo centralizado para detectar solo *wormhole attacks*

Continúa en la siguiente página »

Tabla 2.1 (Continúa de la página anterior)

Ref.	Tema	P	D	M	C	R	Características destacadas
[132]	IDS	×	✓	×	✓	×	Revisión de ataques específicos a SDN en WSN, IDS especializado para SDN
[157]	Elección de CH	✓	×	×	✓	✓	Elección centralizada asistida por UAV de CH para evitar la interferencia de nodos maliciosos
[75]	Recolección	×	✓	×	✓	✓	Se emplea un UAV para recopilar datos y tomar medidas directas.
[146]	Criptografía	✓	×	×	✓	✓	Distribución basada en geometría de claves compartidas a un grupo de nodos usando un robot móvil
[130]	Criptografía	✓	×	✓	✓	✓	El uso de un UAV como centro de distribución y coordinación de claves públicas

2.5.2 Comparativa con el trabajo realizado

No es fácil comparar SNSR, que se describirá con más detalle en el Capítulo 3, con otros trabajos previos. SNSR considera la seguridad de la WSN en su conjunto incluyendo mecanismos de prevención, detección y mitigación desde el establecimiento de la red, mientras que la mayoría de los trabajos previos se centran en aspectos de seguridad concretos. Además, el uso de un robot en WSN solo se considera teóricamente en unos pocos trabajos de manera muy limitada. Aunque SNSR aún no ha sido detallado, en esta sección se va a hacer una comparativa a alto nivel de sus ideas claves, técnicas y funcionalidades respecto a otros trabajos.

La utilización de criptografía asimétrica ECC en WSN ya se ha realizado en [113, 83, 152, 121, 63, 89], y hay diferentes alternativas para autenticar las claves públicas [62, 168, 89]. La arquitectura propuesta podría funcionar con cualquiera de ellas, pero se propone el uso de PKI con reanudación de sesión y prevalidación de certificados por eficiencia. SNSR también va a requerir el empleo de un protocolo de intercambio de claves autenticado, pero es flexible y no impone ninguno. En la implementación realizada, se ha preferido usar el protocolo estándar DTLS, que está ampliamente probado y escrutado.

En SNSR, todas las entidades de la red detectan anomalías (a diferencia de [11] o [58]), de acuerdo con su rol, en línea a lo que hace [158], pero de una manera mucho más simple: reportando fallos de comunicación o comportamientos que no siguen la especificación (sin carga computacional adicional) y aprovechando la observación directa del robot. La detección de anomalías en los nodos implica principalmente costos de transmisión, pero proporciona una detección de ataques más rápida y evita el uso continuo del robot. La recopilación y el análisis de anomalías están centralizados en la BS (a diferencia de [88] o [99]) para liberar recursos en los nodos. La simplificación proporcionada por SNSR hace posible implementar un IDS basado en especificaciones y no excluye el uso de otros IDS como [120] y [76] o detectores para ataques específicos utilizando la infraestructura proporcionada. A diferencia de muchos IDS que solo proponen algoritmos de detección

[59, 88, 153], SNSR también tiene mecanismos de mitigación que pueden cambiar la configuración de la red durante los ataques. Solo [11] y [99] proponen otros mecanismos de mitigación.

SNSR reduce significativamente las tareas que realizan los nodos, el componente más débil desde el punto de vista de la seguridad en WSN, con el ahorro computacional y energético que ello conlleva. Por ejemplo, los nodos no realizan el descubrimiento de vecinos o la BS, una tarea común en otros trabajos [26, 113, 130]. Además, SNSR evita la necesidad de protocolos de enrutamiento dinámicos, lo que reduce aún más la influencia de los nodos. Puede soportar varios mecanismos de enrutamiento. También podría usar enrutamiento estático durante la creación de la red y luego cambiar a un enrutamiento distinto como SDN.

Algunos trabajos asumen que los requisitos de seguridad durante la configuración inicial de la red pueden ser menores que durante el funcionamiento normal porque hay más supervisión. Por ejemplo, en [146], al principio, todos los nodos comparten la misma clave global. Sin embargo, [26] y SNSR consideran en su diseño que los nodos podrían tener software malicioso preinstalado o que siempre es posible realizar ataques pasivos.

En cuanto a seguridad de las WSN con robots, [157] presenta un método para la elección de CH asumiendo que los nodos maliciosos reportan un mayor consumo de energía y no pueden engañar al UAV. [75] requiere que el robot tenga las mismas capacidades de detección que los nodos sensores para determinar la confiabilidad de los nodos. En ambos trabajos la seguridad es un objetivo secundario y se centran únicamente en problemas particulares. [146] distribuye claves simétricas a un grupo de nodos, pero un atacante podría escuchar las claves transmitidas si conoce la clave global preinstalada en todos los nodos. Además, el robot se autentica mediante técnicas basadas en cadenas *hash*, que son susceptibles de sufrir ataques *packet duplication* y *rushing attack*. El trabajo [130] propone un protocolo de acuerdo de claves usando criptografía con claves públicas obtenidas de un UAV para evitar almacenarlas en nodos. Ni [146] ni [130] consideran la captura del UAV. Esto sería fatal para la seguridad ya que el UAV tiene información insustituible, que también está preconfigurada en los nodos. En SNSR, el robot no distribuye claves. Prevalida las claves públicas, por lo que los nodos necesitan menos esfuerzo para detectar claves incorrectas. Además, la captura de un robot no es crítica ya que es posible revocar el certificado del robot capturado y utilizar otro robot con un certificado y claves diferentes.

2.6 Descripción de ZigBee con el perfil de aplicación ZigBee Smart Energy

ZigBee [8] es una especificación de protocolos que se apoyan en el nivel físico y de MAC del estándar IEEE 802.15.4 [1] para LR-WPAN que se usa para crear redes WSN con poco ancho de banda, entre otras aplicaciones. La define el *Connectivity Standars Alliance*, antes llamado *ZigBee Alliance*, desde el 2004. Se han creado varias versiones y en esta Tesis Doctoral se describe la versión de 2015, que era la más reciente con documentos públicos accesibles a la fecha de comienzo de su análisis. Otros autores han analizado las versiones anteriores [170, 10], e incluso algunos detectaron fallos de seguridad [169, 137].

A continuación, se describirán sus características principales, algunas de las cuales las hereda de IEEE 802.15.4, prestando especial atención a aquellas que afecten a su seguridad.

En ZigBee las redes son heterogéneas. Hay 3 tipos de dispositivos lógicos (un dispositivo puede comportarse como un tipo u otro según las circunstancias):

- Coordinador de ZigBee (ZC, *ZigBee Coordinator*): controla la red y las rutas de comunicación, solo puede haber uno en la red.
- Enrutador de ZigBee (ZR, *ZigBee Router*): encamina paquetes entre distintos dispositivos.
- Dispositivo final de ZigBee (ZED, *ZigBee End Device*): no reenvía paquetes de otros dispositivos, solo se comunica con el ZR por el que se unió a la red. Puede pasar la mayor parte del tiempo dormido (en modo de bajo consumo).

Según su capacidad hardware un dispositivo se puede clasificar como un dispositivo de función completa (FFD, *Full-Function Device*), que es aquel que puede funcionar como ZC, y un dispositivo de función reducida (RFD, *Reduced-Function Device*), que no puede funcionar como ZC y suele ser un dispositivo con pocos recursos. Por último, también se distingue entre los dispositivos que siempre están a la escucha, con el receptor radio encendido y pueden recibir mensajes en cualquier momento y los que no, que se ponen en modo de bajo consumo periódicamente y apagan la interfaz radio. Normalmente el ZC y los ZR están siempre a la escucha, aunque pueden existir ZR que no lo hagan. Igualmente, los ZED pueden elegir entre estar siempre a la escucha o no según la fuente de energía que utilicen.

Con los anteriores dispositivos, las redes pueden tener 3 topologías:

- En estrella, con ZC en el centro.
- En árbol, siendo el ZC la raíz. Es posible usar direccionamiento jerárquico asignando rangos de direcciones a cada ZR para que él las asigne y esto permite usar encaminamiento jerárquico que no requiere de tablas de encaminamiento. La desventaja es que el número de dispositivos ZR y ZED que pueden usar un mismo padre dentro del árbol está limitado, lo que puede provocar que algunos dispositivos no puedan unirse a la red.
- Mallada. Se considera la más robusta. Las direcciones se suelen asignar aleatoriamente, salvo para el ZC cuya dirección siempre es la 0. Existen procedimientos para detectar direcciones duplicadas y cambiar de dirección.

En la Figura 2.1 se muestra la arquitectura de la pila de ZigBee que, debido a sus diferencias respecto a otras pilas más conocidas como la de OSI o la de TCP/IP, comentaremos brevemente a continuación.

Los dos primeros niveles, el físico (PHY) y el MAC están definidos en el IEEE 802.15.4. El nivel MAC incluye también las funcionalidades del subnivel Control de Enlace Lógico (LLC, *Logical Link Control*) que suele existir en otros niveles de enlace, que aquí no existe, por lo que equivale al nivel de enlace completo. Para el acceso al medio se usa Acceso Múltiple con Escucha de Señal Portadora (CSMA, *Carrier Sense Multiple Access*) con Prevención de Colisiones (CA, *Collision Avoidance*) (CSMA/CA). Opcionalmente

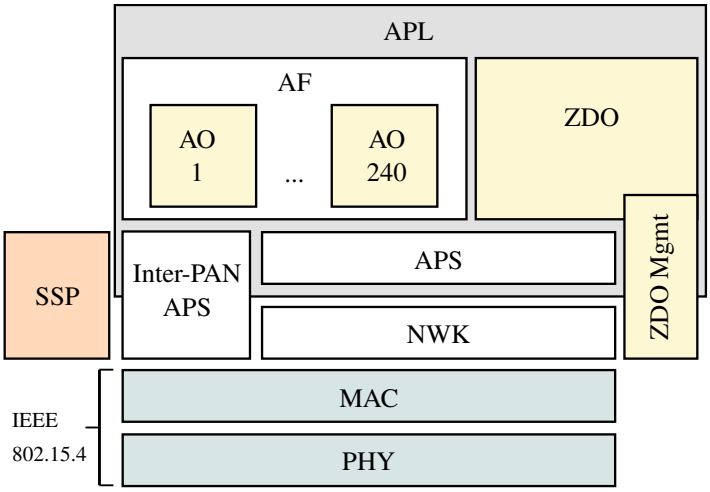


Figura 2.1 Arquitectura de la pila de ZigBee.

se pueden usar balizas de sincronización para reducir el consumo, pero no en topologías malladas. Las balizas indican a los dispositivos cuando deben transmitir y cuando pueden dormir. Los intervalos entre balizas los define el ZC. Cuando se usan balizas también es posible reservar ranuras de tiempo durante las cuales transmite solo un dispositivo. Si no se usan balizas, la transmisión se puede realizar en cualquier momento por lo que los ZR y ZC deben estar a la escucha la mayor parte del tiempo. En un mismo lugar pueden coexistir distintas redes, cada una diferenciada por su identificador de red de área personal (PAN, *Personal Area Network*). Si se detecta que varias redes usan el mismo identificador, existen procedimientos para cambiarlo.

En el nivel de red (NWK) se utilizan direcciones de 16 bits. La comunicación puede ser por unidifusión (directamente o salto a salto), por difusión o por multidifusión (con direcciones de grupo). En todos los anteriores casos se puede limitar el número máximo de saltos. La multidifusión es opcional y también se puede realizar en niveles superiores. Los ZR son capaces de almacenar mensajes de datos destinados a sus hijos ZED que no están siempre activos, y estos sondean cada cierto tiempo a su ZR para obtenerlos. Se usa un protocolo de encaminamiento basado en *Ad hoc On-Demand Distance Vector* (AODV). Las peticiones de rutas se envían por difusión y la respuesta por unidifusión. Los ZR actualizan las rutas incluyendo el coste asociado en una tabla de rutas. Un ZR puede responder en nombre de sus ZED hijos. Existen tres posibles peticiones de rutas: a la dirección única de un equipo (*unicast*), a un miembro de un grupo (*multicast*) y de muchos a uno (*many-to-one*). Esta última permite a un dispositivo *concentrador* (por ejemplo, una BS) hacer que todos los ZR y ZC descubran la ruta a él. Muchos procesos requieren el envío por difusión (inundación), que puede limitarse a un cierto radio (número de saltos), pero que provoca mucho gasto energético.

El nivel de aplicación (APL) se encuentra subdividido. El subnivel inferior se denomina subnivel de soporte de aplicaciones (APS, *Application Support Sublayer*) y tiene funcionalidades típicas de un nivel de transporte como el transporte fiable de información, la

fragmentación de mensajes grandes, la detección de mensajes duplicados y la gestión de la pertenencia a grupos. APS también ofrece el establecimiento de comunicaciones seguras y la vinculación (*binding*) entre dispositivos (unión de dos dispositivos según sus servicios y necesidades, comparable a una conexión de nivel de transporte). También se permiten las comunicaciones Inter-PAN (entre dos redes con PAN distinto) y con dispositivos *Green Power* (cuyo protocolo es diferente), a través del Inter-PAN APS, que genera los mensajes y los entrega directamente al nivel MAC.

El resto del APL son aplicaciones, cada una situada en un punto final (*endpoint*) (identifica a la aplicación dentro del dispositivo en los mensajes transmitidos) y descritas en un perfil de aplicación ZigBee (ZAP, *ZigBee Application Profile*). Los ZAP permiten la interacción estandarizada entre dispositivos y definen tipos de dispositivos, sus funcionalidades, comandos y mensajes (denominados *clusters*) con sus atributos en un dominio de aplicación. Ejemplos de ZAP son *Smart Energy*, *Home automation*, *ZigBee Light link*, etc. que incluyen políticas de seguridad, protocolos adicionales de seguridad, etc. Todos los dispositivos contienen la aplicación objeto de dispositivo ZigBee (ZDO, *ZigBee Device Object*) encargada de inicializar el resto de los niveles y gestionar el dispositivo (parte del ZDO está en el plano de gestión), las claves de seguridad y las políticas. Esta aplicación define el rol del dispositivo en la red (ZC, ZR o ZED) además de implementar los servicios básicos de descubrimiento de dispositivos y servicios y vinculación con otros dispositivos. El ZDO siempre se coloca en el punto final 0, mientras que el resto de las aplicaciones (conocidas como *Application Objects*) pueden estar en los puntos finales desde el 1 al 240. Por último, un dispositivo puede ser elegido para desempeñar uno de los siguientes roles servidores:

- *Trust Center* (TC) (primario y de respaldo). Crea y distribuye claves en la red, se describirá con más detalle más adelante.
- *Network Manager*. Se encarga de decidir cuándo se debe cambiar de canal, a qué canal y comunicarlo al resto de dispositivos que están siempre a la escucha. Los dispositivos, si detectan muchos errores de transmisión o pérdida de paquetes, pueden hacer un escaneo de energía del canal. La detección de que hay mucha energía es un indicador de que pueden estar produciéndose muchas interferencias. Cuando esto ocurre envían informes de estado al *Network Manager*.
- *Binding table cache* (primario y de respaldo). Almacenan información de vinculación de los dispositivos que no tienen capacidad para almacenarla. También tiene un registro de los dispositivos que sí la almacenan internamente.
- *Discovery cache* (primario y de respaldo). Guardan los descriptores con información de las capacidades disponibles en cada uno de los puntos finales de los dispositivos que suelen pasar gran parte dormidos, de tal manera que se puedan descubrir en todo momento.

Los anteriores roles son opcionales. Están implementados en el ZDO de los ZR o ZC y se activan por configuración. Solo puede haber un TC primario y un *Network Manager* en la red y normalmente es el ZC quien realiza esas funciones.

2.6.1 Seguridad en ZigBee

La seguridad de ZigBee parte de la que proporciona IEEE 802.15.4, que usa cifrado simétrico AES con claves de 128 bits usando el modo CCM*, que es una variante de CCM (*Counter with CBC-MAC*, *Counter with Cipher Block Chaining Message Authentication Code*). CCM proporciona autenticación y confidencialidad a un mensaje, calculando primero su CBC-MAC, también conocido como Código de Integridad del Mensaje (MIC, *Message Integrity Code*), y cifrando después, con el modo CTR (*Counter*), la concatenación del mensaje y el CBC-MAC. CCM* lo extiende posibilitando proporcionar solo confidencialidad o solo autenticación. IEEE 802.15.4 permite aplicar seguridad en las tramas del nivel MAC añadiendo una cabecera auxiliar de seguridad a la habitual. ZigBee desde la versión 2006 no especifica cómo usar la seguridad del nivel MAC, así que su uso puede dar lugar a incompatibilidades. Sin embargo, utiliza los mismos métodos (CCM* y cabeceras auxiliares similares) para aplicar seguridad en los niveles NWK y APS mediante el Proveedor de Servicios de Seguridad (SSP, *Security Service Provider*) común a ambos niveles. Las claves utilizadas en cada nivel se deciden en el nivel APL y suelen tener asociados contadores de mensajes entrantes y salientes para garantizar que todos los mensajes transmitidos utilizando una misma clave son diferentes para detectar duplicados y evitar la deducción de la clave por criptoanálisis. Las cabeceras auxiliares tienen campos para incluir los contadores si los datos no lo incorporan ya. Si un contador llegara a su valor máximo, el dispositivo dejaría de transmitir con la clave asociada hasta que no se realice un cambio de clave.

Las aplicaciones que se ejecutan en un mismo dispositivo deben tener confianza mutua, ya que no hay separación lógica entre ellas y no hay ningún mecanismo de seguridad que las aísle. Tienen acceso al resto de niveles y pueden compartir claves.

La seguridad aplicada por cada dispositivo en cada capa debe ser la misma y la misma que en el resto de los dispositivos de la red. Si un dispositivo necesita más seguridad, debe crear otra red. Existen 8 modos de seguridad: sin seguridad, solo autenticación con 3 posibles longitudes de MIC, solo encriptación y encriptación con autenticación con 3 posibles longitudes de MIC.

En NWK se usa una misma clave de red (NK) compartida entre todos los dispositivos de la red. Esta clave se genera aleatoriamente cuando se crea la red y un dispositivo la recibe cuando se une a la red. La NK tiene asociado un número de secuencia que se va incrementando cada vez que se cambia. Si un nodo no recibe una actualización de la NK, detectaría esto por el número de secuencia y debería volver a unirse a la red para recibir la nueva NK. Nunca se transmite esta clave en claro si se usa cifrado. El que esta clave sea compartida, es uno de los puntos más débiles de su seguridad, ya que un dispositivo malicioso en la red con acceso a esta clave podría afectar a muchos de los procesos básicos de la red. En APS se usa una clave de enlace (LK) diferente para cada dispositivo con el que se comunica. A partir de ella se derivan, usando un Código de Autenticación de Mensaje basado en *Hash* (HMAC, *Hash-based Message Authentication Code*), otra exclusivamente para enviar la clave NK a otros dispositivos y otra para enviar claves LK.

La red puede funcionar en dos modos de seguridad: centralizada y distribuida. Con seguridad centralizada, existe en la red un solo elemento centralizado, el TC, que normalmente es el ZC. El TC se encarga de generar claves nuevas, tanto NK como LK entre dos

dispositivos, y autoriza la unión de un nuevo dispositivo a la red, pudiendo usar listas de control de acceso. Con seguridad distribuida, conocido como modo de TC distribuido, cada ZR decide quien puede unirse y distribuye la clave de NK.

Un dispositivo para poder comunicarse en una red primero tiene que unirse y autenticarse. Mientras pertenece a la red puede recibir actualizaciones de NK y establecer comunicaciones seguras con otros dispositivos a nivel de aplicación. Un dispositivo puede salir de la red por decisión propia o por expulsión del TC. A continuación, se describen estos procesos cuando se usa seguridad centralizada.

Unión a una red segura

Durante la unión (*join*) un dispositivo recibe la NK si no la tenía y se le asigna una dirección de red. Los dispositivos se unen a través de un ZR o el ZC, que se convertirá en su *padre*. En IEEE 802.15.4 la red puede estar abierta o cerrada a nuevas incorporaciones. Cuando la red está cerrada, ZigBee permite la unión si el dispositivo previamente se había unido y ha perdido la conexión con su padre: a) cuando el dispositivo aún conserva la NK en vigor y se asocia a través de otro *padre* (*secure rejoin*); b) cuando no tiene la NK en vigor, pero sí una clave LK con el TC (*trust center rejoin*); o c) si la red ha cambiado de canal y el padre y la NK no cambian (*join through orphaning*). También es posible unir dispositivos por configuración, sin necesidad de realizar ninguna comunicación.

Un dispositivo que quiere unirse a la red transmite una baliza, y los ZR y el ZC responden a ella. Posteriormente, el dispositivo envía una petición a uno de los que han contestado (cifrada con la NK si la conoce). Si la respuesta a la petición es positiva y el dispositivo no conocía la NK, deberá autenticarse con el TC a continuación. Si conocía la NK, el nodo se da por autenticado. El TC es informado en cualquier caso de las nuevas uniones y es el encargado de enviar la NK al dispositivo si no la tuviera.

También existe un modo de puesta en marcha y unión en el que se comunican los parámetros de la red mediante comunicaciones de muy corto alcance (conocido como *touchlink*).

Autenticación

Los dispositivos que se unen a la red deben autenticarse ante el TC demostrando que conocen una LK compartida con él (TCLK). La NK se transmite al dispositivo utilizando una clave de cifrado derivada de la TCLK. Existen TCLK únicas (una diferente por dispositivo) y globales (compartida por todos los dispositivos). El estándar propone una TCLK global concreta que todos los dispositivos deben conocer, cuyo uso es bastante inseguro, pero que permite la compatibilidad con perfiles menos exigentes. El TC puede tener listas de dispositivos permitidos o rechazados y decidir en cada momento si un nodo se admite o debe abandonar la red.

En el caso de TCLK únicas, es necesario configurar el TC con estas claves antes de que el dispositivo intente unirse. Para facilitar la tarea al usuario, los dispositivos suelen usar códigos de instalación que normalmente se generan aleatoriamente durante la fabricación y que son visibles para el usuario (mediante impresión o etiquetas en el propio dispositivo). Estos códigos se introducen en el TC y a partir de ellos, usando una función *hash*, se genera una TCLK. Su uso se limita al primer contacto, e inmediatamente después se negocia una

nueva TCLK. El código de instalación no se debería usar más de una vez para evitar que los dispositivos sean clonados.

Actualización de la NK

El TC elige cuándo hay que actualizar la NK, periódicamente o de forma que se garantice que el contador de paquetes de cualquier dispositivo nunca alcance su valor máximo. La TC la comunica al resto de dispositivos de la red por difusión (cifrada con la antigua NK) o uno a uno (cifrando con la TCLK de cada uno) según su configuración y circunstancias. Por ejemplo, la única manera de excluir un dispositivo malintencionado es no darle la nueva clave de red. Los ZR están obligados a almacenar al menos 2 NK (la actual y una alternativa) y permiten el uso de ambas, así que se deben realizar 2 cambios de NK como mínimo para que una clave antigua deje de usarse. La nueva clave recibida se guarda como clave alternativa hasta que se recibe un comando de conmutación de clave desde el TC o se detecta que otro dispositivo ya la está usando.

Establecimiento de claves entre pares

Para crear una LK entre dos dispositivos se requiere la participación del TC. Un dispositivo pide la nueva clave al TC indicando el otro extremo y el TC envía una nueva clave a ambos cifrada con sus respectivas TCLK. Tanto los dispositivos como el TC pueden establecer políticas sobre la aceptación de nuevas LK, incluso puede haber una lista de pares permitidos. El estándar deja la opción a niveles más altos para que puedan establecer claves LK por otros procedimientos. En versiones anteriores también se usaba un esquema de establecimiento de claves basado en la compartición previa de una clave maestra simétrica, pero en la versión analizada solo se menciona como alternativa el esquema de establecimiento de claves basado en certificados (CBKE, *Certificate-Based Key Establishment*).

La actualización de una TCLK única a partir de una TCLK anterior aparece como novedad en la versión analizada, aunque se recomienda usar otros mecanismos como el CBKE, porque puede presentar un problema de sincronización ante la pérdida de un paquete. Usa comandos de APS que no tienen reintentos ni asentimientos, por lo que, si se pierde el mensaje de transporte de clave, el TC y el otro dispositivo no compartirán la misma clave y será, en muchas configuraciones, imposible volver a establecer comunicación de manera automática (habrá que usar otros métodos no especificados).

Salida de la red

Hay 2 modos de salir. Por expulsión (el TC envía un comando solicitándolo) o por propia voluntad informando al *padre* (que informará posteriormente al TC). Los mensajes entre los ZR y los ZED van encriptados con NK, y entre los ZR y el TC con LK si hay o NK si no. Si un dispositivo malicioso tiene la NK, podría falsificar estos mensajes y provocar que los dispositivos se vayan de la red.

2.6.2 ZigBee Smart Energy

ZigBee Smart Energy (ZSE) [113] es un estándar basado en ZigBee que define dispositivos y prácticas estándares para aplicaciones de suministro de energía inteligente necesarias en un entorno residencial o de pequeño comercio, de tal manera que dispositivos de

distintos fabricantes sean compatibles. Los dominios de aplicación clave son la medición, la tarificación, la detección de errores y la respuesta a la demanda y el control de carga. En estos entornos los requisitos de seguridad son altos. Se han creado varias versiones y en esta Tesis Doctoral se describe la versión 1.2a [113], que era la más reciente de la cual se disponían documentos públicos a la fecha de comienzo de su análisis.

En ZSE, el TC siempre debe ser el ZC, que a su vez debe ser un punto de suministro (*Energy Service Interface*) capaz de comunicarse con la red de la compañía suministradora. También se introduce el concepto de redes de área doméstica virtuales que se forman en edificios con múltiples viviendas que comparten red ZigBee. En estas redes, el TC, conocido como TC federado, contiene información de los dispositivos que forman parte de cada red virtual y previene la comunicación a nivel de aplicación entre dispositivos de diferentes redes virtuales.

La principal medida de seguridad de ZSE es que requiere el establecimiento de claves con el TC mediante el uso de CBKE. Esto debe ser realizado en un punto final diferente al empleado por el resto de ZSE, que se denomina *Key Establishment* y que debe de ser descubierto antes de poder realizar el CBKE. El CBKE usa ECC según los algoritmos definidos en *Standards for Efficient Cryptography 1* (SEC 1) [19], con 2 suites de cifrado. La más débil utiliza claves públicas de 22 bytes, con la curva `sect163k1` dando lugar a certificados de 48 bytes. La otra usa claves públicas de 37 bytes, con la curva `sect283k1`, generando certificados de 74 bytes que equivalen a usar claves simétricas de 128 bits. Se requiere fragmentación para poder transmitir estos certificados. Cada dispositivo tiene una clave privada y un certificado digital firmado por una CA. El certificado digital incluye la clave pública, la dirección IEEE extendida del dispositivo de 64 bits e información adicional como el tipo de dispositivo, el identificador de la red, fechas de validez, etc. Los certificados proporcionan un mecanismo para vincular criptográficamente una clave pública con la identidad y las características de un dispositivo. Los dispositivos también tienen la clave raíz pública de la CA para validar los certificados. Los certificados pueden ser generados por el fabricante, el distribuidor o el usuario final. Tras realizarse el procedimiento CBKE entre dos dispositivos, ambos comparten la misma clave LK, se han autenticado, han confirmado que la clave es correcta y que ningún extremo ha controlado la generación de la clave. La nueva LK es única y cumple el requisito de secreto perfecto hacia adelante (PFS, *Perfect Forward Secrecy*).

Durante la puesta en marcha de los dispositivos es obligatorio el uso de TCLK únicas preinstaladas derivadas de códigos de instalación que han sido previamente comunicados al TC. No se permiten uniones a la red con claves globales. La TCLK preinstalada solo se utiliza para obtener la NK y realizar operaciones que solo requieran cifrado en el nivel de red. Solo se utilizan en la primera autenticación y solo se admitirán de nuevo si el dispositivo es expulsado de la red y posteriormente se une. El TC no elimina las claves de un dispositivo al recibir un anuncio de que se marcha, pero sí cuando él lo expulsa. Así evita ataques de DoS si se suplantan las direcciones. Cualquier operación que use el cifrado en el APS requiere la obtención de una nueva TCLK usando CBKE.

En APS es obligatorio el uso de fragmentación debido a la longitud de algunos comandos. Todos los comandos definidos por ZSE van cifrados con la LK, aunque se siguen usando otros comandos previamente definidos que solo usan NK, como los usados en el

descubrimiento de servicios. Se puede (y se recomienda si hay confianza) usar el establecimiento de claves entre dispositivos usando como intermediario el TC. Un dispositivo puede prohibir la comunicación con el punto final *Key Establishment* si el solicitante no es el TC. Así, el TC puede aplicar políticas, realizar comprobaciones y la Lista de Revocación de Certificados (CRL, *Certificate Revocation List*) estaría centralizada. Antes de pedir una clave LK al TC, se intenta hacer una vinculación con el otro destino y solo si le responde con éxito se pone en contacto con el TC. El otro dispositivo aceptaría solo nuevas LK esperadas.

Existen otros aspectos que pueden influir en la seguridad. En ZSE es importante que los dispositivos estén sincronizados por lo que hay servidores de tiempo (ordenados por preferencia) y clientes que se sincronizan a ellos. La sincronización se intenta realizar una vez al día para no sobrecargar la red, utilizando cifrado en el APS. Existe la posibilidad de comunicar alarmas y errores a un dispositivo central. Se define un procedimiento para cambiar el TC estando la red en funcionamiento, haciendo previamente copias de seguridad de toda la información del TC existente, salvo de las LK (se guardan en su lugar los valores devueltos al aplicar una función *hash*). Los dispositivos deben establecer una nueva TCLK usando CBKE tras el cambio. La comunicación Inter-PAN está admitida y la existencia de dispositivos no ZSE con códigos de instalación registrados en el TC, pero que no podrán obtener nuevas LK. Por último, si se permite la actualización remota de firmware, este debe ir firmado digitalmente usando el Algoritmo de Firma Digital de Curva Elíptica (ECDSA, *Elliptic-Curve Digital Signature Algorithm*).

2.7 Descripción de INSENS

El protocolo INSENS (*Intrusion-Tolerant Routing in Wireless Sensor Networks*) [25, 26] es un protocolo que trata de minimizar y tolerar los efectos que pueden provocar nodos maliciosos dentro de una red de sensores. En [25] se describe INSENS para una WSN con una BS que es refinado en [26], donde se le denomina INSENS *básico*. En esta Tesis Doctoral haremos referencia a él simplemente como INSENS. En [26] también se muestra un INSENS *mejorado* para WSN con múltiples BS, que, a pesar de su nombre, es un protocolo totalmente diferente que solo comparte el objetivo con el anterior.

INSENS no intenta detectar atacantes, solo ignorarlos o tolerarlos, alegando que la detección es costosa y difícil de llevar a cabo en las WSN. Un atacante puede hacer que una parte de la red quede inoperativa, por lo que únicamente es adecuado para aplicaciones no críticas con medidas redundantes. El objetivo es mejorar la seguridad de los mensajes procedente de los nodos con destino a la BS. No se considera la comunicación entre nodos distantes ni el envío de mensajes desde la BS a los nodos salvo durante el establecimiento de la red. Para proteger los mensajes, los nodos comparten una clave con la BS, que se utiliza para generar otras claves de cifrado simétrico y de generación de MAC, y un número de secuencia inicial de cadena *hash* unidireccional (OHC, *One-way Hash Chain*) usado para autenticar los mensajes de difusión (adoptado de μTesla [117]). La BS es la única entidad autorizada a iniciar difusiones. Una OHC consiste en una secuencia de números generados por la BS aplicando sucesivamente una función *hash* unidireccional H: $n_1, n_2, \ldots, n_{k-1}, n_k : n_k = H(n_{k-1})$. Se comparte el último número n_k con los nodos y

en cada difusión se incluye un número de secuencia previo. Los nodos pueden verificar que el mensaje ha sido originado por la BS si el número de secuencia guardado o el último visto se obtiene aplicando H repetidamente un número limitado de veces. Este mecanismo es eficiente, pero tiene como inconvenientes la posibilidad de agotar la secuencia a lo largo de la vida de la red o la posibilidad de que un nodo no pueda comprobar la autenticidad si pierde muchos mensajes previos si no se incorporan mecanismos para el reinicio de la secuencia como ocurre en este trabajo. Los sensores no autentican ni cifran los mensajes intercambiados entre sí.

La principal novedad de INSENS es el descubrimiento de rutas, que se realiza en tres rondas de mensajes (todos incluyen el mismo número de secuencia de OHC). Primero la BS envía un mensaje de petición de ruta (REQ) a todos por inundación. Esto se utiliza también para que los nodos detecten a sus vecinos. Los nodos vuelven a reenviar el mensaje REQ si el número de secuencia es válido y no está repetido, guardando la información de sus vecinos. En el mensaje reenviado se modifica el identificador del origen y un MAC usando la clave compartida con la BS y que es calculado a partir del que tenía el primer mensaje recibido (MAC anidado), cuyo origen se convierte en su nodo padre. Los nodos no pueden verificar los MAC recibidos, pero los almacenan para enviarlos posteriormente a la BS. En segundo lugar, los nodos envían un mensaje de *feedback* (FDBK) a la BS con los identificadores y MAC recibidos de cada vecino cifrados con la clave compartida con la BS, utilizando como siguiente salto su nodo padre. Por último, la BS recopila información de la topología a partir del conocimiento parcial que cada nodo le envía, comprobando que es coherente y calcula las rutas. INSENS permite distintas técnicas de cálculo, pero los autores proponen uno para su implementación con 2 rutas (la más corta y otra que no contenga nodos de la primera ni de sus vecinos ni de vecinos de vecinos, en lo posible). La BS envía un mensaje encriptado a cada nodo según el orden seguido por una búsqueda en anchura (BFS, *Breadth First Search*) conteniendo la tabla de reenvío. A diferencia de las tablas de reenvío más habituales que eligen el siguiente salto según el destino, aquí la elección se realiza usando el destino, el origen y el último emisor para asegurar que los mensajes siguen la ruta establecida por la BS.

En el mismo trabajo se reconocen limitaciones de seguridad, por ejemplo, indicando que es vulnerable al ataque *rushing*. Un análisis de seguridad en profundidad de INSENS se realizará en la Sección 5.2. También existen limitaciones de funcionamiento en tanto que no se proponen procedimientos para añadir o eliminar nodos, el reinicio de la OHC en el caso de que se agotara la secuencia y la fragmentación de mensajes largos. Para los experimentos se utilizó un mecanismo sencillo de fragmentación y cifrado RC5 en modo CBC usando 12 rondas que en la actualidad es inseguro [13].

El protocolo INSENS *mejorado* pasa a ser totalmente distribuido. Ahora se utilizan varias BS que no calculan las rutas ni las vecindades ni tienen contraseñas individuales compartidas con cada nodo. Las únicas similitudes con INSENS son el empleo de OHC para autenticar los mensajes REQ empleados para anunciar la presencia de las BS. Los nodos deben mantener un número de OHC por cada BS. La seguridad se basa en el empleo de una clave global preinstalada en todos los dispositivos que estos usan para intercambiar claves aleatorias de manera cifrada. Cada nodo genera una clave individual para cada vecino con el que sea posible una comunicación bidireccional y otra común para reenviar los mensajes REQ mediante difusión. Después del intercambio de claves, se borra la clave

global. Para permitir la introducción posterior de nuevos nodos, se guardan una serie de números aleatorios y sus MAC usando la clave global para poder verificar que los nuevos poseen la clave global original. Los nodos envían los datos a todas las BS simultáneamente siguiendo los caminos inversos a los realizados por los mensajes REQ. Los mensajes se cifran en cada salto usando la clave individual que establecieron cada par de vecinos. Si un atacante consiguiera hacerse con la clave global antes de que fuera borrada por un nodo, toda la red quedaría expuesta a posteriores manipulaciones. Esta es la principal debilidad de este protocolo.

2.8 Conclusiones

La seguridad es crítica en muchas aplicaciones de monitorización basadas en WSN que operan de manera desatendida durante largos períodos de tiempo. La gran superficie de ataque que presentan las WSN y los diferentes requisitos de seguridad han motivado el desarrollo de una amplia variedad de arquitecturas y protocolos para prevenir, detectar y mitigar muchos tipos de ataques. Muchas técnicas de seguridad de redes clásicas se consideran demasiado complejas para ser realizadas con los recursos limitados de las WSN, por lo que muchas soluciones tienden a simplificarlas. Además, muchos trabajos se centran en ataques específicos.

Recientemente, los robots aéreos se han convertido en herramientas comunes en muchas tareas de monitorización, como inspección, detección de fallos o mantenimiento predictivo, y esta tendencia parece ir en aumento. La cooperación entre robots y WSN ha obtenido resultados notables en muchos problemas. Sin embargo, a pesar de sus ventajas potenciales, el empleo de esta cooperación para mejorar la seguridad de las WSN aún está poco investigada.

3 Arquitectura de seguridad

La simplicidad es la máxima sofisticación.

Leonardo da Vinci

Este capítulo está destinado a describir la *Contribución 1* de esta Tesis Doctoral "*Diseño de una nueva arquitectura de seguridad combinando robots y WSN*" y la *Contribución 2* de esta Tesis Doctoral "*Arquitectura SNSR con un robot y una estación base*". En él se presenta el diseño de una nueva arquitectura de seguridad combinando robots y WSN, llamada SNSR, que explota las sinergias entre el procesamiento heterogéneo y las capacidades de comunicación del robot y los nodos sensores con el fin de proporcionar soluciones computacionalmente eficientes y seguras contra los ataques de estas redes. Los robots se emplean para configurar el funcionamiento seguro de la WSN y son responsables de realizar la autenticación de los nodos sensores. La comunicación en un solo salto entre un robot y cada nodo sensor mejora significativamente la seguridad. La arquitectura SNSR se puede adaptar a redes ya existentes y es flexible al proponer alternativas a la hora de realizar muchos procedimientos que pueden ser particularizados en diferentes implementaciones y configuraciones. SNSR se puede generalizar para el uso de múltiples BS y robots simultáneamente. Solo se contempla el caso con un robot y una BS en esta Tesis Doctoral y, por sencillez, SNSR se explica usando este caso.

El capítulo se puede dividir en tres grandes bloques. En la Sección 3.1, se comenzará analizando el problema de la seguridad en las WSN utilizadas en instalaciones industriales o infraestructuras. Después, en la Sección 3.2 se esbozará la idea propuesta.

Comenzando con el segundo bloque, la Sección 3.3 describirá la arquitectura de la red, incluyendo los requisitos que debe cumplir la torre de protocolos de la WSN donde se vaya a implementar y los fundamentos para garantizar una comunicación segura y fiable entre componentes de la red. Al final de este bloque, se dará una visión general de las fases de funcionamiento y las relaciones que se producirán entre los distintos tipos de entidades.

En el último bloque, que comienza en la Sección 3.4, se profundizará en la información de la topología de la red almacenada, así como el funcionamiento y protocolos utilizados en cada una de las fases de la red. Se terminará, en la Sección 3.10, con un cuadro resumen

de todas las opciones que permite la arquitectura SNSR y los parámetros de configuración vistos a lo largo del capítulo.

3.1 Planteamiento del problema

Un conjunto de nodos sensores estáticos (que también se denominarán simplemente *nodos* o, por asociación, *sensores*) están desplegados en unas instalaciones industriales o en una gran infraestructura civil. Cada nodo recopila mediciones que interesan ser supervisadas, filtra las mediciones y las transmite a una estación base (BS, *Base Station*), que también es estática. En la BS las mediciones son registradas, procesadas y/o transmitidas a un centro remoto. Esta es una configuración típica de muchos esquemas existentes en aplicaciones de monitorización. Además de los nodos sensores y la BS, el sistema incluye un robot aéreo, también denominado vehículo aéreo no tripulado (UAV, *Unmanned Aerial Vehicle*) o genéricamente estación móvil (MS, *Mobile Station*), que realiza tareas de inspección y detección de fallos. A los elementos participantes en la red los llamaremos genéricamente entidades y existirán tres tipos: BS, robot y nodo sensor. La Figura 3.1 muestra un ejemplo de red.

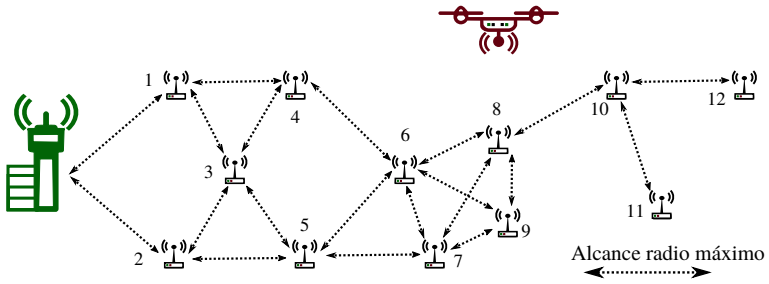


Figura 3.1 Ejemplo de topología de red.

3.1.1 Consideraciones sobre la estación base

Se supone que la estación base es un equipo con suficientes recursos (memoria, potencia computacional, comunicaciones) y no está limitada en cuanto al consumo de energía que puede utilizar por estar conectada directamente a la red eléctrica mediante un sistema de alimentación ininterrumpida. Dispondrá de varios adaptadores de red que le permitirá conectarse a distintas redes usando diferentes tecnologías de comunicación. Podrá conectarse a la WSN como sumidero de información, a la red local de la empresa y, opcionalmente tendrá una conexión de largo alcance radio con el UAV (de mayor alcance que las conexiones que puede tener con los nodos).

Se supone que está lo suficientemente protegida y vigilada de tal manera que sea prácticamente imposible el acceso no autorizado a ella y su manipulación.

3.1.2 Consideraciones sobre los nodos sensores

Se supone que los nodos son dispositivos con limitaciones computacionales y energéticas, porque son de bajo costo o están alimentados con baterías y deben consumir la menor cantidad de energía posible durante su funcionamiento. Cada sensor tiene un rango de alcance radio r de manera que, si la distancia entre dos nodos es menor que r, podrán comunicarse. Un robot o una BS podrían tener un rango de alcance mayor, pero no podrían comunicarse bidireccionalmente con un nodo si se encuentran a una distancia mayor de r.

No se consideran distintos tipos de nodos (al contrario de lo que hace el estándar IEEE 802.15.4 [1]). Todos los nodos contemplados tienen la capacidad de reenviar mensajes. La topología de la red formada por los nodos y la BS es una topología de malla en la que existen nodos que no tienen conectividad directa con la BS, por lo que muchos mensajes deben realizar varios saltos para llegar a su destino. La topología de árbol podría verse como un caso simplificado de topología de malla. La Figura 3.1 muestra un ejemplo de topología de red considerada en esta Tesis Doctoral.

Los nodos sensores están diseñados para operaciones de monitorización a largo plazo y no hay fuertes medidas de seguridad para acceder a ellos, por lo que están expuestos a degradación por el medio ambiente y ataques maliciosos incluyendo la manipulación del propio dispositivo y del software que ejecuta, la extracción de información, la clonación o la destrucción total.

El número de nodos puede ser desde unos cientos de nodos o menos (red mediana) hasta miles de nodos o más (red grande). Los nodos no son móviles y la posición de cada uno de ellos es estática en circunstancias normales.

3.1.3 Consideraciones sobre los robots

Al igual que una BS, un robot es un equipo con muchos recursos (memoria, capacidad computacional, comunicaciones) y su consumo de energía puede ser mucho mayor que la de cualquier nodo, ya sea porque dispone de una batería de gran capacidad o porque la recarga pueda hacerse de manera sencilla y rápida. Pero, a diferencia de una BS, es más probable un acceso no autorizado a él mediante la captura y robo por parte de un agente malicioso.

Un robot dispone de componentes capaces de localizar la posición de las fuentes que emiten los mensajes que recibe con la suficiente precisión para distinguir cada nodo por separado. Un robot es capaz de moverse por la zona donde están desplegados los nodos y es capaz de acercarse lo suficientemente a cada uno de ellos y a la BS como para establecer comunicación (a una distancia menor de r). Se da por hecho que el robot ha recibido información sobre la posición de la BS, la zona donde pueden encontrarse los nodos y el valor de r.

Pueden existir varios robots en una red, pero supondremos que solo uno puede estar en funcionamiento en un determinado instante, pudiendo el resto de robots “suplentes” relevarlo de sus funciones si las condiciones lo requieren.

3.1.4 Consideraciones sobre las comunicaciones

Se considera que la red y el escenario son realistas, por lo que pueden producirse errores de comunicación y fallos en los nodos. La red no es muy dinámica: los nodos sensores no suelen fallar, no se añaden frecuentemente nuevos nodos a la red y los nodos son estáticos. Se asume que el canal de comunicación es no fiable: la pérdida de paquetes puede ser originada por colisiones, interferencias y ruido entre otras causas, y la probabilidad de que ocurra no es despreciable y puede ser diferente en distintos emplazamientos del escenario. El nivel de enlace (LL, *Link Layer*) de cada elemento de la red comprobará, usando Verificación de Redundancia Cíclica (CRC, *Cyclic Redundancy Check*) o un Código de Autenticación de Mensaje (MAC, *Message Authentication Code*), que la trama recibida es correcta y en caso contrario la descartará.

Se supone que los canales de comunicación entre entidades del mismo tipo son simétricos, es decir, si una entidad x puede recibir mensajes de otra entidad y, entonces y también podrá recibir mensajes de x.

Existirán parámetros de comunicación que pueden ser diferentes de una red a otra. Estos afectarán al comportamiento de las comunicaciones. Serán preestablecidos por el propietario de la WSN. Ejemplos de estos parámetros son:

- Tiempos de espera máximos para recibir una determinada respuesta esperada y política de reintentos.
- Protocolos de seguridad empleados en cada nivel de red incluyendo algoritmos de cifrado, tipos de certificados, funciones *hash*, etc.
- Utilización de mecanismos para detectar el fallo de comunicación con el otro extremo (envío de mensajes periódicos o *keepalive*).

Todos estos parámetros serán conocidos por configuración por la BS y el robot antes de su puesta en funcionamiento.

3.1.5 Consideraciones sobre las amenazas

Se pueden capturar físicamente los nodos y extraer toda la información que almacena. También se puede modificar el software de un nodo, de tal manera que ejecuten código malicioso. Pero todo esto no puede hacerse instantáneamente, por lo que estos ataques requieren una cantidad de tiempo no despreciable. La captura de un robot por parte de un agente malicioso se detectaría rápidamente y el atacante no tendría tiempo suficiente para poder utilizar las claves criptográficas contenidas en el robot antes de que sean invalidadas.

Se supone que el rango de alcance radio de un atacante d es mayor que el de un nodo, r y que pueden provocar interferencias en parte de la red. Se considera que los ataques de DoS o con interferencias intencionadas (en inglés, *jamming*) no pueden continuar constantemente sin ser detectados y eliminados.

3.1.6 Requisitos deseados de la solución

Se desea encontrar un método de configurar y mantener una red de manera segura y robusta, para redes medianas o grandes. Los métodos utilizados deben ser eficientes energéticamente y con poca sobrecarga en el funcionamiento normal de la red. La red

debe ser capaz de resistir ataques como los descritos, aunque a consecuencia de estos se comprometan o destruyan nodos o robots.

3.2 Idea general de la solución propuesta

Esta arquitectura combina robots y WSN explotando las sinergias entre el procesamiento heterogéneo y las capacidades de comunicación del robot y los nodos de la WSN con el fin de proporcionar soluciones computacionalmente eficientes y seguras para los ataques más comunes. La arquitectura SNSR contiene importantes novedades sobre los esquemas de seguridad de WSN tradicionales. El robot se emplea para configurar el funcionamiento seguro de la red de sensores y es responsable de realizar la autenticación y revocación de los nodos sensores. En otras palabras, el robot implementa funcionalidades de seguridad que en los esquemas tradicionales son asumidas por la estación base, permitiendo la comunicación en un solo salto entre el robot y cada nodo sensor, lo que mejora significativamente la seguridad.

En la Figura 3.2, se muestra un esquema general con la arquitectura SNSR y los módulos principales de seguridad implementados en cada entidad: el robot, la BS y los nodos sensores. Además de estos módulos de seguridad, cada elemento llevará a cabo el resto de las tareas necesarias para su funcionamiento, aunque no se muestran en la figura, por ejemplo, para el robot, la navegación y para un nodo sensor, la recogida de mediciones.

La arquitectura incluye técnicas de seguridad proactivas, como el uso de un esquema de clave pública, y también técnicas reactivas, como la detección de anomalías, que está descentralizada en cada entidad.

La BS contiene los módulos principales de una PKI X.509 descritos en [14] que incluye:

- Una Autoridad de Certificación (CA, *Certificate Authority*), que es responsable de emitir, actualizar y firmar los certificados digitales.
- Una Autoridad de Registro (RA, *Registration Authority*), que verifica la identidad del propietario de un certificado digital que solicita el uso de la CA.
- Un Repositorio de Certificados (CR, *Certificate Repository*), que almacena e indexa certificados digitales y una CRL que contiene los certificados que han dejaron de ser válidos.

En nuestra arquitectura, se supone por simplicidad que los anteriores tres módulos se sitúan en la BS. Sin embargo, también pueden estar en una entidad externa remota que proporcione mayor seguridad física y de red sin suponer ninguna modificación en el funcionamiento del resto de módulos. La BS también incluye módulos para la sincronización de información con robots, así como la supervisión de la red y recopilación de anomalías. Opcionalmente, también puede establecer la topología de la red decidiendo vecindades y calcular las rutas entre todas las entidades.

Todas las entidades de la red tendrán un certificado digital X.509 firmado por la CA que habrá sido instalado previamente antes de que la entidad comience su funcionamiento. Un certificado digital relaciona la identidad de su propietario con una clave pública, ambos firmados por la autoridad certificadora. Al usar el formato X.509 [14], el certificado digital

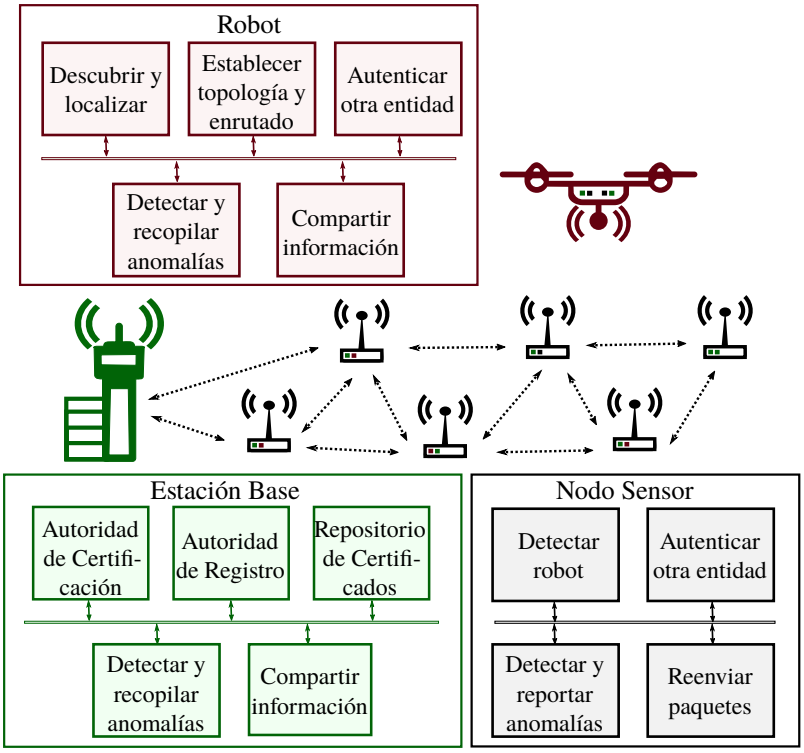


Figura 3.2 Esquema con la arquitectura SNSR y las principales funcionalidades de seguridad implementadas en el robot, la BS y los nodos sensores..

es un documento digital compuesto por un conjunto variable de campos entre los cuales se destacan los siguientes por ser de especial interés en el resto del documento:

- Número de serie: número único asignado por la CA. Se utiliza en la revocación. Una CRL está compuesta por números de serie de certificados revocados.
- Sujeto: identidad del propietario de la clave pública. Está compuesto a su vez por subcampos, siendo el Nombre Común (CN, *Common Name*) el más importante para esta arquitectura.
- Emisor: identidad de la CA que firma el certificado.
- Intervalo de validez: periodo de validez del certificado (fecha de comienzo y fin).
- Clave pública: la clave pública que se quiere asociar con el sujeto.
- Algoritmo de firma: algoritmo que se usará para firmar el certificado.
- Firma: firma del resto de campos realizada por la CA usando el algoritmo indicado.

El CN del certificado de una entidad x será la identidad de x, $\mathbf{ID}_x$, e incluirá el nombre de la entidad, su tipo (robot, BS o nodo sensor) y el nombre de dominio de la red.

Cada entidad autentica al resto de los pares antes de comunicarse con ellos. La autenticación se realiza usando PKI, por lo tanto, cada entidad implementa un módulo para esto.

En la arquitectura SNSR, el robot realiza el descubrimiento y la localización de los nodos sensores. Al robot se le ordena volar siguiendo una trayectoria de manera que pueda comunicarse con cada nodo sensor directamente. En el primer vuelo, el robot descubre y autentica todos los nodos sensores y establece la topología de la red. La ubicación de los nodos sensores se podría conocer a priori o el robot podría estar dotado de técnicas para localizar los nodos sensores, como la descrita en [151]. El robot anuncia su presencia y disponibilidad al resto de entidades transmitiendo tramas de baliza (*beacon*). Cada nodo sensor que no está conectado al robot analiza cada trama de baliza que recibe y, si es correcta, intenta conectarse al robot que la transmitió. Durante la conexión, el robot y el nodo sensor intercambian sus claves públicas, se autentican mutuamente (usando las claves firmadas por la CA) y finalmente establecen un canal de comunicación seguro utilizando claves simétricas aleatorias elegidas durante la conexión. En PKI, la autenticación usa claves asimétricas, que son más seguras que las claves simétricas, pero requieren mayor cómputo [29, 83, 82, 50]. Una vez que las entidades se han autenticado, se establece por mutuo acuerdo una clave simétrica, ya que su uso implica una menor carga computacional: esa clave se utilizará de ahora en adelante para cifrar los datos enviados.

Después de haber descubierto y localizado a los nodos sensores, el robot establece la topología de la red (determina los vecinos de cada nodo sensor) y calcula las rutas entre todas las entidades de la red. Opcionalmente, el robot puede pasar la información descubierta a la BS para que esta realice los cálculos. El robot garantiza una comunicación segura entre nodos sensores autenticados, lo que aumenta la seguridad de las etapas iniciales de la configuración de red y el cálculo de rutas. Después del despliegue, cada nodo sensor espera a ser activado por el robot. Los nodos sensores no establecen comunicación con ninguna otra entidad que no haya sido autenticada previamente por el robot. Asegurar las etapas iniciales de configuración de la WSN es una ventaja de esta arquitectura: las etapas iniciales son críticas en el rendimiento de la red y son objeto de una amplia variedad de ataques [29, 150, 82]. Además, evita que los nodos sensores transmitan paquetes para el descubrimiento de otros nodos, reduciendo significativamente el consumo de energía.

Para mejorar la robustez, dado que el robot tiene una probabilidad de fallo no nula, el robot intercambia con la BS la información obtenida en el descubrimiento de nodos sensores y el establecimiento de la topología. Este intercambio lo realizan los módulos existentes en el robot y la BS de compartición de información. La BS usará esa información para comunicarse con el resto de los nodos sensores. Antes, la BS debe aceptar los datos de establecimiento de la topología y cambiarlos o generarlos si fuera necesario.

Los mecanismos de seguridad reactiva también tienen una gran relevancia en esta arquitectura. Cada entidad implementa módulos para detectar anomalías y ataques maliciosos simples. El robot y la BS funcionan como recopiladores de alarmas, mientras que los nodos sensores implementan funciones para el reporte de alarmas.

3.2.1 Identidad de las entidades

Cada entidad x de la red tendrá asignada una identidad, $\mathbf{ID}_x$, que será una cadena de caracteres compuesta por un nombre único entre las entidades de su mismo tipo, el tipo de la entidad (robot, BS o nodo sensor) y el nombre de dominio de la red (similar a un nombre de dominio absoluto de Internet [100]), utilizando el carácter `'.'` como separador.

Como se dijo antes, el CN del certificado de una entidad x equivale a $\mathbf{ID}_x$.

$$\mathbf{ID}_x = nombre \parallel . \parallel tipo \parallel . \parallel dominio \tag{3.1}$$

Todas las entidades de la red deberían utilizar el mismo nombre de dominio (aunque podría relajarse este requisito para admitir varios) y se habrán definido tres cadenas de caracteres que permitan diferenciar los tres tipos de entidades. En esta Tesis Doctoral se utilizarán las que aparecen en la Tabla 3.1.

Tabla 3.1 Tipos de entidad y cadena de caracteres asociada.

Tipo	Cadena de caracteres
Nodo sensor	`"sn"`
Robot (*mobile station*)	`"ms"`
Estación base	`"bs"`

Si se utiliza como nombre de dominio de la red `"dit.us.es."`, ejemplos de cada uno de los tipos serían: `"robot_x.ms.dit.us.es."`, `"base_y.bs.dit.us.es."` y `"nodo_z.sn.dit.us.es."`.

3.3 Arquitectura

La arquitectura SNSR como *Contribución 1* de esta Tesis Doctoral que se presenta aquí debe verse como una ampliación a las WSN existentes. El objetivo es mejorar y complementar las redes que ya están en uso utilizando elementos móviles adicionales (robots), que en cierto modo pueden verse como una extensión de la BS. Por tanto, la arquitectura se intentará generalizar de tal manera que sea aplicable al mayor número de WSN existentes. Para la descripción de la arquitectura no se supondrá el uso de ninguna WSN o torre de protocolos concretos y se partirá del mínimo común que comparten todas estas redes. Una torre genérica de protocolos para este tipo de redes (e.g. [92, 8]) se muestra en la Figura 3.3. Esta torre de protocolos es similar a la usada por otros autores [5, 171] pero adaptada a los intereses de esta Tesis Doctoral.

En la literatura a veces se denominan *capas* a los niveles pero en esta Tesis Doctoral se prefiere este último término. El Plano de Gestión y Seguridad es transversal a todos los niveles del plano de comunicación. Es en este plano dónde se localizan las principales aportaciones que propone esta arquitectura. Además se modificarán los niveles TL y LL (si aún no soportan la creación de un canal seguro) y el nivel de aplicación (AL, *Application Layer*) deberá incluir nuevos mensajes para la comunicación entre los planos de gestión de distintas entidades. El Plano de Gestión y Seguridad recibirá eventos de todos los niveles del Plano de Comunicación.

Desde el punto de vista del enrutamiento, e independientemente de los planos antes descritos, se dividen lógicamente los elementos, funciones y el tráfico de información destinados a gestionar y mantener el enrutamiento (plano de control) y aquellos destinados a transportar los datos de los servicios y usuarios que utilizan la red (plano de datos o plano

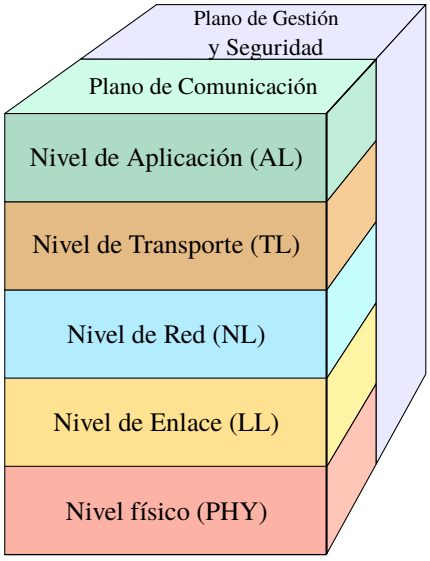


Figura 3.3 Torre de protocolos genérica para un nodo sensor.

de usuario). En esta arquitectura, como en el paradigma Redes Definidas por Software (SDN, *Software Defined Networking*) [61], se desacopla el plano de control, que es el plano que toma las decisiones de cómo enrutar los paquetes, del plano de datos, que es el que realiza el reenvío de paquetes. El plano de control estará implementado exclusivamente en la BS y el robot, mientras que los nodos sensores tan solo tendrán una aplicación de gestión encargada de recibir y aplicar las reglas que se decidan en el plano de control.

La seguridad se aplicará principalmente en los niveles LL y TL. Se soportarán los 6 modos de seguridad indicados en la Tabla 3.2.

Tabla 3.2 Modos de seguridad soportados por la arquitectura SNSR.

		Seguridad en TL		
		No	Excepto vecinos	Sí
Seguridad en LL	No	`SEG_NO`	`SEG_TE`	`SEG_T`
	Sí	`SEG_L`	`SEG_LTE`	`SEG_LT`

Dependiendo del uso que se le vaya a dar a la WSN, su situación física, la criticidad y el grado de confidencialidad deseados, los requisitos de seguridad pueden cambiar. Aplicar siempre la máxima seguridad puede suponer un consumo de recursos innecesario en algunos casos, al igual que no aplicar ningún mecanismo de seguridad puede ser irresponsable en otros. Cada modo de seguridad indica en qué nivel se establecerá un canal de comunicación seguro entre los participantes: ninguno, LL, TL o ambos. En LL o se

aplica seguridad a todos los enlaces o a ninguno, pero en el nivel de transporte se hace la distinción si la comunicación se va a establecer con otro nodo que está en el rango de alcance radio y por tanto será un vecino en el LL. La columna «Excepto vecinos» indica que en el TL se aplicará seguridad solo si los extremos no son vecinos y los motivos por los que se ha incluido son los siguientes:

- Si se está aplicando ya seguridad en LL y se establece una conexión de transporte entre dos vecinos, puede parecer excesivo aplicar de nuevo seguridad para datos que no tienen que pasar por nodos intermedios para llegar a su destino. Aquí se elegiría el modo `SEG_LTE`. Sin embargo, se podrían plantear escenarios en los que aparezcan frecuentes interferencias entre dos nodos por causas aceptables y se pierda la comunicación directa durante periodos cortos. En estos casos se pueden utilizar temporalmente rutas alternativas a elección del NL que sí impliquen el uso de nodos intermedios. Si en estos últimos casos se quiere evitar el acceso de los nodos intermedios a la información transmitida se elegiría el modo `SEG_LT`.
- Si en una instalación se considera segura la transmisión local de mensajes y no se aplica seguridad en el LL, cuando se establezca una conexión de transporte entre vecinos puede parecer innecesario aplicar seguridad. En este caso se elegiría el modo `SEG_TE`. Igual que se explicó en el punto anterior, otras veces se desea asegurar estas conexiones en los casos en los que se pierda la conexión directa y se usen rutas alternativas y así evitar el acceso a la información en nodos intermedios. En este último caso se utilizaría el modo `SEG_T`.

En el caso de que la seguridad esté activada en una conexión (en LL o TL), existen muchas suites de cifrado (*cipher suites*) que se pueden escoger para conseguir en los datos transmitidos:

- Integridad.
- Confidencialidad y privacidad.
- Autenticación y no repudio.

En esta Tesis Doctoral se supondrá el uso de certificados usando criptografía ECC para realizar la autenticación de las entidades. Por consiguiente, se utilizarán las suites de cifrado que incorporen este tipo de criptografía.

En las siguientes subsecciones se describirá la torre de protocolos genérica común a todas las entidades, los componentes funcionales comunes y los datos almacenados de todas las entidades, las características adicionales que incluyen los distintos tipos de entidades y las relaciones entre ellas.

3.3.1 Torre de protocolos

Seguidamente se describen las funcionalidades mínimas requeridas en cada nivel de la torre de protocolos para implementar la arquitectura desarrollada usando una red existente. En la Figura 3.3 puede parecer que un nivel solo se relaciona con aquellos colindantes, pero esto es una visión algo simplista. Puede haber relaciones entre niveles no vecinos también.

Durante las descripciones siguientes se propondrán los campos que deben contener las cabeceras de las unidad de datos de protocolo (PDU, *Protocol Data Unit*) de cada nivel. Los nombres y tamaños de estos campos son meramente orientativos intentando que sean lo más genéricos posibles. Las redes reales posiblemente usarán nombres de campos diferentes con la misma semántica.

3.3.1.1 Plano de Gestión y Seguridad

En este plano está la mayor novedad que propone esta arquitectura. Podría separarse en dos, como se hace en otras arquitecturas [8], uno destinado a la gestión y otro a la seguridad. En esta Tesis Doctoral, debido a la alta interrelación entre ambos, se ha optado por considerarlo uno solo aunque con dos funcionalidades principales claramente diferenciadas.

Este plano tendrá acceso a todos los niveles del plano de comunicación y se comunicará con su par en otras entidades utilizando una aplicación de gestión (en el AL). Proporciona dos servicios: el servicio de gestión y el de seguridad. Todo esto está representado en la Figura 3.4. Mientras que el Plano de Comunicaciones puede ser idéntico en todas las entidades, el Plano de Gestión y Seguridad y la aplicación de gestión dependerá de cada tipo de entidad, tal como se mostró en la Figura 3.2 de la Sección 3.2.

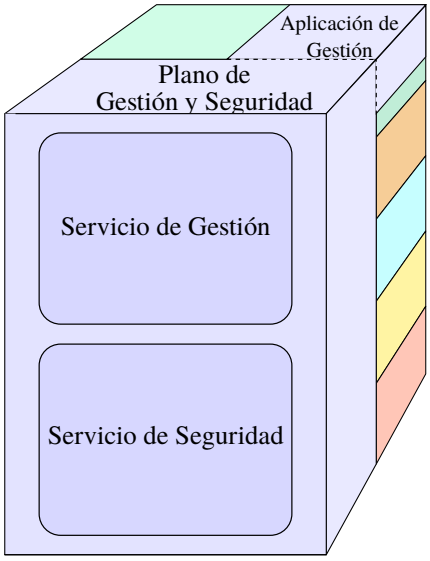


Figura 3.4 Esquema del Plano de Gestión y Seguridad.

A continuación se presentan de forma resumida los servicios incluidos en este plano. Los detalles del funcionamiento y protocolos utilizados se explicarán en la Sección 3.3.2 y siguientes.

Servicio de gestión

Este servicio será diferente según el tipo de entidad. Se encargará de las tareas de:

- Puesta en marcha inicial. Recuperación de los parámetros de configuración, inicialización de todos los componentes y verificaciones iniciales.
- Detección de otras entidades como el robot y comunicación con estas a través de una aplicación de gestión. El funcionamiento con cada entidad dependerá del tipo aunque con una base común.
- Modificación de la configuración posterior para adaptarse a la topología de la red. Los nodos sensores recibirán esta configuración a través del robot o la BS. La BS lo hará con la información de la topología calculada.
- En el caso de BS y robot, cálculo y almacenamiento de la información de la topología de red. Esta información será intercambiada entre el robot y la BS de manera que se mantenga sincronizada y actualizada.
- Monitorización del funcionamiento del resto de componentes y de las comunicaciones. Detección de posibles anomalías. Las anomalías se almacenarán hasta que puedan ser transmitidas a la BS o el robot (preferentemente al que sea vecino). El robot transferirá las anomalías que haya detectado él junto con las detectadas por otros nodos a la BS lo antes posible.

Servicio de seguridad

El servicio de seguridad proporciona al resto de componentes de la entidad:

- La realización del proceso de saludo utilizado para el establecimiento de conexiones locales o remotas que implica la autenticación del otro extremo y la negociación de algoritmos de seguridad y claves a utilizar.
- Si es necesario, según la configuración y utilizando los algoritmos negociados, la generación de campos de cabecera adicionales para proporcionar seguridad a la transmisión de los datos en distintos niveles del plano de comunicación, el cifrado de los datos y la generación del MAC.
- La implementación de los algoritmos de seguridad, de criptografía asimétrica y simétrica, de funciones *hash*, etc.
- La generación de números aleatorios seguros. Muchos algoritmos de seguridad requieren usar números aleatorios que no deberían ser predecibles. También se utilizan en protocolos de comunicación para evitar colisiones, elegir números de secuencia o de transacciones únicos, elección de rutas, etc.
- El almacenamiento de todo el material de seguridad como:
  - Certificados de la autoridad certificadora $\mathbf{Cert}_{CA}$ y el suyo propio firmado por esta $\mathbf{Cert}_{ID}^{CA}$.
  - La clave privada asociada a su certificado $\mathbf{PR}_{ID}$.
  - Una Lista de Revocación de Certificados (CRL, *Certificate Revocation List*). No es necesario que los nodos sensores contengan la lista completa si todos los certificados tienen un CN único. Solo necesitarían los certificados revocados de tipo robot, ya que el resto de las entidades con las que se comunican han

debido ser autenticadas antes de ser configurados. La BS y el robot sí necesitan la lista completa.

- Parámetros de seguridad, algoritmos empleados, números de secuencia, claves maestras y temporales, y números de serie de los certificados de las entidades con las que tiene una conexión segura. Todos estos datos se utilizan durante la comunicación segura entre entidades. Si se utiliza el saludo de reanudación, que se explicará en la Sección 3.3.2.4, también incluirá un testigo que identifique unívocamente estos parámetros. El almacenamiento de los números de serie permite identificar si hay establecida una conexión con una entidad que pertenece a la CRL cuando esta se actualice.
- Dependiendo del modo de difusión elegido (ver Sección 3.3.1.4), es posible que sea necesario almacenar la clave pública de la estación base para poder autenticar los mensajes de difusión.

Cuando un certificado sea comprometido (por ejemplo, un nodo sensor o el robot ha sido capturado), este debe ser revocado por la CA (que en principio está integrada con la BS). La revocación de un certificado implica la generación de una nueva CRL que debe ser transmitida a todas las entidades de la red. La transmisión a los nodos sensores se hará usando uno de los siguientes procedimientos:

- La BS transmite un mensaje conteniendo la nueva versión de la CRL a cada nodo con el que la BS haya establecido una sesión de extremo a extremo.
- La BS inunda la red con un mensaje de difusión que contiene la nueva versión de la CRL. Este mensaje puede ser firmado por la BS para garantizar la autenticidad.
- El robot también puede actualizar la CRL de los nodos que actualmente tienen una sesión establecida con él.

Al recibir una actualización de la CRL, el servicio de seguridad comprobará si algunas de las conexiones actuales están utilizando los nuevos certificados revocados y, si es así, cerrará estas conexiones.

Tener centralizadas todas las funciones de seguridad en un único componente en vez de hacerlo en cada nivel que las necesita tiene ventajas como la reducción del espacio de almacenamiento necesario tanto en código como en memoria. Pero la reutilización de la información requiere que los distintos niveles confíen entre sí, cosa que es habitual en el software de nodos sensores [8].

Este servicio debe ser similar en todos los tipos de entidades y las funciones relacionadas con la seguridad pero específicas de un tipo de entidad quedarían fuera. Por ejemplo, previamente se indicó que la BS podría contener los módulos principales de una PKI (o estar en una entidad externa) y estos módulos no serían incluidos en el servicio de seguridad.

Los niveles de comunicación son independientes del nivel de seguridad empleado y de esta manera el protocolo de seguridad empleado podría cambiar sin influirles. Tener protocolos de seguridad comunes en distintos niveles facilita su mantenimiento, pero implica que los protocolos sean lo suficientemente flexibles para adaptarse a las particularidades de cada nivel. Si se analizan los distintos protocolos de seguridad más utilizados (ver

Sección 3.3.2), se pueden ver que estos tienen dos fases principales: el proceso de saludo y la fase de comunicación posterior. Las mayores diferencias se dan en el proceso de saludo, pero todos se basan en unos principios comunes.

Una vez realizado el proceso de saludo, si en un nivel se añade seguridad, los datos a enviar deberán pasar por el servicio de seguridad que modificará estos si es necesario y añadirá campos adicionales de cabecera. Los campos que habitualmente se añaden son [8, 125, 81]:

- Identificador de la clave o conjunto de claves temporal que se están usando para proteger los datos. A veces también se conoce como época (*epoch*). Las entidades suelen intercambiar una clave maestra que se utiliza para derivar otras claves temporales a partir de un índice (época) usando un algoritmo. Estas claves temporales se renuevan cada cierto tiempo. Este campo es útil sobre todo durante la transición de una clave a otra, para poder decodificar y verificar correctamente los datos. Hay que tener en cuenta que los mensajes pueden llegar desordenados, pueden perderse, etc.
- Un número de secuencia que se incrementa en cada mensaje. El nivel donde se aplica la seguridad podría tener un campo similar, así que se podría utilizar para ambas funcionalidades. Este campo protege ante la duplicación de mensajes y garantiza la frescura de los datos. Para evitar la falsificación también se puede cifrar como se hace en el protocolo QUIC [148].
- Dependiendo de los algoritmos de seguridad utilizados, por ejemplo al usar el modo encriptación autenticada con datos asociados (AEAD, *Authenticated Encryption with Associated Data*), pueden incluirse otros campos:
  - Un número aleatorio que debería ser único y no repetible, que suele llamarse *nonce* o vector de inicialización (IV, *Initialization Vector*).
  - Un MIC o su variante que incluye autenticación MAC. El MAC suele basarse en funciones *hash* cuyo resultado es cifrado posteriormente. Es frecuente colocar este campo al final de los datos, por lo que no sería propiamente un campo de cabecera.
  - Relleno (*padding*). También se suele añadir al final

Así, los datos, una vez protegidos con el modo AEAD, tendrían una estructura como la mostrada en la Figura 3.5, donde el tamaño de los campos es solo orientativo.

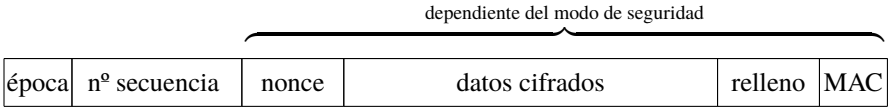


Figura 3.5 Formato de datos con seguridad añadida.

3.3.1.2 Nivel físico (PHY)

En esta arquitectura se supone que el nivel físico (PHY, *Physical layer*) es inalámbrico (usando radiofrecuencia), al menos el usado por el robot con el resto de entidades. Cualquiera de los existentes podría utilizarse en principio. El nivel físico definirá la modulación

radio, la señalización, la tasa de transferencia, los canales a utilizar, cómo se controla el consumo y la potencia de la señal, etc.

3.3.1.3 Nivel de enlace de datos (LL)

La funcionalidad del LL suele dividirse en 2 subniveles:

- Control de Acceso al Medio (MAC, *Media Access Control*).
- Control de Enlace Lógico (LLC, *Logical Link Control*).

A veces, la misma funcionalidad, especialmente el control de errores y flujo, puede estar localizada en distintos subniveles según el protocolo, por lo que desde el punto de esta arquitectura este nivel se verá como un único bloque.

En este nivel se componen las tramas y se transmiten. Una vez recibidas se comprueba su validez, detectando y/o corrigiendo posibles errores. Esta detección debe ser lo suficientemente fiable para evitar requerir comprobaciones adicionales en niveles superiores para datos transmitidos entre dos nodos vecinos.

En cuando al control de acceso al medio, la mayoría de redes inalámbricas utiliza CSMA con CA, así que se supone este protocolo de resolución de contienda. No es necesario que en este nivel se utilicen asentimientos, aunque su uso podría suponer una ventaja en entornos muy ruidosos. En algunas redes se puede sincronizar la comunicación de los participantes utilizando balizas, estructuras de supertramas y ranuras de tiempo reservadas. En esta arquitectura se supondrá que este no es el caso.

Cada dispositivo tendrá asignado una dirección MAC única. Los campos mínimos que debe contener una trama de datos serán (opcionalmente pueden existir otros):

- Dirección MAC destino.
- Dirección MAC origen.
- Datos: unidad de datos de servicio (SDU, *Service Data Unit*) de este nivel.
- Secuencia de comprobación de trama (FCS, *Frame Check Sequence*).

Estos campos se muestran representados en la Figura 3.6.

Figura 3.6 Formato de una trama de datos mínima.

La longitud de los datos podría aparecer explícitamente como un campo adicional o ser calculado implícitamente restando el tamaño de la cabecera de la trama al tamaño total, si es conocido. La longitud y codificación de todos estos campos serán dependientes del protocolo en concreto. El nivel superior tratará las direcciones MAC como datos opacos y podrá obtener del LL el tamaño máximo de SDU, conocido como unidad de transmisión máxima (MTU, *Maximum Transmission Unit*). Se supone que existirá un único protocolo de NL, por lo que no será necesario tener un campo discriminador de protocolo superior.

La comunicación entre dos vecinos va a requerir una asociación previa entre estos. Esta asociación se iniciará con un proceso de saludo que se explicará más adelante. Durante la

asociación entre dos entidades se puede verificar que el otro extremo sigue en funcionamiento obligando a transmitir al menos una trama en un intervalo determinado (trama de pulso o *keepalive*) si no hay datos que transmitir. Este tipo de trama se puede identificar con algún campo de tipo de trama o simplemente transmitiendo unos datos que no puedan ser válidos en el nivel superior. Aquí se propone la transmisión de un único byte de datos con valor 0.

En los modos de seguridad `SEG_L`, `SEG_LTE` y `SEG_LT`, esta asociación añadirá campos adicionales a la trama que permitan proporcionar los mecanismos de seguridad deseados. Esto hará que la MTU disminuya. Antes de generarse la trama, los datos pasarán por el servicio de seguridad que los modificará. En la Figura 3.7 se puede ver el proceso realizado para una trama segura.

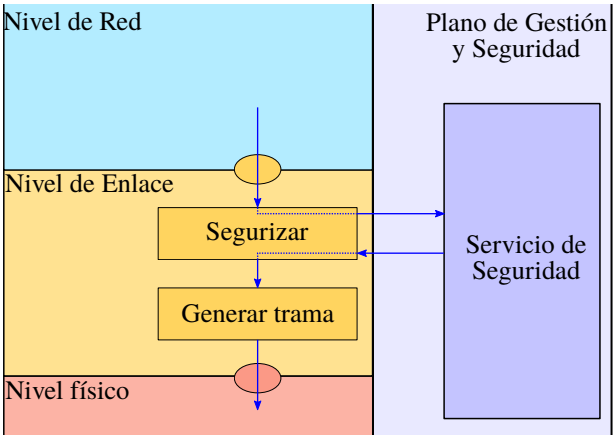


Figura 3.7 Detalle de aplicación de seguridad a una trama.

Debe existir una dirección MAC de difusión que se utilizará exclusivamente para que el robot pueda anunciarse al resto de entidades. Por seguridad, se ha optado por impedir que otro tipo de entidad pueda enviar tramas de difusión. Si una entidad tiene que enviar una misma información a todos los vecinos con los que tiene una asociación previa, desde el nivel superior se deberá solicitar el envío de una trama individualizada a cada vecino con esta misma información. Si las asociaciones están cifradas, la información se cifrará con una clave diferente para cada vecino. Esto reduce el rendimiento, pero mantiene la seguridad. En la Sección 3.3.1.4, se mostrará que este caso se produce infrecuentemente.

Este nivel tendrá interfaces con otros niveles/planos para las siguientes operaciones:

- Con nivel superior:
  - ↕ Recepción y envío de SDU.
  - ↑ Notificaciones de creación/destrucción de asociaciones.
  - ↓ Consulta de estadísticas, información de asociaciones, reportes de calidad de enlaces.
  - ↓ Consulta de la MTU actual.

- Con plano de gestión y seguridad:
  - ↑ Notificación de creación/destrucción de asociaciones.
  - ↕ Realización del proceso de saludo: procesos de cifrado, autenticación, etc.
  - ↓ Configuración de parámetros y asociaciones.
  - ↓ Consulta de estadísticas, información de asociaciones, reportes de calidad de enlaces.
  - ↓ Consulta de la MTU actual.
- Con nivel inferior:
  - ↕ Recepción y envío de PDU.

En el anterior listado, el símbolo ↓ indica que la operación es un servicio ofrecido por el nivel inferior al superior, ↑ que es una notificación/indicación del nivel inferior al superior y ↕ que son operaciones en ambos sentidos.

3.3.1.4 Nivel de red (NL)

Cualquier nivel de red (NL) que permita la comunicación entre dos entidades de la red aunque no sean vecinos sería válido para esta arquitectura. No se pone ningún requisito más por lo que esta arquitectura sería adaptable a prácticamente todos los existentes.

La mayoría de los protocolos de red empleados en WSN proporcionan servicios no orientados a conexión usando datagramas y las principales diferencias se dan en el protocolo de enrutamiento utilizado. En estas redes es habitual usar protocolos de encaminamiento ad hoc, que requieren un esfuerzo colaborativo en los nodos por lo que están expuestos a manipulación por alguno de ellos. Existen protocolos propuestos de encaminamiento ad hoc seguros [164], que no proporcionan seguridad completa ante todos los ataques a costa de reducir el rendimiento de la red e incrementar el procesamiento en los nodos. Aunque se podrían seguir utilizando estos protocolos, esta arquitectura permite utilizar otros protocolos más simples y menos costosos para los nodos manteniendo un nivel de seguridad alto, como:

- Protocolos basados en tablas de encaminamiento almacenadas. Estas pueden ser proporcionadas por configuración por el robot o la BS que tienen una visión global de toda la red e incluir rutas alternativas para incrementar la robustez ante fallos de nodos.
- Protocolos basados en el paradigma SDN como [47]. Los nodos solo deben conocer a priori como comunicarse con el controlador porque pueden preguntar a este qué deben hacer ante paquetes que no saben tratar. En nuestra arquitectura, las funciones de controlador de SDN podrían hacerla la BS o el robot.

Es común asignar una dirección de red diferente a cada una de las entidades de la red y usar esta para hacer el enrutamiento. Esta dirección de red puede ir cambiando durante la vida de un nodo a diferencia de la dirección MAC. Cada nodo deberá guardar una tabla de correspondencia entre la dirección de red y la dirección MAC del resto de entidades conocidas (vecinos principalmente).

La comunicación entre vecinos (con un único salto) debería poder utilizar direcciones de red locales al enlace que podrían repetirse en el resto de la red. Esto permitiría a nodos que aún no tengan asignada una dirección de red o cuya dirección de red sea incorrecta/duplicada comunicarse con vecinos. Estas direcciones no serían nunca reenviadas. De hecho, es preferible que los vecinos utilicen direcciones locales al enlace entre ellos, ya que esto puede ser una garantía de autenticidad (un nodo remoto no podría suplantar la dirección local del vecino).

El protocolo de enrutamiento usado va a imponer la necesidad de incluir determinados campos en las cabeceras de los paquetes que pueden no ser necesarios en otros protocolos. Pero hay una serie de campos que se repiten en todos:

- Límite de saltos, también conocido como tiempo de vida (TTL, *Time To Live*) en algunos protocolos. Campo que indica el límite de saltos que puede dar un paquete antes de ser descartado. Se va decrementando al pasar por cada nodo intermedio y no puede ser 0.
- Un campo con opciones, dependiente del protocolo.
- Dirección de red de origen.
- Dirección de red de destino.

Con estos campos, y suponiendo direcciones de red de 16 bits, un paquete de red básico presentaría el formato mostrado en la Figura 3.8.

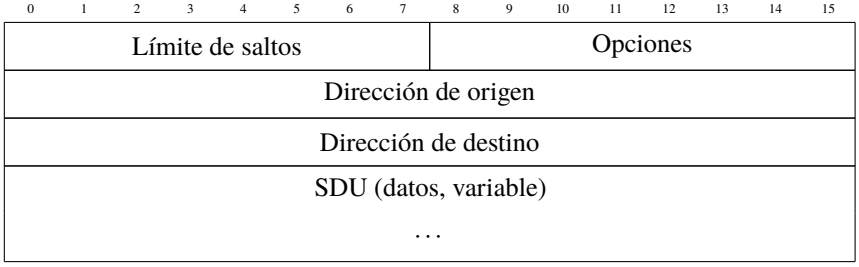


Figura 3.8 Formato de un paquete de red básico.

Difusión y multidifusión

La principal imposición a este nivel es sobre la difusión de paquetes (*broadcast*). En las redes más comunes es habitual implementar la difusión en este nivel apoyándose en los mecanismos de difusión existentes en el LL. Pero, como se indicó en la Sección 3.3.1.3, la difusión en el LL solo se emplea para que el robot anuncie su proximidad.

Por otro lado está el problema de la seguridad en los paquetes de difusión. En redes basadas en IEEE 802.15.4, por ejemplo ZigBee, se utiliza una clave de red (compartida por todas las entidades) para cifrar los paquetes de difusión. Sin embargo, en esta arquitectura no existirán claves compartidas porque el acceso no autorizado a cualquiera de las entidades afectaría a toda la red.

Esta arquitectura puede funcionar sin necesidad de emplear difusión en el NL, pero como se explicará más adelante se va a permitir exclusivamente a la BS enviar mensajes de

difusión al resto de entidades y en situaciones muy concretas e infrecuentes. Esta difusión será controlada por los niveles superiores y el plano de gestión y seguridad. Al NL solo se pedirá:

- que marque los paquetes de difusión, usando un bit del campo opciones o utilizando una dirección de destino especial,
- que notifique la llegada de estos paquetes de manera diferenciada respecto al resto de paquetes,
- y que permita la transmisión de un mismo mensaje a todos los vecinos con la excepción de un conjunto de ellos.

La difusión seguirá un algoritmo de inundación, donde cada entidad retransmitirá un mensaje, una vez verificado, a todos sus vecinos excepto aquellos que se lo hayan enviado a ella (uno como mínimo si la retransmisión es inmediata o puede que varios si se espera un intervalo de tiempo para poder recibir de otras fuentes antes de retransmitir). El propio mensaje debe incorporar información para que pueda ser verificado y autenticado. La verificación incluye al menos que el mensaje no sea duplicado ni antiguo para evitar ataques de reproducción/reinyección (*replay*/*playback*) que consumirían muchos recursos. La autenticación es opcional, pero recomendable y se basará en la firma digital del mensaje utilizando criptografía asimétrica. La verificación de una firma digital es un proceso computacionalmente costoso que es asumible para mensajes esporádicos, pero si se quieren transmitir varios mensajes de difusión consecutivos o el mensaje se quiere fragmentar en varios paquetes, se pueden utilizar técnicas más eficientes, como la descrita en [166], donde una misma firma digital se amortiza entre varios mensajes consecutivos.

En una red densamente poblada, una entidad recibirá el mismo mensaje desde diferentes vecinos lo que proporcionará fiabilidad ante posibles pérdidas de paquetes. En otros casos se puede utilizar asentimientos o repetir la transmisión un número determinado de veces a intervalos.

La multidifusión (*multicast*) no se usa en esta arquitectura, pero si una aplicación la necesitara estaría basada en los mismos mecanismos que la difusión.

Interfaces con otros niveles o planos

Este nivel tendrá interfaces con otros niveles/planos para las siguientes operaciones:

- Con niveles superiores:
  - ↕ Recepción y envío de SDU, para transmisión de difusión o unidifusión (*unicast*).
  - ↓ Consulta de la MTU actual.
- Con plano de gestión y seguridad:
  - ↑ Notificación de creación/destrucción de vecinos.
  - ↓ Configuración de parámetros y vecinos.
  - ↓ Consulta de estadísticas, información de vecinos y enrutamiento.
  - ↓ Consulta de la MTU actual.

- Con nivel inferior:
  - ↕ Recepción y envío de PDU.
  - ↓ Consulta de la MTU actual.
  - ↑ Recepción de notificaciones sobre asociaciones
  - ↓ Consulta de información relevante para el enrutamiento (por ejemplo, alcanzabilidad y calidad de los vecinos).

En el listado anterior, los símbolos tienen el mismo significado que en la Sección 3.3.1.3.

3.3.1.5 Nivel de transporte (TL)

El nivel de transporte debe proporcionar la comunicación entre dos entidades de la red en modo con conexión y preferentemente orientado a flujo de mensajes. La conexión se iniciará utilizando el mismo proceso de saludo que se utiliza en el LL aunque pudiendo cambiar los valores de los temporizadores y retransmisiones. Del mismo modo que en el LL, si el modo de seguridad escogido lo indica, se cifrará la conexión. También se puede configurar la verificación de alcanzabilidad del otro extremo forzando la transmisión de segmentos cada cierto tiempo.

Entre dos extremos solo existirá una única conexión como máximo. Suponemos que todos los usuarios del servicio de transporte comparten el mismo nivel de seguridad, confían mutuamente, no interfieren entre sí maliciosamente y no requieren usar conexiones diferentes. Esto aumenta la eficiencia al no tener que realizar varios procesos de saludo que suelen ser lo más costoso computacionalmente. Además, no es necesario utilizar direcciones de transporte. La conexión entre entidades se identifica por sus 2 direcciones de red. Este nivel multiplexará todos los mensajes del AL en el envío y hará la desmultiplexación en la recepción, si existe más de una aplicación y no se realiza esta función en el AL.

Para implementar la arquitectura SNSR sería deseable que este nivel ofrezca el envío fiable y no fiable de mensajes a un mismo destino sobre la única conexión, para reducir el número de conexiones a utilizar. Si el tamaño de los datos es superior al que se puede transmitir en un único paquete de red, se deberá realizar la segmentación de estos datos en fragmentos antes de enviarlos. En el destino se realizará el reensamblado antes de la entrega al nivel superior. La transmisión fiable de mensajes incluirá mecanismos para la entrega ordenada, el control de flujo y la recuperación ante errores. En la transmisión no fiable de mensajes no se garantiza nada de lo anterior.

No se necesita el envío simultáneo de más de un mensaje en cada dirección. Lo anterior implica que en el caso de la transmisión fiable de varios mensajes consecutivos que deben ser segmentados, los segmentos no se enviarán intercalados.

Si se utiliza una conexión segura, los segmentos son procesados por el servicio de seguridad antes de ser enviados al NL. El proceso de saludo (que se explicará en la Sección 3.3.2) y el procesamiento de la información antes de ser enviada para añadir seguridad es realizado en el servicio de seguridad. En la Figura 3.9 se representa este proceso. Nótese que se hace en orden inverso a como se hacía en el LL. Esto hace que los campos del segmento puedan ser encriptados y/o autenticados. El orden es inverso también respecto al que se utiliza en el protocolo TLS con Protocolo de Control de Transmisión (TCP, *Transmission Control Protocol*) [31]. Esto es posible porque solo se va a tener una

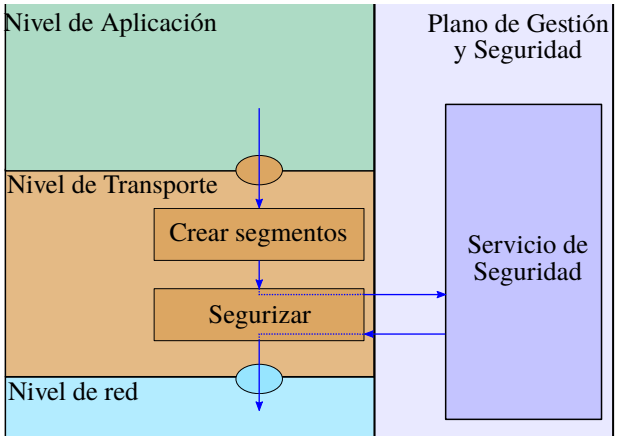


Figura 3.9 Detalle de aplicación de seguridad a un segmento.

única conexión de transporte con unos parámetros de seguridad también únicos entre dos entidades. Aplicar seguridad a los datos antes de crear los segmentos provocaría que los campos de la cabecera de un segmento no estuvieran protegidos y esto conlleva una serie de riesgos de seguridad ante determinados ataques:

- Un nodo intermedio malicioso realizando reenvío selectivo puede seleccionar descartar los segmentos que provoquen más daño. Por ejemplo, un fragmento si no se emplea transporte fiable.
- Se podría alterar los datos de un fragmento de un mensaje y el destino no podría detectar el error hasta que los datos sean reensamblados. Aunque se detecte el error no se podría saber cuál de los fragmentos es el incorrecto y habría que retransmitir todos los fragmentos de nuevo.
- Un atacante podría detectar el inicio de la transmisión de un mensaje fragmentado e inyectar fragmentos erróneos pero con numeración correcta antes de que el origen lo haga (*rushing attack*). El destino creería que le han llegado segmentos duplicados y no sabría cuál es el correcto. Si la transmisión es fiable, el destino enviaría asentimientos al origen de segmentos que pueden que no hayan llegado, por lo que el origen no los retransmitiría.

Sin embargo, segurizar antes de crear los segmentos presenta la ventaja de una posible reducción en el número de bytes transmitidos al fragmentar un mensaje, ya que algunos campos de seguridad se transmitirían solo una vez, como por ejemplo el código de autenticación de mensaje. Esto también podría implicar un menor tiempo de cómputo, aunque en principio esto último podría ser despreciable.

La función de añadir seguridad a la conexión está en el límite entre los niveles de transporte y de red. Es discutible si esta función debería realizarse en uno u otro nivel. En redes TCP/IP, un esquema de funcionamiento similar está implementado en el NL con el protocolo IPSec [81] en su modo de transporte (el nombre del modo ya indica

ambigüedad). En cambio, en ZigBee esta funcionalidad está implementada en el APS, que es el equivalente al nivel de transporte. Por tanto, la adaptación de esta arquitectura a una red existente tendrá que tener en cuenta esta consideración. En esta Tesis Doctoral, se considerará parte del nivel de transporte porque se presupone que el NL ofrece un servicio no orientado a conexión y la creación de un canal seguro va a requerir el establecimiento de una conexión, un proceso de saludo, etc. que son funciones típicas del nivel de transporte.

Con los requisitos antes descritos, un segmento básico podría tener el aspecto mostrado en la Figura 3.10.

Figura 3.10 Formato de un segmento de datos no seguro básico.

Un segmento debe incluir los siguientes campos como mínimo:

- El tipo de segmento (T), indicando si la transmisión es no fiable, fiable, o fiable con asentimiento selectivo. Este último tipo permitiría el envío de asentimientos selectivos (*SACK*) como los empleados en TCP [44]. Si se emplea la transferencia fiable, habría que transmitir además los campos `Nº secuencia` y `Nº asentimiento` que se describen más abajo.
- Un marcador de fragmentación (F), indicando si este segmento no pertenece a un mensaje fragmentado, es el primer fragmento u otro. Si existe fragmentación se incluiría el campo `Nº fragmento` que se describe más abajo.
- Un número de secuencia, que permite la entrega ordenada en el caso de transmisión fiable o identificar un fragmento en caso de transmisión no fiable.
- Un número de asentimiento, que permite confirmar la entrega de un segmento en el destino.
- Un número de fragmento, cuyo valor sería el número de fragmento o el número del último fragmento si es el primer o último fragmento. Se supone que solo se transmite un mensaje fragmentado a la vez y el primer fragmento debe llegar primero al destino para que pueda reservar espacio y detecte el comienzo del nuevo mensaje fragmentado.
- Datos o asentimiento selectivo.

El receptor de un segmento sabrá a priori si debe incluir seguridad o no, y esto no debe venir especificado expresamente en la cabecera del segmento. Sin embargo, la verificación de errores se simplifica si se garantiza que los primeros bits del segmento son diferentes si incluyen seguridad o no.

Este nivel tendrá interfaces con otros niveles/planos para las siguientes operaciones:

- Con nivel superior:

  - ↕ Recepción y envío de SDU, de manera fiable o no.
  - ↑ Notificación de creación/destrucción de conexiones.
- Con plano de gestión y seguridad:
  - ↑ Notificación de creación/destrucción de conexiones.
  - ↕ Realización del proceso de saludo: procesos de cifrado, autenticación, etc.
  - ↓ Configuración de parámetros y conexiones.
  - ↓ Consulta de estadísticas, información de conexiones, reportes de calidad.
  - ↓ Consulta de la MTU actual.
- Con nivel inferior:
  - ↕ Recepción y envío de PDU.
  - ↓ Consulta de la MTU actual.

En el anterior listado, los símbolos tienen el mismo significado que en la Sección 3.3.1.3.

3.3.1.6 Nivel de aplicación (AL)

En el AL de los nodos se situaría la aplicación de usuario encargada de tomar medidas del sensor cada cierto tiempo, su preparación y envío a la BS como caso típico. La arquitectura que se propone en esta Tesis Doctoral supone que es una aplicación que necesita tener comunicación bidireccional con la BS y posiblemente otros nodos sin entrar en su funcionamiento real.

Independientemente de la aplicación de usuario, existirá una aplicación de gestión, utilizada como mecanismo de comunicación del plano de gestión y que recibirá comandos y configuraciones del plano de control situado en el robot y la BS. El protocolo de comunicación de esta aplicación y su funcionamiento es parte principal de esta arquitectura.

Los datos entregados por el TL deben ser entregados a la aplicación adecuada. Para ello los mensajes del AL incorporarán una cabecera inicial que permita distinguir la aplicación destino. En esta Tesis Doctoral se supondrá que esta cabecera estará formada por un único campo con el tipo de mensaje (*TM*). Las distintas aplicaciones tendrán asignado un rango de tipos de mensajes. El tipo de mensaje también puede verse como un valor formado por un prefijo indicativo de la aplicación y un valor único dentro de esa aplicación. La codificación y significado del resto del mensaje será dependiente de la aplicación. Se supone que las aplicaciones confían entre sí. Es decir, no habrá aplicaciones maliciosas que interfieran o extraigan información de las otras en un nodo correcto. Si existiera una aplicación maliciosa se entiende que toda la entidad está comprometida.

Un mensaje de este nivel tendría el formato que aparece en la Figura 3.11.

TM	mensaje (contenido dependiente de TM)

Figura 3.11 Formato de un mensaje del nivel de aplicación.

La aplicación de gestión de la BS podrá transmitir mensajes de difusión utilizando los servicios del NL. La aplicación de gestión del resto de entidades escuchará los mensajes de difusión que pueda enviar la BS y los retransmitirá a sus vecinos si son válidos.

A modo de resumen y utilizando los mismos símbolos que en la Sección 3.3.1.3, este nivel tendrá interfaces con otros niveles/planos para las siguientes operaciones:

- Con plano de gestión y seguridad:
  - ↕ Interfaz de comunicación del plano de gestión con otras entidades.
- Con TL:
  - ↕ Recepción y envío de PDU desde/a aplicaciones de otras entidades.
  - ↑ Notificación de creación/destrucción de conexiones.
- Con NL:
  - ↕ Recepción y envío de mensajes de difusión.

3.3.2 Proceso de saludo

En esta arquitectura las entidades, a excepción del robot, se comunicarán solo con pares autorizados previamente y con el robot. Una entidad no puede iniciar la comunicación con cualquiera. Las entidades descartarán cualquier mensaje recibido de otras entidades no autorizadas. El robot permite que cualquier otra entidad inicie la comunicación con él como parte de la estrategia de descubrimiento de la topología de la red.

El saludo (similar a lo que se conoce como *handshake* en otros protocolos, por ejemplo, TCP) se realizará como fase inicial en la comunicación de cualquiera dos entidades de la red, ya sea entidades vecinas (con comunicación directa) o entidades remotas (con comunicación indirecta a través de intermediarios). Se realizará en el nivel de enlace y en el nivel de transporte.

Esta primera fase de la comunicación permite realizar las siguientes funciones:

- La verificación de la conectividad entre ambos extremos. Los participantes confirman la existencia del otro extremo y su deseo de participar en la comunicación.
- El acuerdo o sincronización de los parámetros de comunicación que se utilizarán para el intercambio de mensajes. En esta arquitectura, cuando se aplique seguridad se acordarán:
  - Los algoritmos a usar para hacer determinados cálculos, como el algoritmo de intercambio de claves y de autenticación, el algoritmo de cifrado y el algoritmo de MAC. Estos algoritmos pueden estar predefinidos en una red de sensores, pero incluso así es conveniente verificar que ambos extremos utilizarán los mismos algoritmos.
  - Las claves de cifrado/MAC, utilizando los algoritmos anteriores. Idealmente las claves utilizadas deberían tener la propiedad de ser un PFS, es decir las claves utilizadas no deben tener relación con otras utilizadas anteriormente. De esa manera si un atacante descubre las claves actuales, no podrá descifrar comunicaciones pasadas o futuras. Las claves se generan durante el saludo de manera aleatoria y ambos extremos confirman su acuerdo. Es común que solo se intercambie una clave «maestra», y a partir de ella se deriven otras claves diferentes.

- La autenticación de uno o ambos participantes. En esta arquitectura, si se va a segurizar la comunicación, se exigirá que ambos se autentiquen mutuamente.

Existen diferentes estándares con recomendaciones sobre protocolos, esquemas, componentes y primitivas para realizar todo lo anterior. Los protocolos de seguridad TLS [31], DTLS [125] e *Internet Key Exchange* (IKE) en su versión 2 [80] que está asociado con IPSec [81], utilizan el estándar IEEE 1363-2000 y sus revisiones y ampliaciones posteriores [67, 71, 68, 69, 70]. En los protocolos usados por el estándar *ZigBee Smart Energy* [113], se toma como referencia el *Standards for Efficient Cryptography 1* (SEC 1) [19]. La arquitectura que se propone en esta Tesis Doctoral está abierta a adoptar aquellos estándares que sean más eficientes y adecuados para los escenarios utilizados.

La funcionalidad buscada con el saludo dependerá del nivel de seguridad deseado. En el caso más simple, donde no se utilice autenticación ni encriptación y solo se desea verificar la comunicación con el par extremo, bastará con que ambos pares envíen/reciban un mensaje (saludo simple). En otro caso, la comunicación requerirá el intercambio de más mensajes y realizar más operaciones (saludo avanzado). El mensaje inicial del saludo, que llamaremos *Hola*, será diferente según el tipo de saludo a realizar. Las entidades de la red tendrán configurado el tipo de saludo a realizar en los niveles de enlace y transporte. Si se recibe un mensaje *Hola* con formato no esperado será descartado.

3.3.2.1 Entrega fiable, duración y reintentos

El nivel de la torre de protocolos donde se realice el saludo (LL o TL) deberá garantizar la protección ante posibles pérdidas de los mensajes del saludo. Para ello se deberá implementar en ese nivel algunos de los métodos habituales: envíos con asentimientos positivos y/o negativos, retransmisiones con temporizadores, etc.

La duración temporal del saludo se limitará de tal manera que si no ha llegado a su finalización en un tiempo máximo definido, el saludo será abortado por el extremo que detecte tal situación. Esto no evita que posteriormente se reintente el saludo. El número de reintentos es un parámetro que se puede utilizar para estimar la calidad de la conexión y detectar posibles anomalías o ataques en la red.

Opcionalmente, también se podrá establecer un tiempo mínimo de duración de saludos con error de tal manera que la tasa de intentos por segundo de saludos de una conexión no supere un umbral máximo establecido. Si el certificado del otro extremo no se considera admisible o durante el saludo algún mensaje recibido es inaceptable, el saludo se finaliza con error. La detección del error puede ser rápida, pero también puede haber requerido realizar cómputos costosos. El otro extremo podría iniciar inmediatamente un nuevo saludo con el mismo resultado y así continuar indefinidamente. Esto podría ser un comportamiento bienintencionado pero no deseado o un ataque de alto consumo energético o computacional similar al que se explica en la Sección 3.6.1. El tiempo mínimo de duración puede ser adaptativo al número de reintentos realizados, pudiendo incluso ignorarse cualquier otro intento hasta reportar esta anomalía al robot o la BS para que se actúe en consecuencia.

3.3.2.2 Saludo simple

El saludo simple consistirá en un único intercambio de mensajes. El mensaje de *Hola* podrá ser cualquier mensaje válido o, en caso de que la entidad no tenga nada que comunicar inicialmente, un mensaje que contendrá un único octeto con todos los bits a 0 (u otro valor

acordado), como se muestra en (3.2). El otro extremo responderá con otro mensaje válido o el mensaje (3.2). Cualquier extremo puede iniciar el saludo.

$$A \rightarrow B : 0 \tag{3.2}$$

3.3.2.3 Saludo avanzado

El saludo avanzado deberá realizar todas las funciones antes presentadas para proporcionar confidencialidad, autenticidad, integridad y no repudio a los mensajes que se transmitan durante la comunicación entre ambos extremos.

Se diferencian dos roles: cliente (el que inicia el saludo) y servidor (se queda a la escucha y responde). Salvo el robot, que siempre actuará como servidor, el resto de elementos adoptarán el rol indicado durante la fase de configuración, pudiendo ser diferente según el otro extremo. Las funciones realizadas por cada rol en el saludo es diferente y por tanto el consumo requerido también puede serlo según los algoritmos utilizados. Además, la entidad que funciona como servidor está más expuesta a posibles ataques DoS o ataques de alto consumo, entre otros. Al calcular la topología de red se decidirá el rol de cada extremo en una comunicación, intentando que el esfuerzo requerido sea lo más balanceado posible entre los nodos y que la BS siempre realice mayor esfuerzo que los nodos.

El servidor está expuesto a ataques de DoS donde se envíen multitud de mensajes *Hola* que puedan obligar a realizar cálculos costosos. Para evitar esto en lo posible, el servidor conocerá previamente la dirección (de enlace o red según el caso) del otro extremo por configuración. Cualquier mensaje proveniente de otra dirección será descartado.

Cuando la conexión no es directa, un atacante también podría enviar mensajes con direcciones falsificadas invalidando la protección anterior. Por este motivo se debe utilizar un método similar al usado en [125] o [80] basado en *cookies* sin estados si el mensaje de *Hola* recibido por el servidor no se puede autenticar:

1. El servidor creará una *cookie* (siguiendo un procedimiento similar al descrito en el protocolo Photuris [79]). La *cookie* incluye información sobre la petición (origen, fecha, etc.). Solo el servidor debe poder generarla correctamente. La generación y verificación de la *cookie* debe ser rápida y eficiente.
2. La *cookie* generada se envía al cliente en un mensaje que le indica que debe reiniciar el saludo.
3. El cliente vuelve a enviar el mensaje *Hola* adjuntando esta vez la *cookie*. El servidor ahora puede verificar que la dirección existe y que el cliente es alcanzable (ya que ha recibido correctamente la *cookie*).

La elección de los algoritmos a utilizar se realiza en un intercambio de mensajes:

1. El cliente propondrá el conjunto de algoritmos que está dispuesto a utilizar.
2. El servidor elegirá uno de ellos y se lo comunicará al cliente en el mensaje de respuesta.

En general, cuando se inicia el saludo, ambos extremos conocen cuál debería ser la identidad del otro por configuración. La excepción es el robot cuando recibe la conexión

de una entidad por primera vez en la fase de descubrimiento. Una entidad x tiene que demostrar su identidad $\mathbf{ID}_x$ presentando una firma digital de unos datos conocidos d, $E_{\mathbf{PR}_x}(d)$, o del resultado de aplicar una función *hash* sobre d para reducir la longitud, $E_{\mathbf{PR}_x}(H(d))$. Esta firma digital debe estar realizada con la clave privada $\mathbf{PR}_x$ asociada a una clave pública $\mathbf{PU}_x$ que pertenece a esa identidad $\mathbf{ID}_x$. La asociación entre $\mathbf{PU}_x$ y $\mathbf{ID}_x$ se obtiene de un certificado que debe estar firmado por la CA, $\mathbf{Cert}_x^{CA}$, y debería tener un número de serie que no esté revocado. Después de haber acordado los algoritmos a utilizar, cada extremo presentará al otro su certificado. El otro extremo verificará que el certificado pertenece a la identidad esperada, está firmado por la CA y no está revocado. Los datos que se podrían firmar digitalmente podrían ser cualquiera, pero por eficiencia se suelen firmar datos que se han transmitido para otras funciones, lo que a su vez garantiza su integridad. Por ejemplo, en TLS 1.2 [31], el servidor firma los parámetros utilizados en el intercambio de claves y el cliente firma todos los mensajes recibidos y transmitidos en todo el proceso de saludo hasta ese momento.

Existe la posibilidad de obtener previamente los certificados por configuración. En este caso, el robot debe recopilar los certificados de los nodos para después pasarlos a otros nodos que vayan a tener contacto con ellos. Esto requiere usar más espacio de almacenamiento en los nodos y existe la posibilidad de que algunos certificados transmitidos al final no se lleguen a utilizar si la comunicación no es posible. Para mayor seguridad, la validación debería hacerse igualmente en los nodos (verificar que no están revocados y que la firma de la CA es válida), ya que no es totalmente descartable que el software de un robot haya sido alterado. En el caso hipotético (y arriesgado) de que se considerara al robot totalmente seguro, el robot podría transmitir directamente a un nodo la clave pública del otro extremo en vez del certificado, lo que conllevaría un ahorro de espacio de almacenamiento y los nodos no tendrían que validar los certificados.

El acuerdo de las claves de cifrado o MAC a utilizar se hará utilizando uno de los esquemas propuestos por los estándares antes comentados. Estos esquemas se basan en los protocolos Diffie-Hellman (DH) [32] y Menezes–Qu–Vanstone (MQV) [86]. El más común de ellos es el DL/ECKAS-DH1, definido en el estándar IEEE 1363-2000 [67]. DL/ECKAS-DH1 es utilizado en IKE, TLS y DTLS.

Durante el saludo, dado que aún no se ha acordado ninguna clave, los mensajes no llevan ningún MAC que proporcione integridad o autenticidad de los datos. Un atacante podría situarse entre los dos extremos y realizar un ataque de intermediario, en inglés *Man in the Middle attack* (MITM). Este ataque permite interceptar los mensajes transmitidos, modificarlos y suplantar uno o los dos extremos. El atacante podría manipular la elección de algoritmos para que se eligieran algoritmos débiles que posteriormente pudiera descifrar con facilidad o podría intervenir en el intercambio de claves para directamente poder descifrar la comunicación. El mecanismo habitual de protección es intercambiar el resultado de aplicar una función de resumen o *hash* a los datos recibidos y transmitidos (sin tener en cuenta repeticiones y considerando que los mensajes no han sido fragmentados) de manera autenticada mediante:

1. La firma digital de esos datos, que a la vez sirve para demostrar la identidad del extremo transmisor. Esto se suele realizar antes de terminar el intercambio de claves

y permite garantizar que el saludo se está realizando sin intervención de ningún agente externo.

2. Cifrando los datos con la clave acordada, que a la vez permite verificar que la generación de la clave ha sido correcta. Normalmente, se realiza como último mensaje del saludo.

Cualquier alteración de los mensajes intercambiados dará lugar a que el resumen generado sea inválido. Dado que la firma digital solo puede realizarla la entidad que posee la clave privada apropiada y que los protocolos de intercambio de claves garantizan que con la mera escucha de la conversación no se puede extraer las claves intercambiadas, se imposibilita el éxito del ataque de intermediario.

No es necesario comprobar explícitamente la conectividad, ya que va implícita en cualquier intercambio de mensajes que se realice para el resto de funcionalidades.

El orden en el que se transmitirán los mensajes del saludo para conseguir toda la funcionalidad descrita en este apartado será decisión de la implementación, pudiendo realizarse uno a uno o en paralelo para reducir el número de viajes de mensajes que se deben realizar. Un mensaje puede transportar varios tipos de información siempre que el tamaño total esté permitido por el nivel de la torre de protocolos usado.

Para SNSR, no es necesario desarrollar un nuevo protocolo de seguridad. Se puede utilizar un protocolo de seguridad ya existente y adaptarlo. Un protocolo maduro que ha sido revisado, objeto del escrutinio de la comunidad científica, garantiza robustez y fiabilidad. Más aún si se trata de un protocolo de seguridad que ha sido atacado y puesto a prueba en múltiples ocasiones. La adaptación no está exenta de peligros si no se comprenden todos los elementos existentes que contribuyen a hacer el sistema seguro o se añaden otros elementos que cancelan los anteriores o incrementan la predictibilidad del sistema.

El formato de los mensajes del saludo estará definido por la implementación elegida, aunque se exige que el primer byte, $TIPO$, sea diferente de cero y el valor o conjunto de valores utilizados sea fácilmente distinguible del resto de mensajes que no formen parte del saludo como se muestra en (3.3)

$$A \rightarrow B : TIPO \parallel \ldots \tag{3.3}$$

3.3.2.4 Saludo avanzado de reanudación

El coste computacional del saludo avanzado puede ser muy alto. Las relaciones establecidas entre las distintas entidades en esta arquitectura suelen ser permanentes por lo que este coste se amortiza con el paso del tiempo. Las entidades también deben detectar cuando se pierde la conectividad, como se discutirá en la Sección 3.3.4. La conectividad se puede perder permanentemente si la otra entidad falla fatalmente o temporalmente por interferencias o porque se mueva a otra posición como en el caso del robot. Las pérdidas de conectividad temporales pueden ser breves y/o frecuentes. En estos casos, utilizar el saludo avanzado cada vez que se restablezca la comunicación es muy costoso. El saludo avanzado de reanudación es una versión simplificada del saludo avanzado donde se reutilizan los datos de una conexión anterior sin necesidad de realizar todos los procesos del saludo completo.

Para poder utilizar el saludo avanzado de reanudación, el servidor debe haber generado un testigo (en inglés recibe el nombre de *token* o *ticket*) durante la conexión anterior. Este testigo será transmitido al cliente que lo almacenará. Para el cliente el testigo es un dato opaco, pero el servidor podrá codificar en él:

1. Un número de referencia interno del servidor que le permita identificar los datos de una conexión. Este mecanismo está sugerido en [31].
2. Los datos de la conexión serializados y encriptados con una clave que el servidor solo conoce. En este caso el servidor no necesitaría almacenar nada una vez terminada una conexión a costa de transmitir más información y forzar al cliente a almacenarlo él. Este mecanismo está sugerido en [38].

En esta arquitectura se utilizará la primera opción por requerir menos operaciones a realizar en el servidor y ser más equitativo en el almacenamiento.

El cliente adjuntará el testigo en el mensaje *Hola*. El servidor validará la validez de este y retomará la comunicación en el punto en el que se dejó utilizando las mismas claves que se habían generado anteriormente.

Dado que las claves deben renovarse cada cierto tiempo por seguridad, el testigo tendrá un tiempo de vida limitado. Si este vence, se obligará a realizar un saludo avanzado completo para garantizar el PFS.

3.3.3 Interacciones entre entidades

Las interacciones que pueden darse entre diferentes tipos de entidades están representadas en la Figura 3.12. Se muestran en rojo aquellas en las que interviene el robot, en verde donde interviene la BS y en negro/gris las interacciones de nodos sensores. También se muestra la torre de protocolos utilizada para la comunicación. Se representa de diferente color la parte fundamentalmente distinta de cada tipo de entidad: la aplicación de gestión y el servicio de gestión. Tanto el robot como la BS pueden implementar una torre de protocolos dual, pudiendo utilizar la propia de la WSN y otra más rápida, de mayor capacidad y con mayor alcance para la comunicación entre BS y robot como redes TCP/IP sobre redes inalámbricas de la familia IEEE 802.11. Este último modo de comunicación es opcional en esta arquitectura, pero deseable sobre todo por el volumen de información que tienen que intercambiar robot y BS.

Existen 4 interacciones diferentes. La finalidad de cada una aparece descrita en la Tabla 3.3.

Antes de que las distintas entidades puedan interactuar, se deberá establecer una conexión entre ellas, que implicará la realización de un proceso de saludo. El proceso de saludo podrá autenticar a ambos extremos de tal manera que cada entidad podrá conocer la identidad y tipo de su interlocutor. Si no se utiliza seguridad en las comunicaciones, un nodo también tendrá conocimiento del tipo del otro extremo porque para comunicarse con el robot deberá haber recibido previamente una baliza de este y la información sobre la BS a utilizar la habrá recibido por configuración. La BS identificaría al robot por su baliza y al resto de nodos por configuración. El robot sabría quién es la BS por configuración y los nodos por descubrimiento o configuración.

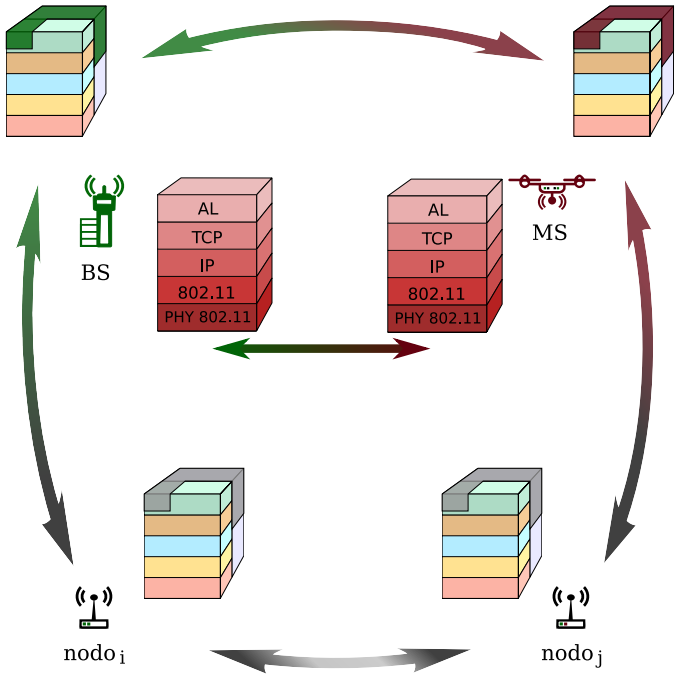


Figura 3.12 Relaciones entre distintas entidades de la arquitectura SNSR.

La duración de una interacción no está definida y las conexiones establecidas se mantendrán abiertas el mayor tiempo posible. Se podrá forzar a que los extremos intercambien datos periódicamente para probar el buen funcionamiento de la comunicación y de los mismos extremos.

En las interacciones se definen una serie de mensajes que utilizarán las aplicaciones de gestión. Estos mensajes se describirán en los apartados siguientes. En la Tabla 3.4 se muestra un listado con los tipos de mensajes definidos, los tipos de entidad origen y destino del mensaje, una descripción breve y el apartado donde se describirán detalladamente.

Los mensajes transmitidos se pueden agrupar en pares petición/respuesta (o notificación/asentimiento en el caso de anomalías). Para identificar correctamente la respuesta a una petición, en las peticiones y respuestas se transmitirá un identificador de transacción *TX_ID* que deberá ser el mismo. El valor del *TX_ID* será un número entero sin signo y nunca podrá ser cero. El valor inicial del *TX_ID* podrá ser un valor aleatorio mayor que 0 e irá incrementándose con cada nueva petición (en una unidad o más de manera cíclica) y será diferente en cada sentido de la comunicación. Un mensaje con un identificador menor o igual al último recibido se considerará duplicado y será descartado (si es el primer identificador recibido, siempre se acepta).

Tabla 3.3 Interacciones entre entidades de la arquitectura SNSR.

Interacción	Finalidad
Robot $\Longleftrightarrow$ BS	• Intercambio de información de la topología de la red. La BS enviará al robot la información conocida inicialmente de la red y el robot enviará a la BS información recopilada tras la detección de nodos. Ambos pueden transmitir al otro la topología calculada. • El robot informará a la BS de anomalías detectadas por él o por nodos sensores. • La BS podrá informar al robot sobre cambios en la CRL y comandos administrativos. • El robot podrá preguntar sobre el estado de la BS y detectarlo como un elemento estático más de la red.
BS $\Longleftrightarrow$ Nodo	• Como funcionamiento básico de una red de sensores, el nodo enviará a la BS las medidas recogidas. • La BS podrá enviar peticiones de estado, de cambio de configuración del nodo y actualizaciones de la CRL. • Los nodos comunicarán a la BS las anomalías que detecten.
Nodo $\Longleftrightarrow$ Nodo	• En la arquitectura SNSR, los nodos solo se comunican entre sí para reenviar paquetes de manera colaborativa hacia su destino.
Nodo $\Longleftrightarrow$ Robot	• El robot puede consultar el estado de los nodos y autenticarlos durante el proceso de descubrimiento y posteriormente pedirles que cambien su configuración o actualicen su CRL. • Los nodos comunicarán al robot las anomalías detectadas si tienen conexión directa con él.

3.3.4 Mantenimiento de la conexión

Si se quiere detectar la conectividad con el otro extremo de una conexión en el LL o TL en periodos de tiempo donde no haya que intercambiar datos se recurrirá al procedimiento habitual de envío de tramas o segmentos periódicos de pulso o *keepalive*. Esto permite que la conexión esté preparada cuando sea necesaria e identificar anomalías en la red. Es posible que este procedimiento u otro similar ya lo proporcione el LL o TL utilizados, ya que es muy habitual.

El mantenimiento de la conexión será opcional en la mayoría de los casos y dependerá de las políticas y funcionamiento esperado de la red de sensores. En las conexiones de

Tabla 3.4 Tipos de mensaje de la aplicación de gestión.

Tipo de mensaje	Sentido	Descripción	Más detalles
TM_STATUS_REQ	robot→BS robot→nodo BS→nodo	Petición de estado	Sección 3.6.2
TM_STATUS_RES	BS→robot nodo→robot nodo→BS	Estado de la entidad	Sección 3.6.2
TM_CONFIGURE_REQ	BS→robot BS→nodo robot→nodo	Petición de cambio de configuración o CRL	Sección 3.8
TM_CONFIGURE_RES	robot→BS nodo→BS nodo→robot	Respuesta al cambio de configuración	Sección 3.8
TM_ANOMALY_STATUS	robot→BS nodo→robot nodo→BS	Notificación de anomalías detectadas	Sección 3.9.1
TM_ANOMALY_ACK	BS→robot robot→nodo BS→nodo	Confirmación de recepción de las anomalías	Sección 3.9.1
TM_GET_ND_REQ	robot→BS	Obtención de información topológica	Sección 3.5.2
TM_GET_ND_RES	BS→robot	Información topológica	Sección 3.5.2
TM_SET_ND_REQ	robot→BS	Petición de modificación de información topológica	Sección 3.5.1
TM_SET_ND_RES	BS→robot	Confirmación de la modificación de la información topológica	Sección 3.5.1
TM_SET_CRL_BCAST	BS→todos (por inundación)	Actualización urgente de la CRL	Sección 3.9.2
TM_MGMT_CMD_REQ	BS→robot	Ejecución de comandos administrativos	Sección 3.5.3
TM_MGMT_CMD_RES	robot→BS	Resultado de la ejecución de comandos administrativos	Sección 3.5.3

TL con entidades vecinas, será innecesario si ya se utiliza en el LL. En las conexiones de LL con el robot, se impone su obligatoriedad por las razones que se discutirán en la Sección 3.6.

En el caso de tener que implementar este procedimiento, se realizaría del siguiente modo:

- Cada extremo de la conexión almacenaría el instante de tiempo de transmisión de la última PDU, TT_{last}.
- Si transcurre un intervalo de tiempo establecido KAI desde TT_{last} sin enviar nada, se enviaría una PDU de *keepalive*. Esta contendrá un único octeto con todos los bits a 0, como se muestra en (3.4). KAI es un valor individual de cada conexión, aunque la entidad tendrá un valor por defecto KAI_g.

$$A \rightarrow B : 0 \tag{3.4}$$

- Cada extremo también anotará el instante de tiempo de recepción de la última PDU, TR_{last}. Se define un tiempo máximo que puede pasar sin recibir ninguna PDU, $KATO$. Este valor será un múltiplo entero de KAI como se muestra en (3.5). El factor multiplicador será un valor predeterminado común en todas las entidades.

$$KATO = KAIF \cdot KAI \tag{3.5}$$

- Si no se recibe ninguna PDU en un intervalo igual o mayor a $KATO$ desde TR_{last}, se considerará que no hay conexión con el otro extremo. Esto se registra como una anomalía, se cierra la conexión y se informa a los niveles superiores y el plano de gestión para que estos actúen en consecuencia.

El robot anunciará el valor de KAI que debe aplicar cada entidad que quiera conectarse a él en la trama de baliza. El resto de conexiones tendrán fijado el valor de KAI a utilizar por configuración o utilizarán el KAI_g si no se indica ninguno explícitamente.

3.3.5 Fases de funcionamiento de la red

Con esta arquitectura la WSN pasará a lo largo del tiempo por 5 fases o etapas. Estas fases aparecen representadas en la Figura 3.13 con las transiciones entre ellas.

Durante la fase ***inicial***, se decidirán los parámetros de configuración estática y de seguridad que irán preconfigurados en todas las entidades, se generarán e instalarán los certificados de cada entidad y de la CA y se desplegarán los nodos sensores en las instalaciones a monitorizar. Si se conoce la lista de nodos sensores autorizados, el robot la cargará localmente o pidiéndola a la BS. Al robot también hay que darle las coordenadas de la zona donde se localizan los nodos sensores, el alcance radio máximo de estos y cualquier otra información que esté disponible y ayude en la localización de los nodos sensores posteriormente (fuentes de interferencia o aislamiento electromagnético, posiciones aproximadas, etc.). Al terminar todas estas tareas iniciales, la red pasará a la fase de *descubrimiento*.

La creación de la red de sensores y su puesta en funcionamiento se producirá en las fases de color naranja: *descubrimiento*, *establecimiento de la topología* y *configuración*. Estas fases se pueden repetir iterativamente hasta que la red de sensores sea completamente funcional y para corregir problemas y ataques que se hayan identificado en la fase de *mantenimiento*.

En la fase de ***descubrimiento*** de nodos sensores, el robot calculará una trayectoria de tal manera que pueda establecer contacto con todos los nodos sensores desplegados en la

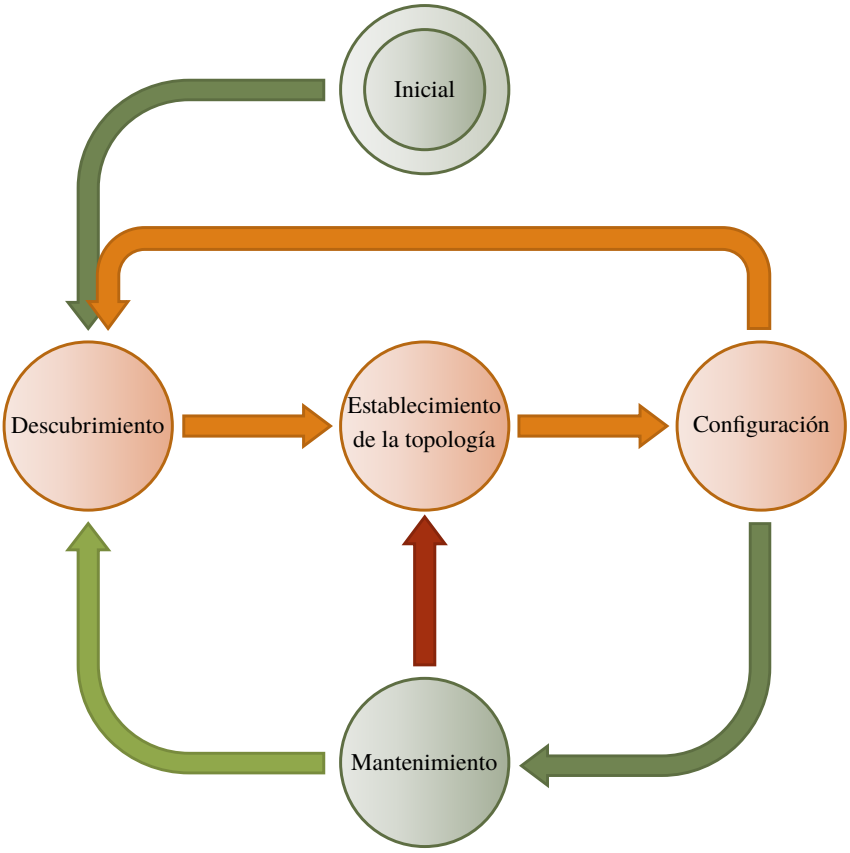


Figura 3.13 Fases de funcionamiento de la red en la arquitectura SNSR.

instalación. Esta trayectoria será más o menos eficiente dependiendo del conocimiento previo que se tenga. Si esta fase se realiza más de una vez, la segunda vez y siguientes la trayectoria se puede optimizar dado que anteriormente ya se localizaron los nodos. El robot comenzará a moverse por la instalación detectando y autenticando todos los nodos. La velocidad debe ajustarse para que los nodos tengan tiempo de conectarse al robot y este pueda recopilar sus estados. En zonas con muchas interferencias, el robot deberá desplazarse más despacio. La principal diferencia con respecto a los esquemas de WSN tradicionales es que, en esta arquitectura, los nodos de sensores no descubren otros nodos. La primera transmisión de cada nodo sensor será para comunicarse con el robot. Esta fase terminará automáticamente cuando se detecten todos los nodos autorizados o se termine de escanear la zona establecida. Con los datos recopilados se pasará a la fase de *establecimiento de la topología*.

El robot o la BS calculará la topología de la red en la fase de ***establecimiento de la topología*** y generará la información, como se verá en la Sección 3.4, que servirá para configurar todos los nodos sensores. Dependiendo del tamaño de la red y el número de factores a optimizar este proceso puede ser más o menos costoso computacionalmente. La

información recopilada en la fase anterior también permitirá calcular trayectorias mejores para visitar a todos los nodos. Una vez hechos los cálculos, la información de la topología se sincronizará entre la BS y el robot, y cuando estos estén de acuerdo se pasará a la siguiente fase.

En la fase de ***configuración***, el robot enviará la configuración a cada nodo. Para ello, volverá a moverse por la instalación de tal manera que tenga comunicación directa con cada nodo y así transmitir la configuración sin intermediarios. Los nodos, cuando apliquen la configuración, empezarán su modo de funcionamiento normal, donde se intentarán conectar con la BS para enviar sus medidas e informar de cualquier anomalía detectada en la red. Hasta este momento el funcionamiento del nodo había sido el de conectarse al robot y enviar su estado por lo que su consumo será muy reducido. Esta fase terminará cuando todos los nodos previamente descubiertos sean configurados.

Es probable que la primera topología calculada no sea totalmente correcta, sobre todo en casos en los que se asignen vecinos a nodos que realmente no sean alcanzables por distintos motivos: el alcance radio máximo puede ser optimista, interferencias no contempladas, etc. Es posible que, aunque con defectos, todos los nodos puedan comunicarse con la BS y en este caso, gracias a los informes de anomalías enviados por los nodos, se podría calcular una nueva topología corregida. Pero otras veces pueden quedarse nodos aislados que serán reconfigurados a través del robot. Además, la comunicación con el robot es directa y no se ve afectada por problemas que puedan producirse durante el transitorio conforme se van configurando los distintos nodos. Así, para tener mayores garantías, se puede volver a hacer una iteración por las tres fases de color naranja. La fase de *descubrimiento* serviría para validar la configuración enviada, comprobando el estado en el que se encuentra cada nodo (que incluye los vecinos que son alcanzables). La fase de *establecimiento de la topología* corregiría los errores detectados y por último se volvería a enviar la configuración modificada. Una vez que la configuración se considere correcta, se pasaría a la fase de *mantenimiento*.

La fase de ***mantenimiento*** sería la más estable y es donde la red permanecería más tiempo según los supuestos de esta Tesis Doctoral. Se analizarán las anomalías detectadas y el estado de los nodos. Si se produjera algún cambio en la topología de la red habría que volver a reconfigurar la red. En el caso de eliminación de un nodo o actualización de la CRL se pasaría a la fase de *establecimiento de la topología*. Si un nodo es añadido a la red o cambia su posición, se iniciaría de nuevo la fase de *descubrimiento*.

El funcionamiento de los nodos sensores en esta arquitectura es muy sencillo. Para un nodo solo habría dos modos de funcionamiento: con o sin configuración recibida.

En las siguientes secciones se detallará el funcionamiento de cada entidad en las diferentes fases en las que se va a poder encontrar la red de sensores, los mensajes transmitidos y las secuencias de mensajes intercambiados.

3.4 Información de topología de la red

El robot conforme explora la WSN debe ir recopilando información sobre ella. Esta información se basará en la identificación de los nodos y en el estado en el que se encuentren

estos. Aquellos nodos que no estén autorizados a formar parte de la red serán ignorados. Si se usa la autenticación con certificados digitales se verificarán que estos no estén en la CRL. El robot intercambiará la información obtenida de la red con la BS. Ambos mantendrán un mapa virtual de la WSN a partir de esa información.

La información topológica de la red contiene todos los parámetros de configuración (vecinos, tablas de encaminamiento,...) que deben tener los nodos y la BS para un correcto funcionamiento. Esto se almacena en una base de datos de topología de red y simultáneamente pueden existir distintas instancias o versiones que contengan: a) restricciones iniciales, b) datos obtenidos durante la fase de *descubrimiento*; o c) configuración calculada en distintas ejecuciones de la fase de *establecimiento de la topología*. En a) y b) la información estará incompleta.

La Figura 3.14 es un diagrama de Entidad-Relación en Lenguaje Unificado de Modelado (UML, *Unified Modeling Language*) [109] donde se muestran las relaciones entre los distintos tipos de información almacenada sobre la topología de la red. No se especifica el formato ni dónde se almacenará físicamente esta información. Aunque en el diagrama todas las relaciones son de agregación, donde un elemento contiene a los otros, también sería posible utilizar relaciones de asociación, como las relaciones entre distintas tablas de una base de datos relacional. A continuación, se describen cada una de ellas.

`NetworkData` Es el elemento raíz que contiene al resto de elementos: una CRL, una lista de nodos y la versión de la lista. Aquí el término «nodo» se refiere a nodo de la red que incluye a la BS. La topología puede ir cambiando con el tiempo por distintos motivos: incorporación de nuevos nodos, eliminación de nodos, cambio de rutas o de vecinos, etc. El número de versión de la topología actual será un parámetro que se configurará en todas las entidades de la red. La comparación del número de versión permite determinar si una entidad está actualizada o no, y, si no lo está, saber qué cambios necesita realizar para actualizarse. De una versión a otra debe producirse un cambio en la lista de nodos. No hay límite máximo al número total de versiones que almacenará el robot y la BS, pero si se establece un mínimo de 2 versiones:

1. Versión 0 (inicial), que debe entenderse como la lista de nodos permitidos en la red con los datos necesarios para su verificación (identificación de las entidades y CRL inicial), pero que normalmente no contendrá información sobre vecinos, rutas ni posiciones. Si la lista de nodos está vacía, cualquier nodo que descubra el robot con un certificado válido estará autorizado a pertenecer a la red.
2. Última versión (actual) calculada. Al principio coincidirá con la versión inicial, pero posteriormente irá actualizándose. Las nuevas versiones se calculan cada vez que se entra en la fase de *establecimiento de la topología*.

Durante el cálculo de una nueva versión, será necesario almacenar una versión temporal extra.

`CRLData` y `SerialData` Es la CRL utilizada en la red propuesta. Contiene los números de serie (`SerialData`) de los certificados revocados. Los números de serie son valores enteros positivos únicos asignados a los certificados por la CA con un tamaño

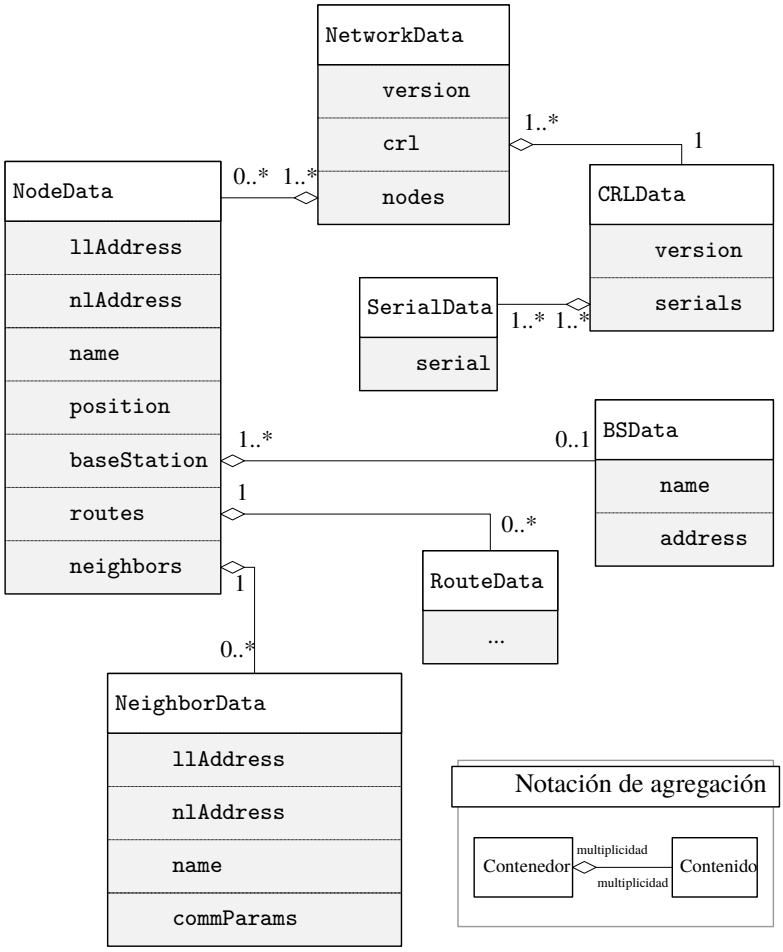


Figura 3.14 Información de la topología de la red.

variable no superior a los 20 octetos [14]. Una CRL se identifica por su número de versión, que es independiente del número de versión del elemento `NetworkData`: una misma CRL puede ser utilizada por distintas versiones de la topología y la CRL puede ser actualizada sin necesidad de cambiar la versión de la topología. Las CRL son incrementales: si la versión de una CRL X es mayor que la de otra Y, contendrá todos los números de serie de la versión Y además de otros más.

`NodeData` Contiene información de un nodo sensor o de la BS con la configuración que debe tener en la topología actual. La información consiste en:

- la dirección física, que no debería cambiar durante la vida de la entidad,

- la dirección de red, que puede ser diferente según la topología calculada,
- la identidad (nombre), que debe aparecer en su certificado,
- y la posición espacial detectada por el robot o preconfigurada que permitirá estimar su rango de alcance radio.

Y la configuración para esta entidad contiene:

- la BS a la que debe transmitir datos en el caso de que sea un nodo sensor,
- información de encaminamiento,
- e información necesaria para conectarse a sus vecinos autorizados.

`BSData` Contiene la información de la BS a la que debe transmitir datos un nodo sensor. Si solo hay una BS, esta será la misma para todos los nodos sensores. Incluye el identificador de la BS (nombre), utilizado para validar el certificado de esta, y la dirección de red.

`RouteData` Información de encaminamiento utilizada en el NL para determinar qué debe hacer con los paquetes a enviar/reenviar. Esta arquitectura es agnóstica al NL usado, por lo que la información será dependiente de ese NL y no se define aquí (debe particularizarse según la implementación concreta).

`NeighborData` Es la información sobre el vecino, necesaria para establecer una conexión con él y reenviar paquetes destinados a él. Está compuesta por:

- la dirección física,
- la dirección de red,
- la identidad (nombre), que permite validar su certificado,
- y parámetros adiciones para establecer la comunicación que dependerán del nivel de enlace, tipo de saludo usado en la conexión, mantenimiento de la conexión, política de reintentos, etc. Deberán ser particularizados por la implementación concreta.

3.5 Intercambio de información entre el robot y la estación base

El intercambio de información entre el robot y la BS se hace normalmente a petición del robot, a excepción de actualizaciones urgentes de la CRL. La BS debe centralizar el almacenamiento de información de la red, por lo que los datos recopilados y generados por el robot serán finalmente transferidos a la BS. El robot es un elemento que debería poder ser sustituido en cualquier momento. El robot sustituto puede utilizar la información almacenada en la BS como punto de partida, aunque también podría recibir los datos iniciales por configuración o utilizar otros medios administrativos que permitan modificar los datos que contiene el robot.

Se distinguen los siguientes tipos de información intercambiada:

1. Información de topología de red descrita en la Sección 3.4. Puede ser la información inicial con las entidades permitidas en la red y posiciones aproximadas, la información

recopilada durante el *descubrimiento* o la información de red calculada para el funcionamiento adecuado de la red.

2. Información de monitorización de red: estadísticas y anomalías detectadas. En la Sección 3.9 se detallará más este tipo de información y los procedimientos asociados.
3. Información de estado de la BS, útil para la construcción de la topología de la red.
4. Actualizaciones de la CRL.

Cada vez que hay cambios en la información de la topología, la BS modificará su configuración de manera acorde: estableciendo conexiones con sus nuevos vecinos y permitiendo la comunicación solo con los nodos autorizados y con las direcciones de red asignadas.

En la Figura 3.15 se muestran los procesos asociados al intercambio de información de topología de red. Estos procesos son independientes y en la figura se muestran separados y no se deben interpretar como consecutivos. En las siguientes secciones se describen cada uno de ellos.

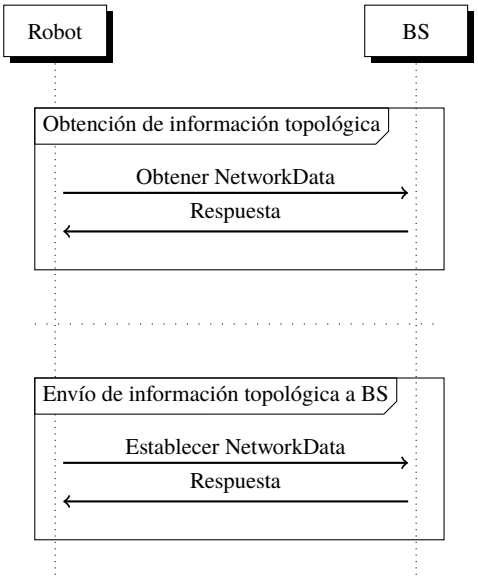


Figura 3.15 Intercambio de información de topología.

3.5.1 Envío de información topológica a la estación base

La secuencia de mensajes transmitidos durante el proceso de envío de información topológica a la BS aparece representada en la parte inferior de la Figura 3.15. Como se puede observar consiste en un intercambio de mensajes: el robot envía una petición de tipo `TM_SET_ND_REQ` y la BS responde con una respuesta del tipo `TM_SET_ND_RES`. El contenido del mensaje de petición de establecimiento de la información topológica se muestra en (3.6), y el de la respuesta en (3.7).

$$R \rightarrow BS : TM\_SET\_ND\_REQ \parallel TX\_ID \parallel ND \quad (3.6)$$
$$BS \rightarrow R : TM\_SET\_ND\_RES \parallel TX\_ID \parallel ACCEPTED \quad (3.7)$$

Estos mensajes serán transmitidos usando el servicio de envío fiable del TL. Los dos primeros campos de cada mensaje son el tipo de mensaje de aplicación y el identificador de transacción, como se explicó en la Sección 3.3.3. El resto de campos aparecen descritos en la Tabla 3.5.

Tabla 3.5 Campos específicos de los mensajes de establecimiento de información topológica.

Campo	Descripción
ND	Son los datos de información topológica serializados. El formato de la serialización no se define aquí y es dependiente de la implementación.
$ACCEPTED$	Es un valor booleano que indica si la BS ha aceptado la nueva versión. Los motivos por lo que los datos no serían aceptados son: • Los datos son inconsistentes o se ha producido un error en la decodificación. • La versión de `NetworkData` o `CRLData` de la información enviada no es mayor que la última disponible. • La BS va a realizar los cálculos y/o los datos enviados por el robot solo contiene información obtenida durante el descubrimiento.

Este proceso se iniciará después que el robot termine la fase de *descubrimiento* y, opcionalmente, calcule una nueva topología. El robot debe enviar la información topológica a la BS para que esta la acepte. Si la BS la acepta, el robot podría pasar a la fase de configuración de la red. Si no la acepta, el robot deberá preguntar cuál es la información topológica que debe utilizar a la BS utilizando el procedimiento explicado en la siguiente sección.

3.5.2 Obtención de información topológica de la estación base

La secuencia de mensajes transmitidos durante el proceso de obtención de información topológica de la BS aparece representada en la parte superior de la Figura 3.15. Como se puede observar, consiste en un intercambio de mensajes: el robot envía una petición de tipo `TM_GET_ND_REQ` y la BS responde con una respuesta del tipo `TM_GET_ND_RES`. El contenido del mensaje de petición de obtención de información topológica se muestra en (3.8), y el de la respuesta en (3.9).

$$R \rightarrow BS : TM\_GET\_ND\_REQ \parallel TX\_ID \parallel INITIAL \tag{3.8}$$
$$BS \rightarrow R : TM\_GET\_ND\_RES \parallel TX\_ID \parallel ND \tag{3.9}$$

Estos mensajes serán transmitidos usando el servicio de envío fiable del TL. Los dos primeros campos de cada mensaje son el tipo de mensaje de aplicación y el identificador de transacción, como se explicó en la Sección 3.3.3. El resto de campos aparecen descritos en la Tabla 3.6.

Tabla 3.6 Campos específicos de los mensajes de obtención de información topológica.

Campo	Descripción
INITIAL	Es un valor booleano que indica si se desea obtener la versión 0 (inicial) de la información topológica o la última versión calculada.
ND	Son los datos de información topológica serializados. El formato será similar al utilizado en el mensaje `TM_SET_ND_REQ`.

Este proceso se iniciará en los siguientes casos:

- Cuando el robot arranca y aún no está en la fase de descubrimiento, si no dispone de la versión inicial de información topológica y se conecta con la BS. El robot puede obtener esta versión inicial por configuración u otros medios también.
- Si el robot envía la información topológica a la BS y es rechazada, por los motivos descritos en la sección anterior, el robot deberá actualizar su información topológica utilizando la que tiene la BS. Esta nueva versión será utilizada en la fase de configuración.
- El robot desea asegurarse que la CRL que tiene está actualizada. La BS contendrá siempre la CRL más reciente.

3.5.3 Envío de comandos administrativos al robot

El robot puede recibir órdenes de la BS, por ejemplo para iniciar la fase de *descubrimiento*. En SNSR se define únicamente la forma de enviar comandos administrativos, sin tener en cuenta el contenido. Los comandos que puede recibir el robot se deja a criterio de la implementación, como también la posibilidad de que el robot pueda recibir órdenes de otras entidades distintas y por otros medios diferentes.

El intercambio de mensajes es el siguiente: la BS envía una petición de tipo `TM_MGMT_CMD_REQ` y el robot responde con una respuesta del tipo `TM_MGMT_CMD_RES`. El contenido de los mensajes se muestran en (3.10) y (3.11).

$$BS \rightarrow R : TM\_MGMT\_CMD\_REQ \parallel TX\_ID \parallel ... \quad (3.10)$$

$$R \rightarrow BS : TM\_MGMT\_CMD\_RES \parallel TX\_ID \parallel ... \quad (3.11)$$

Estos mensajes serán transmitidos usando el servicio de envío fiable del TL. Los dos primeros campos de cada mensaje son el tipo de mensaje de aplicación y el identificador de transacción, como se explicó en la Sección 3.3.3. El resto de campos no se muestran por ser dependientes de la implementación.

3.6 Descubrimiento de nodos sensores

El robot se mueve en el escenario transmitiendo periódicamente una trama baliza que contiene información del robot (nombre, número de serie de su certificado, canal de comunicación compatible). Si el nodo sensor que recibe esa trama todavía no se ha conectado al robot, transmite una trama *Hola* que activará el proceso de saludo. Si se utiliza seguridad en el LL, el robot y el nodo se autentican entre sí, establecen claves de cifrado y MAC. Para finalizar, el robot pregunta al nodo sensor sobre su estado (versión de configuración, versión de la CRL, entre otros) y registra toda la información recibida. El robot puede configurarse para admitir solo conexiones de nodos con un certificado válido no revocado.

Las conexiones con el robot siempre se mantendrán activas por el procedimiento descrito en la Sección 3.3.4. Esto puede servir para determinar el rango de alcance del nodo. El robot siempre preguntará por el estado del nodo sensor cada vez que se inicie una conexión, ya que si se pierde la conexión no puede asegurar que el nodo no haya sido reiniciado. Mientras que la conexión está abierta con el robot, el nodo también puede enviarle anomalías.

La Figura 3.16 muestra un diagrama con los mensajes intercambiados asumiendo que todos llegan sin error. Los mensajes del LL son de color azul y los del AL están en negro. No se muestra el procedimiento de saludo completo, ya que depende del modo de seguridad y la implementación utilizada.

3.6.1 Balizas

Las balizas permiten al robot anunciar su presencia a las entidades de la red que lo rodean. Las entidades que reciben una trama de baliza supondrán que es posible establecer comunicación con el robot en ese momento. Los nodos y la BS que ya hayan creado una conexión con el robot ignorarán las siguientes balizas recibidas.

El formato de las balizas difundidas por el robot es el siguiente:

$$R \rightarrow * : 0 \parallel CN \parallel SN \parallel NA \parallel CP \quad (3.12)$$

Los campos que forman parte de una baliza están descritos en la Tabla 3.7.

Las balizas no van cifradas ni incluyen ningún campo que permita comprobar la autenticidad de las mismas. Sin embargo, esto no afecta significativamente a la seguridad de la arquitectura SNSR. El cifrado implicaría distribuir previamente a todas las entidades

Figura 3.16 Intercambio de mensajes durante la etapa de descubrimiento.

una clave, que podría fácilmente obtenerse capturando uno de los nodos. Además la información que viaja en una baliza no es crítica y se puede obtener por observación de los mensajes transmitidos. La autenticación y el cifrado supone un coste computacional que habría que realizar por cada baliza recibida. Si hubiera que comprobar la autenticidad, se podría lanzar un ataque donde se transmitieran balizas erróneas que obliguen a los nodos a realizar operaciones con alto consumo energético. Las balizas deben ser fáciles de procesar o descartar y la autenticación se realiza, si es necesaria, durante el proceso de saludo.

El robot transmitirá balizas continuamente a intervalos no constantes pero no superiores a un máximo fijado BI_{max}. El objetivo de añadir una componente aleatoria al intervalo entre balizas es evitar que un adversario malicioso pueda deducir cuándo se va a transmitir la siguiente baliza. De este modo no podrá crear interferencias en el momento exacto de transmisión de una baliza. Se define el tiempo que esperará el robot antes de transmitir la siguiente baliza, TE_{baliza}, mediante la ecuación:

$$TE_{baliza} = \frac{BI_{max}}{2} + random\left(0, \frac{BI_{max}}{2}\right) \tag{3.13}$$

La función $random(x, y)$ devuelve un valor aleatorio comprendido en el intervalo $[x, y]$.

Inicialmente, en la red solo transmite el robot, y el resto de entidades se quedan a la espera de recibir una baliza. Por tanto, lo primero que se recibirá en esas entidades será una baliza. Si la red no está siendo atacada y no hay interferencias, no puede producirse ninguna colisión con la primera baliza que emita un robot y debería ser detectada con facilidad por

Tabla 3.7 Campos de una baliza del robot en la arquitectura SNSR.

Campo	Descripción
0	Es un octeto con todos los bits a 0. Esto permite identificar rápidamente una baliza del resto de tramas que no pueden empezar así.
CN	Nombre común del $\mathbf{Cert}_R^{CA}$.
SN	Es el número de serie del $\mathbf{Cert}_R^{CA}$. CN y SN se utilizan para verificar que la baliza procede de un robot válido: el tipo debe ser el de un robot, el dominio de la red debe ser correcto y el número de serie no debe estar en la CRL. Si se usa autenticación en la conexión, la entidad comprobará que el CN del certificado digital del robot es igual al transmitido en la baliza.
NA	(*network address*) Es la dirección de red del robot.
CP	(*communication parameters*) Es un campo con parámetros de comunicación a utilizar durante la conexión. Dependerá de la implementación realizada y de los parámetros que ya estén preconfigurados en los nodos, y que por tanto no sean necesarios volver a indicar. Por ejemplo, puede incluir el valor en segundos del intervalo máximo que debe transcurrir entre mensajes transmitidos para mantener abierta la conexión, KAI.

las entidades que se encuentren en el alcance radio *r*. Una vez que las entidades empiecen a intentar la conexión con el robot, la probabilidad de que se produzcan colisiones con la baliza aumentará.

Conforme el robot se mueve por el escenario los nodos con los que podrá comunicarse irán variando. Muchas de las conexiones que el robot había establecido se cerrarán por los mecanismos de detección de fallos de comunicación al vencer los temporizadores asociados. Las entidades que pierdan la conexión con el robot volverán a ponerse a la escucha de balizas.

El funcionamiento de un nodo va a ser muy sensible a la recepción de la primera baliza recibida, que le permitirá conectarse con el robot y recibir la configuración inicial de él. Un nodo que no reciba nunca una baliza no podrá formar parte de la red. No se comunicará con ninguna otra entidad. Si un robot malicioso se pusiera en contacto primero con el nodo, podría evitar que el nodo se conectara a robots legítimos. Para evitar los problemas descritos se disponen las siguientes medidas:

1. Las balizas deberán ser validadas antes de dar lugar a un intento de conexión con el robot emisor.

2. El robot, según el modo de seguridad elegido, será autenticado al establecer la conexión y se verificará que la baliza no contenía datos falsos.

3. Un nodo deberá recibir y validar correctamente una baliza legítima en un tiempo finito si el robot se encuentra dentro de su alcance radio, a pesar de que haya otros

agentes maliciosos transmitiendo balizas falsas (siempre que las balizas falsas no saturen el canal).

La BS es más inmune ante ataques con balizas falsas. La BS puede conocer cuál es el robot operativo en cada momento, por lo que puede distinguir cuál es la baliza legítima. Si en el ataque se envían duplicados de la baliza legítima sin estar próximo el robot, el efecto también sería mínimo (hacer que la BS transmita una trama *Hola*).

Se han desarrollado dos métodos de detección de balizas: uno básico y otro avanzado. Este último ofrece mayor protección ante ataques en los que un atacante se haga pasar por un robot. A continuación se describe cada método y posteriormente se mostrará una tabla comparativa. Se podrá elegir el método más apropiado para cada red por configuración.

En Figura 3.17 se muestra el diagrama de estados utilizado en la detección de balizas y conexión con el robot que tendría lugar en los nodos o BS para un funcionamiento básico. Inicialmente, la entidad entra en el estado de espera de nueva baliza. Cada vez que se recibe una baliza, esta es verificada: pertenece a un robot de un dominio permitido, el número de serie del certificado no está revocado y los parámetros de comunicación son adecuados. Mientras que no se reciba una baliza correcta, la entidad continuará esperando. Si la baliza es correcta, se esperará un tiempo de contención (*back-off*) aleatorio, entre 0 y un tiempo máximo configurado (TC_{max}), antes de comenzar el proceso de saludo. Esta espera de contención, que es opcional si $TC_{max} \equiv 0$, tiene dos objetivos:

1. Evitar colisiones. Es posible que muchos nodos reciban simultáneamente una misma baliza, por lo que intentarán iniciar el saludo a la vez. Los mecanismos de prevención de colisiones del nivel de enlace pueden no ser suficientes en casos extremos, por lo que esta espera aleatoria puede mitigar los efectos.
2. Evitar que en el robot se reciban un número excesivo de respuestas a la baliza que puedan saturar sus recursos (fenómeno conocido como *implosión*).

Transcurrido el temporizador de espera de contención, se iniciaría el saludo. Si no tiene éxito, se volvería a esperar una nueva baliza. En caso contrario, la conexión a nivel de enlace con el robot se establecería. Cuando la conexión se pierda, se volverá al estado `Espera_Baliza`.

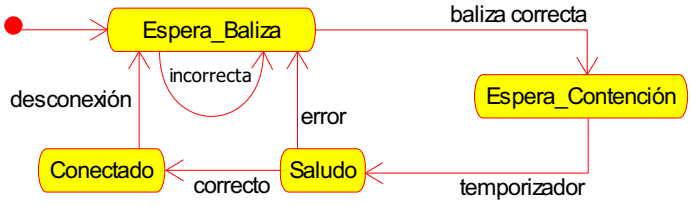


Figura 3.17 Diagrama de estados de detección de baliza básica.

Hay que notar que cualquier baliza recibida fuera del estado `Espera_Baliza` será descartada automáticamente. Esto reduce el consumo en los nodos porque no hay que verificar las balizas, pero este funcionamiento básico presenta una serie de dificultades, sobre todo en el caso donde un atacante también transmita balizas:

1. Espera entre reintentos: es necesario recibir una nueva baliza cuando falla el saludo, aunque posiblemente se hayan recibido nuevas balizas recientemente. Esta espera estará determinada por el intervalo entre transmisión de balizas BI_{max}.

2. Ocultación de balizas legítimas: si un atacante consigue que un nodo detecte su baliza en primer lugar, podría conseguir con una alta probabilidad que la siguiente fuera también otra suya y repetir este proceso sucesivamente sin necesidad de transmitir balizas a una tasa superior a la del robot legítimo.

   Aunque un robot transmita balizas con cierta aleatoriedad, debido a que el nodo ha iniciado el proceso de saludo con el atacante, el atacante conocerá el instante en el que el saludo se dé por terminado, por lo que podrá transmitir una baliza justo después, siendo esta baliza la que se detecte a continuación con mucha probabilidad.

   El atacante también podría controlar la duración del saludo forzando a realizar reintentos, de tal manera que el saludo termine justo después de que el robot haya transmitido una baliza que habrá sido descartada.

3. Ataque de alto consumo energético o computacional (*power attacks*) si se utiliza seguridad en el LL: un atacante provoca que un nodo realice el proceso de saludo constantemente. En el proceso de saludo se realiza la autenticación mutua, que implica la validación de certificados digitales utilizando costosas operaciones criptográficas. Siguiendo el proceso de ataque descrito en el problema anterior, un atacante con certificados incorrectos podría hacer que un nodo, después de detectar su baliza, entrara en el ciclo `Espera_Contención` $\rightarrow$ `Saludo` $\rightarrow$ `Espera_Baliza` $\rightarrow$ `Espera_Contención` ..., reduciendo al mínimo el tiempo que el nodo se encuentra en el estado `Espera_Baliza` y reduciendo el tiempo transcurrido entre saludos.

Aunque en la Sección 3.1.5 se considera que los ataques de DoS son temporales, el comportamiento del sistema sería más robusto si se modifica el diagrama de estados de la Figura 3.17 por el que aparece en la Figura 3.18. La idea principal es no descartar balizas durante los estados `Espera_Contención` y `Saludo` ya que aún no se sabe si la conexión tendrá éxito. Estas balizas no descartadas sirven para mantener un posible candidato en caso de que el saludo falle y así no tener que esperar a la llegada de una nueva baliza. El candidato se elige aleatoriamente entre las balizas recibidas diferentes a la baliza que se está probando en ese instante, utilizando un algoritmo del tipo *reservoir sampling* como los descritos en [156]. Los algoritmos *reservoir sampling* permiten elegir aleatoriamente una muestra de n elementos de un conjunto con un número de elementos desconocidos. En este caso nos permiten elegir una baliza aleatoria entre todas las balizas recibidas hasta un instante determinado (todas con la misma probabilidad de ser elegidas). Esto resuelve los dos primeros problemas del diagrama de estados inicial: no hay espera entre reintentos si hay un candidato disponible y un atacante no puede ocultar balizas legítimas que ahora son utilizadas como posible siguiente reintento.

Por último, el proceso de saludo se deberá modificar para que tenga un tiempo mínimo de ejecución –y así evitar ataques de alto consumo– y un tiempo máximo de ejecución menor del habitual para reducir el tiempo que puede perder un nodo intentando conectarse con un robot falso. La reducción del tiempo máximo se puede conseguir reduciendo el

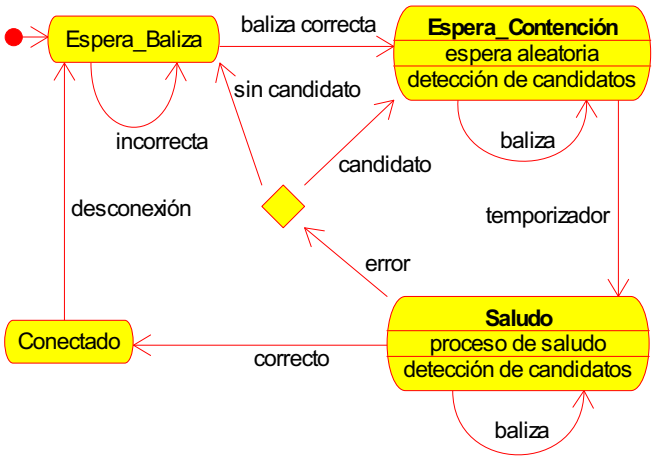


Figura 3.18 Diagrama de estado de detección de baliza avanzada.

número de reintentos permitidos (o incluso no haciendo reintentos) o reduciendo los temporizadores de espera de respuesta.

Se podría pensar que recordar las balizas que han dado lugar a un proceso de saludo fallido, para descartarlas al ser recibidas posteriormente, podría ser una solución ante posibles ataques con balizas falsas (lista negra de balizas). Sin embargo, esto presenta más inconvenientes que ventajas y es una defensa fácilmente eludible:

- Un atacante podría suplantar al robot legítimo antes de que estuviera presente. Más adelante, los nodos descartarían las balizas legítimas, con lo que se impediría la conexión.
- Un atacante podría emitir balizas con datos distintos, de tal manera que la memoria reservada para la lista negra de balizas se llenara. A partir de ese momento, el nodo debería o borrar las entradas más antiguas –que ahora se pueden reutilizar – o no añadir las siguientes balizas erróneas a la lista negra –con lo que se desactivaría esta defensa.

Para terminar, se muestra un cuadro comparativo de ventajas e inconvenientes de cada método en la Tabla 3.8. Por configuración se podrá seleccionar el método elegido utilizando un parámetro llamado `BEACON_DETECTION`.

3.6.2 Petición de estado

La petición de estado permite conocer las versiones de configuración y de CRL de un nodo o la BS, su dirección de red y los vecinos con los que actualmente tiene comunicación. Solo el robot o la BS están autorizados a hacer la petición. La secuencia de mensajes transmitidos aparece representada en la parte inferior de la Figura 3.16. El robot envía

Tabla 3.8 Comparativa entre métodos de detección de baliza.

Detección de baliza	Ventajas	Desventajas
Básica	• Simplicidad. • Menor memoria necesaria.	• Mayor tiempo de espera para reintentar con otra baliza distinta. • Posibilidad de ataques de ocultación de balizas legítimas y ataques de alto consumo energético o computacional.
Avanzada	• Resistente a ataques como los descritos: garantía de conexión, menor tiempo necesario para la conexión y menor consumo energético.	• Más complejo. Uso de más memoria. • Mayor número de reintentos de conexión si se pierden tramas si se limita el tiempo máximo del proceso de saludo.

una petición de tipo `TM_STATUS_REQ` que es respondido con un mensaje del tipo `TM_STATUS_RES`. El contenido del mensaje de petición de estado se muestra en (3.14) y el de la respuesta en (3.15).

$$\left.\begin{array}{r} R \rightarrow BS \\ R \rightarrow nodo \\ BS \rightarrow nodo \end{array}\right\} : TM\_STATUS\_REQ \parallel TX\_ID \tag{3.14}$$

$$\left.\begin{array}{r} BS \rightarrow R \\ nodo \rightarrow R \\ nodo \rightarrow BS \end{array}\right\} : TM\_STATUS\_RES \parallel TX\_ID \parallel CFGV \parallel CRLV \parallel NA \parallel NBL \tag{3.15}$$

Estos mensajes serán transmitidos usando el servicio de envío fiable del TL. Los dos primeros campos de cada mensaje son el tipo de mensaje de aplicación y el identificador de transacción, como se explicó en la Sección 3.3.3. El resto de campos aparecen descritos en la Tabla 3.9.

El mensaje de respuesta se podría extender con más información relevante para una instalación concreta. Por ejemplo, el nivel de batería, estadísticas de uso y de la calidad de las comunicaciones, etc.

La petición de estado se enviará en los siguientes casos:

Tabla 3.9 Campos específicos del mensaje de respuesta de estado.

Campo	Descripción
$CFGV$	Número de versión de la configuración actual o 0 si la entidad no ha sido configurada nunca.
$CRLV$	Número de versión de la CRL instalada o 0 si no hay ninguna.
NA	Dirección de red de la entidad.
NBL	Lista de vecinos autorizados alcanzables. Si la entidad no está configurada estará vacía. Será una lista de elementos `NeighborData` descritos en la Sección 3.4, que a su vez será un subconjunto de la lista de vecinos autorizados que se obtuvo por configuración (los elementos no incluidos son vecinos no alcanzables).

- En las fases de *descubrimiento* y de *configuración*, el robot enviará una petición de estado tras conectarse una entidad a él. Esto se hace incluso aunque la entidad se haya conectado previamente al robot, ya que el robot no puede saber con certeza si ha cambiado algo desde la última conexión.
- En la fase de *mantenimiento*, la BS y el robot podrían hacer peticiones puntuales que permitan identificar la causa de anomalías detectadas. Si la BS emite una nueva CRL por difusión, una petición de estado posterior le permite verificar el estado de los nodos. También se puede enviar periódicamente para monitorizar el estado de los nodos.

3.7 Establecimiento de la topología

En la fase de *establecimiento de la topología*, se analizarán los datos recopilados durante la fase de descubrimiento, los datos aportados administrativamente u otros medios y se verificarán que son coherentes. Por ejemplo, la detección de nodos con el mismo nombre o certificado indicaría que se ha clonado un nodo así que serían descartados, ya que su material de seguridad habría sido comprometido. Administrativamente también se puede limitar la pertenencia a la red a una serie de nodos aunque otros también tengan un certificado válido.

Una vez que se dispone de un mapa con los nodos válidos, se calcula la topología de la red. Los procedimientos para determinar los vecinos de cada entidad o para calcular las tablas de enrutamiento no están predeterminados. La topología adecuada para una red de sensores dependerá del uso que se le vaya a dar y las características de sus entidades. Por ejemplo, si se quiere optimizar la conectividad y la robustez de la red, se intentará conectar cada nodo con todos los nodos que estén a su alcance radio y la tabla de encaminamientos tendrá rutas redundantes al resto de destinos. En los casos donde la energía disponible en cada nodo sea limitada, se puede recopilar información sobre el nivel de energía de cada nodo e intentar que aquellos con menos energía no se utilicen como nodos intermedios. También se puede monitorizar el estado de los enlaces con los nodos vecinos (calidad,

tiempo de respuesta,...) y que cada nodo solo se conecte a sus mejores vecinos. En definitiva, el cálculo de la información topológica buscará maximizar unos objetivos o reducir el coste según unos parámetros dependientes de la aplicación.

Después de realizar los cálculos, si hay cambios, se obtendrá una nueva versión de la información de la topología como la descrita en la Sección 3.4. Esta debe ser sincronizada entre el robot y la BS utilizando los mensajes descritos en la Sección 3.5. Si la BS no estuviera al alcance del robot, este tendría que moverse hacia ella para poder hacer la sincronización.

Durante esta fase las conexiones se mantendrán activas para poder ser utilizadas inmediatamente tras el cambio a la fase de *configuración*.

3.8 Configuración de los nodos

Para configurar los nodos, el robot o la BS enviarán un mensaje de petición de configuración a los nodos. Cuando un nodo recibe el mensaje de petición de configuración, realiza los cambios y confirma el éxito del cambio de configuración devolviendo un mensaje de respuesta. La BS solo podrá realizar este proceso si ya tiene conexión con el nodo a configurar, no hay cambios en las direcciones de red y el enrutamiento sigue siendo válido. En otro caso, el cambio podría afectar a las comunicaciones con el resto de nodos y sería preferible utilizar el robot. El robot, al comunicarse directamente con los nodos, no se ve afectado por fallos de enrutamiento y se puede usar siempre. La primera configuración siempre será realizada por el robot.

En la fase de *configuración* el robot debe volver a moverse por la instalación para poder configurar directamente cada uno de los nodos sensores. La trayectoria seguida y la velocidad debe garantizar lo anterior basándose en la posición de los nodos que se debe conocer al finalizar la fase de *descubrimiento*. La trayectoria por tanto puede cambiar respecto a la realizada anteriormente y optimizarse en función de distintos criterios:

- Distancia más corta y por tanto menor tiempo en recorrerla.
- Con la información topológica se puede analizar la dependencia de unos nodos respecto a otros para la comunicación con la BS. Este análisis se puede emplear para priorizar el envío de la configuración a los nodos que van a soportar más tráfico. En general los nodos más alejados de la BS dependerán de los más cercanos, por lo que la configuración de estos últimos es más prioritaria. Si en la fase anterior el robot se ha tenido que acercar a la BS para hacer la sincronización de la información topológica, la trayectoria ya empezará cerca de los nodos más cercanos a esta.
- Otros criterios propios de las necesidades de la operativa específica de la WSN pueden priorizar algunos nodos o zonas de la instalación.

Conforme el robot se mueve, la interacción con los sensores será inicialmente similar a la fase de *descubrimiento*. Irá preguntando el estado de los nodos y determinará si necesitan ser configurados o no. En caso afirmativo, enviará la configuración apropiada a cada nodo. El intercambio de mensajes se muestra en la Figura 3.19. En el caso de que la BS se encargara de la configuración el funcionamiento sería similar, salvo que el nodo ya se encontraría conectado a la BS.

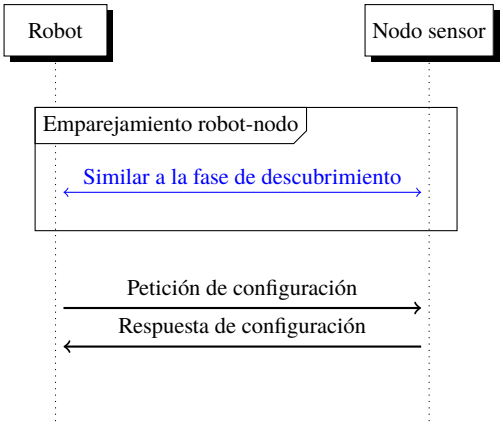


Figura 3.19 Intercambio de mensajes durante la etapa de configuración.

La configuración enviada al nodo se obtendrá de la información de la topología de la red explicada en la Sección 3.4 y, dependiendo del estado del nodo, podrá incluir:

1. La CRL actualizada (elemento `CRLData`).
2. Número de versión de la configuración (versión de `NetworkData`) y configuración del nodo a partir del elemento `NodeData` exceptuando los campos `llAddress`, `name` y `position`. Los campos excluidos ya los conoce el nodo o no los necesita. Aquí se incluye la información de la BS, información de encaminamiento y lista de vecinos autorizados con sus parámetros de conexión.

Opcionalmente, en vez de enviar la configuración completa, se podrían enviar solo los cambios que se han producido desde la versión que hay en el nodo. Esto podría reducir los datos que hay que transferir. Por consiguiente, habría 2 modos de transferencia (completa o diferencial) y 3 tipos de información enviada (solo CRL, solo configuración de red o ambas) que darían lugar a 6 tipos posibles de mensajes de configuración.

Como se observa en la Figura 3.19, la configuración se realiza en un intercambio de mensajes: el robot/BS envía una petición de tipo `TM_CONFIGURE_REQ` y el nodo responde con una respuesta del tipo `TM_CONFIGURE_RES`. El contenido del mensaje de petición de configuración se muestra en (3.16) y el de la respuesta en (3.17).

$$\left.\begin{array}{r} R \rightarrow nodo \\ BS \rightarrow nodo \end{array}\right\} : \quad TM\_CONFIGURE\_REQ \parallel TX\_ID \parallel T\_CFG \parallel CRL\_DATA \parallel CFGV \parallel NODE\_DATA \tag{3.16}$$

$$NODE\_DATA \equiv \quad NA \parallel BS\_DATA \parallel NBL \parallel NO\_NBL \parallel ROUTES \parallel NO\_ROUTES$$

$$\left.\begin{array}{l} nodo \rightarrow R \\ nodo \rightarrow BS \end{array}\right\} : TM\_CONFIGURE\_RES \parallel TX\_ID \parallel CFGV \parallel CRLV \qquad (3.17)$$

Estos mensajes serán transmitidos empleando el servicio de envío fiable del TL. Los dos primeros campos de cada mensaje son el tipo de mensaje de aplicación y el identificador de transacción, como se explicó en la Sección 3.3.3. Los campos del mensaje de respuesta son equivalentes a los del mensaje de respuesta de estado, que aparecen descritos en la Tabla 3.9. El resto de campos del mensaje de petición aparecen descritos en la Tabla 3.10 y la Tabla 3.11.

Tabla 3.10 Campos específicos del mensaje de petición configuración.

Campo	Descripción
T_CFG	Tipo de la petición de configuración. Determina si deben aparecer el resto de campos, que son opcionales, y si el modo de transferencia es completa o diferencial.
CRL_DATA	CRL actualizada utilizando el mismo formato que se almacena en la información de la topología.
$CFGV$	Número de versión de la configuración. Equivale a la versión del elemento `NetworkData` de la información topológica a la que pertenece esta configuración.
$NODE\_DATA$	Información tomada del elemento `NodeData` de la información topológica para este nodo con los campos descritos en la Tabla 3.11.

El mensaje de petición se podría extender con más información relevante para una instalación concreta.

Los nodos deben aplicar la configuración enviada, cerrando conexiones con vecinos que no deberían serlo y estableciendo conexiones con nuevos vecinos. Si un vecino se mantiene como tal pero sus datos o los parámetros de comunicación con él cambian, el nodo puede optar por reabrir la conexión o reutilizar la existente modificándola. Si se actualiza la CRL verificará que no haya establecido ninguna comunicación con entidades cuyos certificados hayan sido revocados. La respuesta del nodo incluirá las nuevas versiones de configuración y CRL como confirmación que han sido aplicadas.

Desde que comienza la configuración del primer nodo hasta que se termina de configurar el último puede existir un periodo transitorio de inestabilidad. Un nodo configurado puede que tenga asignados vecinos que aún no lo están, lo que implicará que tendrá que reintentar la conexión. Entre los efectos temporales que se pueden producir en la red destacan:

- Congestión: los nodos empezarán a conectarse a sus vecinos y a transmitir información a la BS. Se realizarán simultáneamente muchos procesos de saludo por lo que la colisión de tramas es más probable. El robot tiene que tener en cuenta que los nodos pueden tardar más en conectarse a él.

Tabla 3.11 Subcampos del campo $NODE\_DATA$.

Subcampo	Descripción
NA	Dirección de red.
BS_DATA	Información de la BS a la que debe transmitir datos con el mismo formato que se utiliza en la información de la topología.
NBL	Lista de los vecinos autorizados y sus parámetros de comunicación. Lista completa o lista que hay que añadir a los existentes (modo diferencial). El formato es el mismo que el utilizado en la información de la topología.
NO_NBL	Lista que incluye el listado de las direcciones de red de los vecinos que hay que eliminar en el caso de modo diferencial.
$ROUTES$	Información de encaminamiento a utilizar (completa) o que hay que añadir (diferencial). El formato es el mismo que el utilizado en la información de la topología.
NO_ROUTES	Información de encaminamiento a eliminar en el caso de modo diferencial. El formato es el mismo que el campo anterior.

- Bucles: la información de enrutamiento suele estar calculada suponiendo que todos los nodos tienen la misma versión de configuración y en el transitorio se mezclan nodos funcionando con distintas versiones, con direccionamientos de red posiblemente incompatibles.

Al final, este transitorio termina y la red se estabiliza. Si en una implementación concreta se quiere evitar y detectar estos posibles problemas temporales, se proponen las siguientes soluciones opcionales que reducen estos efectos:

- Un nodo no reenviará paquetes que tengan su dirección como origen o, si es posible, lo marcará para que esto no se repita indefinidamente.
- Se establecerá un límite de saltos en los paquetes del NL acorde a las dimensiones de la red. Este es un valor que puede calcularse a partir del diámetro de la red.
- Las conexiones con los vecinos no se iniciarán inmediatamente a la recepción de la configuración, sino que se retardarán un tiempo aleatorio.
- Los reintentos también utilizarán retardos aleatorios.
- Las anomalías no se empezarán a enviar hasta haber recibido correctamente el primer mensaje de la BS a menos que el robot o la BS sean vecinos del nodo.

A pesar de este transitorio el robot no debería tener dificultad en comunicarse directamente con los nodos situándose próximo a estos.

Terminada la configuración de los nodos, se esperaría un tiempo prudencial para que los nodos pudieran ponerse en contacto con la BS. Si alguno de los nodos no consigue

comunicarse con la BS en ese tiempo, se repetiría la fase de *descubrimiento*. En otro caso, también se puede volver a repetir la fase de *descubrimiento* para detectar errores y mejorar la configuración, o pasar a la fase de *mantenimiento*.

3.9 Mantenimiento y monitorización de la red

En la fase de *mantenimiento*, el robot no será necesario todo el tiempo y la BS estará alerta para detectar cualquier cambio que requiera cambiar la configuración. Los mensajes de control transmitidos se reducirán significativamente respecto a los mensajes de datos. El robot puede realizar reconocimientos periódicos o bajo demanda localizando y conectándose a los nodos, y recolectando estados y anomalías que posteriormente reenvía a la BS.

Para la detección de cambios o ataques, la BS se vale de los siguientes mecanismos:

1. Recepción de anomalías del resto de entidades de la red y generación propia de anomalías por observación directa.
2. Consulta del estado de los nodos, utilizando mensajes de petición de estado. Obtención de estadísticas.
3. Notificación administrativa de incorporación, borrado o movimiento de nodos.

Las acciones a realizar una vez detectado un cambio o ataque que implique la reconfiguración podrían ser:

1. Pasar a la fase de *descubrimiento*, cuando se añada o mueva un nodo, se pierda el contacto con uno o más nodos, se quiera confirmar/descartar ataques potenciales o simplemente obtener más información de los errores reportados directamente.
2. Pasar a la fase de *establecimiento de la topología* si se elimina uno o más nodos administrativamente, para actualizar los vecinos de cada nodo y su información de enrutamiento.
3. Actualización urgente de la CRL, si se detecta el robo del robot o la copia de sus claves.
4. Avisar a un operador para que el decida. Por ejemplo, un operador podría forzar acciones administrativas, cambios de fases, etc.

Los procesos administrativos y la obtención de estadísticas no se especifican aquí y su definición se deja a la implementación concreta de esta arquitectura.

A continuación se detalla el proceso de detección y notificación de anomalías y la actualización urgente de la CRL.

3.9.1 Anomalías

Cada entidad puede implementar módulos para detectar anomalías concretas, tales como en [16, 105, 114, 163], además de detectar anomalías/fallos en las comunicaciones y en los mecanismos y protocolos de SNSR. Los nodos irán almacenando las anomalías detectadas

y cada cierto tiempo las transmitirán a la BS o al robot. Se enviarán preferiblemente al que esté conectado directamente y así se intentará reducir el uso de otros nodos. El robot almacenará sus anomalías y las recibidas de otros nodos hasta que pueda transferirlas a la BS. De esa manera toda la información se centraliza en BS. La BS tendrá todos los datos para decidir cómo reaccionar ante la anomalía y podrá aplicar con facilidad técnicas basadas en inteligencia artificial, IDS o Sistema de Prevención de Intrusos (IPS, *Intrusion Prevention System*), heurística, etc. para detectar problemas de seguridad y operativos más complejos. SNSR presenta como ventaja a los sistemas IDS existentes el uso del robot para asegurar la recopilación de anomalías detectadas y poder obtener observación directas de posibles amenazas. Los IDS suelen requerir técnicas complejas para inferir los ataques [16] en las WSN. Con SNSR es posible utilizar técnicas más sencillas al reducirse la superficie de ataque posible.

En la Figura 3.20 se muestra el camino escogido para notificar las anomalías A, B, C y D. La anomalía A se envía directamente a la BS al estar el nodo conectado a ella. La anomalía B se envía a la BS utilizando nodos intermedios, ya que el nodo que la genera no tiene conexión directa ni con la BS ni con el robot. La anomalía C se envía al robot debido a que el nodo donde se ha generado está conectado a él. El robot retransmitirá C junto con la anomalía D, generada en el robot, a la BS.

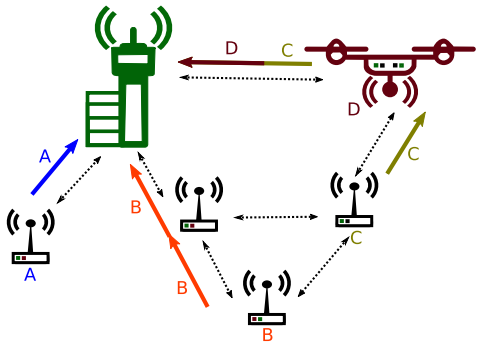


Figura 3.20 Notificación de anomalías.

En los nodos, las anomalías almacenadas están limitadas a un número máximo para evitar ataques que busquen saturar la memoria del nodo a base de generar muchas anomalías. En el caso de que el almacenamiento de anomalías esté lleno, se optará por:

- Agrupar anomalías similares en una sola.
- Eliminar las anomalías menos críticas.
- Borrar anomalías más antiguas.

La transferencia de anomalías a la BS comenzará cuando los nodos reciban la configuración. Para evitar saturar la red, las anomalías se enviarán a intervalos de tiempo predefinidos o cada vez que se produzca una interacción correcta con la BS.

La información que se almacenará de una anomalía genérica estará formada por los campos mostrados en la Tabla 3.12, aunque podrá extenderse por necesidades de la instalación en la que se utilice esta arquitectura.

Tabla 3.12 Campos de la información almacenada de una anomalía genérica.

Campo	Descripción
$OBSERVER$	El observador es la identidad de la entidad que ha generado esta anomalía o vacío si es la propia entidad que la almacena o envía.
ID	El identificador de anomalía es un número secuencial único dentro del observador. La pareja formada por la identidad del observador y el identificador de anomalía debe ser única dentro de la red de sensores.
$TYPE$	El tipo de la anomalía permite clasificar las anomalías por categorías y criticidad. Los tipos predefinidos se recogen en la Tabla 3.13.
$OBJECT$	El objeto de la anomalía es un identificador del objeto que ha provocado la generación de la anomalía. Por ejemplo, si se cae la comunicación con un vecino, sería la identidad del vecino.
$INFO$	El campo información contendrá datos dependientes del tipo de anomalía. Por ejemplo, si se cae una conexión, contendría parámetros de configuración de esa conexión.
TS	El sello de tiempo (*timestamp*) es la fecha y hora del instante en que se produjo la anomalía.
$CFGV$	Es la versión de configuración que estaba instalada en el momento en el que se produjo la anomalía.
$REPEATED$	Este campo es opcional e indica el número de veces que se ha repetido la misma anomalía si se usa la agrupación de anomalías. Si es mayor que 0, el sello de tiempo hace referencia al tiempo de la última ocurrencia.

Los tipos genéricos de anomalías aparecen en la Tabla 3.13. Los dos últimos tipos mostrados son anomalías que detectará exclusivamente el robot. El resto de tipos son fácilmente detectables por todas las entidades de la red, ya que son errores producidos durante la conexión con otras entidades.

La secuencia de mensajes transmitidos durante la notificación de anomalías de un nodo a la BS aparece representada en la Figura 3.21. La notificación de un nodo al robot o del robot a la BS sería similar. La entidad que quiere notificar anomalías enviaría un mensaje de tipo `TM_ANOMALY_STATUS` y el receptor asentiría su recepción con un mensaje de tipo `TM_ANOMALY_ACK`. Las anomalías asentidas pueden ser borradas del almacenamiento local.

El contenido de los mensajes de notificación y asentimiento de anomalías se muestran respectivamente en (3.18) y (3.19).

Tabla 3.13 Tipos genéricos de anomalías.

Tipo	Descripción
`CONN_ERROR_TO_INIT`	Expiró el tiempo de espera máximo para el proceso de saludo sin recibirse datos desde el otro extremo.
`CONN_ERROR_TO_HS`	Expiró el tiempo de espera máximo para el proceso de saludo, aunque se habían recibido datos desde el otro extremo.
`CONN_ERROR_TO_KEEP`	La conexión se cerró porque pasó un tiempo mayor a $KATO$ sin recibir datos o mensajes de *keepalive*.
`CONN_ERROR_CERT`	El certificado es incorrecto o está revocado.
`CONN_ERROR_PEER`	El otro extremo cerró la conexión.
`CONN_ERROR_SHUTDOWN`	La conexión se cerró a petición del plano de gestión.
`CONN_ERROR_HS`	Error durante el proceso de saludo.
`NETWORK_E_BROADCAST`	Error de validación en mensaje de difusión.
`MGMT_PROTOCOL_ERROR`	Error de protocolo de gestión.
`NAME_ERROR_DUP`	Identidad duplicada.
`NODE_DISPLACEMENT`	Desplazamiento del nodo.

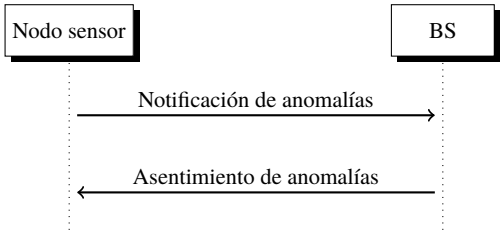


Figura 3.21 Intercambio de mensajes durante la notificación de anomalías.

$$\left.\begin{array}{r} nodo \rightarrow BS \\ nodo \rightarrow R \\ R \rightarrow BS \end{array}\right\} : TM\_ANOMALY\_STATUS \parallel ANOMALY\_LIST \tag{3.18}$$

$$\left.\begin{array}{r} BS \rightarrow nodo \\ R \rightarrow nodo \\ BS \rightarrow R \end{array}\right\} : TM\_ANOMALY\_ACK \parallel LAST\_OBS \parallel LAST\_ID \tag{3.19}$$

Estos mensajes no requieren usar el servicio de envío fiable del TL. Si no llega el asentimiento, no se borrarán las anomalías del almacenamiento local y serán reenviadas posteriormente. El primer campo de cada mensaje es el tipo de mensaje de aplicación, como se explicó en la Sección 3.3.3. El resto de campos aparecen descritos en la Tabla 3.14.

Tabla 3.14 Campos específicos de los mensajes de anomalías.

Campo	Descripción
$ANOMALY\_LIST$	Lista de anomalías en orden cronológico de creación. El formato de serialización es dependiente de la implementación
$LAST\_OBS$	Campo $OBSERVER$ de la última anomalía asentida.
$LAST\_ID$	Campo ID de la última anomalía asentida. Junto con el campo $LAST\_OBS$ permitirá identificar unívocamente la última anomalía asentida. La entidad receptora podrá borrar la última y todas las anteriores.

3.9.2 Actualización urgente de la CRL

Con las suposiciones iniciales, la amenaza más grave de seguridad que podría ocurrir en esta arquitectura es la manipulación o la sustracción del robot. Normalmente, el robot va a estar vigilado durante su tiempo de operación y su desaparición sería rápidamente detectada. Si se supone que el robot no es resistente a la manipulación, los atacantes podrían extraer las claves que almacena y utilizarlas para reconfigurar la WSN según sus intereses. La manipulación de un nodo no es tan crítica ya que sus efectos son solo locales y no puede cambiar el funcionamiento de otras entidades. Para evitar el uso indebido de las claves posiblemente sustraídas, los certificados asociados deben ser revocados actualizando la CRL en todas las entidades.

Una vez revocado el certificado, no se podrá establecer nunca más una conexión con él. Los certificados revocados no pueden ser readmitidos posteriormente. Si se recupera posteriormente la entidad y se verifica que su estado es correcto, la única manera de volver a admitirla en la red es instalando un nuevo certificado.

Para la actualización de la CRL, la BS utilizará un mensaje que transportará la CRL completa o solo con los últimos cambios, pudiendo ser los destinatarios el robot o nodos. Se contemplan dos modos de envío de este mensaje al resto de entidades:

1. Utilizando una conexión punto a punto.
2. Utilizando un paquete de difusión en el NL.

A continuación, se describirá cada modo y se hará una comparativa entre ellos.

3.9.2.1 Actualización mediante conexión punto a punto

En fase de *mantenimiento* la BS debería tener conectividad con todos los nodos sensores de la red. La BS puede enviar un mensaje a cada uno de los nodos (y el robot si está

al alcance) usando una conexión punto a punto en el TL de manera fiable. Se usa el mensaje de petición de configuración que se mostró en (3.16). El campo T_CFG (tipo de configuración) indicará que solo se transmitirá la CRL. El mensaje iría protegido con las mismas medidas de seguridad utilizadas para el resto de mensajes transmitidos en esa conexión. La entidad receptora aplicaría los cambios y respondería con el mensaje de respuesta mostrado en (3.17).

3.9.2.2 Actualización mediante difusión

En este caso la BS utilizaría difusión por inundación. Enviaría el mensaje de actualización a sus vecinos en un paquete de difusión con un número máximo de saltos, estos a su vez lo reenviarían a sus vecinos después de comprobar su validez y así sucesivamente. Cada nodo podría recibir más de una copia del mensaje. En este modo no se utiliza la información de enrutamiento.

Para poder utilizar difusión con el mismo nivel de seguridad que tienen los mensajes punto a punto, hay que resolver previamente una serie de dificultades. A continuación se muestran las dificultades detectadas y las soluciones propuestas:

- Al NL no se le exige que soporte la fragmentación de paquetes. Esta arquitectura evitará usar este método para mensajes superiores a la MTU del NL, si el NL no soporta fragmentación. Se podría utilizar fragmentación en el AL y técnicas como la descrita en [166] opcionalmente.
- El identificador de transacción TX_ID empleado debe ser único para todas las entidades de la red. Este identificador permite detectar duplicados y mensajes antiguos. Las entidades guardarán el último identificador de transacción enviado por la BS en un mensaje de difusión independientemente del que utilicen en la conexión punto a punto.
- Los datos no pueden ir cifrados, aunque como todas las entidades deben conocer la misma CRL esto no es necesario. En esta arquitectura no se van a utilizar claves simétricas compartidas por todas las entidades, ya que cualquier otra entidad de la red podría falsear un mensaje de difusión.
- Los mensajes en el NL no van autenticados. El mensaje de actualización debe permitir incluir firmas digitales.
- Si todas las entidades de la red responden simultáneamente, se puede producir el problema típico de implosión de asentimientos. Por consiguiente, no se responderá a estos mensajes. Si es necesario, la comprobación se hará posteriormente con mensajes de estado.

Además de comprobar que el identificador de transacción no sea duplicado ni antiguo, la verificación de la autenticidad del mensaje de difusión recibido se hará dependiendo de un parámetro de configuración adicional, `BVT` (*broadcast verify type*) cuyos valores aparecen en la Tabla 3.15, junto con las verificaciones que se llevarán a cabo en cada mensaje de difusión. El administrador de la WSN deberá seleccionar el tipo más apropiado en cada caso.

El contenido de la petición de actualización de la CRL con difusión se muestra en (3.20).

Tabla 3.15 Tipos de verificación de la autenticidad del mensaje de difusión.

Valor de `BVT`	Verificaciones realizadas
`BVT_FORBIDDEN`	La difusión está prohibida, por lo que todos los mensajes de difusión son ignorados. Solo se permite la actualización de la CRL mediante conexión punto a punto.
`BVT_KNOWN_BS`	Se verifica que la dirección de red origen es la de la BS. Esta sería la verificación mínima.
`BVT_KNOWN_BS_SIGN`	Además de la verificación anterior, el mensaje debe incorporar el $\mathbf{Cert}_{BS}^{CA}$ y una firma digital. Ambos deben ser verificados. El $\mathbf{Cert}_{BS}^{CA}$ incluye la clave pública que permite verificar la firma digital aunque no se haya establecido comunicación previamente con la BS.
`BVT_CONNECTED_BS_SIGN`	Similar al caso anterior, salvo que no se transmite el $\mathbf{Cert}_{BS}^{CA}$ y se reduce el tamaño del mensaje. La clave pública necesaria para verificar la firma digital debe haberse obtenido anteriormente durante el proceso de saludo de una conexión con la BS.

$$* \rightarrow * : TM\_SET\_CRL\_BCAST \parallel TX\_ID \parallel CRL\_DATA \parallel CERT \parallel SIGN \tag{3.20}$$

Los tres primeros campos del mensaje son equivalentes a los del mensaje de petición de configuración descritos en la Tabla 3.10. El resto de campos aparecen descritos en la Tabla 3.16.

Tabla 3.16 Campos adicionales en la petición de actualización de CRL por difusión.

Campo	Descripción
$CERT$	Es el $\mathbf{Cert}_{BS}^{CA}$. Contiene la clave pública que se utilizará para verificar la firma digital. Solo se transmite si el valor de `BVT` es `BVT_KNOWN_BS_SIGN`.
$SIGN$	Firma digital, realizada con la clave privada asociada al $\mathbf{Cert}_{BS}^{CA}$, de todo el resto del mensaje y la dirección origen.

3.9.2.3 Comparativa

Si la difusión está permitida (`BVT` no es `BVT_FORBIDDEN`) la BS elige entre un modo u otro pudiendo utilizar como criterios:

- El coste computacional.
- El número de paquetes transmitidos.
- Si se desea que entidades ajenas a la red puedan ver o no la información.
- La autenticación deseada.
- La fiabilidad ante fallos de enrutamiento.
- El tiempo necesario para configurar toda la red, que puede ser dependiente de todo lo anterior.

En difusión, si se quiere autenticar los mensajes, se tendría que utilizar criptografía asimétrica que es más costosa computacionalmente. Cada entidad necesita verificar el mensaje tan solo una única vez ya que los mensajes duplicados se detectarían fácilmente. Sin embargo, un atacante que haya comprometido uno de los nodos podría enviar mensajes falsos forzando a otras entidades a comprobar cada uno de ellos, consumiendo sus recursos hasta que no se ponga remedio. Usando la conexión punto a punto se utilizarían las claves simétricas negociadas durante el proceso de saludo como cualquier otro mensaje.

El número de mensajes transmitidos con un modo u otro dependerá de la topología de la red. Por ejemplo, si la topología es en árbol, la difusión sería muy eficiente. En cambio, en redes con mucha densidad, la difusión generaría muchas copias aunque los nodos esperen un cierto tiempo antes de volver a difundir el mensaje. El envío por difusión no es fiable y a veces es necesario repetir la difusión para minimizar la probabilidad de pérdida, como se hace en [8]. Con punto a punto, los nodos intermediarios más cercanos a la BS tendrían más carga pero se garantiza la entrega al utilizar envío fiable.

Con difusión, los datos estarán cifrados para entidades ajenas a la red solo si esto se hace en el LL, mientras que en punto a punto se puede elegir el nivel donde se cifra.

Con punto a punto, la BS obtiene respuesta a la petición y puede gestionar el envío para que no se produzca la implosión de asentimientos, sin embargo si hubiera algún problema de enrutamiento fallaría. Usando difusión, no habría problemas de enrutamiento pero habría que verificar que todas las entidades se han actualizado posteriormente.

Se deja al criterio de la implementación y al del administrador de la red decidir el modo más apropiado en cada caso.

3.10 Opciones y parámetros de la arquitectura

La arquitectura SNSR pretende ser lo más flexible posible, permitiendo que se pueda adaptar a distintos tipos de redes existentes y a las necesidades de seguridad de la instalación donde se vaya a utilizar. A lo largo de este capítulo se ha mostrado que muchos procedimientos de la arquitectura SNSR son opcionales y/o configurables. Se dejan a la elección de la implementación concreta de la arquitectura SNSR y a la configuración del administrador de la red, de tal manera que se adapten a los requisitos de una red de sensores en particular. En esta sección se hará un resumen de todo esto.

Las decisiones dependientes de la implementación y/o configuración son las siguientes:

- Modo de seguridad. Si se usa seguridad, también hay que decidir: los algoritmos permitidos, el tamaño de las claves, la validez temporal máxima de las claves y el proceso de saludo a utilizar.
- Contenido de la CRL de los nodos (completa o solo de certificados de robots).
- Método de detección de balizas.
- Método de actualización urgente de CRL.
- Tipo de verificación de mensajes de difusión.
- Tipo de enrutamiento.
- Algoritmos para el cálculo de trayectorias a seguir por el robot.
- Algoritmos para el cálculo de la topología de la red más eficiente.
- Tipo de entidad que realiza el cálculo de la topología.
- Número de versiones de la información topológica que deben almacenar la BS y el robot.
- Parámetros de comunicación por defecto para el mantenimiento de las conexiones. Reintentos, temporizadores y límites de saltos.
- Parámetros estáticos (preconfigurados) o dinámicos (configurables mediante la petición de configuración).
- Envío de configuración diferencial o completa.
- Información de estado adicional a recopilar.
- Anomalías a detectar. Límites del almacenamiento de anomalías. Políticas de eliminación y envío de anomalías.
- Algoritmos para el procesamiento de anomalías.

En la Tabla 3.17 se muestran los parámetros de configuración de todas las entidades. Algunos de ellos admiten la posibilidad de modificarse posteriormente. No se incluyen aquellos parámetros que son dependientes de la implementación.

Tabla 3.17 Cuadro resumen de parámetros de configuración.

Parámetro	Descripción
`ID`	Identidad de la entidad.
`CERT_CA`	Certificado de la CA.
`CERT`	Certificado propio.
`PR`	Clave privada propia.

Continúa en la siguiente página »

Tabla 3.17 (Continúa de la página anterior)

Parámetro	Descripción
DOMAIN	Nombre de dominio de la red.
BS_TYPE	Campo *tipo* de identidades de tipo BS, (3.1).
MS_TYPE	Campo *tipo* de identidades de tipo robot, (3.1).
SN_TYPE	Campo *tipo* de identidades de tipo nodo sensor, (3.1).
SEG	Modo de seguridad. Tabla 3.2.
CIPHER_SUITES	Conjunto de algoritmos de seguridad permitidos.
SIG_TYPE	Tipo de firmas digitales usadas en los certificados.
HASH_TYPE	Tipo de funciones *hash* usadas en los certificados.
MIN_KEY_LEN	Longitud de clave mínima permitida.
TMP_KEY_DUR	Duración de una clave temporal, intervalo de renovación de claves temporales en una conexión.
CRL	CRL inicial.
BVT	Tipo de verificación de mensajes de difusión.
B_DELAY	Tiempo a esperar antes de reenviar un mensaje de difusión.
LL_ADDRESS	Dirección MAC.
NL_ADDRESS	Dirección de red. Salvo en el robot, inicialmente será 0.
DEFAULT_HOP_LIMIT	Número de saltos máximos al enviar nuevos paquetes.
DEFAULT_KAI	KAI_g usado en las conexiones si no se indica otro expresamente.
KAI_FACTOR	$KAIF$ para calcular el $KATO$ de una conexión.
MAX_NB	Número máximo de vecinos.
MAX_ANOMALIES	Número máximo de anomalías a almacenar.
MIN_ANOM_INTERVAL	Tiempo mínimo que debe transcurrir antes de volver a enviar anomalías sin haber recibido un asentimiento.
MAX_MS	Número máximo de conexiones con robots y de candidatos. Ignorado por el robot.
BEACON_DETECTION	Detección de baliza básica o avanzada (no para el robot).

Continúa en la siguiente página »

Tabla 3.17 (Continúa de la página anterior)

Parámetro	Descripción
TC_MAX	Tiempo de contención máximo al recibir una baliza (no para el robot).
BI_MAX	Intervalo máximo entre balizas (solo para el robot).
MS_FAST_HS	Reducción en reintentos y temporizadores para conexiones con el robot.
TOP_CALC	Tipo de entidad que realiza el cálculo de la topología.

3.11 Conclusiones

Este Capítulo ha presentado el diseño de la arquitectura SNSR, que se corresponde con la *Contribución 1* de esta Tesis Doctoral, y su adaptación para el caso de una BS y un robot, que se corresponde con la *Contribución 2*. Para realizar este diseño ha sido necesario un análisis minucioso del problema que se pretendía solucionar así como de las tecnologías existentes. Adaptar una red de sensores que ya se está utilizando suele ser menos costoso que sustituirla por completo. El diseño de la arquitectura propuesta ha intentado ser lo suficientemente genérico y flexible para poder reutilizar la infraestructura existente y mejorarla aprovechando las capacidades del robot. Esta nueva arquitectura, denominada arquitectura SNSR, tiene como misión principal mejorar la seguridad de las WSN permitiendo elegir el nivel deseado de seguridad, ya que la seguridad implica un coste energético. Sin embargo, gracias al uso del robot, el coste de un mismo nivel de seguridad se reduce respecto a las tecnologías de WSN existentes. Es más, incluso si no se deseara ningún nivel de seguridad, se puede mejorar la eficiencia de las redes habituales al reducir la carga de trabajo de los nodos sensores. Esta arquitectura también es la base de un nuevo Sistema Operativo de Red (NOS, *Network Operating System*), dando lugar a nuevas familias de WSN. Se han detallado el software que debe ejecutarse en cada una de las entidades de la red, las comunicaciones entre ellas y el funcionamiento en conjunto.

La seguridad de una cadena es la del eslabón más débil. La arquitectura SNSR reduce la dependencia de unos nodos respecto a otros. Los nodos sensores van a ser el elemento más débil y, por tanto, más fácil de atacar. Si un atacante se hace con el control de un nodo, el efecto es mucho más limitado que en otras arquitecturas de WSN. Un nodo malicioso no puede alterar el funcionamiento de la red, como mucho puede provocar una degradación del funcionamiento a nivel local. El reparto de responsabilidad en temas de seguridad se hace de acuerdo a la debilidad de la entidad, aprovechando la heterogeneidad del sistema estudiado. Aplicar soluciones de redes ad hoc donde todos los componentes tienen la misma responsabilidad en el mantenimiento de la red supondría un uso no eficiente de recursos por un lado y un mal análisis de riesgos.

Por último, la gran cantidad de opciones que permite esta arquitectura, va a permitir experimentar con ellas, sin limitar la incorporación posterior de mejoras.

4 Implementación

No basta tener un buen ingenio, lo principal es aplicarlo bien.

René Descartes

En este capítulo se detallan la *Contribución 3* de esta Tesis Doctoral "*Sistema operativo de red genérico*", la *Contribución 4* de esta Tesis Doctoral "*Aplicaciones de gestión de robots, estaciones base y nodos sensores para SNSR*", y la *Contribución 5* de esta Tesis Doctoral "*Compositor/orquestador de escenarios y sistema de gestión remota de dispositivos*" . Se desarrolla un Sistema Operativo de Red (NOS, *Network Operating System*) capaz de ejecutarse en distintos equipos (robot, BS y nodos sensores) que implementa las funcionalidades descritas en la arquitectura SNSR. De esta manera se puede probar en equipos reales. La implementación tiene alta flexibilidad y permite probar distintas configuraciones, algoritmos, protocolos, entre otros.

Para realizar la implementación ha sido importante tener en cuenta los escenarios donde se van a realizar las pruebas. Uno de los objetivos principales de esta Tesis Doctoral es la experimentación y validación de la arquitectura desarrollada en dispositivos físicos y condiciones reales, pero sin depender de una red de sensores concreta.

En los escenarios objetivos reales el robot aéreo o terrestre estará equipado con un computador compacto, mientras que la BS será una estación de trabajo. Para los nodos sensores, se dispone de dispositivos con menor capacidad Raspberry PI 1 Model B. En esta Tesis Doctoral se utilizarán redes WiFi para la comunicación inalámbrica, en modo ad hoc, pudiendo modificar la MTU para asimilarlo a las WSN. Se ha creado un portal web que permite monitorizar las versiones instaladas del software en cada equipo, así como realizar actualizaciones de este. También permite cargar distintas configuraciones para poder hacer pruebas en distintos escenarios.

Al mismo tiempo se desean hacer pruebas masivas que involucren cientos de nodos sensores y con múltiples topologías diferentes. Dado que la realización de pruebas reales es más lenta y costosa, otro objetivo es poder realizar escenarios simulados, donde un mismo computador pueda ejecutar múltiples nodos sensores aislados lógicamente, pero utilizando el mismo software que se utilizaría realmente. Para esto, se ha desarrollado un

compositor/orquestador de escenarios específico, que se ha denominado *lanzador*, para probar estas redes sin necesidad de hacerlo en un entorno real. Este *lanzador* permite simular las propiedades de los canales de comunicación y se puede emplear para probar también otras arquitecturas. En escenarios simulados, la topología se podrá definir manualmente para probar casos límites, como podría ser una topología lineal o circular. También se podrán utilizar generadores automáticos de topologías que permitan generar grafos de cuadrículas (rectangulares y triangulares) y aleatorios para un número determinado de nodos sensores, permitiendo grafos fuertemente conexos o no y con un mínimo grado (número de conexiones con vecinos) para cada nodo. Las propiedades de los canales de comunicación también podrán ser modificadas (retardos de propagación, probabilidad de pérdida de tramas, duplicación y envío desordenado).

En la la Sección 4.1 se detallan las decisiones tomadas a la hora de implementar la arquitectura. En el resto del capítulo se explica la implementación de todo el software utilizado en esta Tesis Doctoral, comenzando en la Sección 4.2 con las aplicaciones de usuario que se ejecutarán en cada tipo de entidad. A continuación, en la Sección 4.3, se explica el NOS completo que se ejecuta en las entidades incluyendo las aplicaciones de gestión fundamentales para la arquitectura SNSR. Dado que en algunos escenarios se realizan ataques, en la Sección 4.4 se incluye una breve descripción de las herramientas utilizadas para esos ataques. Una vez descritas las entidades participantes en los escenarios, la Sección 4.5 se dedicada al *lanzador*, utilizado principalmente para escenarios simulados, aunque también es de ayuda en la preparación de escenarios reales. Para terminar, en la Sección 4.6, se muestra el sistema de gestión remota que permite controlar los dispositivos empleados en escenarios reales.

4.1 Decisiones de implementación

En la Sección 3.10 se mostraba la alta flexibilidad de la arquitectura SNSR, permitiendo distintas alternativas en la implementación y configuración. En esta sección se describirán las decisiones tomadas para esta implementación y las opciones que son configurables, ya sea en compilación, inicialización o ejecución.

Se desea hacer una implementación de la arquitectura genérica, que no dependa de una tecnología de red de sensores concreta. Además, debe mantenerse suficientemente flexible para que pueda ser adaptada posteriormente a redes reales. La implementación debe correr sobre un sistema operativo existente, ya que no es el objetivo de esta Tesis Doctoral desarrollar uno. Aunque existen sistemas operativos específicos para WSN, como los descritos en [39], se ha decidido utilizar el sistema operativo Linux para la ejecución de programas que actúen como las distintas entidades de la red. Linux es un sistema operativo de propósito general de código abierto y libre, que permite modificarlo y adaptarlo a distintos requisitos. Al estar soportado por todos los dispositivos utilizados, va a permitir simplificar las tareas de desarrollo. La implementación de una entidad como un programa de usuario del sistema operativo servirá posteriormente para simular escenarios permitiendo que un mismo ordenador ejecute distintas entidades simultáneamente.

Para mantener la implementación genérica, tampoco se va a utilizar una pila de protocolos existente. En su lugar, se va a implementar una nueva que dé más libertad para hacer

pruebas. Pero sin llegar a los niveles más bajos PHY ni LL, que requieran trabajar directamente con el hardware de los dispositivos. Así que se reutilizarán los niveles inferiores existentes ya soportados por el propio sistema operativo. Sobre un LL concreto se creará un nivel de adaptación que proporcione las funcionalidades que espera la arquitectura.

Se utilizan los lenguajes C/C++ para la codificación. Se busca obtener una implementación eficiente y rápida. Estos lenguajes se utilizan frecuentemente en el desarrollo de sistemas operativos [39] y pilas de protocolos (como la del propio sistema operativo Linux) y es habitual también encontrarlos en simuladores de red de eventos discretos como el *ns-3* [128]. Existen compiladores para prácticamente todos los microprocesadores por lo que el código se podrá portar de manera sencilla a otros dispositivos y sistemas. A partir del código fuente puede generarse un único programa que pueda actuar como cualquier tipo de entidad por configuración, aunque también se puede generar un programa individual para cada tipo de entidad. Dentro del código existen constantes (macros del preprocesador) que hacen que se compilen unas partes del código y otras no (compilación condicional). Esto se ha utilizado para modificar el comportamiento de las entidades, pudiendo seleccionar en tiempo de compilación distintas alternativas para implementar algunas funcionalidades. El programa generado se ha llamado `Node`. Además, para realizar pruebas, se ha creado otro programa partiendo del mismo código fuente, pero modificando determinadas partes para simular nodos maliciosos, al que se ha llamado `Badnode`.

La configuración en compilación es el modo de configuración más estático, ya que, una vez generado el programa, no se puede modificar. Esto reduce la flexibilidad, pero también limita la posible modificación malintencionada de un nodo y permite reducir el tamaño del ejecutable al no incluirse código que no se desea utilizar. La reducción del tamaño, la simplificación del análisis de los parámetros de configuración y las pruebas que se han realizado son los motivos por los que estas constantes no se han configurado en la inicialización.

El programa `Node` recibe la configuración en la inicialización a través de los argumentos de la línea de comandos o de un fichero de configuración. Algunos de los parámetros configurables pueden ser modificados en ejecución, pero otros no y conservarán su valor durante la ejecución del programa. Estos últimos suelen ser parámetros cuyo valor no se desea cambiar a posteriori, y que no podrían ser modificados con un mensaje de petición de configuración. En el listado 4.1 se muestra la ayuda que presenta el programa sobre los argumentos admitidos.

Código 4.1 Argumentos de configuración inicial.

```
1 Use: Node [OPTIONS] | [CONFIG_FILE]
2
3 Run a node.
4
5 -h, --help            shows this help,
6
7 Required options:
8 -n, --name=name       name of the node (CN certificate's field),
9 -c, --cert=file       file name of the certificate in DER format,
```

```
-k, --key=file         file name of the private key in DER format,
-m, --ca=file          file name of the CA public certificate,
-a, --netaddr=address  network address of the node (unsigned short),

Optional options (0 or more):
-p, --pid=file         save pid in file name
-x, --crl=file         file name of a CRL
-i, --input=string     listen to UDP socket. Format: IPv6#UDPport
                       listen to ETHERNET socket. Format: ifname or mac
-r, --route=string     add route entry. Format: subnet#bits#gw#metric
-b, --bs=address[#name] add network address of a base station. If name
                         is given, data will be encrypted.
-l, --link=string      add link. UDP: c|s|p#IPv6#UDP#name#addr#keepalive
                            ETHERNET: c|s|p#ETH#mac#name#addr#keepalive
-L, --securell=[01]    secure link layer (LL) data:
                         NO=0 (default), YES=1
-T, --securetl=[012]   secure transport layer (TL) data:
                         NONE=0, ALL=1, EXCEPT_NEIGBORS=2
-s, --suffixes=string  suffixes to differentiate node types. Format:
                         commonCNsfx#nodeCNsfx#managerCNsfx#bsCNsfx
-A, --autolinks=string config autolinks. Format:
                        [CNsuffix]#maxAutolinks#[secure[0|1]][#interval]
                        if CNsuffix is omitted, managerCNsfx+commonCNsfx
                        if secure is omitted, uses securell option
                        interval between attempts in seconds (optional)
-B, --beacons=string   send beacons. UDP IPv6#UDP#keepalive#interval
                                     ETH mac#keepalive#interval
-u, --unknown=[0123]   action for unknown packets:
                         DISCARD=0, DEFAULT_LINK=1, NEW=2, NEW_SECURE=3
                       default: 0 (types 'sn' & 'bs'), 2+securell ('ms')
-K, --keepalive=int    default keepalive interval for new links (0)
-U, --unknownpeer      allow unkwnown peers data in TL.
                         default: false (types 'sn' & 'ms'), true ('bs')
-P, --allowedpeers=x-y network addresses of allowed peers in TL. 0=all
-t, --type=string      application layer to run:
                         'al' (generic AL), 'sn', 'ms' (robot), 'bs'
-C, --command=string   command to execute on node at startup.
-v, --bvt=[0123]       broadcast verify type, source must be a BS:
                       FORBIDDEN=0,KNOWN=1,KNOWN_SIGN=2,CONNECTED_SIGN=3
-E, --cipherlist=string cipher suite list, colon-delimited: suite:suite
                         default is 'ECDHE-ECDSA-AES128-GCM-SHA256'
-F, --cipherlisttl=str cipher suite list for TL.
                         default is the same as --cipherlist
-d, --loglevel         trace=0, debug=1, info=2, warn=3, err=4,
                         critical=5, off=6
-f, --logfile          file name where logs are written
-H, --hoplimit         default hop limit used in network packets [1-255]

(c) 2016-2022 - Francisco José Fernández Jiménez. DIT.US.ES
```

La BS y los nodos sensores permiten una configuración posterior cuando se establezca la topología de la red: vecinos y cómo comunicarse con ellos, enrutamiento y la CRL principalmente. El robot también puede actualizar la CRL y recibir una lista de nodos a descubrir antes de entrar en la fase de descubrimiento.

Por último, cada tipo de entidad admite un conjunto de comandos administrativos, que pueden recibirse durante la ejecución por la entrada estándar (por defecto). Cada entidad proporciona una consola de comandos con la que un administrador, de manera manual o a través de otro programa, puede interactuar para obtener información del estado actual o modificar la configuración. El listado 4.2 muestra los comandos administrativos implementados comunes y específicos para cada tipo de entidad.

Código 4.2 Comandos administrativos.

```
=== Node commands ===
anomaly type => add anomaly.
beacons IP#PORT#KEEPALIVE#INTERVAL => config. UDP beacons.
beacons mac#KEEPALIVE#INTERVAL => config. ETH beacons.
cl c|s|p#IPv6#UDP#name#addr#ka => create new UDP link.
cl c|s|p#ETH#mac#name#addr#ka => create new ETH link.
ct c|s|p#name#addr => create new transport connection.
da id => delete anomaly.
dl IP#PORT => delete link (UDP).
dl mac => delete link (ETHERNET).
dn netaddr => delete neighbor.
dt netaddr => delete transport connection.
echo string => print string.
help => list available commands.
pa => print anomalies.
pals => print autolink statistics.
pb => print base stations (AL).
ping netaddr => send ping.
pn => print neighbors (NL).
pp => print peers (AL).
pr => print routes (NL).
ps => print statistics.
restart => restart node with same arguments.
stop => stop and exit.
tr netaddr => send trace.
transfer netaddr,num,size,reliable => transfer test.

=== Sensor commands ===
last-ping => timestamp of last ping response received.
pbs => print connected base stations.
reset-last-ping => reset last ping.
sa => force to send anomalies.
start-pings interval => send pings to base station.
```

```
stop-pings => stop sending pings to base station.

=== Robot commands ===
dump file => dump network data to file.
dumpnd file => dump last calculated network data to file.
fname [value] => get functionality as string [all if no value passed].
functionality value => set functionality to value.
functionality => get functionality.
load file => load network data from file.
prepare [file] => calculate network data [and optionally save to file].
pun => show unprocessed nodes.
radius float => set radius of propagation to calculate neighborhood.
sa => force to send anomalies.
savend prefix => save calculated network data to file prefix+CFG.bin.
status netaddr => send status request.
showpos [ID1#ID2...] => show positions of all nodes or nodes passed as
    IDs.
updpos ID1,x,y,z#[ID2...#]=> update position of nodes. ID can be a LL
    address or a name.
wfconfirm 0|1 => set to 1 to do not change automatically to
    functionalities DISCOVER, PREPARE or CONFIGURE. Wait explicit
    command.

=== Base Station commands ===
bcrl version => broadcast CRL version (0=last).
load file => load network data from file.
mcrl version => send CRL version to peers (0=last).
```

La implementación realizada permite combinaciones de parámetros que puedan dar lugar a que una entidad presente comportamientos diferentes a los descritos en la arquitectura SNSR. Esto se ha decidido así para comprobar la resistencia de la arquitectura ante configuraciones erróneas o maliciosas. Lo anterior también ha permitido probar ataques sencillos. Para otros más complejos se ha tenido que recurrir a herramientas externas y al programa `Badnode`, como se describe en la Sección 4.4 .

4.1.1 Modos de seguridad

Los modos de seguridad soportados por la arquitectura SNSR se mostraban en la Tabla 3.2. Todos los modos descritos se soportan y pueden emplearse en esta implementación.

La configuración del modo de seguridad se hace en la inicialización con dos parámetros: uno para indicar la seguridad en el LL y otro similar para el TL. En una misma red todas las entidades deben estar configuradas con el mismo modo de seguridad que debe haberse decidido con anterioridad al despliegue de los nodos (se supone que una posterior modificación de la configuración del robot y la BS es más sencilla). Cuando se establece la topología de la red, el modo de seguridad afectará a la configuración generada. Un nodo que recibe una petición de configuración descarta conexiones con vecinos que no sean compatibles con el modo de seguridad. Al establecerse una conexión en el LL o el TL, ambos extremos deben realizar el mismo proceso de saludo para que esta se establezca.

Por defecto, tal como define la arquitectura, ni la BS ni los nodos sensores aceptan conexiones desconocidas en el LL. En el TL, los nodos sensores solo aceptan conexiones de robots vecinos por defecto, y la BS acepta conexiones desde cualquier dirección origen. El robot acepta en ambos niveles aquellas conexiones que cumplan el modo de seguridad establecido. Estos comportamientos por defecto pueden ser modificados con parámetros de inicialización. Incluso puede especificarse un rango de direcciones de red permitidas para conexiones en el TL.

En el caso de querer asegurar un nivel o los dos, se debe decidir también el grado de seguridad, el proceso de saludo empleado, los algoritmos a utilizar, etc. Esto se realiza con ayuda del servicio de seguridad en el Plano de Gestión y Seguridad. La interfaz del servicio de seguridad es agnóstica al nivel de la torre de protocolos al que se está aplicando. Por tanto, las propiedades y el funcionamiento deben ser válidos en el LL y el TL.

En esta implementación, no se van a desarrollar protocolos nuevos para la segurización de las comunicaciones. La implementación se va a apoyar en un protocolo ya existente que cumpla todos los requisitos analizados en la Sección 3.3.1.1 y la Sección 3.3.2. Se ha elegido el protocolo DTLS en su versión 1.2 [125], aunque con vistas a una implementación futura de la versión 1.3 [127] que, entre otras cosas, reduce la sobrecarga del protocolo. DTLS está diseñado para funcionar sobre protocolos de datagramas no orientados a conexiones, lo que lo hace fácilmente adaptable al LL y al TL tal como se propone en la arquitectura SNSR.

Existen muchas implementaciones de DTLS. Para esta implementación se ha elegido la biblioteca de código `wolfssl Embedded SSL/TLS` [162], siendo la versión 4.3.0 la utilizada durante la redacción de estas líneas. Según sus propios autores es "una biblioteca ligera de SSL/TLS escrita en ANSI C y diseñada para entornos integrados, sistemas operativos en tiempo real y con recursos limitados, principalmente debido a su pequeño tamaño, velocidad y conjunto de características". Incluye además una biblioteca de funciones criptográficas (`wolfCrypt`) y es de código abierto licenciado con la versión 2 de la Licencia Pública General de GNU (GPL, *GNU General Public License*).

Se supondrá el uso de certificados usando ECC para realizar la autenticación de las entidades. Por consiguiente, se limitan las suites de cifrado posibles a aquellas que incorporen este tipo de criptografía. En compilación, se puede configurar el tamaño mínimo de las claves ECC permitidas, tiempo de vida máximo para el testigo utilizado en el saludo avanzado de reanudación y una lista de conjuntos de algoritmos de seguridad permitidos (*cipher suites*). En la Tabla 2 de [9] se muestra una comparativa del grado de seguridad proporcionado por diferentes algoritmos y tamaños de clave. Se ha utilizado como valor por defecto de tamaño mínimo de clave un valor de 256 bits, que según la anterior tabla equivaldría a una clave de 128 bits en Estándar de Encriptación Avanzado (AES, *Advanced Encryption Standard*) o 3072 bits en RSA. Para el tiempo de vida máximo del testigo se ha utilizado un valor por defecto de 600 segundos, que se considera pequeño para la mayoría de los casos, pero que para los experimentos realizados en esta Tesis Doctoral es más que suficiente. Para el conjunto de algoritmos de seguridad, es conveniente tomar como referencia las recomendaciones proporcionadas por la *Internet Assigned Numbers Authority* (IANA) en [65]. Solo hay 3 conjuntos recomendados, que son los que ofrecen mayor seguridad, que utilicen ECC tanto para el intercambio de claves como para la firma digital:

1. `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
2. `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
3. `TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256`

De entre los cuales, se ha elegido el primero por defecto por disponer de ordenadores con implementación de AES en hardware para hacer pruebas. Este conjunto está definido en [126] y comprende los siguientes algoritmos:

- Intercambio de claves: Diffie-Hellman de Curva Elíptica Efímero (ECDHE, *Elliptic-Curve Diffie–Hellman Ephemeral*).
- Firma digital: ECDSA.
- Cifrado simétrico: AES con clave de 128 bits usando *Galois/Counter Mode* (GCM).
- Resumen o *hash*: Algoritmo de Hash Seguro (SHA, *Secure Hash Algorithm*) 256.

La configuración de inicialización puede tener una lista de conjuntos de algoritmos de seguridad permitidos diferentes al establecido por defecto y debe incluir la identidad de la entidad $\mathbf{ID}$, el certificado de la autoridad certificadora $\mathbf{Cert}_{CA}$, el certificado propio firmado por esta $\mathbf{Cert}_{ID}^{CA}$ y la clave privada asociada a él $\mathbf{PR}_{ID}$. Es posible especificar un conjunto de algoritmos de seguridad diferente para el LL y el TL, pero no para conexiones individuales.

4.1.2 Codificación de los mensajes transmitidos

En la arquitectura SNSR no se establece la codificación que deben tener los mensajes definidos. Se entiende como codificación la serialización de los campos del mensaje para su transmisión. En un apartado introductorio sobre las convenciones utilizadas ya se proponían distintas técnicas.

En general, para la mayoría de los mensajes la codificación se ha realizado utilizando un formato binario compacto propietario. El formato de cada mensaje se detallará más adelante. Sin embargo, para los mensajes del AL definidos en la arquitectura SNSR se ha utilizado *Protocol Buffers* [53], con la versión de protocolo 3 y utilizando el motor en tiempo de ejecución *Lite* que consume menos recursos.

Protocol Buffers es una tecnología de serialización de datos desarrollada por Google con licencia de código abierto. Utiliza una codificación del tipo TLV usando un formato binario muy compacto donde la longitud puede ir implícita en el tipo. Permite crear protocolos extensibles: los campos pueden omitirse, no tienen que estar en un orden determinado y se pueden añadir nuevos campos posteriormente sin afectar a programas ya funcionando. Es independiente de la plataforma y del lenguaje de programación utilizados y se puede convertir fácilmente a otras codificaciones más legibles por humanos como *JavaScript Object Notation* (JSON). Los mensajes generados tienen un tamaño mayor que si se usara un formato compacto propietario, pero a cambio se aprovechan sus características y su facilidad de codificación, como [119] estudió en el dominio del IoT. Hay otros autores, como [129, 91, 144], que lo han utilizado en WSN y existen implementaciones para sistemas embebidos como nodos sensores, por ejemplo, `nanopb` [4].

4.1.3 Contenido de la CRL

Todas las entidades almacenan la CRL completa, sin distinguir los certificados revocados por tipo de entidad. La CRL inicialmente se puede importar a partir de un fichero. En el estándar RFC 5280 [14] se define un formato estándar para CRL, pero su uso no es obligatorio. El formato estándar usa Notación Sintáctica Abstracta 1 (ASN.1, *Abstract Syntax Notation One*) que, entre otros campos, incluye un conjunto de números de serie de certificados revocados y una firma digital para poder autenticar al emisor. Aunque el software utilizado soporta esta codificación, la implementación ha optado por usar un formato de fichero mucho más simple con las siguientes características:

- Basado en texto, por lo que se puede editar fácilmente con cualquier editor de texto.
- La primera línea contiene el número de versión.
- En cada una del resto de líneas habrá un número de serie de un certificado revocado como una cadena hexadecimal.

El robot y la BS también pueden cargar una CRL como parte de la información de la topología de red completa desde un fichero codificado con *Protocol Buffers*. Esta codificación también se va a usar para la transmisión de parte o toda la información de la topología y por tanto cuando se envíe la CRL en una petición de configuración.

Según las consideraciones indicadas en la Sección 3.1, es posible transmitir la CRL sin incluir una firma digital, ya que los nodos sensores confían en la BS y el robot y, si se usa seguridad en la comunicación, los mensajes pueden incluir implícitamente la autenticación del emisor.

4.1.4 Envío y detección de balizas

Respecto al envío de balizas, el intervalo máximo de transmisión BI_{max} puede tener valores de hasta 4 segundos. La componente aleatoria del tiempo de espera TE_{baliza} puede deshabilitarse en compilación, aunque no está permitido en SNSR para evitar que un adversario malicioso pueda deducir cuándo se va a transmitir la siguiente baliza. También es posible que cualquier entidad pueda transmitir balizas, aunque no sea un robot (único autorizado en SNSR), si se indica en la inicialización.

Dentro de los campos que forman parte de una baliza (véase la Tabla 3.7), el campo CP, que transporta los parámetros de comunicación a utilizar, solo necesita incluir el parámetro KAI (véase la Sección 3.3.4) utilizado para el mantenimiento de la conexión. Las entidades que se quieran conectar al emisor de la baliza, usarán la dirección origen de esta como extremo remoto y el uso de seguridad vendrá dado por el modo de seguridad en el LL configurado en el inicio.

Un emisor de balizas puede iniciar el envío en la inicialización o, posteriormente, tras solicitarlo un comando administrativo. Los parámetros que hay que indicar son: la dirección de difusión destino, el BI_{max}, y el parámetro KAI a incluir en la baliza. Si BI_{max} es 0, se detiene el envío de balizas.

En cuanto a la detección de balizas y el siguiente intento de conexión, se han implementado tanto la detección básica (Figura 3.17) como la avanzada (Figura 3.18). El tipo

de detección, el tiempo de espera de contención máximo TC_{max} y la utilización o no de reintentos durante la conexión son parámetros configurables en compilación.

Una entidad que se quiera conectar automáticamente a emisores de balizas, recibirá en la inicialización el número máximo de conexiones a establecer (1 si suponemos que solo va a existir 1 robot) y opcionalmente: el sufijo que debe tener el identificador de un emisor autorizado (por defecto, el sufijo de un robot de la misma red que el receptor), si la conexión va a ser segura (por defecto, el valor indicado por el modo de seguridad) y un intervalo mínimo entre intentos de conexión (por defecto, no se establece un mínimo).

4.1.5 Actualización urgente de la CRL y autenticación de los mensajes de difusión

Se soportan los dos modos de actualización urgente de la CRL descritos en la Sección 3.9.2: mediante conexión punto a punto o mediante difusión. La ejecución de uno u otro se realiza tras la recepción de comandos administrativos, es decir, se requiere de intervención externa.

Si se utiliza difusión para el anterior cometido, existen distintos tipos de verificación de la autenticidad de los mensajes enviados (véase la Tabla 3.15). Todos están soportados. Por defecto se utiliza `BVT_KNOWN_BS` y su valor puede modificarse en la configuración inicial. Es evidente que tanto el emisor (BS) como los receptores/retransmisores deben estar configurados con el mismo tipo de verificación para que el mensaje de difusión sea aceptado.

En los tipos que requieren el uso de firmas digitales para la autenticación, se utiliza ECC, dado que los certificados de las entidades están basados en este tipo de criptografía. Los datos que se firman son los devueltos por una función resumen o *hash* aplicada al resto del mensaje. La función *hash* puede configurarse en compilación (por defecto se usa SHA 256).

4.1.6 Tipo de enrutamiento

El enrutamiento en la arquitectura, en conjunto, podría clasificarse como de tipo adaptativo centralizado, con similitudes al paradigma SDN, ya que la decisión de enrutamiento se toma en una entidad centralizada. Pero a nivel local, una vez configurados, los nodos funcionan de manera autónoma hasta el siguiente cambio de configuración.

Localmente, para condiciones normales se ha implementado un enrutamiento estático basado en tablas de encaminamiento (enrutamiento proactivo, ya que cuando un paquete va a ser reenviado, la ruta debe ya existir). Sin embargo, si un nodo detecta un bucle porque la dirección de origen de un paquete recibido es la de sí mismo, como puede ocurrir en los transitorios de la fase de *configuración*, en vez de descartar el paquete utiliza un enrutamiento adaptativo aislado que va a seleccionar una ruta alternativa. Por tanto, el enrutamiento es el resultado de dos componentes: un enrutamiento *normal* y otro *alternativo*.

El funcionamiento en el enrutamiento *normal* es parecido al utilizado en las redes Protocolo de Internet (IP, *Internet Protocol*) pero simplificado. Sus principales características son:

1. Direcciones de 16 bits, habitual en otras WSN como ZigBee [8] o *IPv6 over Low-Power Wireless Personal Area Networks* (6LoWPAN) [101]. La dirección 0 se considera de enlace local y se utiliza siempre para la comunicación entre vecinos.

2. No soporta fragmentación, enrutamiento de origen, calidad de servicio ni notificación de errores.
3. Límite de número de saltos configurable. El valor máximo es 255.
4. Enrutamiento plano (no jerárquico). Todas las entidades participan del mismo modo.
5. Soporte de difusión. Cuando se recibe un paquete de difusión se reenvía a todos los vecinos excepto al que lo ha enviado.
6. La tabla de rutas está indexada por subred destino (dirección de subred y máscara) y contiene el vecino al que hay que pasar el paquete. Pueden existir distintas alternativas para un mismo destino ordenadas por: a) un valor de métrica y b) el orden de creación. Esta tabla se puede rellenar en la inicialización o durante la fase de *configuración*. Una entrada de la tabla puede desactivarse temporalmente de manera manual o porque no haya conectividad con el vecino. Se busca en la tabla de rutas la entrada con métrica menor, prefiriendo que el siguiente salto no sea a un vecino cuya dirección de red es la misma que la dirección origen del paquete y que no salga por el mismo enlace por el que se ha recibido. Si no se cumplen las anteriores condiciones, primero se prefiere enviar al vecino con dirección la del origen del paquete y después al enlace de entrada.

Cuando el emisor de un paquete detecta un bucle, utiliza un campo de la cabecera para indicar que a ese paquete hay que aplicarle un enrutamiento *alternativo*. El valor de este campo es un número natural que puede ser como máximo 3 e indica el número de veces que el paquete ha vuelto al origen. Si supera el valor máximo, el paquete es descartado. El enrutamiento *alternativo* no requiere interacción con otros nodos y se basa en un algoritmo heurístico subóptimo aislado desarrollado ex profeso para esta Tesis Doctoral. Este algoritmo intenta probar caminos alternativos al que seguiría el paquete normalmente (menos prioritarios), cada vez que se detecte un bucle, sin necesidad de guardar estados. Para ello trata de evitar siguientes saltos que retornen al origen o que devuelvan el paquete hacia atrás. Con enrutamiento *alternativo* el próximo salto de un paquete será un vecino que cumple alguna de las siguientes condiciones (de mayor a menor preferencia):

1. N-ésimo vecino de mayor prioridad que no cumpla el resto de condiciones. En el nodo origen, N será el número de vueltas detectadas más 1. En los nodos de tránsito N siempre será 1. Si N es mayor que el número de alternativas posibles para ese destino, se elige al vecino de menor prioridad que no cumpla el resto de condiciones.
2. Vecino que se utilizaría para enviar un paquete al origen, en los nodos de tránsito.
3. Vecino origen del paquete.
4. Vecino conectado al enlace de entrada del paquete.

El enrutamiento *alternativo* es un método poco costoso que soluciona algunos problemas que se han observado durante el transitorio, pero no todos, como bucles en los que no participe el nodo origen. Para reducir el efecto de otros problemas, se puede configurar en la inicialización el número máximo de saltos por defecto de los nuevos paquetes.

Para ilustrar el funcionamiento del enrutamiento se muestran en la Figura 4.1 4 casos representativos usando topologías de red sencillas (podrían ser parte de una mayor) en las se consigue que un paquete llegue a su destino a pesar de existir enlaces caídos. Las líneas negras sólidas son conexiones establecidas entre entidades y las punteadas son conexiones configuradas aún no establecidas. Las flechas rojas sólidas representan la transmisión de un paquete con enrutamiento *normal* y las punteadas con *alternativo*. El color de fondo de los nodos representa la distancia menor en saltos a la BS suponiendo que todas las conexiones configuradas estuvieran bien. Suponemos que todos los nodos tienen tantas rutas configuradas a la BS y a N3 como nodos vecinos, ordenadas de menor a mayor número de saltos teóricos. Con esos supuestos, el nodo N3 (con borde rojo) envía un paquete a la BS (con fondo rojo).

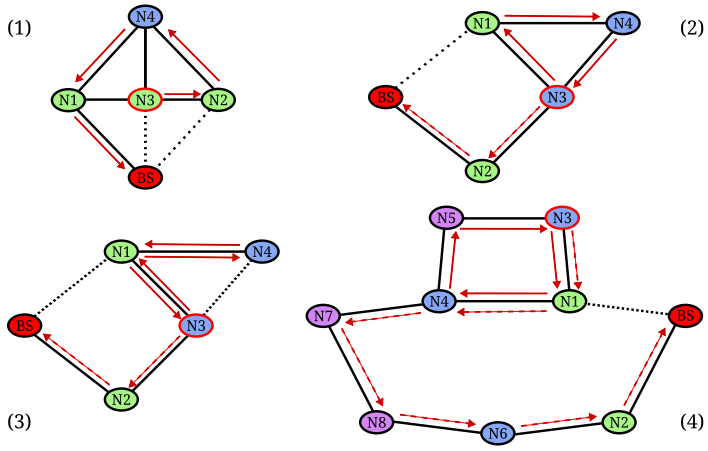


Figura 4.1 Funcionamiento del enrutamiento implementado. La BS se muestra en rojo y el color de los nodos depende de la distancia a la BS.

En el caso (1) solo se usa enrutamiento *normal*. N3 aún no ha establecido comunicación directa con la BS por lo que envía el paquete a N1 o N2 según el orden de configuración, sin conocer el estado de los enlaces de sus vecinos. Suponiendo que lo envía a N2 y este tampoco tiene la conexión directa con la BS, N2 elige enviarlo a N4 antes que al origen. N4 prefiere enviarlo a N1 antes que al origen o por el enlace de entrada. Por último, N1 lo envía al destino.

En el caso (2), N3 podría enviar el paquete a N1 o N2 igual que en (1). Si lo envía a N1, este lo reenviará al N4. N4 prefiere enviarlo al origen antes que por el enlace de entrada. N3 detectará el bucle y activa el enrutamiento *alternativo*, por lo que lo enviará usando la segunda ruta más prioritaria, que es a N2. N2 lo entrega al destino final.

El caso (3) es similar a (2), pero ahora no está establecida la conexión entre N3 y N4. N4 tiene ahora que reenviar el paquete por el enlace de entrada y N1 preferirá enviarlo al origen antes que al nodo anterior. N3 detecta el bucle y el resto es igual.

Por último, en el caso (4), N3 lo envía a N1, pero este no tiene la conexión directa con la BS operativa. Así, que N1 lo envía a N4 y N4 a N5, porque es la ruta más corta sin usar

el enlace por el que llegó el paquete. N5 no tiene más remedio que reenviarlo al origen que activa el enrutamiento *alternativo*, pero se ve forzado a reenviarlo de nuevo a N1. N1 lo reenvía a N4 y ahora N4 cambia su decisión, porque prefiere enviarlo a N7 antes que a N5, que sería el mismo vecino que usaría si tuviera que enviarlo al origen (esto no se hace en enrutamiento *normal*). N7 lo reenvía a N8 y así sigue hasta la BS.

4.1.7 Cálculo de la topología de la red

En esta implementación el objetivo principal es maximizar la robustez de la red. Para ello, se busca que la BS y los nodos tengan el número máximo posible de conexiones con vecinos (aunque su número puede limitarse en compilación) y que dispongan de rutas principales y alternativas con el resto de entidades. El orden de las rutas se decide intentando distribuir el trabajo de reenvío de paquetes entre todos los nodos. El robot va a ser el encargado de realizar el cálculo. Después comparte el resultado con la BS que lo acepta o rechaza verificando los números de versiones. A la BS también se le puede pasar información topológica de manera administrativa. El proceso completo se realiza en 7 pasos:

1. Generación de un grafo representativo de la red.
2. Cálculo de las rutas.
3. Ordenación de las rutas.
4. Asignación de direcciones de red.
5. Definición de los parámetros de comunicación entre vecinos.
6. Compresión de las rutas.
7. Actualización de la CRL y rellenado del resto de campos.

En las siguientes secciones se describen esos pasos.

4.1.7.1 Generación de un grafo representativo de la red

Lo primero que se realiza es una representación de la red usando un grafo $G = (V,A)$ formado por un conjunto de vértices V y aristas A. Cada vértice $v \in V$ es una entidad de la red y cada arista $a \in A$ representa la vecindad o adyacencia entre dos entidades. En esta implementación, los grafos se representan con listas de adyacencia: cada vértice $v \in V$ tiene una lista con los vértices adyacentes a él, que denotaremos como $N_G(v)$. La adyacencia se puede obtener de tres modos:

1. Obtenida de la versión inicial de la información de topología, que habrá sido establecida externamente.
2. Calculada a partir del rango de alcance radio *r* de los nodos. De manera optimista, se considera que dos nodos pueden ser vecinos si se encuentran a una distancia menor que *r*. Este modo se utilizará después de la primera fase de *descubrimiento* si en la versión inicial de información topológica no se incluían datos de vecindad.
3. A partir de la información de estado facilitada por las propias entidades al robot en una fase de *descubrimiento* posterior a la primera fase de *configuración*. Cada

entidad intenta conectarse con los vecinos indicados e informa con cuáles se ha conectado.

4.1.7.2 Cálculo de las rutas

En segundo lugar, se calculan las rutas más cortas desde cada vértice $v \in V$ al resto de vértices $v' \in V \setminus \{v\}$ para cada uno de los vértices adyacentes $w \in N_G(v)$ como siguiente salto. Por ejemplo, si un nodo x tiene n vecinos, se intentaría calcular n rutas hacia cada posible destino, siendo la ruta i la más corta que tiene al vecino $i \in N_G(x)$ como primer salto. Para lograr lo anterior se utiliza un algoritmo basado en búsqueda en profundidad (DFS, *Depth First Search*) [60] que también permite verificar si el grafo es fuertemente conexo [145]. Una ventaja de DFS es que requiere muy poca memoria para recorrer el grafo. El grafo se recorre desde cada uno de los vértices anotando simultáneamente el número de saltos de la ruta más corta a cada destino partiendo desde cada vecino (si existe). Esta información sirve para determinar con qué prioridad se elegirá un vecino u otro para enviar un paquete a un destino. La ventaja de tener múltiples alternativas almacenadas en la tabla de rutas es que el encaminamiento de paquetes es muy robusto. Si un vecino falla, un nodo tendrá disponible otra ruta, en principio la siguiente más corta (si no se han producido más caídas de las que no es consciente), sin necesidad de comunicarse con otras entidades. Esto a su vez, reduce el número de reconfiguraciones necesarias que pueden implicar una nueva fase de *descubrimiento* con el robot. El inconveniente es la escalabilidad. Está demostrado que en un enrutamiento basado en rutas más cortas el tamaño de la tabla de rutas crece linealmente con el tamaño de la red en el mejor de los casos [85]. Normalmente el crecimiento es supra lineal. Un ejemplo similar muy conocido de este problema es el que se produce en las tablas del protocolo de puerta de enlace de frontera (BGP, *Border Gateway Protocol*) de Internet. En este caso, con rutas alternativas, el tamaño de las tablas de enrutamiento son del orden de $\mathrm{O}(|V|\, g)$, siendo $|V|$ el número de elementos de la red y g el grado medio. Según las características de las redes (como aquellas del tipo *scale-free* o *small-world*) se pueden encontrar soluciones más escalables si no se fuerza a usar las rutas más cortas [85]. El número de rutas también se puede reducir si se tiene en cuenta las aplicaciones que se van a ejecutar sobre la red. Por ejemplo, si se conoce que los nodos se comunican exclusivamente con la BS y no entre ellos, se pueden excluir todas las rutas que usen la BS como punto intermedio.

4.1.7.3 Ordenación de las rutas

A continuación, las rutas calculadas se ordenan. La métrica asociada a cada ruta es proporcional a su longitud (en número de saltos). De esta manera las rutas más cortas tendrán una métrica menor y por tanto serán más prioritarias. Puede ocurrir que existan varias rutas alternativas de longitud mínima. En estos casos las rutas se ordenan intentando que el tráfico se distribuya lo más equitativamente posible entre los posibles nodos de tránsito en el caso más probable: ningún nodo ha fallado y siempre se utilizan las rutas más prioritarias. También se ha supuesto que todos los nodos van a generar el mismo tráfico y que todas las entidades pueden hablar entre sí. Para realizar lo anterior se siguen los siguientes pasos:

1. Se define la carga de un vértice $l_v \quad \forall v \in V$, como el número de veces que el vértice es elegido el primer salto de la ruta más prioritaria entre otros dos vértices $x \in V, y \in V$. Inicialmente es 0.

2. Sea $r^f_{x,y} \quad x \in V, y \in V, f \in N_G(x)$ la primera ruta más corta encontrada entre x e y usando f como primer salto. La longitud mínima de la ruta desde x a y vendrá dada por $min_{x,y} \stackrel{\text{def}}{=} \text{mín}\left\{\left|r^f_{x,y}\right|\right\} \quad \forall f \in N_G(x)$. No se tienen en cuenta las rutas que no existen, sería como suponer que tienen longitud infinita. Si $min_{x,y} = \infty$ el resto del proceso no se aplicaría.

3. Sea el conjunto de rutas de longitud mínima entre el vértice x y el y, $R^{min}_{x,y} \stackrel{\text{def}}{=} \bigcup r^f_{x,y} \quad : x \in V, y \in V, x \neq y, f \in N_G(x), \left|r^f_{x,y}\right| = min_{x,y}$. Sea U el conjunto formado por todos los conjuntos anteriores $U \stackrel{\text{def}}{=} \left\{R^{min}_{x,y}\right\} \quad \forall x,y \in V, x \neq y$.

4. Se ordenan los elementos U según el número de rutas alternativas, $\left|R^{min}_{x,y}\right| \quad x \in V, y \in V, x \neq y$, de menor a mayor.

5. Siguiendo el orden anterior, se ordenan los elementos de cada $R^{min}_{x,y} \quad x \in V, y \in V, x \neq y$ según la carga del vértice usado como primer salto, $l_f \quad f \in N_G(x), r^f_{x,y} \in R^{min}_{x,y}$. A continuación se incrementa en una unidad el valor de carga menor: $l_{f'} = l_{f'} + 1$ donde $l_{f'} = \text{mín}\left\{l_f\right\} \quad \forall f \in N_G(x), r^f_{x,y} \in R^{min}_{x,y}$.

Este proceso se puede particularizar para aplicaciones concretas de la red de sensores, donde se conozca el tráfico que va a generar cada nodo y las comunicaciones entre ellos, obteniendo resultados mejores haciendo ingeniería de tráfico.

4.1.7.4 Asignación de direcciones de red

El siguiente paso es la asignación de direcciones de red. Estas se van a representar en hexadecimal con cuatro cifras. A la BS siempre se le asigna la dirección `0x0001`. El robot debe tener una ya asignada desde el inicio, comprendida en el rango [`0x0002`, `0x00ff`], que no cambia. Para los nodos sensores se establecen dos planes de numeración sencillos:

1. Cada nodo x tiene una dirección de red NA_x con dos componentes del tipo `0xHHNN`, donde `HH` es el número de saltos de la ruta mínima a la BS y `NN` es un número que se va asignando consecutivamente a los nodos situados a la misma distancia de la BS. Si existen más de 256 nodos con la misma distancia, se utilizan las direcciones con prefijo `HH+1`. Si el número de nodos con la misma distancia es superior a 512, se utiliza el segundo plan de numeración. Los nodos que no tengan conectividad con la BS toman una dirección consecutiva en el rango [`0xfe00`,`0xffff`], dejando sin numerar aquellos que no quepan en ese rango.

2. En el caso de que no se pueda aplicar el plan de numeración anterior, se asigna una dirección a cada nodo que tenga conectividad con la BS, en orden de menor a mayor cercanía a esta, en el rango [`0x0100`,`0xfdff`]. La numeración para los nodos sin conectividad con la BS es similar al plan anterior.

El primer plan de numeración permite visualizar rápidamente la topología y es el que se ha utilizado en los experimentos presentados en el Capítulo 5 de esta Tesis Doctoral. En ambos casos, se verifica que si x e y son dos nodos sensores, entonces:

$$\text{Si } \exists x,y \in V : min_{x,BS} > min_{y,BS} \Longrightarrow NA_x > NA_y \tag{4.1}$$

es decir, la dirección de un nodo más cercano a la BS siempre será menor que uno más alejado (si la distancia es igual no se puede garantizar). En la Figura 4.2 se muestra una red aleatoria con 50 nodos con las direcciones asignadas como un ejemplo de aplicación del plan de numeración, donde los nodos con el mismo prefijo de dirección `HH` aparecen con el mismo color.

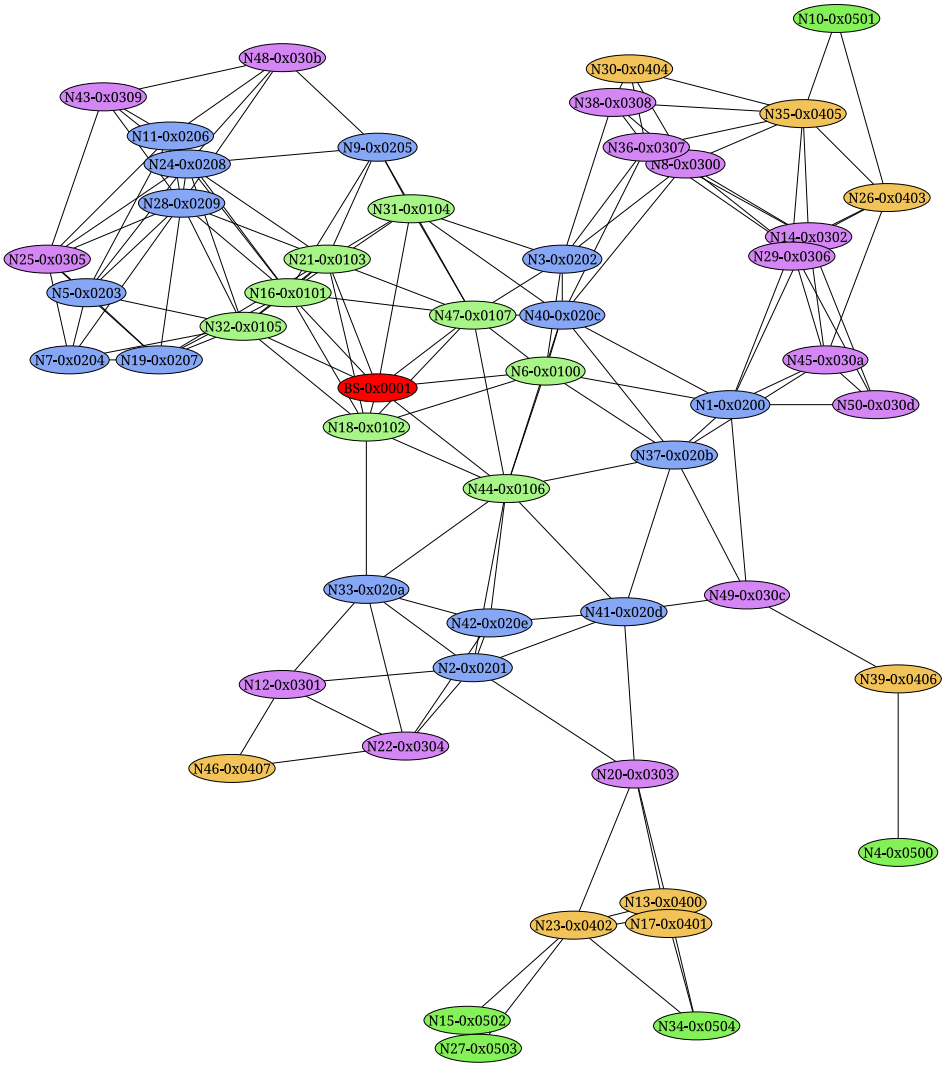


Figura 4.2 Ejemplo de asignación de direcciones de red. La BS se muestra en rojo y los nodos con el mismo prefijo de dirección `HH` aparecen con el mismo color.

4.1.7.5 Definición de los parámetros de comunicación entre vecinos

Si se usa el saludo avanzado en el LL, hay que definir el papel que jugará cada extremo de la comunicación: cliente o servidor. Esta información estará contenida dentro de los parámetros adicionales para establecer la comunicación con cada vecino. Asumiendo que en el intercambio de claves se va a usar ECDHE, el coste computacional es similar para el cliente y el servidor si el proceso de saludo tiene éxito [107]. Pero el cliente transmite un mensaje más por término medio y el servidor está más expuesto a ataques de alto consumo. Se ha preferido intentar asignar el papel del servidor al extremo con más recursos o con menos carga de red (reenvío de paquetes). La BS y el robot siempre actuarán como servidores. Cuando ambos extremos sean nodos sensores, se ha tenido en cuenta que es más probable que sea mayor el tráfico que pasa por un nodo más cercano a la BS que el que pasa por uno más alejado. Aprovechando (4.1), la regla simple que se aplica en estos casos es que el nodo con dirección de red más alta sea el servidor.

Independientemente del proceso de saludo, también se establece el parámetro de mantenimiento de la conexión KAI. El valor es común a todas las conexiones y es pasado al robot durante la inicialización.

4.1.7.6 Compresión de las rutas

El número de rutas obtenidas en el segundo paso puede ser muy elevado, lo que afecta a la memoria necesaria para la tabla de rutas y al tamaño del mensaje de petición de configuración. El enrutamiento basado en rutas más cortas se ha demostrado que es incomprensible de manera teórica [52], en el sentido de que no hay ningún algoritmo que funcione de manera universal en todas las redes posibles. Aun así, en muchas redes se puede reducir el tamaño de la tabla de rutas con el procedimiento descrito a continuación.

La información de las rutas se enriquece con las direcciones de red asignadas a cada vértice. El conjunto de rutas de un origen x a un destino y, $R_{x,y}$, se va a representar como un conjunto de pares $\langle métrica, NA_f \rangle$ ordenado por métrica, donde el NA_f es la dirección del vecino $f \in N_G(x)$ que es el siguiente salto de la ruta si existe y no se repite. Este conjunto de pares puede repetirse para otros destinos. Se va a denominar *multiruta* al conjunto de destinos que comparten el mismo conjunto de pares $\langle métrica, NA_f \rangle$, incluyendo ese conjunto de pares común. El procedimiento de compresión realiza las siguientes tareas:

1. Se forman el menor número de *multirutas* posibles.
2. Se agrupan las distintas direcciones destino de cada *multiruta* en subredes virtuales de mayor tamaño, siempre que no afecte a otros destinos.
3. La *multiruta* con mayor número de destinos se toma como ruta por defecto del nodo (con un solo destino, que es la subred 0/0).

4.1.7.7 Actualización de la CRL y rellenado del resto de campos

A la información de la topología se añade la CRL actualizada y se rellenan los campos que falten. Por ejemplo, los datos de la BS que va a utilizar cada nodo sensor serán los mismos. Y para terminar se incrementa el número de versión.

4.1.8 Almacenamiento de la información topológica

La BS y el robot guardan las 2 versiones mínimas indicadas en la Sección 3.4: la versión inicial y la actual. El robot también mantiene una versión temporal durante el cálculo de la topología a la espera de ser confirmada por la BS.

Ambas entidades admiten la importación y exportación de cualquiera de las versiones a ficheros. Se utiliza *Protocol Buffers* para la serialización de los datos, ya sea para el almacenamiento en el sistema de ficheros o para la transmisión usando mensajes del AL. El robot también permite exportar una versión de la información topológica abreviada, con solo datos obtenidos durante la fase de *descubrimiento*, que puede incluir información de nodos que no están autorizados, pero que se hayan conectado.

Por último, se ha creado una utilidad complementaria, llamada `NetworkDataExport`, que permite la conversión de ficheros codificados en binario con *Protocol Buffers* a JSON y viceversa. Esta utilidad permite a los administradores visualizar y modificar directamente la información almacenada en ficheros binarios con cualquier editor de textos.

4.1.9 Parámetros de comunicación por defecto

Aunque anteriormente se han nombrado parámetros que afectan a la comunicación, existen otros que merece la pena destacar:

1. El valor de KAI_g es configurable en el inicio. Este valor se aplica en las conexiones en el TL. En el LL se utiliza el valor comunicado por el robot o en la información de la topología.
2. El valor de $KAIF$ es una constante de compilación. Por defecto se usa el valor 4.
3. El número de reintentos máximos de una conexión en el LL es:
   - 0, para conexiones con el robot.
   - 2, para conexiones creadas administrativamente.
   - Sin límite, para conexiones creadas según la información topológica.

   En el TL no se hacen reintentos automáticos, pero se informa al AL cuando una conexión se cierra para que pueda ser reiniciada si es necesario.
4. Una conexión en el LL puede ponerse en *modo ignorar* durante un tiempo cuando se detecten un número predeterminado de errores similares consecutivamente. En este modo todas las tramas recibidas son descartadas, por lo que se evita tener que realizar saludos que posiblemente terminen con el mismo error y así se reduce el consumo energético. Esto es configurable en compilación y por defecto se ha establecido una duración de 100 segundos cuando ocurran 10 errores consecutivos en procesos de saludos relacionados con la validación del certificado o de los datos intercambiados en el proceso.
5. La MTU del LL se puede configurar en compilación para el caso más favorable. A partir de este valor y de manera dinámica se calcula la MTU de cada enlace. Este valor se propaga por los niveles superiores, restando el tamaño de las cabeceras de las PDU. Por ejemplo, en el TL se utiliza para determinar cuándo una SDU debe fragmentarse.

6. El límite de saltos máximos en el NL por defecto se puede configurar en el inicio y en compilación.

4.1.10 Mensajes de aplicación

Los mensajes de aplicación definidos en la arquitectura SNSR presentan campos opcionales, campos dependientes de la implementación y la posibilidad de ser ampliados.

En el mensaje de respuesta de estado no se va a incorporar información adicional a la ya indicada, ya que no es necesario para el cálculo de la topología.

El mensaje de petición de configuración no soporta el modo de transferencia de configuración diferencial, tampoco ha sido extendido con más información y la información de encaminamiento puede incluir rutas individuales o *multirutas*.

Se ha definido un nuevo mensaje, *PING*, adicional a los definidos en la arquitectura para las siguientes operaciones:

1. comprobar la conectividad entre dos entidades en el AL,
2. medir el tiempo de ida y vuelta (RTT, *Round-Trip Time*),
3. intercambiar identificadores de entidad,
4. y hacer pruebas de transferencias fiables o no y con fragmentación o no.

El formato está representado en la Figura 4.3.

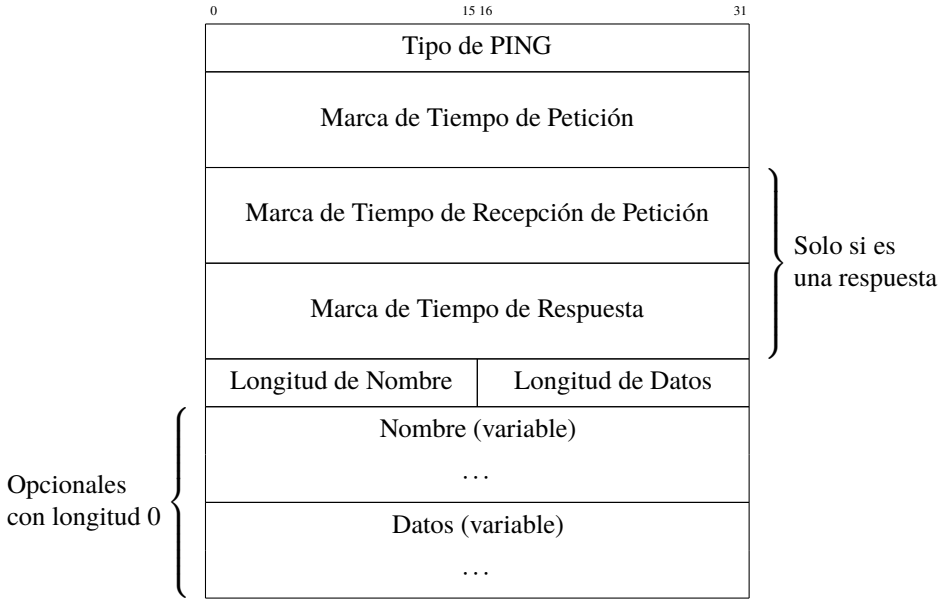


Figura 4.3 Formato de un mensaje *PING*.

Las marcas de tiempo indican instantes de tiempo con una precisión de microsegundos. Son el número de segundos transcurridos desde el 1 de enero de 1970, codificando la parte

entera en los primeros 32 bits y la parte fraccionaria en los restantes como un número de microsegundos.

El tipo del mensaje *PING* es un número natural codificado en binario. Existen 3 tipos:

1. Petición (tipo 0): utilizado para solicitar al otro extremo un mensaje de respuesta con el objetivo de comprobar la conectividad, medir el RTT e intercambiar identificadores. Se envía una marca de tiempo indicando el instante en el que se envió el mensaje, T_P.
2. Respuesta (tipo 1): respuesta a un mensaje de petición. Se copia la marca de tiempo de la petición y se añade una marca de tiempo del momento en que se recibió la petición, T_{RP} y otra indicando cuando se envió la respuesta, T_R.
3. Test de transferencia (tipo 2): permite hacer pruebas de transferencias de datos. El receptor tan solo registra su recepción y no debe responder.

El mensaje *PING* incluye la identidad del emisor y opcionalmente unos datos adjuntos. Son cadenas de caracteres variables y sus longitudes se codifican en sus respectivos campos auxiliares. Si la longitud de un campo es 0, no se envía.

Cuando una entidad recibe un *PING* de respuesta puede calcular con precisión el RTT sin necesidad de que los relojes de ambos extremos estén sincronizados. Se anota el instante de tiempo de recepción de la respuesta, T_{RR}. En la Figura 4.4 se muestra el intercambio de mensajes *PING* entre dos nodos con los instantes de tiempos marcados. Se supone que el reloj del nodo X tiene una desviación desconocida o con respecto al del nodo Y.

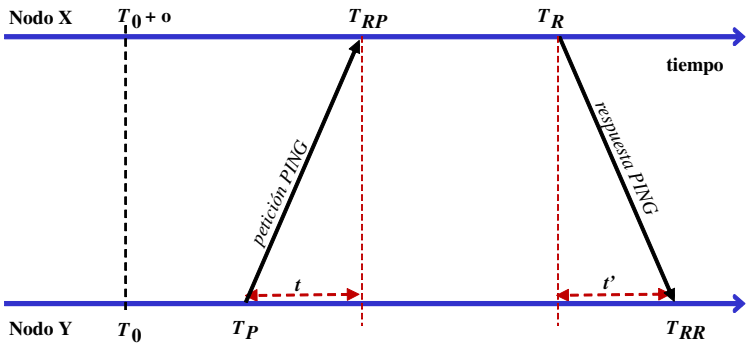


Figura 4.4 Envío y recepción de mensajes *PING*.

Las marcas de tiempo del nodo X estarán afectadas por la desviación o que habría que restar. El cálculo de RTT sería:

$$\begin{aligned} RTT &= t + t' \\ t &= (T_{RP} - o) - T_P \\ t' &= T_{RR} - (T_R - o) \\ RTT &= T_{RR} - (T_R - o) + (T_{RP} - o) - T_P \\ RTT &= T_{RR} - T_R + T_{RP} - T_P \end{aligned} \tag{4.2}$$

Por tanto, la desviación de los relojes no afecta al valor obtenido de RTT.

4.1.11 Anomalías detectadas y su procesamiento

Todas las anomalías de la Tabla 3.13 se detectan. Además, se pueden introducir nuevas anomalías mediante comandos administrativos para hacer pruebas que validen la implementación. Se ha realizado un procesamiento básico de las anomalías para probar el funcionamiento descrito en la arquitectura SNSR.

La BS va a ir almacenando las anomalías recibidas para que queden registradas sin hacer ningún otro procesamiento por el momento, aunque permite acceder a ellas desde otros procesos para su posterior análisis. No se han establecido límites en el número de anomalías a almacenar y por tanto no se van a agrupar (el campo $REPEATED$ no se usa) y no se eliminan anomalías que no hayan sido asentidas de manera automática. Se pueden borrar anomalías mediante un comando administrativo.

El tamaño de la lista de anomalías enviada en una notificación de anomalías se limita para evitar que el mensaje se fragmente.

El robot transmite todas las anomalías que tiene almacenada a la BS cada vez que esta se conecta con él. Continúa transmitiendo las anomalías en bloques de acuerdo al límite antes comentado mientras que la BS las siga asintiendo. Si algún nodo le envía anomalías, envía el correspondiente asentimiento e inmediatamente las retransmite a la BS.

Los nodos comienzan el envío automático de anomalías cuando establecen comunicación con la BS. También se puede forzar administrativamente que el robot o los nodos intenten enviar sus anomalías inmediatamente.

4.1.12 Módulo de localización y movimiento del robot

En esta Tesis Doctoral no se va a desarrollar un módulo real de localización y movimiento para el robot. El software implementado está preparado para colaborar con uno existente. Se ha desarrollado un componente que simula este módulo en los escenarios simulados. Este módulo debe tener las siguientes funcionalidades:

- Obtención de la posición actual.
- Localización de fuentes de emisión a su alcance.
- Cálculo de la trayectoria a seguir por el robot en la fase de *descubrimiento* y *configuración*.
- Movimiento del robot siguiendo trayectorias definidas.

La comunicación de este módulo con la implementación del robot realizada en esta Tesis Doctoral se realiza en ambas direcciones:

- El módulo debe ir almacenando la dirección MAC junto con la posición de cada emisor. Esta información debe estar disponible para el robot antes de iniciar el cálculo de la información de topología. Para ello, se actualiza la versión inicial de la información de topología por vía administrativa.
- La aplicación de gestión del robot permite la consulta, también por vía administrativa, del estado actual del robot y las entidades con las que está pendiente de establecer

el contacto. Con esta información, el módulo puede determinar qué movimientos debe realizar el robot.

- Se puede indicar al robot el fin de una fase, forzando un cambio de estado. El robot puede desconocer el número de nodos que tiene que descubrir y debe ser informado cuando se ha completado la trayectoria objetivo para cambiar a la siguiente fase. Si el robot conoce los nodos, el cambio de fase es automático cuando todos los nodos se han comunicado con él.

4.1.13 Algoritmos para el cálculo de trayectorias a seguir por el robot

Como esto no es un objetivo principal de esta Tesis Doctoral se van a suponer solo dos casos simples. La trayectoria seguida por el robot afecta al tiempo necesario para completar las fases de *descubrimiento* y *configuración*, pero no a la seguridad o el funcionamiento final de la red.

La zona donde se encuentran los nodos sensores y la BS se supone conocida y de planta rectangular. El robot sobrevuela el escenario en un plano horizontal a una altitud superior a la de cualquier sensor y libre de obstáculos. El robot se va a mover de manera lineal y las trayectorias descritas más adelante se van a componer de segmentos lineales en solo dos dimensiones. También se supone que se usarán antenas omnidireccionales y que se conoce el rango de alcance radio *r* de los nodos.

Si se desconocen las posiciones de los nodos, el robot debe recorrer toda la zona detectando los nodos que existan. La trayectoria que sigue el robot en este caso se denominará *trayectoria de descubrimiento*.

Si los nodos ya están localizados y se desea establecer comunicación con todos ellos (por ejemplo, para configurar o preguntar por el estado), los movimientos del robot pueden ser más precisos. Se denominará *trayectoria de configuración* al camino seguido por el robot en este caso.

La resolución de la mejor trayectoria en cada caso es un típico problema de planificación de ruta de cobertura (*coverage path planning*), conocido también como problema del cortacésped. Su clase de complejidad es *NP-hard* y se puede solucionar con distintos métodos [46]. Se ha utilizado un método exacto de descomposición celular basado en un movimiento de zigzag, también conocido como movimiento de esparcidor de semillas, como los mostrados en [90], o movimiento bustrófedon [20].

Para calcular la trayectoria, es necesario conocer el área que cubre el robot desde un punto. Se ha tomado de referencia el círculo a nivel del suelo que corta con una esfera de radio *r* con centro en la posición del robot, al que se denominará radio efectivo, r_e. Se define también un radio de seguridad r_s, utilizando un factor de seguridad adicional σ dentro del intervalo $[0, 1]$, tal que $r_s = \sigma r_e$.

En la *trayectoria de descubrimiento* la zona se divide en celdas horizontales de ancho $2r_s$ que se recorren en su totalidad siguiendo una línea central, desde el lado más próximo al robot al más alejado. Estas zonas se recorren de arriba a abajo o viceversa dependiendo de si el robot está más cercano a la parte superior o inferior respectivamente, con un movimiento bustrófedon.

En la *trayectoria de configuración* la división de la zona es similar, pero la línea central no recorre toda la celda, sino solo donde existan nodos a una distancia menor de r_s, pero

manteniendo un movimiento bustrófedon. En la Figura 4.5 se muestra la *trayectoria de configuración* de un robot (línea verde) moviéndose a una altitud de 5 m, con $r = 50m$ y $\sigma = 0{,}8$, en un escenario con 100 nodos (puntos negros) y una BS (punto rojo). El área verde representa el área cubierta según r_e, observándose áreas recorridas más de una vez debido a σ.

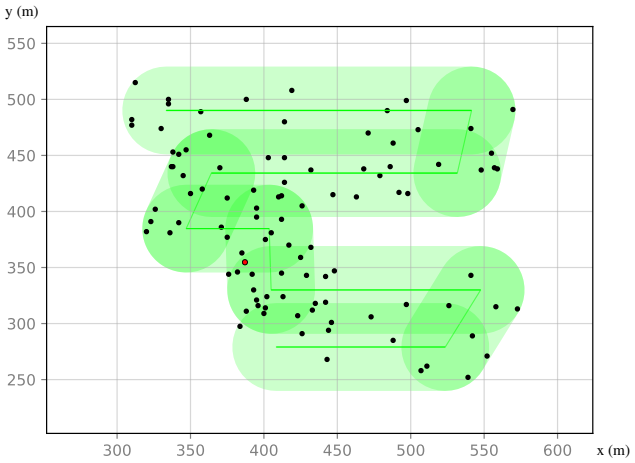


Figura 4.5 Ejemplo de trayectoria del robot en fase de *configuración*.

La velocidad máxima del robot durante la ejecución de la trayectoria también es importante para que la comunicación se pueda completar. Normalmente, la velocidad en descubrimiento debe ser menor que en configuración. Si se conocen los nodos con los que hay que comunicarse y uno de ellos no lo ha hecho, el robot puede acercarse a la posición del nodo y esperar a que se establezca la comunicación. Así, una vez terminada la *trayectoria de configuración*, el robot se dirige a las posiciones más próximas de los nodos que falten, empezando por el más cercano.

4.2 Aplicación de usuario de los nodos sensores

Además de la aplicación de gestión descrita en la arquitectura SNSR, los nodos tienen una aplicación de usuario encargada de realizar su tarea operativa habitual. La actividad de los nodos en la WSN depende de la aplicación concreta que se le vaya a dar a la red. Para probar la arquitectura se va a simular el caso típico de nodos que envíen medidas periódicas a la BS.

Los nodos envían un mensaje *PING* de petición donde la medida es el instante de tiempo que transporta este mensaje. La BS responde con un mensaje *PING* de respuesta, permitiendo comprobar el RTT y que la comunicación es bidireccional. La BS registra los mensajes *PING* recibidos sin realizar más operaciones.

El modus operandi anterior comienza automáticamente cuando el nodo finaliza el proceso de saludo con la BS y se realiza de manera periódica en intervalos configurables: el valor por defecto en compilación es de 10 segundos pero se puede modificar mediante un comando administrativo.

4.3 Sistema operativo de red (NOS)

En esta sección se describe el software que va a implementar una pila de red genérica adecuada a la arquitectura SNSR y la aplicación de gestión correspondiente en cada una de las entidades. Este software se debe ejecutar junto con una configuración adecuada para que el comportamiento en conjunto de la red completa sea el especificado.

Como se comentó al inicio del capítulo, se utiliza Linux como sistema operativo base sobre el que se realizan modificaciones. La pila de protocolos implementada parte de un LL proporcionado por el sistema operativo. El LL puede ser real o virtual (creando un túnel virtual sobre otro protocolo que no tiene que ser de LL). Sobre esto se sitúa un nivel de adaptación al nivel de enlace (LLAL, *Link Layer Adaptation Layer*). Este nivel es distinto dependiendo del LL que se vaya a usar, de tal manera que la pila de red pueda funcionar en distintos sistemas operativos y tipos de redes. Este nivel de adaptación abstrae el nivel de enlace real al resto de niveles superiores, permitiendo que el funcionamiento de estos sea homogéneo a pesar de las diferencias reales. Este nuevo nivel aparece representado en la Figura 4.6. Esta figura parte de la Figura 3.3 vista anteriormente, aunque desde un punto de vista plano. El servicio de seguridad y el servicio de gestión se muestran esta vez separados por simplicidad para representar que ambos interactúan con el LLAL pero no con el LL real. También se observa que el AL puede interactuar directamente con el NL.

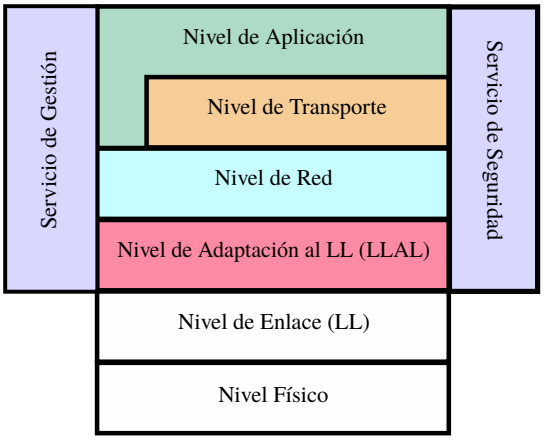


Figura 4.6 Arquitectura por niveles.

Los requisitos funcionales necesarios que debía cumplir una torre de protocolos para implementar la arquitectura SNSR aparecían descritos en la Sección 3.3.1. La implemen-

tación cumple todos los requisitos mencionados allí. A continuación, se describen las propiedades no funcionales principales de la implementación.

4.3.1 Propiedades no funcionales

A diferencia de los requisitos funcionales que especifican qué hay que implementar, los requisitos o propiedades no funcionales precisan cómo debe ser realizado (eficiencia, seguridad...).

Ante todo, no solo ha sido necesario analizar las características del entorno donde se ejecutarán las pruebas, sino que se desea sentar las bases para una ejecución futura en nodos sensores con recursos aún más limitados. En [39] se describen las características de varios sistemas operativos para nodos sensores. Estos sistemas operativos sirven como referencia en cuanto a las funcionalidades mínimas que se pueden esperar. Entre los principales sistemas operativos destacan *Contiki* [35] y *TinyOS* [87].

Existen trabajos previos que han desarrollado pilas de red para WSN, como en [36] donde se presentan dos pilas de red para TCP/IP, llamadas *IwIP* y *uIP*. Ambas están pensadas para nodos con pocos recursos. *IwIP* implementa más funcionalidades que *uIP*, siendo esta última muy ligera con un consumo de recursos mínimo. De hecho, *uIP* se utiliza en *Contiki*. En esta Tesis Doctoral, a pesar de que las características de los protocolos son diferentes, las propiedades no funcionales van a ser parecidas y se encuentran en un punto intermedio entre *IwIP* y *uIP*.

El código que se ejecuta en los nodos sensores está preparado para ser portado a otros sistemas operativos, especialmente los específicos para WSN nombrados anteriormente. Esto incluye tanto la pila de red como la aplicación de gestión de los nodos sensores. La pila de red también se ejecutará en el robot y la BS para no tener que hacer dos implementaciones distintas. El código se va a clasificar, según los requerimientos de recursos, como *código restringido* y *código no restringido*. En el *código restringido* se utilizan clases de abstracción/emulación que encapsulen las particularidades del sistema operativo. En el *código no restringido* no se ponen restricciones y se pueden aprovechar todas las facilidades proporcionadas por Linux.

De nada sirve hacer una arquitectura de seguridad si el código que la implementa presenta fallos de seguridad. Por ese motivo, se han utilizado herramientas para la detección de errores, y entre ellos los que puedan dar lugar a vulnerabilidades de seguridad. Se han utilizado las herramientas de análisis estático de código `cppcheck` [93], `flawfinder` [160] e `Infer` [17] y la herramienta de análisis dinámico `valgrind` [27].

Se supone que hay confianza entre las aplicaciones y la pila de red que se ejecuta en una misma entidad por lo que no es necesario disponer de mecanismos de protección de la memoria.

El NOS incorpora un sistema de registros altamente configurable, basado en `spdlog` [98]. `spdlog` es una biblioteca de registro muy rápida realizada en C++. Todos los registros llevan:

1. Un sello de tiempo con una precisión de un microsegundo.
2. Información de la gravedad del registro con los siguientes valores posibles: traza, depuración, información, aviso, error y error crítico.

3. Datos del emisor (nivel de red, enlace, conexión de transporte, etc.).
4. Datos del suceso o estadísticas.

Estos registros se utilizan durante los experimentos realizados para obtener datos deseados como estadísticas, contadores, estados, etc. Los registros se pueden desactivar selectivamente cuando no sean necesarios, ya que pueden afectar a la precisión del cálculo de la duración de los procesos.

Además de lo anteriormente comentado, la pila de red tiene las siguientes propiedades no funcionales:

1. Aplicaciones de usuario. La pila de red incluye un AL simplificado con soporte de una única aplicación, pero con posibilidad de ser sustituido fácilmente. A partir de este AL genérico se crean otros específicos para cada tipo de entidad acorde a los recursos disponibles.
2. Funcionamiento en tiempo real relajado. Aunque muchos de los S.O. para nodos sensores no soportan aplicaciones en tiempo real, se intentan utilizar, siempre que sea posible, mecanismos en tiempo real, aunque sin garantizar su cumplimiento en todas las situaciones. En Linux se utilizan las extensiones POSIX para tiempo real [66], encapsuladas en clases que permitan su portabilidad.
3. Monoprocesador y monotarea. Muchos nodos sensores disponen de microprocesadores con un solo núcleo y la multitarea suele ser costosa o con soporte limitado [39]. El modelo de programación es dirigido por eventos con entrada/salida asíncrona. El motor gestor de eventos permite el registro de nuevos eventos y sus procesadores asociados externos a la pila de red. Todos los procesadores de eventos tienen la misma prioridad y no se pueden interrumpir, pero se ha procurado que el tiempo de procesamiento sea pequeño para reducir la posibilidad de retrasar otros eventos que se deben ejecutar en tiempo real.
4. Ausencia de mecanismos de sincronización. La programación dirigida por eventos (*event-driven*) asíncrona permite el acceso a los recursos de manera serializada por lo que no ha sido necesario utilizar sincronización. Es posible utilizar la pila de red desde distintas tareas siempre que sea una única tarea la que envíe datos a la red o que se garantice que no haya más de una tarea utilizando una misma conexión DTLS, ya que la biblioteca `wolfssl` no lo soporta. Se ha garantizado que en las distintas entidades lo anterior siempre se cumple.
5. Uso preferente de memoria estática en vez de dinámica. En los S.O. para nodos sensores la gestión de la memoria dinámica suele ser costosa. Se evitan copias innecesarias de datos, utilizando operaciones de copia cero (*zero-copy*). Por ejemplo, si no se usa ningún modo con seguridad, solo es necesario un trozo de memoria estática del tamaño de la MTU para procesar los mensajes recibidos hasta el AL.
6. Uso del patrón de diseño software *observador* para realizar notificaciones. Esto permite un bajo acoplamiento entre los distintos niveles de abajo hacia arriba. Las clases que puedan notificar eventos admiten como máximo un observador, pudiendo los observadores formar posteriormente listas de observadores, pasándose la notificación entre ellos.

7. Publicación como software libre utilizando la versión 3 de la GPL [140].

El AL de los nodos sensores se considera *código restringido*. Se ha implementado con las mismas propiedades que la pila de red y solo permite el uso de una única aplicación por simplicidad, con la composición de las funcionalidades de usuario (envío de medidas periódicas) y las de gestión. Utiliza el mismo motor gestor de eventos de la pila de red por lo que no requiere usar nuevas tareas ni procesos. Tampoco realiza operaciones con el sistema de ficheros durante su funcionamiento.

El AL en la BS y el robot es *código no restringido*. En la BS sigue siendo monotarea y hace uso del motor gestor de eventos, pero usa más memoria (almacenamiento de información de la topología) y usa el sistema de ficheros para guardar/recuperar información.

En el robot se va a realizar el cálculo de la topología de red. La duración de este cálculo depende de la complejidad de la red y debe ser realizado en paralelo al funcionamiento de la pila de red. Por lo tanto, el AL del robot va a ser el más complejo y el que más recursos necesita de todos. Además de mayor consumo de memoria y uso del sistema de ficheros, requiere usar computación paralela.

4.3.2 Nivel de adaptación al nivel de enlace (LLAL)

Se ha contemplado el uso de los siguientes LL como punto de partida de la pila de red, sobre los cuales se situará el LLAL:

- IEEE 802.3 (*Ethernet*) y los protocolos inalámbricos de la familia IEEE 802.11 (*Wi-Fi*). Las tramas de IEEE 802.3 e IEEE 802.11 son diferentes, pero muchos campos son similares. Se puede usar la misma interfaz de Linux para recibir y enviar datos en ambas, salvo por el hecho de que en IEEE 802.3 los datos deben tener una longitud mínima de 46 bytes añadiendo relleno si es necesario. Se ha utilizado el valor 0x88B5 como campo *EtherType* que está registrado en la IANA con la descripción *«IEEE Std 802 - Local Experimental Ethertype»*.
- Enlace de datos virtual a través de Protocolo de Datagramas de Usuario (UDP, *User Datagram Protocol*). Aunque UDP es un protocolo de TL, su uso permite la ejecución en una misma máquina de múltiples entidades de red aunque no disponga de interfaces de red diferenciadas para cada una de ellas, ya que en este caso se puede utilizar una dirección IP junto con un puerto UDP como dirección MAC. Así, se puede probar la comunicación a través de interfaces de red virtuales como la interfaz *loopback* o a través de interfaces reales con equipos separados.

Estos LL darían lugar a las torres de protocolos simplificadas que aparecen en la Figura 4.7.

Ambos LL proporcionan las siguientes funcionalidades comunes:

1. Transferencia de información no fiable entre dos máquinas conectadas directamente (física o virtualmente).

2. Detección de tramas erróneas, sin medidas correctoras.

3. Uso de direcciones origen y destino en cada trama (con diferente formato).

Y el resto de requisitos de LL para la arquitectura deben ser proporcionados por el LLAL.

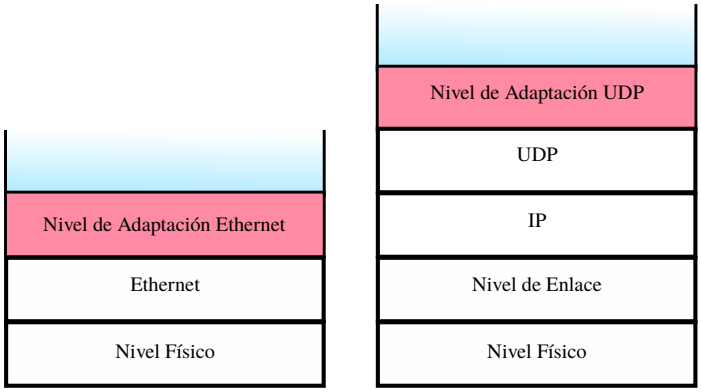


Figura 4.7 Niveles de adaptación para Ethernet y UDP.

En la Figura 4.8 se muestra un diagrama de las clases que forman parte del LLAL, el servicio de gestión y el servicio de seguridad en formato UML [109]. Solo se muestran las relaciones entre ellos y las relaciones con elementos de otros niveles se verá más adelante. Por brevedad, no se enumeran los atributos ni las operaciones de cada clase.

La clase principal del LLAL es `LinkLayer` que es, a todos los efectos, el LL virtual que usará el NL. Debe existir un único objeto de esta clase, es el contenedor del resto de objetos y ofrece los servicios de este nivel a niveles superiores.

La comunicación con el nivel inferior, LL, se realiza a través de objetos que implementan la interfaz `Connector` (conectores). Esta interfaz permite acceder a la funcionalidad común de los distintos LL soportados de una manera homogénea. El objeto `LinkLayer` debe contener al menos un conector, pero puede contener más. En este último caso, se pueden recibir tramas simultáneamente de todos ellos de manera asíncrona, pero solo se utilizará el primero para enviar tramas. En `LinkLayer` está el motor gestor de eventos asíncronos que se encarga de recibir todas las entradas que recibe la entidad (sin incluir ficheros) y las señales de vencimiento de temporizadores de manera eficiente utilizando una única tarea. Lo anterior se apoya en el uso de las funciones `epoll` (siempre que sea posible) o `pselect` de Linux. Se han creado 4 tipos diferentes de conectores:

1. `ConnectorUDP`: para utilizar UDP como LL. Asociado a una dirección IP y un puerto UDP. Para poder usar difusión con este tipo de conector es necesario transmitir a una dirección *multicast* y un puerto donde escuchan simultáneamente varias entidades.
2. `ConnectorEthernet`: para utilizar *Ethernet* o *Wi-Fi* como LL. Se asocia a una interfaz de red y utiliza su dirección MAC como dirección origen.
3. `ConnectorEthernetRaw`: es similar al anterior, pero puede transmitir tramas con direcciones origen MAC diferentes a la de la interfaz de red. La principal utilidad de este tipo de conector es poder hacer pruebas de ataques en los que se falsifique el origen de las tramas. La utilización de este tipo o el anterior es excluyente y se configura en compilación.
4. `ConnectorStdIO`: permite recibir datos de la entrada estándar. Este conector está pensado para recibir comandos administrativos y no para la comunicación con otras

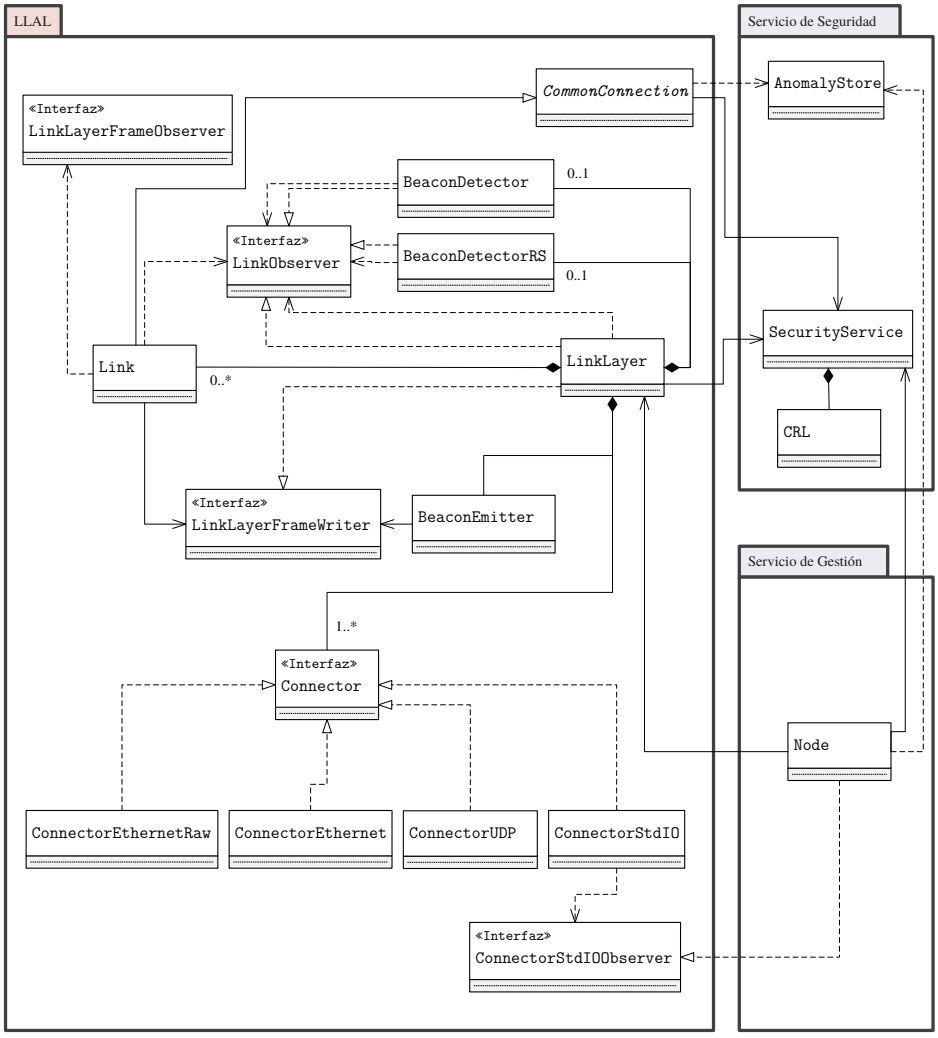


Figura 4.8 Diagrama de clases centrado en LLAL.

entidades. Cuando se recibe un comando administrativo por la entrada estándar se notifica a un observador del tipo `ConnectorStdIOObserver` pero estos datos no son procesados por `LinkLayer`. El servicio de gestión, a través de la clase `Node` está subscrito a estas notificaciones. El motivo por el que se ha hecho de esta manera es para no tener que crear nuevos hilos que se queden a la espera de recibir comandos administrativos, aprovechando así el motor gestor de eventos asíncronos.

En general, dado que cada conector utiliza un tipo distinto de direcciones MAC y diferentes MTU, las entidades no utilizan conectores de distintos tipos, a excepción de al menos un conector del tipo `ConnectorStdIO`.

La comunicación con los vecinos autorizados (enlaces) se realiza con objetos de clase `Link`. Los enlaces tienen mucha funcionalidad común con las conexiones del TL. La parte común se ha implementado en la clase abstracta `CommonConnection` de la que se derivan las conexiones en el LL y TL. En el código común se realiza el proceso de saludo y la aplicación del modo de seguridad elegido (ayudado por el servicio de seguridad cuando sea necesario), el mantenimiento de la conexión, los reintentos de conexión y la detección de anomalías. Los enlaces creados de manera automática con los robots requieren un tratamiento especial y se van a denominar *autolinks* (enlaces automáticos). El número máximo de enlaces se configura en compilación, mientras que el número de *autolinks* es configurable en inicialización.

La emisión y detección de balizas se realizan en clases separadas pero contenidas en `LinkLayer`. `BeaconEmitter` se encarga de la emisión de balizas cuando sea necesario. Para la detección de baliza básica se usa `BeaconDetector` y para la avanzada `BeaconDetectorRS`. Solo hay un único detector activo en un mismo instante.

Las notificaciones al nivel superior se hacen utilizando el patrón *observador* con dos interfaces:

1. `LinkLayerFrameObserver`: utilizado para notificar la recepción de datos. Los datos son procesados por el objeto `Link` (para desencriptarlos, autenticarlos, etc.) antes de pasarlos al nivel superior.

2. `LinkObserver`: para notificar la creación/destrucción de enlaces con entidades vecinas. Además del nivel superior, internamente, tanto `LinkLayer` como el detector de balizas utilizado están interesados en recibir estas notificaciones. El detector de balizas necesita conocer cuántos *autolinks* están en activo para determinar si debe seguir detectando balizas o no. Para que todos los elementos reciban la misma notificación, se crean listas de observadores que se van pasando las notificaciones. Y por tener que crear listas, tanto `LinkLayer` como el detector de balizas deben implementar esta interfaz y, a la vez, hacer uso de ella. Para enlaces normales la cadena es `LinkLayer`→nivel superior. Para *autolinks* la cadena es `BeaconDetector[RS]`→`LinkLayer`→nivel superior.

La interfaz `LinkLayerFrameWriter` se utiliza únicamente para desacoplar las clases `Link` y `BeacomEmitter` de la clase `LinkLayer`. Esta implementada por `LinkLayer` y la utilizan las clases que tienen que transmitir tramas a la red. El objeto de clase `LinkLayer` utilizará el primer conector configurado para transmitirla.

En cuanto al servicio de gestión, el único componente visible es la clase `Node`. Debe haber un único objeto de esta clase por entidad. Este objeto se encarga de iniciar/reiniciar/destruir los distintos niveles y el servicio de seguridad. Además, a partir de él se puede acceder de manera directa o indirecta a todo el sistema operativo de red. Los comandos administrativos, leídos de la entrada estándar y notificados a través de la interfaz `ConnectorStdIOObserver`, son procesados por él directamente (comandos comunes a todas las entidades) o son pasados a la aplicación de gestión para que esta los procese (comandos particularizados para cada tipo de entidad).

El servicio de seguridad realiza todas las funciones descritas en la Sección 3.3.1.1 con 3 clases:

1. `AnomalyStore`: su funcionalidad es almacenar anomalías (tanto de la misma entidad como de otras), su posterior recuperación para ser enviadas y su borrado tras la confirmación de recepción.
2. `CRL`: como su nombre indica almacena los números de serie de certificados revocados, además de ofrecer métodos para comprobar la existencia de un número de serie y la posibilidad de importar números de serie desde ficheros. Cada vez que se inicia un enlace seguro o se detecta una baliza se verifica que el número de serie del certificado no está revocado.
3. `SecurityService`: realiza el resto de funciones particularizadas para el uso de DTLS para asegurar los datos y ECC para la autenticación e intercambio de claves. El formato de los datos con seguridad añadida, el proceso de saludo avanzado, los algoritmos soportados y otros mecanismos de seguridad están descritos en [125]. Se ha utilizado la biblioteca `wolfssl` [162] como implementación de DTLS. El equivalente al saludo avanzado de reanudación recibe el nombre de *Session-Resuming Handshake*, que es utilizado también en TLS y está definido en [31], y se basa en que el servidor asigna un identificador de sesión a cada conexión y lo comunica al cliente. Esta clase también contiene una caché de sesiones asociadas a identificadores de otras entidades. Cuando se establece una conexión se comprueba esta caché para usar el saludo avanzado de reanudación cuando sea posible.

Hay que resaltar que las únicas clases dependientes de la biblioteca `wolfssl` son *`CommonConnection`* y `SecurityService`. El reemplazo del protocolo o la biblioteca utilizada sería relativamente sencillo.

4.3.2.1 Formato de tramas y balizas

El formato de los datos cuando se usan los modos con seguridad en LL es similar al del protocolo DTLS. Cuando se usa UDP como LL subyacente, no se añade ninguna cabecera adicional. Si se usa *Ethernet* es necesario añadir un campo a la cabecera `Ethernet` de tamaño 2 octetos con la longitud de los datos codificada en binario. No es necesario indicar si la trama va protegida o no. Según la arquitectura SNSR, los vecinos deben conocen el modo de seguridad que hay que aplicar antes de iniciar la comunicación.

Una SDU desde o para el NL, nunca puede empezar con el primer octeto con todos los bits a cero (más adelante se verá que este octeto indica el límite de saltos). La anterior propiedad se utiliza para distinguir las tramas de datos de las de control. Todas las tramas de control son similares a las tramas de datos a excepción de que o solo contienen un octeto o el primer octeto de datos en claro es 0. Existen tres tipos de tramas de control:

1. Tramas de saludo. En el saludo básico es una trama con un único octeto de datos de valor 1. En el saludo avanzado se usan los formatos definidos en el protocolo DTLS (existen varios).
2. Tramas de desconexión. Si se ha usado el saludo básico es una trama con un único octeto de datos de valor 2. En el saludo avanzado se usan los formatos definidos en el protocolo DTLS.
3. Tramas de pulso o *keepalive*. Similar a las tramas de saludo básico pero con valor 0.

4. Balizas. Contiene los campos descritos en (3.12) particularizados, con el formato mostrado en la Figura 4.9.

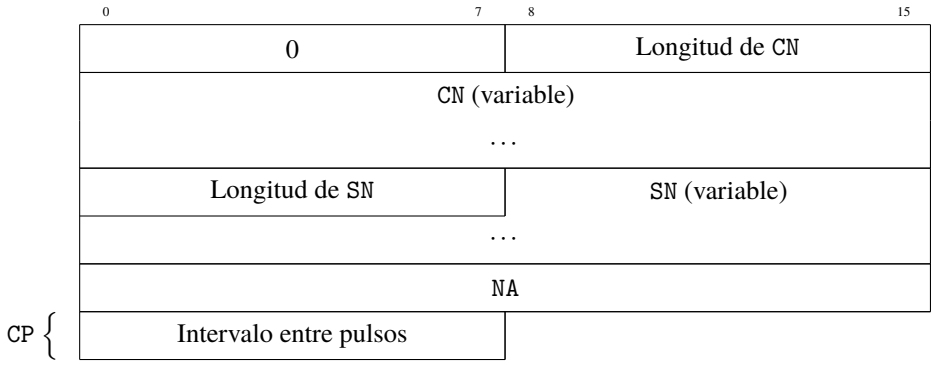


Figura 4.9 Formato de una baliza en la implementación.

Se están utilizando certificados X.509 [14], así que la longitud del campo `CN` está limitada a 64 octetos y la del campo `SN` a 20 octetos. El campo `CP` (*communication parameters*) está formado por un único subcampo con el valor del intervalo entre envíos de pulsos en segundos para mantener la conexión una vez establecida.

4.3.3 Nivel de red (NL)

En este nivel se ha implementado la parte local del enrutamiento descrito en la Sección 4.1.6, que es un enrutamiento estático basado en tablas de encaminamiento con un modo *normal* y otro *alternativo*. En la Figura 4.10 se muestra un diagrama de clases UML centrado en el NL y sus relaciones con el nivel inferior y el servicio de gestión. No se hace uso del servicio de seguridad.

La clase principal es `NetworkLayer`. Esta clase ofrece todos los servicios de este nivel a niveles superiores y utiliza los servicios de `LinkLayer`. Para recibir notificaciones del LLAL, implementa las interfaces `LinkLayerFrameObserver` (recepción de paquetes) y `LinkObserver` (creación/destrucción de enlaces). Cada enlace (objeto `Link`) en este nivel se corresponde a un vecino de red codificado como un objeto de la clase `NetworkNeighbor`, que no es más que la asociación de una dirección de red a un objeto de tipo `Link`. Si se desconoce la dirección de red real de un vecino se utiliza el valor 0 (dirección de enlace local). También se admite la modificación posterior de la dirección de red. Existirán tantos objetos `NetworkNeighbor` como objetos `Link` existan y su número máximo es el mismo. El nivel superior puede enviar datos de tres maneras: a una determinada dirección de red, directamente a un vecino usando la dirección de enlace local o por difusión.

La tabla de enrutamiento, implementada en la clase `RadixTreeRoutingTable`, está basada en una estructura de datos dinámica conocida como árbol compacto binario de prefijos (*binary radix tree* o *binary radix trie*) que permite hacer búsquedas del prefijo más largo coincidente muy eficientes tanto en velocidad como en cantidad de memoria

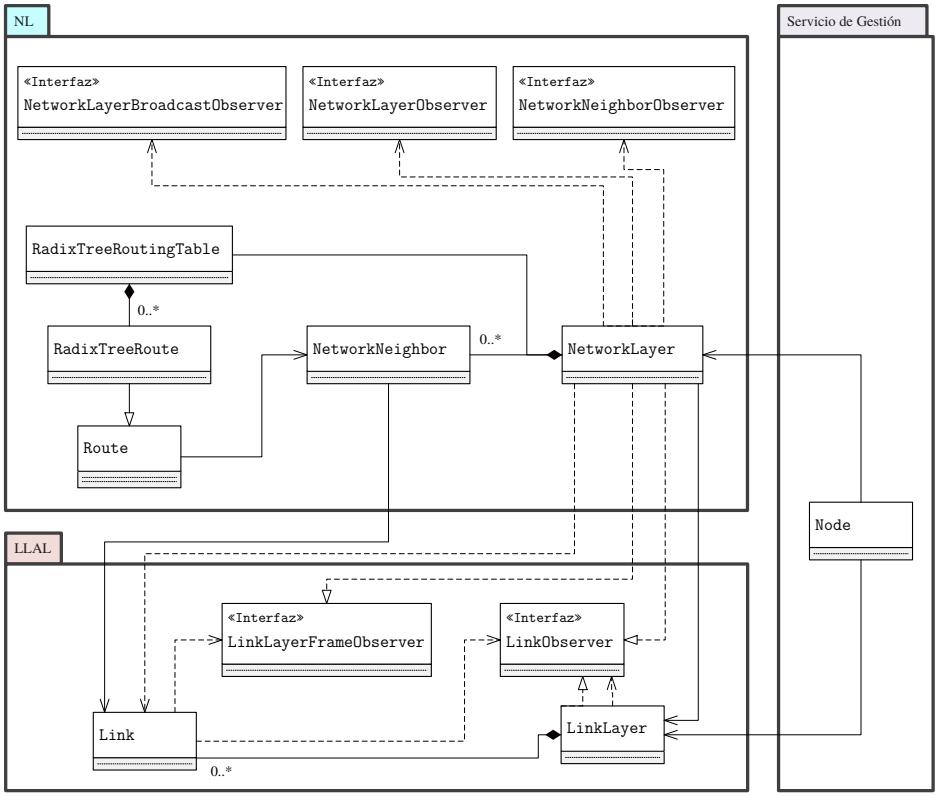


Figura 4.10 Diagrama de clases centrado en NL.

utilizada. Este tipo de estructuras o variantes de ella ya se han utilizado anteriormente para este problema, como se describe en [139, 106, 165]. Las rutas que están en esta tabla, están almacenadas en objetos de clase `Route`. Las rutas alternativas a una misma subred destino se almacenan en un mismo objeto `Route` ordenadas de mayor a menor preferencia. Cada ruta alternativa contiene datos del vecino que se debe usar como siguiente salto, la métrica asociada (a menor métrica, más preferencia), si es una ruta estática (no se ha creado automáticamente), si es directa a un vecino y si está habilitada o no (una ruta puede deshabilitarse temporalmente). Las rutas estáticas se pueden crear durante la inicialización del nodo o al recibir una nueva configuración. A través del objeto `NetworkNeighbor` de una ruta también se puede consultar si el enlace asociado está preparado para enviar datos. La clase `RadixTreeRoute` es una clase derivada de `Route` con extensiones para que una ruta se pueda almacenar en el árbol compacto binario de prefijos.

Cuando se recibe un nuevo paquete, ya sea para el nivel superior o para reenviarlo a otra entidad, en `NetworkLayer` se realizan comprobaciones para evitar la suplantación de direcciones, sobre todo las de los vecinos en los modos de seguridad `SEG_TE` y `SEG_LTE` (seguridad en TL excepto en vecinos). Entre estas comprobaciones se incluyen:

- El uso de la dirección de enlace local (0) es obligatorio en los paquetes enviados entre

vecinos. Esto es transparente al nivel superior, que siempre trabaja con direcciones globales cuando son conocidas. `NetworkLayer` se encarga de hacer las sustituciones oportunas en ambos sentidos de manera automática. Cualquier mensaje de un vecino que no cumpla lo anterior es descartado. Los mensajes con direcciones origen y destino 0 nunca se reenvían.

- No se permite el enrutamiento *alternativo* con vecinos. Es decir, nunca se reenviará un paquete con la dirección de la entidad como origen y dirección destino de un vecino porque no pueden producirse bucles.

El NL ofrece 3 conjuntos de notificaciones a los niveles usuarios, accesibles mediante la implementación de 3 interfaces diferentes de tipo *observador*:

1. `NetworkLayerBroadcastObserver`, para la recepción de paquetes de difusión.
2. `NetworkLayerObserver`, para la recepción de datos destinados a la entidad y la notificación de la eliminación de un vecino de red.
3. `NetworkNeighborObserver`, que como la anterior permite notificar la eliminación de un vecino, pero también la creación o modificación.

La difusión sigue los principios descritos en la Sección 3.3.1.4. Se utiliza la dirección 0 como dirección destino y se añade una marca de difusión en la cabecera del paquete. La difusión es por inundación, que implica enviar el mismo paquete a cada uno de los vecinos y estos a su vez lo reenvían a sus vecinos. Pero este reenvío no es automático en el NL, sino que solo se notifica al observador de clase `NetworkLayerBroadcastObserver`. Este debe verificar la autenticidad según el valor del parámetro `BVT` (ver Tabla 3.15) y solicitar la redifusión según el resultado.

Para terminar, el objecto de clase `Node` del servicio de gestión crea y tiene acceso también al objeto `NetworkLayer`.

4.3.3.1 Formato de paquetes

Los paquetes presentan el formato mostrado en la Figura 4.11. Los paquetes están compuestos por una sencilla cabecera de tan solo 6 octetos seguida de los datos del nivel superior. La MTU en este nivel solo es 6 octetos menor que en el LL. Solo hay un único formato de paquete. Los campos de la cabecera están descritos en Tabla 4.1

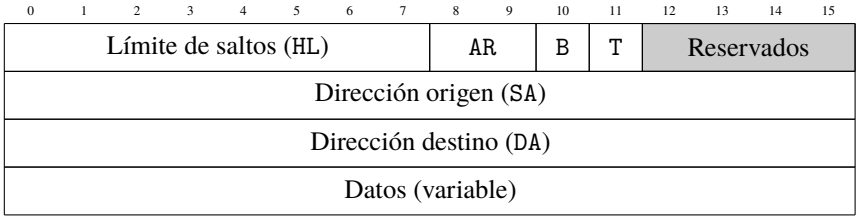


Figura 4.11 Formato de un paquete de red.

El bit `T` de traza se ha utilizado durante el desarrollo para verificar la ruta seguida hacia un destino. Cuando se recibe un paquete con la marca de traza se anotan sus datos en el

Tabla 4.1 Campos de la cabecera de un paquete.

Campo	Descripción
`HL`: límite de saltos (*Hop Limit*)	Indica el número máximo de saltos permitidos. Se va decrementando en una unidad cada vez que un paquete es reenviado. El paquete se descarta al llegar este campo a 0 en general o 1 si el siguiente salto no es el destino (con certeza se va a descartar y se evita una transmisión).
`AR`: ruta alternativa (*Alternative Route*)	Indica que hay que aplicar enrutamiento *alternativo* a este paquete. El valor indica el número de veces que el paquete ha vuelto al origen, con un valor máximo 3.
`B`: Difusión (*Broadcast*)	Indica que el paquete es de difusión. En estos paquetes el campo `DA` deben valer 0.
`T`: Traza (*Trace*)	Bit usado para trazar la ruta que sigue un paquete. No se debe usar en una red real por motivos de rendimiento.
Reservados	Bits reservados para un posible uso futuro. Actualmente deben ser 0.
`SA`: Dirección origen (*Source Address*)	Es la dirección de red del emisor. Si el paquete va destinado a un vecino, este campo y `DA` deben ser 0. Nunca se reenvían paquetes con estos dos campos a 0.
`DA`: Dirección destino (*Destination Address*)	Es la dirección de red de la entidad destino.

registro. Posteriormente se pueden analizar los registros para ver la ruta. Sin embargo, en un entorno de producción no es necesario y podría abusarse de este comportamiento, así que, por defecto, el uso del bit `T` está desactivado en compilación (se considera parte de los bits reservados).

4.3.4 Nivel de transporte (TL)

Este nivel cumple todos los requisitos descritos en la Sección 3.3.1.5. Se ha creado un nuevo protocolo de comunicación entre entidades remotas mediante conexiones que tiene las siguientes propiedades:

- Las conexiones están orientadas a mensajes, similar a UDP y Protocolo de Transmisión de Control de Flujo (SCTP, *Stream Control Transmission Protocol*) [143], y no como TCP.
- Según el modo de seguridad elegido, se añade seguridad o no a la conexión. Como se vio en la Tabla 3.2, existen tres modos de seguridad en el TL: sin seguridad, seguridad en todas las conexiones o seguridad en todas las conexiones excepto

con entidades vecinas. Al aplicar la seguridad también se protege la cabecera del segmento.

- Utilización del mismo proceso de saludo y mantenimiento de la conexión que en el LL.
- Una misma conexión transporta simultáneamente dos flujos de datos en cada dirección: un flujo no fiable y otro fiable. Lo equivalente en una red TCP/IP, desde el punto de vista de la fiabilidad, sería unificar en un mismo protocolo UDP y TCP.
- Solo se permite una única conexión entre 2 entidades, y, por tanto, no es necesario utilizar direcciones de transporte (o puertos). Una conexión puede estar asociada a un vecino o a la dirección de red del otro extremo. Si está asociada a un vecino, siempre se usa la dirección de enlace local (0) por lo que dos vecinos pueden establecer una conexión sin conocer sus direcciones de red.
- Segmentación y reensamblado automático de mensajes de tamaño superior a la MTU de una conexión. En el TL es posible tener conexiones con seguridad o no por lo que es posible que distintas conexiones tengan distintas MTU.

El transporte de mensajes no fiables no ofrece ninguna garantía de entrega u orden. Los fragmentos de mensajes fragmentados pueden llegar desordenados, pero la pérdida de uno de ellos hace que el mensaje no se entregue. Solo se permite un mensaje fragmentado pendiente de reensamblar y la llegada del primer fragmento de otro mensaje provoca el descarte del anterior. Tampoco se utiliza ningún campo de detección o corrección de errores y se supone que los niveles inferiores ya han realizado esta labor.

El transporte de mensajes fiables adapta funcionalidades típicas de otros protocolos fiables, en especial de TCP y QUIC[1] [73] para proporcionar:

- Entrega garantizada de los mensajes, utilizando asentimientos normales (ACK) y selectivos (SACK). La implementación de asentimientos selectivos se basa en la implementación usada en TCP descrita en la RFC 2018 [44] y sobre todo en la usada por QUIC [72].
- Entrega ordenada usando recursos del emisor. Un emisor no transmitirá un nuevo mensaje hasta haber recibido asentimiento del anterior. Esto reduce la necesidad de memoria en el receptor y evita ataques que intenten agotarla. El receptor sí realiza la ordenación de fragmentos en mensajes fragmentados.
- Control de flujo basado en ventana deslizante de tamaño fijo (configurable mediante parámetro de compilación).
- Control de congestión y recuperación ante errores basado en la medición del RTT y temporizadores dinámicos. Está basado en parte en los mecanismos usados por QUIC [72].

En la Figura 4.12 se muestra un diagrama de clases UML centrado en el TL y sus relaciones con niveles inferiores, el servicio de gestión y el servicio de seguridad. Si se compara la Figura 4.12 con la Figura 4.8, puede verse que las relaciones con el servicio de seguridad son prácticamente idénticas.

[1] QUIC no es un acrónimo, sino el nombre del protocolo.

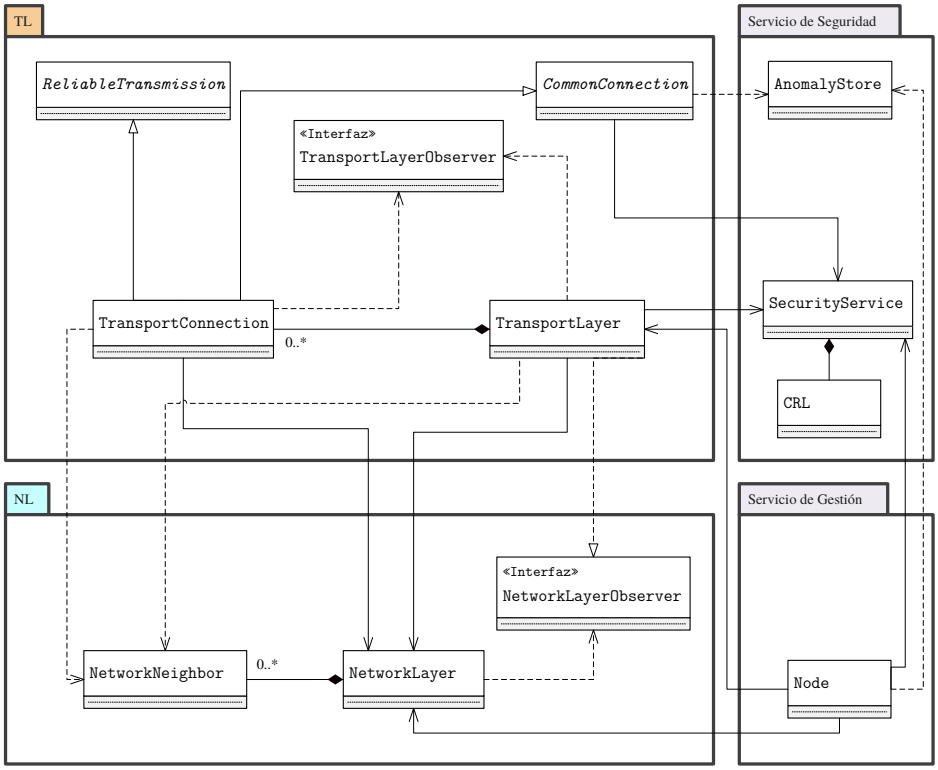


Figura 4.12 Diagrama de clases centrado en TL.

Los objetos de la clase `TransportConnection` implementan las conexiones de transporte antes descritas. Esta clase es la equivalente a la `Link` del LLAL. Hereda de las clases abstractas *`CommonConnection`* y *`ReliableTransmission`*. *`CommonConnection`* se comparte con el LLAL (realmente no pertenece a esos paquetes pero, por simplicidad, se ha representado así en la figura). En *`CommonConnection`* se desarrollan las tareas asociadas con la seguridad, el proceso de saludo, el mantenimiento de la conexión y detección de anomalías. En *`ReliableTransmission`* está implementada la gestión del flujo fiable de la conexión. El resto de funciones, incluyendo la gestión del flujo no fiable, están en la propia clase `TransportConnection`. El envío de segmentos de transporte se hace directamente usando el objeto `NetworkLayer` y si una conexión está asociada a un vecino se accede a los datos de este en el constructor. Los objetos de esta clase pueden ser utilizados directamente desde el nivel superior para enviar mensajes, aunque la recepción se hará mediante notificaciones siguiendo el patrón *observador*. `TransportConnection` también proporciona métodos para generar identificadores únicos que los niveles superiores pueden añadir a los mensajes transmitidos en la conexión (por ejemplo, el valor *TX_ID* asociado a las peticiones/respuestas del nivel de aplicación descrito en la arquitectura) y comprobar el último transmitido.

La clase principal de este nivel es `TransportLayer`. Se encarga de gestionar las conexiones de transportes (creación, modificación, búsqueda y eliminación) y de procesar notificaciones sobre la eliminación de un vecino y la llegada de segmentos, implementando la interfaz `NetworkLayerObserver` del NL. Las conexiones de transporte asociadas a un vecino eliminado deben ser eliminadas también. Los segmentos entrantes son distribuidos a sus correspondientes conexiones. Por configuración en la inicialización se puede especificar el comportamiento deseado cuando llegue un segmento que no pertenezca a ninguna conexión y sea un saludo: descartar o crear una nueva conexión de manera automática (según dirección origen o cualquiera).

Las notificaciones al nivel superior se realizan a través de la interfaz `TransportLayerObserver`. Se producen notificaciones cuando:

- Una nueva conexión de transporte ha sido establecida y el proceso de saludo ha terminado con éxito.
- Una conexión se ha cerrado.
- Una conexión va a ser eliminada. El nivel superior puede abortar la eliminación y reiniciar la conexión.
- Un nuevo mensaje entrante ha llegado.
- Se desea verificar la identidad asociada a una dirección de red.

Para terminar, el objecto de clase `Node` crea y tiene acceso también al objeto `TransportLayer`.

4.3.4.1 Formato de los segmentos

Existen 5 tipos de segmentos:

1. De flujo no fiable.
2. De flujo no fiable fragmentado.
3. De flujo fiable.
4. De flujo fiable no fragmentado.
5. De asentimiento selectivo (SACK).

La cabecera de los segmentos se muestra en la Figura 4.13. Todos los tipos comparten un primer octeto común al que se le añaden campos opcionales según el tipo de segmento. La descripción de los campos está recogida en la Tabla 4.2.

En una transmisión fiable se ha seguido la política de marcar cada segmento transmitido con un número de secuencia (`SEQ`) único aunque sea una retransmisión. Esto hace el cálculo del RTT más sencillo, ya que no hay ambigüedad en el caso de segmentos retransmitidos cuando llega el asentimiento. Sin embargo, exige otro número de secuencia (`ORD`) que permita distinguir mensajes duplicados e identificar fragmentos de un mismo mensaje, cuyo rango no necesita ser amplio porque el emisor no envía un nuevo mensaje hasta recibir el asentimiento de otro (pero aun así pueden llegar duplicados en distinto orden). El valor de `SEQ` se inicia con un número aleatorio, se incrementa en una unidad y puede desbordarse

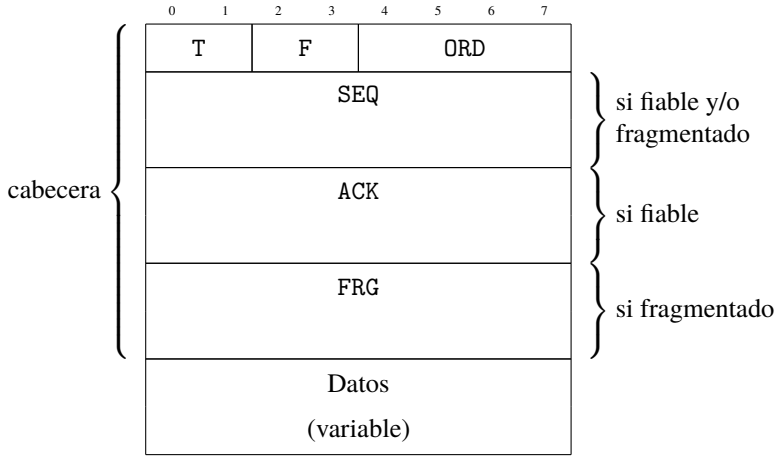


Figura 4.13 Formato de un segmento.

Tabla 4.2 Campos de la cabecera de un segmento.

Campo	Descripción
T	Tipo de flujo (fiable o no) y asentimiento. Los posibles valores se pueden consultar en la Tabla 4.3 (izda.).
F	Indicación de fragmento y posición (primero u otro). Los posibles valores se pueden consultar en la Tabla 4.3 (dcha.).
ORD	Número de orden de un segmento del flujo fiable o 0 si el flujo no es fiable. En caso de fragmentación, todos los fragmentos del mismo mensaje llevarán el mismo número de orden. Permite distinguir mensajes diferentes.
SEQ	Número de secuencia de 16 bits. En un flujo fiable todos los segmentos llevan uno diferente, aunque sean retransmisiones, que se utiliza posteriormente en los asentimientos. En un flujo no fiable tiene el mismo uso que el campo ORD, pero con mayor rango.
ACK	Asentimiento del número de secuencia del último segmento que se ha recibido.
FRG	Número de fragmento. En el primer fragmento de un mensaje es el número de fragmento del último (número de fragmentos menos uno). En otro caso es el número de fragmento actual. Nunca puede valer 0.

aunque nunca puede valer 0. Un número de secuencia $\texttt{SEQ}_x$ es posterior a otro $\texttt{SEQ}_y$ si se verifica que, usando una resta sin signo con desbordamiento: $\texttt{SEQ}_x - \texttt{SEQ}_y < 2^{15}$. El campo ORD en cambio empieza en 0, puede valer 0 y también admite desbordamiento.

En una transmisión no fiable no es suficiente el campo ORD para distinguir mensajes

Tabla 4.3 Valores de los campos `T` y `F` de un segmento.

`T`	Significado
`00`	Error
`01`	No fiable
`10`	Fiable
`11`	Fiable SACK

`F`	Significado
`00`	No es un fragmento
`01`	Primer fragmento
`10`	Otro fragmento
`11`	Reservado

diferentes porque el número de segmentos perdidos puede ser muy superior a 7 (diferencia máxima entre dos segmentos para identificar que uno es posterior a otro usando el campo `ORD`).

El campo `FRG` a pesar de ser de 16 bits puede limitarse en compilación, de tal manera que el número máximo de fragmentos esté predefinido. Esto tiene implicaciones en la cantidad de memoria necesaria para almacenar temporalmente los fragmentos y en el tamaño máximo de un mensaje que puede transmitirse.

En el flujo fiable, todos los segmentos llevan el campo `ACK` con el número de secuencia del último segmento recibido en orden, que se utiliza como un asentimiento implícito. Si no hay datos que transmitir y se quiere asentir un único segmento del flujo fiable, puede transmitirse un segmento sin datos. Si se tienen que asentir varios segmentos se emplea un segmento SACK. Un segmento SACK tiene la misma cabecera que un segmento fiable no fragmentado y el contenido de los datos son todos los números de secuencia que se van a asentir con el formato compacto que se muestra en la Figura 4.14.

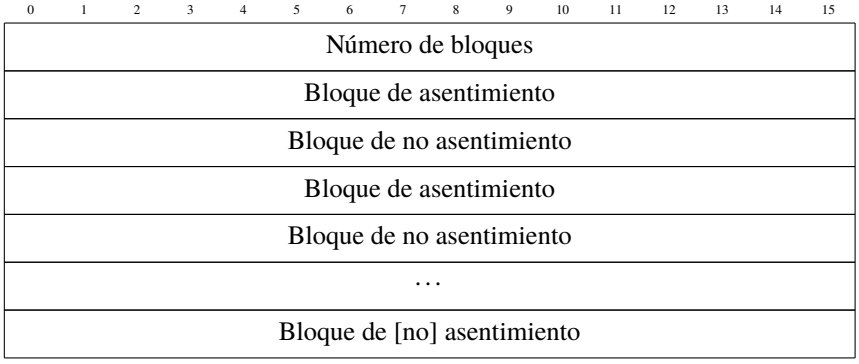


Figura 4.14 Formato de los datos de un segmento SACK.

Los datos de un segmento SACK están formados por una secuencia de bloques de asentimiento ($\texttt{B}_{ACK}$) y no asentimiento ($\texttt{B}_{NACK}$) que se van alternando precedida del número de bloques totales. Todos los campos son enteros sin signo de 16 bits y el número máximo de bloques está determinado por la MTU de este nivel, ya que un segmento SACK no puede fragmentarse. El número mínimo de bloques es 1, es decir, debe existir al menos

el primer bloque de asentimiento.

Un bloque de asentimiento indica cuantos segmentos continuos se están asintiendo que son precedentes al de mayor número de secuencia asentido determinado por el bloque anterior (bloque de no asentimiento) o campo `ACK` en caso del primer bloque. El número de secuencia del segmento menor que se está asintiendo se puede calcular como:

$$\mathtt{SEQ}^{i}_{min} = \mathtt{SEQ}^{i-1}_{max} - \mathtt{B}^{i}_{ACK} \tag{4.3}$$

Un bloque de no asentimiento indica el número de segmentos continuos que no se están asintiendo que preceden al segmento con número de secuencia una unidad menor que el menor que se asentía en el bloque de asentimiento previo. Por tanto, el número de segmentos no asentidos es uno más del valor indicado por el bloque. Este bloque implica el asentimiento del anterior segmento a este bloque con número de secuencia que se calcula como:

$$\mathtt{SEQ}^{i}_{max} = \mathtt{SEQ}^{i-1}_{min} - \mathtt{B}^{i}_{NACK} - 2 \tag{4.4}$$

Un receptor guardará los números de secuencia de los segmentos recibidos hasta que el emisor asienta el último asentimiento normal o selectivo, de tal manera que es posible que un mismo número de secuencia esté contenido en varios asentimientos enviados.

4.3.5 Nivel de aplicación (AL) y aplicaciones de gestión

A diferencia del resto de niveles, el AL es diferente en cada tipo de entidad, aunque algunas funciones son similares. Otra novedad es que los mensajes enviados a excepción del *PING*, es decir, los que se describieron en el Capítulo 3, usan para la codificación *Protocol Buffers* [53] y no una codificación binaria específica como la de otros niveles.

En la Figura 4.15 se muestra un diagrama de clases UML centrado en el AL y sus relaciones con niveles inferiores, el servicio de gestión y el servicio de seguridad. En el diagrama solo se muestran las clases más representativas del AL y solo las clases de otros niveles que tienen relación con las anteriores.

A nivel de paquetes UML, en la Figura 4.15 además del AL aparecen dos nuevos paquetes UML que son utilizados por el AL:

1. El paquete `NetworkData` incluye todas las clases necesarias para almacenar y transmitir la información de la topología de la red, con los mismos nombres que aparecieron en la Sección 3.4 y con las particularidades descritas al comienzo de este capítulo. La clase principal que contiene a todas las demás se llama también `NetworkData`. Todas las clases de este paquete usan para la serialización *Protocol Buffers*.

2. El segundo paquete contiene las clases necesarias para la codificación/decodificación de los mensajes de gestión descritos en el Capítulo 3. Igual que antes, se usa *Protocol Buffers*. Los mensajes transmitidos punto a punto son de la clase `SnsMessage` y contienen uno de los tipos de mensajes definidos (existe una clase por tipo). Esta clase es la encargada de añadir el campo tipo de mensaje (*TM*) que permite distinguir los diferentes mensajes. Los mensajes recibidos por difusión utilizan otra clase llamada `SnsBroadcastMessage` (no aparece en la figura), que incluye los campos necesarios para realizar la verificación de autenticidad. Los mensajes de gestión

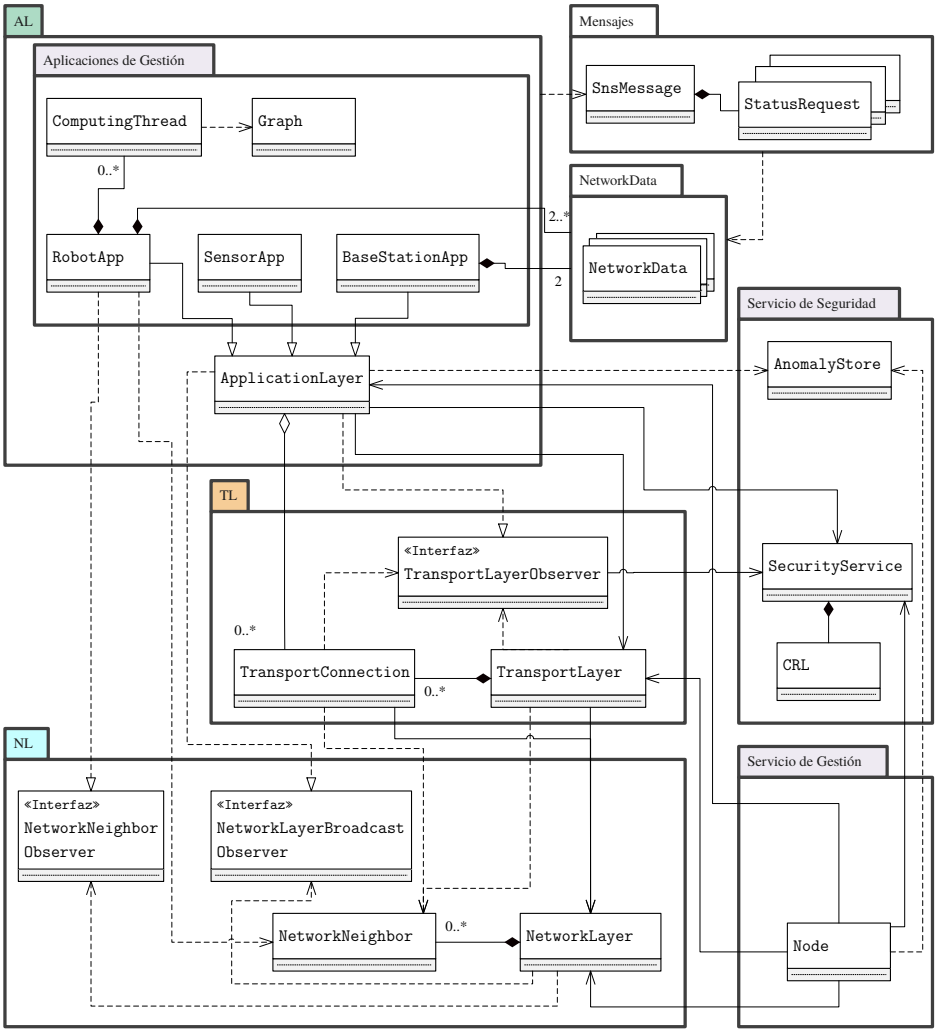


Figura 4.15 Diagrama de clases centrado en AL.

pueden transportar información de la topología, por lo que utilizan las clases definidas en el paquete `NetworkData`.

A nivel interno del AL, existe una implementación genérica de AL y una implementación específica para cada tipo de entidad. `ApplicationLayer` es la genérica que sirve como base a las implementaciones específicas, que heredan de ella. Esta clase base realiza todas las funciones comunes, que son las siguientes:

- Registro de datos de estaciones base asociadas a la entidad. Aunque en el Capítulo 3 se describe la arquitectura para una única BS, el código está preparado para poder asociar varias aunque solo se utilice una en un determinado instante, manteniendo

el resto como reserva en caso de fallo. Se mantiene una referencia a la conexión de transporte con cada BS para acceder rápidamente a ella y el *TX_ID* del último mensaje de difusión recibido para descartar mensajes duplicados.

- Recepción de datos procedentes del TL. Para ello implementa la interfaz `TransportLayerObserver` y se registra como observador en `TransportLayer` (que a su vez pasa la referencia a las distintas conexiones `TransportConnection`). El TL solo admite un único observador, de tal manera que, si existieran varias aplicaciones, la distinción de la aplicación destino de un mensaje se realizaría en el AL. En las distintas especializaciones de AL también existe una única aplicación, la aplicación de gestión, que puede también funcionar como aplicación de usuario básica en los nodos como se ha comentado antes.
- Comprobación de la correspondencia entre un mensaje de petición y un mensaje de respuesta analizando el campo *TX_ID* de ambos.
- Envío, recepción, verificación y redifusión de mensajes de difusión. Para la recepción de mensajes de difusión es necesario implementar la interfaz `NetworkLayerBroadcastObserver` del NL. Si el tipo de verificación configurado requiere el uso o comprobación de firmas digitales en los mensajes, se usan los servicios de `SecurityService`. El tratamiento del mensaje recibido debe ser implementado en la clase derivada.
- Operativa de transmisión de anomalías y sus asentimientos. El AL genérico no incluye la lógica para iniciar estas operaciones de manera automática, sino que es el AL específico el que la aporta. Cuando se genera un mensaje `TM_ANOMALY_STATUS` se utilizan las anomalías más antiguas no asentidas con un máximo tal que el mensaje resultante no se fragmente en el TL.
- Mantenimiento de la comunicación con la BS, detección de cierre de la conexión con esta y, de manera automática, selección de otra BS como principal si hay otra disponible o reconexión de la actual si no.
- Envío de *PING* y respuesta automática a ellos, así como realización de pruebas de transferencia de datos.
- Y otros métodos de utilidad y una estructura básica para que las clases derivadas puedan particularizar el procesamiento de mensajes y otras notificaciones.

Esta implementación genérica se puede emplear para probar la pila de red, y el programa `Node` permite seleccionarla como tipo de AL a ejecutar. Sin embargo, no se puede utilizar en la arquitectura SNSR, ya que no respondería a ningún mensaje de gestión.

El funcionamiento diferencial de los tres tipos de entidades se realiza en el paquete de aplicaciones de gestión, en sendas clases derivadas de `ApplicationLayer`. Estas clases son:

1. `SensorApp`. Es la aplicación de gestión utilizada en los nodos sensores y, además, realiza las funciones de simulación de aplicación de usuario habitual de recogida y envío de medidas a la BS a intervalos configurables utilizando mensajes *PING*, que son automáticamente respondidos por la BS. El funcionamiento como aplicación de usuario se inicia automáticamente cuando se establece la conexión con la BS.

2. `BaseStationApp`. Es la aplicación de gestión de la BS. Almacena dos copias de la información de la topología de la red (la inicial y la actual) usando la clase `NetworkData`. Hay que destacar también que es la única aplicación que es capaz de enviar por difusión y está preparada para aceptar conexiones de transporte desde cualquier origen, aunque verifica en modo seguro que los extremos usan identificadores y direcciones de red correctos según la información de la topología. Si las conexiones de transporte no incorporan seguridad, se utiliza un mensaje *PING* para obtener la identidad del otro extremo.

3. `RobotApp`. La aplicación de gestión del robot es la más compleja. El robot permite la conexión de cualquier entidad y la aplicación de gestión debe tener conocimiento de todos los cambios que se produzcan en su vecindario, sobre todo en las fases de *descubrimiento* y *configuración*. Para esto implementa la interfaz `NetworkNeighborObserver` del NL. Al igual que la BS requiere tener un mínimo de dos copias de `NetworkData` (inicial y actual) pero en esta implementación es el robot quien calcula la nueva topología de la red, así que puede necesitar copias temporales auxiliares. El cálculo de la topología de la red se hace en paralelo usando objetos de la clase `ComputingThread`, donde la red se representa como grafos (codificados como objetos `Graph`).

Por supuesto, las aplicaciones de gestión anteriores cumplen las especificaciones y el comportamiento descrito en la arquitectura SNSR. El servicio de gestión (objeto `Node`), como con el resto de niveles, también crea, configura y tiene acceso al objeto `ApplicationLayer` o cualquiera de sus derivados.

La lista de comandos administrativos soportados por una entidad dependerá de la variante del AL utilizada. El objeto `Node` es capaz de ejecutar un conjunto de comandos administrativos comunes a todas las entidades. En el Código 4.2, que muestra la ayuda proporcionada al ejecutar el comando `help`, estos comandos comunes aparecen en el apartado *"Node commands"*. Los comandos no reconocidos se pasan a un método del `AL` para que los procese. En consecuencia, el `AL` puede añadir comandos administrativos adicionales. Los comandos adiciones de cada tipo de entidad se muestran también en el Código 4.2 (realmente la ayuda solo mostraría los comandos válidos para una entidad). Cada comando administrativo puede devolver una respuesta en forma de cadena de texto. Por defecto, la respuesta se guarda en el registro del programa, pero se puede solicitar que se envíe a un fichero añadiendo el prefijo `<fichero>:` al nombre del comando o a la salida estándar añadiendo el prefijo `:`.

Tanto en `BaseStationApp` como en `RobotApp` la información de la topología almacenada en los objetos `NetworkData` puede ser leída de un fichero y guardada en otro, de tal manera que se pueden cargar datos ya calculados o analizar los cálculos realizados.

El programa `Node` solo permite ejecutar una implementación de AL simultáneamente y en compilación se puede elegir entre permitir al usuario escoger el tipo de entidad durante la configuración inicial o forzar a utilizar solo `SensorApp`, para reducir el tamaño del ejecutable resultante que se utiliza en los nodos sensores.

4.3.5.1 Formato de los mensajes

Los mensajes de unidifusión siguen el formato mostrando en la Figura 3.11, utilizando 8 bits para el campo *TM*. El valor de este campo que se usa en los diferentes tipos de mensajes aparece en la Tabla 4.4. Los mensajes de gestión se serializan con *Protocol Buffers* y los valores de sus campos *TM* están determinados en parte por la manera en que *Protocol Buffers* codifica los identificadores y tipos de campos [53].

Tabla 4.4 Valores del campo *TM* de un mensaje de unidifusión.

TM	Valor
TM_STATUS_REQ	0x0a
TM_CONFIGURE_REQ	0x1a
TM_ANOMALY_STATUS	0x2a
TM_GET_ND_REQ	0x3a
TM_SET_ND_REQ	0x4a
TM_MGMT_CMD_REQ	0x5a
TM_PING	0xff

TM	Valor
TM_STATUS_RES	0x12
TM_CONFIGURE_RES	0x22
TM_ANOMALY_ACK	0x32
TM_GET_ND_RES	0x42
TM_SET_ND_RES	0x52
TM_MGMT_CMD_RES	0x62

Los campos de los mensajes se han visto previamente en el Capítulo 3 y en la Figura 4.3, y las opciones de implementación están descritas en la Sección 4.1.10. El tipo exacto de los campos de los mensajes de gestión están recogidos en un documento de descripción del protocolo usando la síntaxis de *Protocol Buffers* llamado `sns.proto` que a su vez incorpora `networkDataLite.proto` con la descripción del formato de `NetworkData`. El contenido abreviado de ambos ficheros se muestra en el Código 4.3 y el Código 4.4 respectivamente.

Código 4.3 Descripción del protocolo.

```
1  //================================================
2  // Name        : sns.proto
3  // Copyright   : ...(omitido por brevedad)...
4  // Description : SNSR application layer messages
5  //================================================
6
7  syntax = "proto3";
8  option optimize_for = LITE_RUNTIME;
9  package fjfj.sns.messages;
10 import "networkDataLite.proto";
11
12 message Empty {}
13
14 message StatusRequest {
15     uint32 txid = 1;
```

```
}

message StatusResponse {
    uint32 txid = 1;
    uint32 cfgVersion = 2;
    uint64 crlVersion = 3;
    uint32 nlAddress = 4;
    uint32 bsAddress = 5;
    repeated NeighborData neighbor = 6;
}

message ConfigureRequest {
  enum ConfigurationType {
        FULL = 0;
        DIFFERENTIAL=1;
    }
    uint32 txid = 1;
    uint32 cfgVersion = 2;
    ConfigurationType type = 3;
    uint32 nlAddress = 4;
    BaseStationData baseStation = 5;
    CRLData crl = 6;
    repeated NeighborData neighbors = 7;
    repeated uint32 noNeighbors = 8;  //(DIFFERENTIAL mode only) list of NL addresses
    repeated RouteData routes = 9;
    repeated MultiRouteData multiRoutes = 10;  //more compact routes
    repeated RouteData noRoutes = 11; //(DIFFERENTIAL mode only)
    repeated MultiRouteData noMultiRoutes = 12; //(DIFFERENTIAL mode only)
}

message ConfigureResponse {
    uint32 txid = 1;
    uint32 cfgVersion = 2;
    uint64 crlVersion = 3;
}

message TimeStamp {
    uint32 sec = 1;
    uint32 usec = 2;
}

message Anomaly {
    string observer = 1;
    uint32 id = 2;
    int32 type = 3;
    string object = 4;
    string info = 5;
    TimeStamp ts = 6;
    uint32 cfgVersion = 7;
}

message AnomalyStatus {
    repeated Anomaly anomaly = 1;
}

```

```
message AnomalyAck {
    string lastObserver = 1;
    uint32 lastId = 2;
}

message GetNetworkDataRequest {
    uint32 txid = 1;
    bool initial = 2;
}

message GetNetworkDataResponse {
    uint32 txid = 1;
    bytes networkData = 2;
}

message SetNetworkDataRequest {
    uint32 txid = 1;
    bytes networkData = 2;
}

message SetNetworkDataResponse {
    uint32 txid = 1;
    bool accepted = 2;
}

message MgmtCmdRequest {
    uint32 txid = 1;
    bytes command = 2;
}

message MgmtCmdResponse {
    uint32 txid = 1;
    bytes output = 2;
}

message SnsMessage {
    oneof snsMessage {
        StatusRequest statusReq = 1;
        StatusResponse statusRes = 2;
        ConfigureRequest configureReq = 3;
        ConfigureResponse configureRes = 4;
        AnomalyStatus anomalyStatus = 5;
        AnomalyAck anomalyAck = 6;
        GetNetworkDataRequest getNetworkDataReq = 7;
        GetNetworkDataResponse getNetworkDataRes = 8;
        SetNetworkDataRequest setNetworkDataReq = 9;
        SetNetworkDataResponse setNetworkDataRes = 10;
        MgmtCmdRequest mgmtCmdRequest = 11;
        MgmtCmdResponse mgmtCmdResponse = 12;
  }
}

message SnsBroadcastMessage {
    uint32 txid = 1;
    bytes content = 2;
    bytes sourceCert = 3;
```

```
127     bytes signature = 4;
128 }
129
130 message BroadcastContent {
131     oneof snsMessage {
132         CRLData crl = 3;  //same as ConfigureRequest
133     }
134 }
```

Código 4.4 Descripción de `NetworkData`.

```
1  //================================================
2  // Name        : networkDataLite.proto
3  // Copyright   : ...(omitido por brevedad)...
4  // Description : SNSR network data
5  //================================================
6
7  syntax = "proto3";
8  option optimize_for = LITE_RUNTIME;
9  package fjfj.sns.messages;
10
11 message CRLData {
12     uint64 version = 1;
13     repeated bytes serials = 2;
14 }
15
16 message BaseStationData {
17   uint32 address = 1;
18   string name = 2;
19 }
20
21 message NeighborData {
22   enum NeighborType {
23       PEER = 0;
24       SERVER = 1;
25       CLIENT = 2;
26   }
27   string llAddress = 1;
28   uint32 nlAddress = 2;
29   string name = 3;
30   NeighborType type = 4;
31   uint32 keepaliveInterval = 5;
32 }
33
34 message RouteData {
35   uint32 subnet = 1;
36   uint32 mask = 2;
37   uint32 gwAddress = 3;
38   uint32 metric = 4;
39 }
40
41 message SubnetData {
42   uint32 subnet = 1;
43   uint32 mask = 2;
```

```
}

message NextHopData {
  uint32 gwAddress = 1;
  uint32 metric = 2;
}

message MultiRouteData {
    repeated SubnetData destinations = 1;
    repeated NextHopData nextHops = 2;
}

message Coordinates {
    double x = 1;
    double y = 2;
    double z = 3;
}

message NodeData {
  string llAddress = 1;
  uint32 nlAddress = 2;
  string name = 3;
  BaseStationData baseStation = 4;
  repeated NeighborData neighbors = 5;
  repeated RouteData routes = 6;
  repeated MultiRouteData multiRoutes = 7;  //more compacted routes
  Coordinates position = 8;
}

message NetworkData {
    uint32 cfgVersion = 1;
    CRLData crl = 6;
    repeated NodeData nodes = 7;
    bool networkByteOrder = 8; // Network addresses in network byte order?
}
```

Los mensajes de gestión se transmiten usando el flujo fiable de la conexión de transporte, a excepción de `TM_ANOMALY_STATUS` y `TM_ANOMALY_ACK` que utilizan el flujo no fiable (ya que la notificación de anomalías se reintenta si no son asentidas). Los mensajes *PING* de petición y respuesta usan el flujo no fiable, mientras que el de test de transferencia puede utilizar ambos (para probar la transferencia en ambos flujos).

En cuando a los mensajes de difusión, la arquitectura SNSR define un único mensaje de tipo `TM_SET_CRL_BCAST`, usado para la actualización de la CRL y cuyos campos aparecen en (3.20). Sin embargo, la implementación permite la difusión de mensajes desconocidos por la AL siempre que se verifique la autenticidad del mensaje. De esta manera otras aplicaciones de usuario pueden enviar sus propios tipos de mensajes. La codificación del mensaje `TM_SET_CRL_BCAST` se realiza usando también *Protocol Buffers* y, por motivos de eficiencia, los dos primeros campos aparecen en orden inverso y se usa el mismo valor de tipo de mensaje que para el mensaje de configuración (`0x1a`).

4.3.5.2 Aplicación de gestión del robot

En esta implementación de la arquitectura SNSR, la aplicación de gestión del robot tiene la mayor relevancia, al concentrar las principales tareas incluyendo el cálculo de la topología

de la red, que podría haber realizado la BS. El funcionamiento del robot está totalmente condicionado a la fase en la que se encuentra la red (descritas en la Sección 3.3.5). En cambio, el resto de los tipos de entidad están diseñadas para no necesitar conocer la fase actual. En una red real la BS sí debería ser consciente de ella, pero actualmente solo se ha implementado lo imprescindible para probar la arquitectura y es necesario un operador humano para realizar operaciones como la actualización de la CRL mediante comandos administrativos.

En la Figura 4.16 se muestra un diagrama de estados de `RobotApp`. En ella se muestran los estados con su nombre y número y las transiciones automáticas que ocurren entre ellos. Se puede forzar de manera manual el paso desde cualquier estado a otro, finalizar la aplicación o reiniciarla con comandos administrativos, pero esto no está representado. Los colores de los estados son similares a los usados en la Figura 3.13 para mostrar la correspondencia con las fases: el estado `NONE` se corresponde con la fase *inicial* y *mantenimiento*, `DISCOVER` con la fase de *descubrimiento*, `PREPARE` con el inicio de la fase de *establecimiento de la topología* y `CONFIGURE` con la de *configuración*. Los otros estados son estados auxiliares intermedios entre los anteriores estados principales. A continuación, se describen brevemente los estados y sus transiciones salientes:

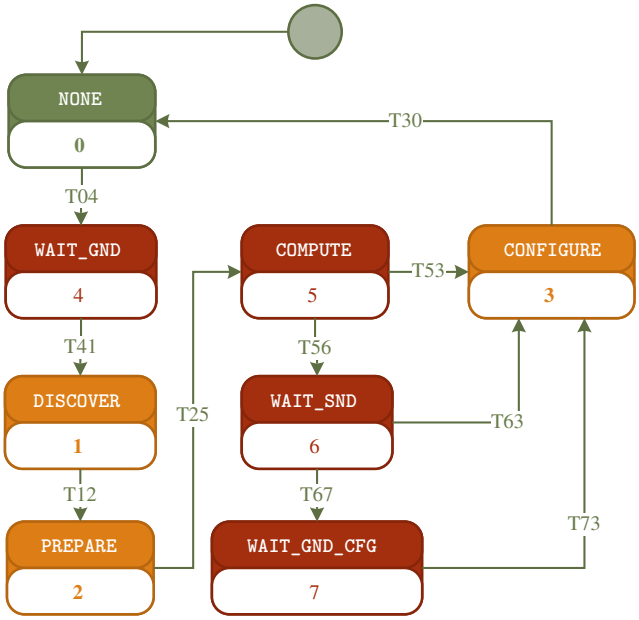


Figura 4.16 Diagrama de estados de `RobotApp`.

0. `NONE`. Estado de reposo para el robot.

Transiciones salientes:

T04 Si el robot no dispone de la versión inicial de la información de la topología de la red y detecta una BS, envía un mensaje `TM_GET_ND_REQ` a esta.

1. `DISCOVER`. El robot recorre el escenario detectando entidades y obteniendo información de ellas.

Transiciones salientes:

T12 Si el robot dispone de una versión inicial de la información de la topología de la red, que sirve como listado de entidades autorizadas en la red, y descubre todas las entidades registradas en ella, pasa a la siguiente fase.

2. `PREPARE`. El robot inicia la preparación de una nueva versión de la información de la topología de la red con los datos disponibles.

Transiciones salientes:

T25 Los preparativos han terminado y puede empezar el cálculo de la topología.

3. `CONFIGURE`. El robot recorre el escenario enviando la nueva configuración a todas las entidades reconocidas.

Transiciones salientes:

T30 Todas las entidades están usando la nueva configuración.

4. `WAIT_GND`. El robot está esperando una respuesta del tipo `TM_GET_ND_RES` de una BS con la versión inicial de la información de la topología de la red.

Transiciones salientes:

T41 El robot recibe una respuesta `TM_GET_ND_RES` correcta.

5. `COMPUTE`. Se calcula la topología de la red como se ha descrito en la Sección 4.1.7. Los cálculos se realizan en paralelo de tal manera que se siguen atendiendo las conexiones abiertas y registrando cambios.

Transiciones salientes:

T53 Los cálculos han terminado, pero el número de versión de la información de la topología es menor que el actual (la versión actual de `NetworkData` ha sido actualizada de manera administrativa).

T56 Los cálculos han terminado y deben ser enviados a la BS con un mensaje `TM_SET_ND_REQ` para su aprobación.

6. `WAIT_SND`. El robot está esperando confirmación de la BS a través de un mensaje `TM_SET_ND_RES`.

Transiciones salientes:

T63 La BS acepta la nueva versión.

T67 La BS no la acepta. El robot solicita la versión que debe utilizar mediante un mensaje `TM_GET_ND_REQ`.

7. `WAIT_GND_CFG`. El robot espera la última versión de la información de la topología.

Transiciones salientes:

T73 Se recibe la última versión de la información de la topología actualizada mediante un mensaje `TM_GET_ND_RES`.

Se puede configurar también el robot para que la transición a los estados `DISCOVER`, `PREPARE` y `CONFIGURE` nunca sea automática y se requiera el envío de un comando administrativo para autorizar la transición a ellos. Por simplicidad, esto no está representado en la figura anterior.

El robot también debe mantener el estado e información de cada una de las entidades descubiertas: incluyendo versiones de la configuración y de la CRL, *TX_ID* del último mensaje enviado, etc. La información obtenida de las propias entidades se fusiona con la de otras fuentes, como la localización, datos de la versión inicial de la información topológica, etc.

Cada vez que una entidad establece un enlace con el robot, `RobotApp` es notificado de la incorporación de un nuevo vecino. Esto provoca que `RobotApp` inicie una nueva conexión de transporte con el nuevo vecino para poder mandar mensajes de gestión. Esta conexión no necesita enviar pulsos si ya se está realizando en el LL. Cuando se pierde la conectividad con el vecino, la conexión de transporte también se elimina.

Por último, `RobotApp` también puede detectar anomalías adicionales a las comunes detectadas por todas las entidades. En concreto, la detección de nodos clonados (con el mismo identificador) es reportada como anomalía. La conexión de entidades no autorizadas o con datos erróneos se ignora actualmente.

4.4 Herramientas para la realización de ataques

Previamente se han comentado las posibilidades de configuración que ofrece la implementación del NOS. Existen combinaciones de parámetros que permiten ejecutar comportamientos distintos a los que se definieron en la *Contribución 1* de esta Tesis Doctoral, permitiendo probar ataques sencillos con el mismo programa `Node`.

Muchos ataques se pueden realizar utilizando herramientas externas al NOS, por lo que no requiere la modificación de este último. Otros ataques que involucran la existencia de un NOS malicioso pueden ser simulados con herramientas externas también, consiguiendo los mismos efectos. Un ejemplo típico del anterior caso es el ataque de reenvío selectivo (*selective forwarding* o *greyhole*), donde un nodo reenvía algunos mensajes y otros los descarta incumpliendo el protocolo, que puede sustituirse por un canal con mayor tasa de pérdidas de tramas o un cortafuegos situado en el propio nodo. En [30] se hace un estudio más detallado de los ataques que se pueden simular de esta manera. Las herramientas externas usadas en esta Tesis Doctoral son las siguientes:

1. Las herramientas de captura de tráfico de red *tcpdump* [74] y *wireshark* [21], que utilizan ambas la biblioteca *libpcap* [74].
2. *Scapy* [12], un programa que permite la inyección, captura y manipulación de paquetes, también utilizando la biblioteca *libpcap*.
3. El cortafuegos del S.O. Linux [159].
4. *NetEm* [57], que permite emular distintas propiedades de transmisión en redes, como retardos, pérdidas, corrupción, duplicación y reordenación de paquetes. Está integrado en el núcleo de Linux y se controla a través de la herramienta *tc* (*Linux Traffic Control*).

5. *WmediumD* [22] como aplicación que permite aplicar probabilidades de pérdidas a enlaces radio simulados con el módulo *mac80211_hwsim* de Linux. Se ha utilizado en redes virtuales creadas con *Mininet-WiFi* [45], que a su vez añade la capacidad de crear redes Wi-Fi virtuales al «sistema de orquestación de emulación de red» *Mininet* [55].

Las dos últimas herramientas se han utilizado para la construcción del compositor/orquestador de escenarios que se presentará más adelante.

Por último, se ha creado una versión del NOS modificado y con él se ha generado el ejecutable `Badnode`. Esta versión añade la posibilidad de especificar manualmente todos los campos de una baliza o generarlos de manera aleatoria en cada transmisión. Además permite la falsificación de la dirección MAC origen si se usa Ethernet (o equivalentes) como LL, usando para ello el conector `ConnectorEthernetRaw`.

4.5 Compositor/orquestador de escenarios

Un escenario es una instalación industrial real o simulada donde se han desplegado una serie de nodos sensores que quieren establecer comunicación con una BS con un nivel de seguridad especificado utilizando al menos un robot. En un escenario también pueden participar agentes maliciosos. El escenario define la posición inicial de cada entidad participante (aunque sea desconocida por las propias entidades), las características de las comunicaciones radio entre entidades y cualquier otra configuración deseada.

Un escenario simulado va a seguir utilizando software real, aunque principalmente en un entorno virtualizado. Una entidad es la ejecución del programa `Node` en un entorno que puede ser real o virtual. Es posible utilizar simultáneamente entidades reales y virtuales en un mismo escenario simulado. La simulación de las propiedades de los canales de comunicación dependiendo de la posición de cada entidad sí debe realizarse y esto incluye simular la potencia de transmisión, las características de las antenas, los retardos de propagación y las probabilidades de error, pérdida de tramas, duplicación y envío desordenado.

La construcción de la topología de los escenarios simulados puede ser manual o automática. El diseñador del escenario, de manera manual, debe poder especificar la posición exacta de cada entidad, y así probar casos límites. La topología también se puede especificar indicando los vecinos que inicialmente tendrá cada entidad, sin importar las posiciones exactas ni el alcance radio máximo. Muchas veces se desea que el grafo de la topología en su conjunto cumpla determinadas propiedades o introducir un componente aleatorio, para lo que se pueden utilizar generadores automáticos.

En esta sección se explica en qué consiste la ejecución de un escenario y se describe el software compositor/orquestador de escenarios desarrollado para esta Tesis Doctoral que ha permitido realizar miles de ejecuciones de escenarios simulados y facilitar las pruebas reales. Para abreviar, este software se ha llamado *lanzador* (*launcher*) de escenarios. Existen dos versiones del *lanzador*, que se diferencian en el modo de simular las propiedades de los canales de comunicación y cómo se virtualizan las entidades, que se han llamado `launcher` y `launchermn`. Para la implementación se ha utilizado el lenguaje de programación Python. El código es compatible tanto para la versión 2 de Python como para la 3. Se han creado otros programas auxiliares para gestionar certificados, convertir

un fichero binario con información topológica serializada a JSON y viceversa, etc. que no serán descritos ya que no se consideran fundamentales para esta Tesis Doctoral.

4.5.1 Secuencia de ejecución de un escenario

La utilización de la arquitectura SNSR en un escenario determinado requiere unos pasos previos antes de entrar en funcionamiento. Estos pasos son:

- Tener una PKI para gestionar los certificados firmados por la CA que usa cada entidad y la CRL.
- Instalar en cada entidad una versión apropiada del programa `Node` (es necesario tener en cuenta las opciones de compilación adecuadas para el escenario) y las bibliotecas compartidas de las que depende si no se ha compilado estáticamente.
- Configurar cada entidad con su certificado y clave privada, el certificado de la CA, la CRL inicial, los parámetros de inicialización, etc.
- Y por último, ejecutar el programa `Node` con la configuración anterior.

Durante el funcionamiento de un escenario interesa poder ejecutar comandos administrativos en las distintas entidades, sobre todo en el robot y la BS. Otras veces interesa conocer el estado actual para verificar que el funcionamiento es correcto. Además, muchos comandos administrativos permiten probar comportamientos anómalos, caídas de nodos, etc.

No solo es interesante poder ejecutar comandos administrativos manualmente sino también la programación de la ejecución de estos en instantes determinados con antelación. La programación de eventos permite repetir escenarios con las mismas condiciones.

Al terminar la ejecución de un escenario se debe recopilar información para poder hacer un análisis posterior. Entre la información a recopilar se encuentran:

- Los registros generados por todas las entidades.
- Las posiciones de las entidades estáticas y los movimientos realizados por el robot.
- Las configuraciones utilizadas y las versiones del software utilizado.
- Los eventos producidos manualmente o de manera programada, entre los que se encuentran los comandos administrativos ejecutados.
- Las distintas versiones de la información topológica.
- Opcionalmente, una captura de red desde el punto de vista de cada entidad cuando sea posible.

El tiempo necesario para realizar todas las acciones comentadas anteriormente de manera manual es elevado y aumenta linealmente con el número de nodos que forman parte del escenario. Por lo anterior, ha sido necesario automatizar todo el proceso.

4.5.2 Simulación de los canales de comunicación y el movimiento del robot

Para este cometido se utilizan dos de las herramientas descritas en la Sección 4.4 por separado: *NetEm* y *WmediumD* (parte de *Mininet-WiFi*). Cada una tiene una serie de ventajas e inconvenientes, que se mostrarán más adelante, que las hace más o menos idóneas para determinadas pruebas. La versión del *lanzador* que usa *NetEm* fue la primera en implementarse y el módulo principal se llama `launcher`. La segunda versión, cuyo módulo principal es `launchermn`, es una extensión de la primera adaptada para usar `Mininet-WiFi`. Según la tecnología empleada, el movimiento del robot se va a simular también de distinta manera.

A continuación, se describe cómo se ha realizado la simulación en cada una de las versiones.

4.5.2.1 `launcher`

NetEm [57], controlado a través de la herramienta *tc*, permite aplicar a los paquetes que cumplan determinadas características reglas que especifican cómo se transmitirán, incluyendo retardos, pérdidas, corrupción, duplicación y reordenación.

En este *lanzador* solo se puede utilizar *NetEm* en paquetes de salida a través de la interfaz local (`lo`). Esto quiere decir que solo se simulan los canales entre las distintas entidades virtualizadas en el mismo equipo y la comunicación con entidades externas nunca se altera. Al usar una única interfaz, no es posible hacer una captura individualizada de los paquetes vistos por una sola entidad, aunque sí una captura global. Las entidades deben usar el protocolo UDP como LL virtual. Cada entidad tendrá el conector principal enlazado a la misma dirección IPv6 pero usando un puerto UDP exclusivo. Cada puerto UDP simula una interfaz virtual. De esa manera los mensajes intercambiados entre entidades se enviarán a través de la interfaz local.

La difusión de tramas, para el envío de balizas, se realiza haciendo que las entidades utilicen un conector UDP adicional escuchando en un puerto común y que el robot transmita a la dirección de multidifusión de todos los equipos (`FF02::1`) y el puerto común. El conjunto de entidades que usan el mismo puerto común se va a denominar «zona de difusión». En un mismo escenario es posible que existan varias zonas de difusión y una entidad puede pertenecer a varias zonas simultáneamente. Un robot solo puede emitir balizas en una zona de difusión, de tal manera que solo las entidades de esa zona podrán detectar su presencia.

En la configuración realizada, los paquetes se dividen en clases de tráfico que pueden a su vez subdividirse o asociarse a colas de paquetes. Estas últimas están gobernadas por algoritmos que deciden cuando enviar cada paquete. El conjunto de una cola y su algoritmo se denomina «disciplina de cola». La disciplina de cola utilizada es *Hierarchical Token Bucket* (HTB) [28], ya que permite la subdivisión en clases del tráfico y no admite prioridades, que es lo que se pretende.

Se definen los siguientes tipos de tráfico (clases) a los que se les puede aplicar distintas reglas de *NetEm*:

1. Entre entidades (enlace), pudiendo ser simétrico o asimétrico. En un enlace simétrico las reglas *NetEm* se aplican en ambos sentidos. Por el contrario, en un enlace asimétrico se pueden especificar reglas diferentes en cada sentido.

2. A zona de difusión.
3. De entidad, incluyendo el tráfico saliente de la entidad y con destino a ella.
4. Global o común, aplicable a todo el tráfico de todas las entidades.

Las reglas de tráfico global se aplican siempre y son independientes del resto de reglas. En cambio, el resto de tipos de tráficos son excluyentes. Por ejemplo, si un paquete se clasifica como perteneciente a un enlace, ya no se puede clasificar como tráfico de entidad y solo se aplicarían las reglas del enlace y las globales y no las de la entidad. El orden de clasificación sigue el orden de la lista anterior: enlace, zona de difusión y entidad.

La creación de las clases, filtros y disciplinas de colas se realiza al comienzo de la ejecución del escenario y solo para los tipos indicados de manera manual en el fichero de configuración del escenario. Es decir, no se crean automáticamente. Las reglas de *NetEm* pueden ser modificadas una vez comenzada la ejecución mediante comandos manuales o mediante la programación de eventos en tiempos determinados.

Gracias a que las reglas se pueden modificar dinámicamente es posible simular ataques por interferencias temporales, congestión y los cambios producidos en las comunicaciones por el movimiento del robot. En concreto, la simulación del movimiento del robot se articula de la siguiente manera:

- Se eligen los puntos a donde se moverá el robot. Se simula un movimiento discreto del robot (no continuo), es decir, el robot se moverá de un punto a otro instantáneamente. Las entidades alcanzables vía radio desde cada punto formarán parte de una zona de difusión, pudiendo existir zonas de difusión sin ninguna entidad. Deben elegirse los puntos de tal manera que cada entidad pertenezca a una zona como mínimo. No existe ningún proceso automatizado de elección de puntos, por lo que el diseñador del escenario debe realizar la elección manualmente y esto a su vez lo hace incompatible con escenarios donde las posiciones de los nodos se generen aleatoriamente si se busca realismo.
- Cada vez que el robot se mueve de un punto a otro, o lo que es lo mismo, de una zona de difusión a otra, se modifica el destino de las balizas transmitidas y se modifican las reglas de *NetEm* para que sea posible la comunicación con las entidades que pertenezcan a la nueva zona y no sea posible con el resto.

Una vez terminada la ejecución del escenario, se muestra un resumen de los paquetes (y bytes) de cada clase que han sido procesados, incluyendo los que fueron descartados, desordenados y los que han superado los límites establecidos.

También hay que decir que este *lanzador* permite opcionalmente ejecutar escenarios ignorando las reglas de *NetEm* configuradas.

4.5.2.2 `launchermn`

A pesar de que la anterior versión del *lanzador* permite validar el funcionamiento de la arquitectura a grosso modo, su utilización requiere el cálculo de las propiedades de los canales manualmente y no permite interfaces inalámbricas virtuales. La búsqueda de un entorno más realista llevó al desarrollo de una segunda versión del *lanzador* (`launchermn`) basada en *Mininet-WiFi*.

Mininet-WiFi se basa en el emulador de red *Mininet* que hace uso de las características de virtualización ligera existentes en el núcleo de Linux, denominadas espacios de nombres (*namespaces*), similares a las usadas por otras tecnologías de contenedores. En concreto, para cada equipo emulado de la red virtual se crea un espacio de nombre de red donde el grupo de procesos que se ejecutan en él tienen acceso a una pila de red exclusiva con interfaces, tablas de rutas, etc. que solo ellos pueden utilizar. A menos que se configure expresamente, no se utilizan otros tipos de espacios de nombres, por lo que los usuarios, disco, identificadores de procesos, etc. son compartidos. *Mininet-WiFi* añade a lo anterior, entre otras cosas, la utilización de interfaces WiFi virtuales con el módulo de Linux *mac80211_hwsim* y el uso de *WmediumD* para aplicar modelos de pérdida de propagación y retardos a los datos transmitidos entre las interfaces virtuales sin necesidad de hacer cálculos manuales. *Mininet-WiFi*, de manera alternativa, permite usar `tc` también para hacer la simulación del canal, pero no está recomendado para redes ad-hoc [33] y su uso es diferente al utilizado en la primera versión del *lanzador* (las reglas están pensadas para aplicarse a toda la interfaz y no permite diferenciar según el destino del mensaje).

WmediumD admite varios modelos de pérdidas de propagación, entre los cuales se ha elegido por defecto el modelo *log-distance path loss* [123], que se utiliza para calcular las pérdidas en espacios interiores. Este modelo se utiliza con un exponente de valor 4, que equivaldría a un interior industrial con gran contenido metálico [123]. La utilización de otros modelos como el de la Unión Internacional de Telecomunicaciones (ITU, *International Telecommunication Union*), el modelo *log-normal shadowing*, etc. también sería posible. Además de lo anterior, se ha habilitado el cálculo de interferencias provocadas por otras estaciones cercanas para hacer más realista la simulación.

Una vez creada la red virtual, es posible modificar la posición de los equipos. Al producirse el cambio de posición, se informa a *WmediumD* para que automáticamente cambie las propiedades de los canales. Se puede simular el movimiento de un equipo simplemente cambiando su posición cada cierto tiempo. Cuanto más pequeño sea el intervalo de cambio de posición, más suave es el movimiento. *Mininet-WiFi* incluye una funcionalidad básica para realizar movimientos de esta manera, pero no era suficiente para los escenarios previstos. Así que se ha desarrollado un módulo adicional que permite movimientos complejos, como los que debe seguir el robot. Los movimientos permiten la indicación de velocidades o tiempos de ejecución, el destino en coordenadas absolutas o relativas a otra entidad y la indicación de trayectorias formadas por varios segmentos. Todo esto se puede además representar gráficamente en tiempo real, para ver la posición de cada entidad y su alcance radio.

El *lanzador* `launchermn` incorpora el módulo de localización y movimiento del robot descrito en la Sección 4.1.12. Se comunica mediante comandos administrativos con el robot y realiza la simulación de los movimientos necesarios.

No es necesario crear distintas zonas de difusión con este *lanzador* y se puede usar directamente el LL de la propia interfaz virtual usando el conector Ethernet (aunque se puede usar también el conector UDP). Además, es posible hacer capturas de tráfico en cada entidad por separado.

4.5.2.3 Comparativa entre versiones

Como se comentó antes, la principal diferencia de las dos versiones es cómo se simulan las propiedades de los canales de comunicación (con *NetEm* o *WmediumD*) y cómo se virtualizan las entidades. Ante todo, existían más opciones a la hora de implementar el *lanzador*. En la segunda versión también podrían haberse utilizado simultáneamente reglas de *NetEm*. Sin embargo, no ha sido necesaria la realización de esta posibilidad en esta Tesis Doctoral. Ambas versiones son necesarias y útiles, aunque dependiendo de la prueba a realizar es preferible usar una u otra como se explica a continuación.

El *lanzador* `launcher` se puede usar con o sin simulación de los canales:

- Usado sin simulación es idóneo para escenarios híbridos en los que se desea ejecutar varias entidades en el mismo equipo interactuando con entidades externas. Si existen tantas interfaces de red como entidades, se pueden usar conectores Ethernet además de los conectores UDP. También se utiliza como paso previo a la ejecución de escenarios reales, ya que permite generar fácilmente todos los ficheros de configuración a instalar en las entidades reales y generar información topológica a cargar en el robot. Además, permite probar escenarios en condiciones de comunicación ideales donde todas las entidades pueden comunicarse con el resto donde es posible hacer pruebas de estrés del robot. Sin embargo, no es posible simular el movimiento del robot a diferencia del resto de opciones.

- Con *NetEm* solo se soportan conectores UDP. Los cálculos de los parámetros de los canales de comunicación deben hacerse manualmente. Debido a esto se suele utilizar para probar condiciones extremas (reordenación de mensajes, duplicación, retardos elevados) o alteraciones temporales más que para intentar hacer simulaciones realistas. El movimiento del robot tampoco es realista, pero lo suficiente para hacer pruebas completas de los protocolos utilizados en la arquitectura SNSR.

En ambos casos el consumo de recursos del *lanzador* es bastante bajo y las ejecuciones de escenarios se inician casi instantáneamente. Esto posibilita ejecutar escenarios con un número elevado de nodos (se han utilizado hasta 500 nodos).

Con el segundo *lanzador*, `launchermn`, se obtiene mayor realismo en la simulación de los canales de comunicación y el movimiento del robot. Se calcula automáticamente la probabilidad de pérdidas y retardos a partir de la distancia, potencia de transmisión, antena y modelo de pérdidas de propagación, incluyendo interferencias producidas. Sin embargo, lo anterior tienen un coste computacional mayor. Durante el movimiento del robot, *WmediumD* debe estar recalculando constantemente los parámetros y si se supera el 100 % de uso de un núcleo de la CPU empieza a comportarse mal. Por el anterior motivo el número de nodos máximo que se pueden simular es bastante inferior al de la primera versión (se han utilizado hasta 50 nodos). El inicio de la ejecución del escenario también se demora bastante más (del orden de decenas de segundos), ya que se deben crear todas las interfaces virtuales y posteriormente configurarlas. No es fácil de usar en escenarios híbridos, ya que en principio cada entidad está aislada y habría que utilizar otro elemento que sirviera como pasarela. Tampoco es fácil modificar temporalmente o manualmente las características de los canales de comunicación.

4.5.3 Argumentos de los *lanzadores*

A un *lanzador* se le pasan los ficheros con la descripción de los escenarios que hay que ejecutar (como mínimo uno). Estos escenarios se ejecutan en orden secuencial. Además, cada *lanzador* admite una serie de argumentos opcionales en línea de comandos. En el listado 4.5 se muestra la ayuda proporcionada por `launcher`.

Código 4.5 Argumentos del *lanzador* `launcher`.

```
usage: launcher.py [-h] [-i] [-e EXECUTABLE] [-t TERMINAL] [-o OUTPUT]
                   [-r REPEAT] [-n] [-N]
                   file [file ...]

SNS Scenario launcher.

positional arguments:
file                  Configuration file

optional arguments:
-h, --help            show this help message and exit
-i, --ignoretc        Ignore tc rules (default: False)
-e EXECUTABLE, --executable EXECUTABLE
                      Change configuration executable for this (default:
                      None)
-t TERMINAL, --terminal TERMINAL
                      Terminal executable (default: xterm)
-o OUTPUT, --output OUTPUT
                      Output directory (default: /tmp/)
-r REPEAT, --repeat REPEAT
                      Repeat several times (default: 1)
-n, --noclean         No clean files from previous run (default: False)
-N, --nocleanrepeat No clean files from previous run only when repeated
                      (default: False)
```

En el *lanzador* `launchermn` la opción `ignoretc` no existe pero se añade la opción mostrada en el listado 4.6.

Código 4.6 Argumento adicional de `launchermn`.

```
-p, --plot            Plot network graph (default: False)
```

Las opciones anteriores permiten modificar algunos valores de la configuración del escenario: ejecutable que se utilizará por defecto en cada entidad (`executable`) y directorio donde se crearán los ficheros de salida (`output`).

Por defecto, el *lanzador* trata de abrir un terminal por cada entidad para mostrar los registros que se van generando en tiempo real. La opción `terminal` permite modificar el programa terminal que se ejecuta (por defecto, `xterm`). Si el número de nodos es

muy elevado no es conveniente abrir tantas ventanas en el sistema, por lo que se puede deshabilitar usando el valor `null`.

En la primera versión se puede deshabilitar el uso de *NetEm* con la opción `ignoretc`. En la segunda versión, utilizando la opción `plot`, se puede mostrar un gráfico con la posición de cada entidad y visualizar el movimiento del robot.

La ejecución de cada uno de los escenarios se puede repetir tantas veces como se indique con la opción `repeat`. También se puede repetir la ejecución simplemente volviendo a invocar al *lanzador* con los mismos parámetros. En ambos casos se pueden reutilizar los datos generados de la invocación anterior, ya que así es posible utilizar los mismos datos aleatorios o evitar la realización de cálculos. En los escenarios se pueden utilizar valores aleatorios para:

- Generar topologías aleatorias. Las posiciones de la BS y los nodos se generan aleatoriamente. Esto a su vez influye en los movimientos del robot y en la información topológica que se genera.
- Seleccionar aleatoriamente un grupo de nodos. Posteriormente, se puede realizar una serie de acciones sobre este grupo.

Para reutilizar los datos de una invocación anterior se utilizan las opciones `noclean` y `nocleanrepeat`. La diferencia entre usar una opción u otra radica en que la primera siempre reutiliza los datos anteriores y la última no lo hace la primera vez y sí en las repeticiones. A continuación, se muestran ejemplos de uso de estas opciones:

- Si existe un escenario donde la topología se genera aleatoriamente y se quiere probar una misma topología varias veces se puede utilizar la opción `nocleanrepeat`. En la primera ejecución se genera una topología aleatoria y en las repeticiones se reutiliza.
- Si se disponen de los ficheros de salida de la ejecución de un escenario realizada previamente y se quiere volver a repetir sin volver a realizar cálculos, se copiarían los ficheros al directorio de salida y se invocaría al *lanzador* con la opción `noclean`.

4.5.4 Descripción del escenario

Un escenario se describe en un fichero JSON, que se conoce como fichero de configuración o descripción del escenario. La estructura general de este archivo se muestra en el listado 4.7 de manera simplificada (los puntos suspensivos indican que podrían añadirse más elementos). Esta estructura es común para ambas versiones del *lanzador* porque los campos no utilizados son ignorados. No se va a describir toda la sintaxis permitida, solo las partes principales para mostrar las posibilidades que ofrece.

Código 4.7 Fichero de descripción de escenario.

```
{
    "id": "Identificador",
    "name": "Nombre del escenario",
    "common": {
        "output_dir": "/tmp/",
```

```
        "exe": "Release/Node",
        "env": {
          "nombre_var": "valor_var", ... },
        "certs_path": "../certificados/",
        "ca_cert": "../certificados/ca-ecc-cert.pem",
        "crl_file": "../certificados/crlPropioEj.txt",
        "ll_security": true,
        "tl_security_type": "EXCEPT_NEIGHBORS",
        "broadcast_verify_type": "KNOWN_SIGN",
        "log_level": 0,
        "capture_packets": false,
        "netem": "delay 10",
        "max_neighbor_distance": 20.
        ...
    },
    "zones": {
        "z1": {
          "multicast_address": "::#20001", "netem": "" },
        ...
    },
    "nodes": {
        "b1": {
          "name": "base1.bs.dit.us.es.",
          "certificate": "base1-cert.der",
          "key": "base1-key.der",
          "link_address": "::#10001",
          "zones": ["z1"],
          "network_address": 1,
          "type": "bs",
          "netem": "",
          "remote": false,
          "commands": [ "load" ],
          "preargs": [],
          ...
        }
        ...
    },
    "node_sets": {
      "nombre_set": "lista", ... },
    "links": [
        { "disabled": true, "client": "b1", "server": "r1",
          "netem": "loss 100%"
        },
        ...
    ],
    "events": [
        { "time": 4, "type": "command", "element": "r1",
          "content": "ps" },
        { "time": 10, "type": "netem", "element": "z1",
```

```
          "content": "loss 0%" },
        ...
    ],
    "network_data": [
      { "cfgVersion": 0,
        "calculateNLAddress": true,
        "useCoordinates": true,
        "showGraph": true,
        "generator": {
            "type": "random",
            "strong_conected": true,
            "min_neighbors": 1,
            "random_X": [0, 1000],
            "random_Y": [0, 1000],
            "random_Z": [0, 10]
          },
        "crl": {
          "version": 1, "serials": [ "01" ] },
        "nodes": [
            { "ref": "b1"},
            ...
          ]
      },
      { "cfgVersion": 1,
          ...
        "nodes": [
          { "ref": "b1",
            "coordinates": {
                "x":5.0, "y":15.0, "z":0.0}},
            ...
        ]
      }
    ]
}
```

Al principio del fichero aparecen datos identificativos del escenario, entre ellos destaca el campo `id` que sirve como prefijo para todos los ficheros generados durante la ejecución.

En el campo `common` se agrupan parámetros por defecto que afectan a todas las entidades y canales de comunicación:

- Directorio por defecto donde se guardan los ficheros de salida. Si no existe, se utiliza el pasado como argumento de línea de comandos.
- Programa principal que se ejecuta en cada entidad por defecto. Puede ser modificado con argumentos de línea de comandos. Una entidad puede utilizar otro distinto si lo especifica explícitamente en sus parámetros.
- Variables de entorno que se utilizan durante la ejecución y sus valores por defecto. Estas variables de entorno se pueden crear antes de la invocación del *lanzador*, de lo contrario el *lanzador* las crearía con el valor por defecto. Las variables de entorno

se pasan a todos los programas ejecutados por el *lanzador* y se pueden utilizar posteriormente en comandos o eventos.

- Directorio por defecto donde se almacenan los certificados digitales, claves privadas y ficheros con CRL. Si el valor de algún parámetro que guarde el nombre de un fichero relacionado con lo anterior no incluye el carácter `'/'`, se supone que es un fichero que debe estar en este directorio por defecto.
- Parámetros que sirven para crear la configuración de inicialización de cada entidad. Estos parámetros pueden personalizarse en cada entidad explícitamente. Habitualmente se incluyen los modos de seguridad, el nivel de registros y en general todos los parámetros que son idénticos en todos los nodos sensores.
- Reglas comunes de *NetEm* que se usarán en `launcher` o activación de la captura de paquetes en `launchermn`.
- Y otros como la distancia máxima entre vecinos, zona de búsqueda para el robot, dirección de difusión por defecto, rango inalámbrico, etc.

El campo `zones` se utiliza solo en `launcher` y en él se definen las zonas de difusión y las reglas de *NetEm* que se van a aplicar (parámetro `netem`). En general, es necesario añadir el parámetro `netem`, aunque tenga un valor vacío, a los elementos que vayan a ser controlados por `NetEm` para que las reglas sean creadas. Cada zona de difusión tendrá un nombre simbólico dentro del escenario que se podrá utilizar en el resto del fichero o en comandos para referirse a ella.

En el campo `nodes` están definidas todas las entidades (o nodos de red de manera genérica) participantes en el escenario. A cada entidad se le asigna un nombre simbólico, similar a como se hace con las zonas de difusión. Se indican los parámetros necesarios para la configuración inicial de cada entidad, a los que se añaden los definidos en el campo `common` si no aparecen explícitamente. El conector principal del LL se define con el parámetro `link_address` y su tipo (UDP o Ethernet) lo determina su valor. Si se usa un conector UDP y se aplica seguridad en el LL, se puede utilizar el valor `::` como dirección IPv6 para indicar que debe asignarse una dirección IPv6 de manera automática. La mayoría de los parámetros se traducen en argumentos que se pasarán al programa principal cuando se ejecute la entidad, pero además existe otros para:

- Indicar si la entidad debe gestionarla el *lanzador*, o se considera una entidad «remota», es decir, entidad que debe funcionar independientemente del *lanzador* dando lugar a escenarios híbridos.
- Comandos administrativos a ejecutar en la entidad al iniciarse.
- Reglas *NetEm* que se aplican inicialmente a todo el tráfico en el que participe la entidad.
- Argumentos adicionales que se deben añadir antes y después cuando se ejecute el programa principal. Esto permite ejecutar otros programas distintos a `Node`, el programa `Node` con la herramienta de análisis dinámico `valgrind` [27] y entidades maliciosas.

El *lanzador* puede inferir argumentos que hay que pasar al programa `Node` a partir del tipo de entidad, el modo de seguridad y otros parámetros de tal manera que la configuración sea lo más sencilla posible, pero permite también declararlos explícitamente. En el caso más simple, un nodo sensor necesita solo especificar 4 parámetros que difieren respecto al resto de nodos: nombre (`name`), ruta de su certificado (`certificate`), ruta de su clave privada (`key`) y dirección de enlace (`link_address`).

Es posible crear grupos de nodos en el campo `node_sets`. La finalidad es poder realizar acciones sobre todos sus miembros utilizando para ello el nombre del grupo. En el campo `node_sets` se definen los nombres de los grupos y la lista de nodos pertenecientes a cada uno. El *lanzador* permite indicar la lista de nodos con distintas expresiones que se pueden combinar entre sí y admiten recursividad. Estas expresiones permiten seleccionar nodos individuales o uniones de otros grupos, todos los de un determinado tipo de entidad, nodos aleatorios (del total, de un tipo o de otro grupo), nodos cuyo nombre simbólico cumpla un determinado patrón usando un rango de valores y todos excepto los que cumplan determinadas condiciones. Además de simplificar la escritura de comandos al grupo, permite guardar una lista de nodos aleatorios durante la ejecución del escenario.

Los nodos de la red pueden tener conexiones de enlace desde el inicio sin necesidad de que el robot los configure. Esto se puede hacer en la definición del nodo (`nodes`) de manera individual o usando el campo `links` por parejas. En este último campo se definen los enlaces entre nodos indicando los extremos y sus correspondientes roles, seguridad, parámetro KAI y reglas de *NetEm*. Si el enlace se define como deshabilitado (`disabled`), no se crea realmente al inicio, pero se crean las reglas de *NetEm* necesarias si se quieren simular determinadas condiciones del canal más adelante. Por defecto no se crean reglas *NetEm* para cada posible pareja de nodos, solo para los enlaces definidos en este campo. A los enlaces creados de esta manera se les asocia un nombre simbólico formado por el carácter `'l'` y el índice dentro del campo `links`.

Cuando el escenario empiece a ejecutarse, el *lanzador* presenta al usuario una interfaz de línea de comandos (CLI, *Command-Line Interface*) para poder interactuar con él. Casi todos los comandos que pueden ejecutarse en la CLI también pueden ser programados como eventos (ver Sección 4.5.6). En el campo `events` se definen estos eventos en el orden en el que deben ser ejecutados. Los parámetros de cada evento son el tiempo desde el inicio de la ejecución del escenario en el que debería ocurrir, el tipo de evento (comentario, comando, modificación de reglas *NetEm*, parada del gestor de eventos o fin de la ejecución del escenario), el elemento o elementos usando sus nombres simbólicos (se admiten expresiones como las usadas en el campo `node_sets`) y un contenido que dependerá del tipo de evento.

En el último campo mostrado, `network_data`, están las distintas versiones de información topológica que deben crearse antes de ejecutarse el escenario. Cada versión será almacenada en un fichero independiente en el formato adecuado para ser cargadas por el robot o la BS. No es estrictamente necesario almacenar la información topológica en el fichero de descripción del escenario, ya que se pueden utilizar ficheros externos. Sin embargo, el *lanzador* no solo traduce la información topológica del formato JSON al formato binario de *Protocol Buffers*, sino que también ofrece las siguientes facilidades para su escritura:

1. En vez de escribir directamente los datos de un nodo es posible usar una referencia con su nombre simbólico. El *lanzador* se encarga de completar la información topológica con los datos adecuados de tal forma que no haya que copiar los mismos valores varias veces. Esto evita posibles inconsistencias.
2. El *lanzador* implementa los mismos algoritmos que utiliza el robot para el cálculo de la topología de la red que se mostraron en la Sección 4.1.7. A partir de los datos iniciales proporcionados (posición o vecindad de cada nodo) es posible solicitar el cálculo automático de las direcciones de red, la BS asignada, los vecinos según el alcance radio máximo y las tablas de rutas que le corresponderían a cada nodo.
3. La asignación automática de las posiciones de las entidades estáticas de la red para formar distintas topologías. Existen tres generadores de topologías:
   a) Generador aleatorio: asigna un valor aleatorio a cada una de las coordenadas de la posición de un nodo dentro de unos rangos permitidos. Es posible forzar a que cada nodo tenga un número mínimo de vecinos alcanzables vía radio y/o que el grafo resultante sea conexo. En la Figura 4.2 se muestra una topología generada de esta manera.
   b) Generador de cuadrícula rectangular: genera una topología con forma de celosía o rejilla en el plano cartesiano. Hay que indicar el número de columnas deseado y las distancias de separación en los ejes x e y entre nodos. En la Figura 4.17 (izda.) se muestra un ejemplo.
   c) Generador de cuadrícula triangular: es similar al anterior, pero en este caso las filas pares están desplazadas la mitad de la distancia de separación en el eje x, formando estructuras triangulares. En la Figura 4.17 (dcha.) se muestra un ejemplo.

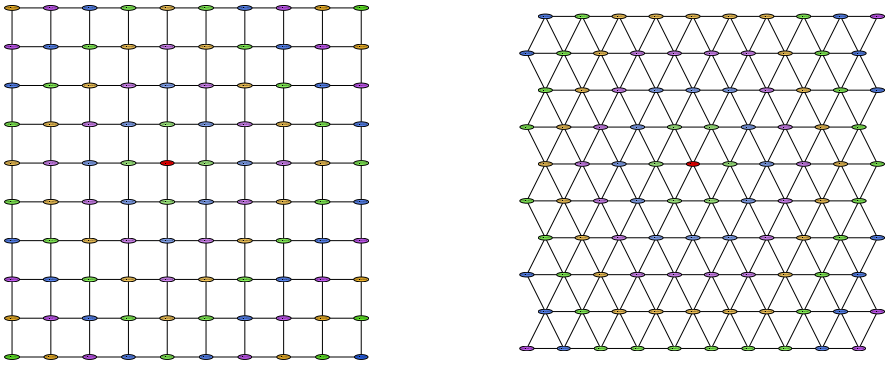


Figura 4.17 Topologías creadas con el generador de cuadrículas rectangulares (izda.) y triangulares (dcha.). La BS se muestra en rojo y el color de los nodos depende de la distancia a la BS.

Junto a cada fichero codificado con *Protocol Buffers* se genera una representación del grafo de la red en formato Graphviz [49] y un gráfico de él con el formato Gráficos Vectoriales Escalables (SVG, *Scalable Vector Graphics*) [43].

4.5.5 Creación de la red y las entidades

Tras procesar el fichero de descripción del escenario, el *lanzador* tiene toda la información necesaria para crear la red con las condiciones que se quieren simular en los canales e iniciar el software que se va a ejecutar en cada una de las entidades.

Como se ha explicado antes, la simulación de los canales de comunicación se hace de manera diferente según la versión del *lanzador* usada:

- En `launcher` hay que crear las clases de tráfico, filtros y disciplinas de colas de acuerdo con los parámetros `netem` encontrados en el fichero de descripción.
- En `launchermn` hay que iniciar la herramienta *Mininet-WiFi*, crear las interfaces virtuales y configurarlas, e indicar la posición, antenas y potencia de transmisión de cada estación inalámbrica para que *WmediumD* calcule las propiedades de propagación.

En cada nodo de la red virtual se ejecuta un programa principal (la entidad) con los argumentos generados a partir del fichero de descripción siempre que el nodo no sea remoto. En `launchermn` se ejecuta en el espacio de nombre de red creado para el nodo. El proceso se ejecuta como un proceso hijo del *lanzador*. Sus entradas y salidas se conectan al *lanzador* mediante tuberías no bloqueantes, para poder interactuar posteriormente con él.

Opcionalmente también se ejecuta una ventana terminal para monitorizar los registros generados por cada entidad y el programa `tcpdump` para capturar el tráfico de la red en cada interfaz virtual. Se guarda el identificador de todos los procesos ejecutados para poder enviarles señales y finalizar su ejecución.

4.5.6 Interacción con el escenario

Una vez que la red está creada y las entidades no remotas están funcionando, el *lanzador* comienza a procesar los eventos programados en los tiempos establecidos y presenta al usuario una CLI interactiva. El procesamiento de eventos y la interfaz interactiva se ejecutan en hilos independientes por lo que, en general, los comandos y eventos se ejecutan simultáneamente, excepto aquellos que involucren movimiento del robot que son procesados de manera síncrona utilizando una cola intermedia. Cualquier interacción con el escenario se registra con un sello de tiempo.

En la Figura 4.18 se muestra una captura de pantalla de `launchermn` simulando una red con 12 nodos sensores. A la izquierda se observa, de arriba a abajo, un gráfico con la posición en cada entidad y su alcance radio máximo en tiempo real, un gráfico que representa la red en su estado final ideal y la CLI del *lanzador*. El resto de ventanas muestran los registros en tiempo real del robot, BS y 4 de los nodos. La captura se tomó durante la fase de *configuración*. El aspecto de `launcher` es similar salvo que no muestra el gráfico de la red en tiempo real.

Los eventos se ejecutan uno a uno en el orden en el que aparecen en el fichero intentando cumplir el tiempo en el que están programados. Sin embargo, es posible que cuando finalice un evento ya se haya cumplido el tiempo del siguiente. En este caso, el siguiente se ejecutaría con retraso, justo después de la finalización del evento actual. En `launchermn`

Figura 4.18 Simulación de un escenario con `launchermn`.

es posible ejecutar 2 eventos en paralelo, con un hilo adicional que lee comandos de una cola y los ejecuta en orden en segundo plano. Esto permite seguir interactuando con el escenario mientras se ejecutan operaciones que duran mucho (por ejemplo, movimientos del robot) y ejecutar comandos justo cuando terminen otros.

La interfaz interactiva está a la espera de que el usuario introduzca una línea con un comando. Cuando esto ocurra, se ejecuta el comando bloqueando la entrada hasta su finalización. Se guarda un histórico de los comandos introducidos para facilitar la repetición de estos de las siguientes maneras:

- Si el usuario introduce una línea vacía, se ejecuta el último comando válido.
- Las teclas cursor arriba y abajo navegan por el historial hacia atrás y hacia adelante respectivamente
- La combinación `Ctrl-R` permite hacer búsquedas.

Los comandos que se pueden ejecutar en la CLI o por eventos pueden estar asociados o no a una entidad y se pueden clasificar según su funcionalidad en las siguientes categorías:

C1. Comandos de control de eventos: parar, ejecutar manualmente el siguiente evento sin importar el tiempo, encolar comandos para ser ejecutados en segundo plano o finalizar la ejecución del escenario de manera ordenada.

C2. Comandos de control de reglas de *NetEm*: permiten mostrar las reglas actuales y modificarlas.

C3. Comandos de la CLI de *Mininet-WiFi*: permiten manipular el funcionamiento de *Mininet-WiFi* y ejecutar programas en el espacio de nombre de red de un nodo. Solo son válidos en `launchermn` y actualmente no pueden usarse como eventos.

C4. Comandos administrativos para las entidades: los soportados por cada entidad. Algunos comandos administrativos pueden escribirse de manera simplificada y en estos casos el *lanzador* los completa con valores adecuados. Las entidades pueden enviar el resultado de los comandos administrativos a un fichero, la salida estándar o el registro. Por defecto, la CLI solicita que se haga a la salida estándar, que puede leer a través de una tubería y mostrar por pantalla. En los eventos, por defecto, el resultado se envía al registro de la entidad.

C5. Comandos informativos: para obtener información de los nodos de la red (nombres) y visualizar sus registros (con o sin actualización automática). No pueden usarse como eventos.

C6. Comandos de espera de cambios de estado: envío de comandos administrativos a entidades periódicamente hasta que el resultado sea el esperado o se supere un tiempo de espera máximo.

C7. Comandos de movimiento: movimientos en líneas rectas a una determinada velocidad, movimiento siguiendo la trayectoria de descubrimiento o de configuración (generadas con los algoritmos descritos en la Sección 4.1.13) e inicio de modo colaborativo con el robot. En el modo colaborativo, el *lanzador* pregunta al robot por su estado y las entidades con las que quiere comunicarse. Se elige la más cercana, se mueve hasta ella y espera en su cercanía un máximo de 10 segundos o hasta que el robot deje de solicitarla y pasa a la siguiente. Las entidades pueden no estar disponibles, haber finalizado o ser destruidas, así que se probará con el resto de entidades pendientes antes de volver a otra que previamente no haya contestado. El modo colaborativo continúa mientras el robot continúe en el mismo estado o se supere un tiempo máximo de espera. Estos comandos solo son válidos en `launchermn` y se ejecutan en segundo plano.

C8. Comandos o eventos desconocidos: solo se registran, pero no se hace nada. Usados para añadir comentarios a los registros.

Los comandos cuyos sujetos son nodos o elementos de *NetEm* permiten especificar una lista de sujetos (elementos separados por comas y expresiones). Además, es posible utilizar las variables de entorno definidas en el fichero del escenario dentro de la lista de sujetos o el propio comando, de tal manera que son sustituidas por su valor antes de ejecutar el comando.

En la Tabla 4.5 se listan los comandos no asociados a entidades, indicando la categoría funcional (columna «Cat.»), la sintaxis de uso y una breve descripción. Si un comando puede usarse como comando interactivo y como evento programado, se muestran las sintaxis de ambos tipos. En la descripción de uso, las palabras encerradas entre los símbolos <> denotan parámetros variables y el texto encerrado entre corchetes es opcional. Por ejemplo, `<T>` sería el instante de tiempo a ejecutar un evento. Los eventos se describen en formato JSON, tal como aparecerían en un fichero de escenario.

En la Tabla 4.6 se listan los comandos asociados a entidades de manera similar a la tabla anterior. La primera fila describe los formatos genéricos del comando interactivo y del evento. Todos admiten una lista de entidades `<LID>` que pueden incluir un grupo de nodos, expresiones y nombres simbólicos de entidades separados por comas. Un nombre

Tabla 4.5 Comandos del *lanzador* no asociados a entidades.

Comando o Evento	**Cat.**	**Uso y Descripción**
`l`	C5	`l` Lista los nombres simbólicos de todas las entidades.

`x`
`exit`
	C1	`x`
		`{"time": <T>, "type": "exit"}`
		Finaliza la ejecución del escenario. Pulsar `Ctrl-C` equivale a escribir `x`.

`s`
`stop`
	C1	`s`
		`{"time": <T>, "type": "stop"}`
		Para la ejecución automática de eventos. Si se desean ejecutar el resto de eventos configurados hay que usar el comando `c`.

`c`	C1	`c` Ejecuta el siguiente evento deshabilitando el uso del tiempo programado. Después, los restantes eventos solo se pueden ejecutar uno a uno manualmente volviendo a introducir el comando `c`.

`n`
`netem`
	C2	`n l`
		`n <IDNE>,<R1>[,<R2]`
		`{"time": <T>, "type": "netem", "element": "<LIDNE>", "content": "<R1>[,<R2]"}`
		Lista las reglas actuales de *NetEm* (solo como comando interactivo) y permite modificar las reglas de *NetEm*. El comando interactivo para modificaciones admite solo un único identificador `<IDNE>` de elemento *NetEm* (para reglas comunes, entidades, zonas de difusión o enlaces), mientras que como evento admite una lista de identificadores `<LIDNE>`. Se adjunta la regla *NetEm* `<R1>` a aplicar o la pareja de reglas `R1,R2` en caso de que el elemento se trate de un enlace asimétrico.

simbólico nunca puede ser igual que un comando de la Tabla 4.5 porque se confundiría. El resto de filas describen únicamente el comando en sí `<C>`.

Tabla 4.6 Comandos del *lanzador* asociados a entidades.

Comando o Evento	**Cat.**	**Uso y Descripción**

`<LID> <C> command`	C4	`<LID> <C>`
		`{"time": <T>, "type": "command", "element": "<LID>", "content": "<C>" }`
		Comando genérico `<C>` asociado a las entidades de la lista `<LID>`. En general, `<C>` es un comando administrativo que se envía a cada una de las entidades de la lista. Los comandos soportados se muestran en el listado 4.2. Sin embargo, algunos comandos son modificados previamente por el *lanzador* antes de ser enviados o son ejecutados por el *lanzador*. Estas excepciones son descritas aparte.

`enqueue`	C1	`enqueue <C'>` Añade a la cola de eventos a ejecutar en segundo plano un comando. Si `<C'>` es `end_scenario` se finaliza la ejecución del escenario, de lo contrario se ejecuta como se describe en el resto de la tabla.
`mn`	C3	`mn` Inicia la consola interactiva de *Mininet-WiFi*.
`dump`	C4	`dump` Pide al robot que grabe la información topológica que ha recopilado hasta el momento en un fichero con nombre generado automáticamente. Es una manera simplificada de ejecutar el comando administrativo `dump` soportado por el robot.
`dumpnd`	C4	`dumpnd` Similar al anterior, pero esta vez se graba la última versión de la información topológica calculada.
`savend`	C4	`savend` Configura el robot para que guarde automáticamente en un fichero la información topológica con un prefijo basado en el directorio de salida y el nombre del escenario.
`load`	C4	`load [<N>|<F>]` Carga ficheros con información topológica generados en el escenario en el robot o la BS sin necesidad de escribir el nombre del fichero manualmente. Por defecto, carga el primer fichero generado. Opcionalmente se le puede pasar un número `<N>` que indica el índice del fichero a cargar, o el nombre de un fichero `<F>` estático.
`beacons`	C4	`beacons <P>` Facilita la configuración del envío de balizas completando automáticamente la dirección de difusión por defecto si no aparece en los parámetros `<P>`.

Continúa en la siguiente página »

Tabla 4.6 (Continúa de la página anterior)

Comando o Evento	C	Uso y Descripción
`radius`	C4	`radius [<R>]` Facilita la configuración del radio de alcance usado en el cálculo de la topología en el robot, utilizando el valor configurado en el escenario si se omite el valor `<R>` del radio.
`node-info`	C5	`node-info` Muestra información de las entidades en formato JSON (configuración, posición, etc.).
`xterm`	C5	`xterm` Abre terminales que muestran los registros de las entidades en tiempo real, comenzando desde las 10 últimas líneas. Si ya estaban abiertas, pero ocultas por otra ventana, las trae a primer plano.
`log`	C5	`log` Visualiza los registros de las entidades desde el comienzo de la ejecución de manera paginada. En este caso los registros no se actualizan automáticamente.
`waitoutput`	C6	`waitoutput <CA> <O> <T>` Envía un comando administrativo `<CA>` a entidades cada 10 segundos mientras la salida no sea la esperada `<O>` (se admiten expresiones regulares) o pase un tiempo máximo especificado `<T>`. Se intenta ejecutar en segundo plano siempre que sea posible.
`waitpings`	C6	`waitpings <T>` Espera a que todos los nodos sensores hayan recibido un mensaje *PING* de respuesta durante un tiempo máximo especificado `<T>`. Se ejecuta en segundo plano.

`move`	C7	`move to <P> in <T> [seconds]`
		`move to <P> at <V> [m/s]`
		Mueve una entidad en línea recta a la posición `<P>` (puede ser una posición absoluta, relativa a la actual o relativa a otra entidad) a una velocidad `<V>` dada o en un tiempo `<T>` (es otra manera de indicar la velocidad a la que debe moverse).

`trajectory`	C7	`trajectory at <V> [m/s] [while <E>] points <P1> [<P2>...]` Mueve el robot siguiendo una trayectoria de segmentos rectos pasando por los puntos indicados a una velocidad `<V>` constante. La trayectoria termina al llegar al último punto o, si se indica un estado `<E>`, mientras el robot no cambie a otro estado. Los estados posibles se detallan en la Sección 4.3.5.2.

Continúa en la siguiente página »

Tabla 4.6 (Continúa de la página anterior)

Comando o Evento	C	Uso y Descripción

`search`	C7	`search area at <V> [m/s] [while <E>]`
		`search from <P1> to <P2> at <V> [m/s] [while <E>]`
		Mueve el robot siguiendo la *trayectoria de descubrimiento* descrita en la Sección 4.1.13 dentro del área configurada en el escenario o definida por los puntos `<P1>` y `<P2>`, a la velocidad `<V>`. Este comando se ejecuta hasta que la trayectoria termina o, si se indica un estado `<E>`, mientras el robot no cambie a otro estado.

`tour`	C7	`tour <V> [m/s] [while <E>]` Mueve el robot siguiendo la *trayectoria de configuración* descrita en la Sección 4.1.13 calculada con la posición de todos los nodos, a la velocidad `<V>`. Este comando se ejecuta hasta que la trayectoria termina o, si se indica un estado `<E>`, mientras el robot no cambie a otro estado.
`trypun`	C7	`trypun <E> at <V> in <T>` Inicia el modo colaborativo descrito previamente a una velocidad `<V>` y durante un tiempo máximo `<T>` mientras el robot continúe en el estado `<E>`.

4.5.7 Finalización del escenario y recopilación de datos

La ejecución de un escenario termina cuando se recibe un comando o evento de finalización. Tras esto, se intenta dejar el sistema en el estado en el que se encontraba antes de ejecutarse el escenario. Para ello se deshacen los cambios realizados. Esto implica terminar todos los procesos controlados por el *lanzador* y eliminar las reglas de simulación de canales de comunicación y la virtualización de la red. En `launcher` se deshacen todas las reglas de *NetEm* que existieran y en `launchermn` se destruye la red virtual y se finaliza *WmediumD*.

El programa principal de cada entidad se termina en 2 fases: primero enviando el comando de parada ordenada al programa o la señal POSIX 2 (`SIGINT`), y luego enviando la señal de terminación POSIX 15 (`SIGTERM`). La terminación del resto de programas se hace enviando directamente la señal `SIGTERM`.

Después de parar la ejecución, los ficheros generados se guardan comprimidos en formato ZIP en un fichero de nombre `SNS_<ID-escenario>_<fecha>_<hora>.zip`. Por defecto, el nombre de todos los ficheros generados durante la ejecución tendrán como prefijo el identificador del escenario y un guion bajo (`<ID-escenario>_`). Lo anterior permite distinguir los ficheros de distintos escenarios, aunque se use el mismo directorio de salida. El *lanzador* permite al usuario utilizar nombres de ficheros que no cumplan lo anterior, pero estos ficheros no se guardan automáticamente. Los ficheros que se generan normalmente durante la ejecución son los siguientes (por brevedad se omite el prefijo común):

- Copia del fichero de descripción del escenario: `config.json`.
- Datos calculados, para su reutilización en repeticiones: `calculated.json`

- Registro del *lanzador*: `launcher.log`.
- Por cada versión de información topológica generada, siendo `<X>` su índice en el fichero de descripción:
  - `nd<X>.bin`, que contiene la información topológica codificada con *Protocol Buffers*. `<X>` se omite si vale 0.
  - `<X>.gv`, que contiene la descripción del grafo de la red en formato Graphviz.
  - `<X>.gv.svg`, que contiene el diagrama de la red en formato gráfico SVG.
- Por cada nodo de la red, siendo `<nombre>` el nombre de este:
  - `<nombre>.args`, que contiene los argumentos pasados al programa principal de la entidad. Se genera siempre, aunque el nodo sea remoto y no sea ejecutado por el *lanzador*. Este fichero puede utilizarse para ejecutar los nodos de manera manual o bien para configurar un dispositivo real.
  - `<nombre>.log`, que es el fichero de registro del nodo.
  - `<nombre>.pid`, que guarda el identificador del proceso del programa principal. Permite matar el proceso de manera manual o adjuntarle un depurador de programas mientras esté funcionando.
  - `<nombre>.log.pcap`, que guarda la captura de red desde el punto de vista de ese nodo, siempre que se haya habilitado.
- Si se usa *NetEm*, un fichero, `tc.log`, con el listado de los comandos de `tc` ejecutados y otro, `tc.log.s`, con las estadísticas finales de `tc`.
- Ficheros de volcado de memoria (*core dump*) si se hubiesen generado por algún error fatal.

4.6 Control de escenarios de prueba en entorno real

En una instalación real, la modificación del software o firmware instalado en los nodos no debería ser algo habitual. Tampoco debería serlo el querer volver a reiniciar la red a su configuración inicial u otra. Sin embargo, durante la realización de pruebas son tareas que deben hacerse frecuentemente para reutilizar los mismos dispositivos en pruebas distintas. Si las pruebas son en entornos reales, donde se desea causar el menor impacto en el funcionamiento normal de las instalaciones, y el tiempo de uso del personal y de los recursos suponen un coste importante, estas modificaciones deben realizarse rápidamente.

Otra diferencia entre una instalación de prueba y una real es que en la segunda es necesario recoger datos de las entidades que deben ser transmitidos a un punto central para su análisis. Además, las entidades deben tener los relojes sincronizados para que las marcas de tiempo de los registros sean coherentes.

Usar actualizaciones por el aire (OTA, *Over-the-Air*) sería una posible solución al primer problema, como se hace en ZigBee [172], pero en la *Contribución 1* y la *Contribución 2* de esta Tesis Doctoral no se aborda esta cuestión y no se pretende hacer en esta Tesis Doctoral.

La solución a las necesidades anteriores ha sido crear un sistema de gestión remota de dispositivos independiente de la implementación de las entidades. Todos los dispositivos que se usan en esta Tesis Doctoral utilizan Linux y podemos hacer uso de sus capacidades para este cometido. El sistema de gestión remota tiene las siguientes funcionalidades:

- Visualización de información de los dispositivos activos.
- Actualización del software instalado en los dispositivos.
- Cambio de la configuración del programa `Node` (entidad) que se ejecuta en cada dispositivo, incluyendo sus dependencias: certificados, claves privadas, CRL y ficheros con información topológica.
- Control del programa `Node`. Esto incluye tanto el inicio y parada del programa como el envío de comandos administrativos.
- Recuperación de los registros generados durante la ejecución de `Node`.
- Envío de comandos al sistema operativo Linux del dispositivo.
- Sincronización de fecha y hora simple en los dispositivos.

En general, las operaciones se aplican a todos los dispositivos a la vez. Sin embargo, en el envío de comandos al sistema operativo o al programa `Node` se puede optar por el envío a un único dispositivo o a todos los que sean alcanzables.

El sistema de gestión remota sigue un modelo cliente/servidor con dos componentes:

1. Aplicación web de gestión remota (servidor).
2. Agente instalado en cada dispositivo (cliente).

La aplicación web está desarrollada en el lenguaje Java, utilizando *Java Enterprise Edition* (Java EE), principalmente con *servlets* y páginas *JavaServer Pages* (JSP). En cambio, el agente está desarrollado en Python. Hay que recalcar que este sistema de gestión, en su estado actual, está pensado para instalaciones de prueba, ya que se usa el Protocolo de Transferencia de Hipertexto (HTTP, *Hypertext Transfer Protocol*) (no seguro), no se autentican los componentes ni existe un control de acceso a la web. En una instalación real habría que usar al menos el Protocolo Seguro de Transferencia de Hipertexto (HTTPS, *Hypertext Transfer Protocol Secure*) y autenticación mutua.

4.6.1 Agente

El agente está constituido por 2 módulos que se instalan en cada uno de los dispositivos: `nodectl` (controlador de nodo) y `client` (cliente).

El módulo `client` se ejecuta periódicamente utilizando el programador de tareas de Linux `cron`. El periodo no debe ser mayor de 10 minutos. También es posible iniciarlo manualmente, pero solo puede haber una instancia como máximo en ejecución. Se utiliza un fichero de bloqueo para evitar más de una instancia.

El cliente se conecta a la aplicación web mediante el protocolo HTTP para sincronizar la fecha y hora, enviar datos del dispositivo y recibir información adicional e instrucciones adicionales que debe ejecutar (en forma de otros módulos de Python). La ejecución completa consiste en los siguientes pasos:

1. Prueba de conexión con la aplicación web. El cliente puede recibir los datos del servidor al que debe conectarse (nombre o dirección IP y puerto) como primer argumento. En caso contrario, utiliza la dirección IP de la primera pasarela por defecto de la tabla de rutas del sistema como dirección del servidor. Si no hubiera ninguna ruta por defecto configurada esperaría 10 segundos y lo volvería a intentar. La prueba consiste en hacer una petición HTTP a la ruta `"/SNSWeb/announce"` del servidor. Si se utiliza la dirección IP de la pasarela y no se recibe respuesta se intentaría con la siguiente dirección IP. Si no se recibe respuesta en ningún caso, el programa terminaría.
2. Sincronización de fecha y hora. El cliente solicita la fecha y hora actual del servidor a través de la ruta `"/SNSWeb/date"` y con ella ajusta la del sistema.
3. Envío de información del dispositivo al servidor. Se obtienen los siguientes datos del sistema: dirección IP de la interfaz usada con la pasarela por defecto, dirección MAC de esa interfaz, arquitectura del procesador, versiones de las bibliotecas de `wolfssl` y `protobuf`, versión del propio cliente, identificación del escenario actual y resumen SHA del ejecutable `Node` instalado. Todo esto se envía a la ruta `"/SNSWeb/info"` del servidor. Si el dispositivo es conocido por el servidor, este responde con información adicional que el cliente debe guardar para que otros programas (por ejemplo, `nodectl`) la usen. La información adicional incluye el nombre del dispositivo utilizado durante las pruebas, el identificador del escenario actual y el localizador de recursos uniforme (URL, *Uniform Resource Locator*) raíz de la aplicación web.
4. Bucle de ejecución de instrucciones adicionales. El cliente repite los siguientes pasos mientras que la aplicación web siga contestando y no se produzcan errores:
   a) Envío de la información del dispositivo a la ruta `"/SNSWeb/announce"`.
   b) La respuesta de la aplicación web es la ruta a un recurso adicional que el cliente debe descargar o está vacía si no hay nada más que ejecutar. La aplicación web determina qué debe ejecutarse según la información del dispositivo.
   c) Si la respuesta no está vacía, el cliente descarga el recurso y trata de ejecutarlo como si fuera un módulo de Python.

El módulo `nodectl` controla la ejecución del programa `Node`. Algunas de sus funciones requieren que el cliente se haya conectado previamente y la información adicional obtenida en el tercer paso esté disponible. Las operaciones que puede realizar son:

1. Detectar si `Node` está en ejecución y qué configuración se está usando.
2. Iniciar `Node` con la configuración actual, en segundo plano y redirigiendo la entrada y salida estándar para que posteriormente puedan enviarse comandos administrativos. Por defecto, si el programa ya está en ejecución, no se hace nada.
3. Enviar comandos administrativos a `Node` y mostrar su respuesta.
4. Parar `Node` de manera ordenada, comprimir los registros generados y enviar el fichero comprimido a la aplicación web (a la ruta `"/upload"`) que los almacenará. Si el envío es correcto, los ficheros de registros son borrados.

El cliente y el controlador se ejecutan con los privilegios de superusuario. El cliente usa estos privilegios para poder hacer modificaciones en el sistema operativo y el controlador para poder ejecutar `Node` usando el conector Ethernet.

El agente completo puede ser actualizado de manera remota después de ser instalado.

4.6.2 Aplicación web

La aplicación web de gestión remota, a la que se ha denominado `SNSWeb`, ofrece sus servicios tanto a la persona que administra la instalación de prueba como a los agentes instalados en los dispositivos. La aplicación no dispone de un control de acceso avanzado, así que la distinción entre un tipo de usuario y otro viene dada por los recursos que solicita. Los agentes son identificados usando la dirección MAC proporcionada por el cliente del agente en sus peticiones. La aplicación web tiene una relación de direcciones MAC asociadas a los nombres simbólicos que se utilizan en los escenarios. Un agente que no tenga asignado un nombre simbólico es ignorado.

La aplicación web requiere como único parámetro de configuración la ruta de un directorio de datos donde deben existir:

1. El fichero `nodes.json`, con una lista de dispositivos identificados por su nombre simbólico. Para cada dispositivo se deben enumerar las interfaces de red con sus direcciones MAC a través de las cuales el agente puede acceder a la aplicación web.
2. El fichero `versions.properties` que contiene información de las versiones de software y el identificador del escenario que se desea tener instalados en los dispositivos, junto con los nombres de los programas que habría que ejecutar para actualizar cada componente.
3. El programa `Node` que se debe ejecutar en los dispositivos.
4. El subdirectorio `logs` donde se guardarán los registros enviados por los dispositivos.
5. El subdirectorio `conf` donde están almacenados los ficheros de configuración y sus dependencias para cada escenario y dispositivo. Generalmente, los ficheros de configuración e información topológica son generados automáticamente por el *lanzador*.
6. Los programas indicados en `versions.properties` así como cualquier otro fichero auxiliar que estos necesiten.

Al iniciarse, la aplicación web lee los ficheros `nodes.json` y `versions.properties` y calcula el resumen SHA de `Node`. Esta información puede ser recargada posteriormente a petición del administrador cuando se produzcan cambios. Todos los ficheros del directorio anterior se pueden descargar usando peticiones HTTP a rutas del tipo `"/resource/<subruta>"`, donde `<subruta>` es la ruta relativa del fichero a descargar desde el directorio anterior. Si la petición anterior incluye un parámetro llamado `hash`, en vez de descargarse el fichero se devuelve el valor del resumen SHA del fichero solicitado. Lo anterior permite comprobar que un fichero ha sido descargado correctamente, verificando que el resumen proporcionado por la aplicación web es el mismo que el que resulta de hacer el cálculo localmente.

La interacción con el agente ha sido descrita en la Sección 4.6.1 de manera general. Como resultado de esta, el agente puede descargar y ejecutar cualquier programa en el dispositivo, sin establecer ninguna limitación a lo que el programa descargado puede realizar. Sin embargo, en esta aplicación web, la funcionalidad de los programas descargados se limita a actualizar el software y configuración usados en esta Tesis Doctoral. La aplicación web decide qué debe realizar cada agente comparando la información del software y configuración que se encuentra actualmente en el dispositivo con los valores deseados. Las siguientes tareas son llevadas a cabo:

1. Actualización del agente. Se utiliza su número de versión para determinar si debe actualizarse.
2. Actualización de las bibliotecas de las que depende `Node`. Se compara la versión de cada biblioteca con la deseada.
3. Actualización de `Node`. Se compara el resumen SHA de `Node` con el que se debe utilizar en el escenario.
4. Actualización de la configuración de `Node` para el escenario actual. Esto implica parar el proceso si ya se estaba ejecutando, enviar los registros a la aplicación web, descargar la nueva configuración y cualquier otro fichero del que dependa y volver a ejecutar `Node` con la nueva configuración. Se utiliza el identificador del escenario que se quiere ejecutar y el nombre simbólico del agente para determinar la configuración que hay que utilizar.

La aplicación web mantiene una lista con información de los agentes que han establecido contacto con ella, incluyendo la fecha y hora de la última interacción.

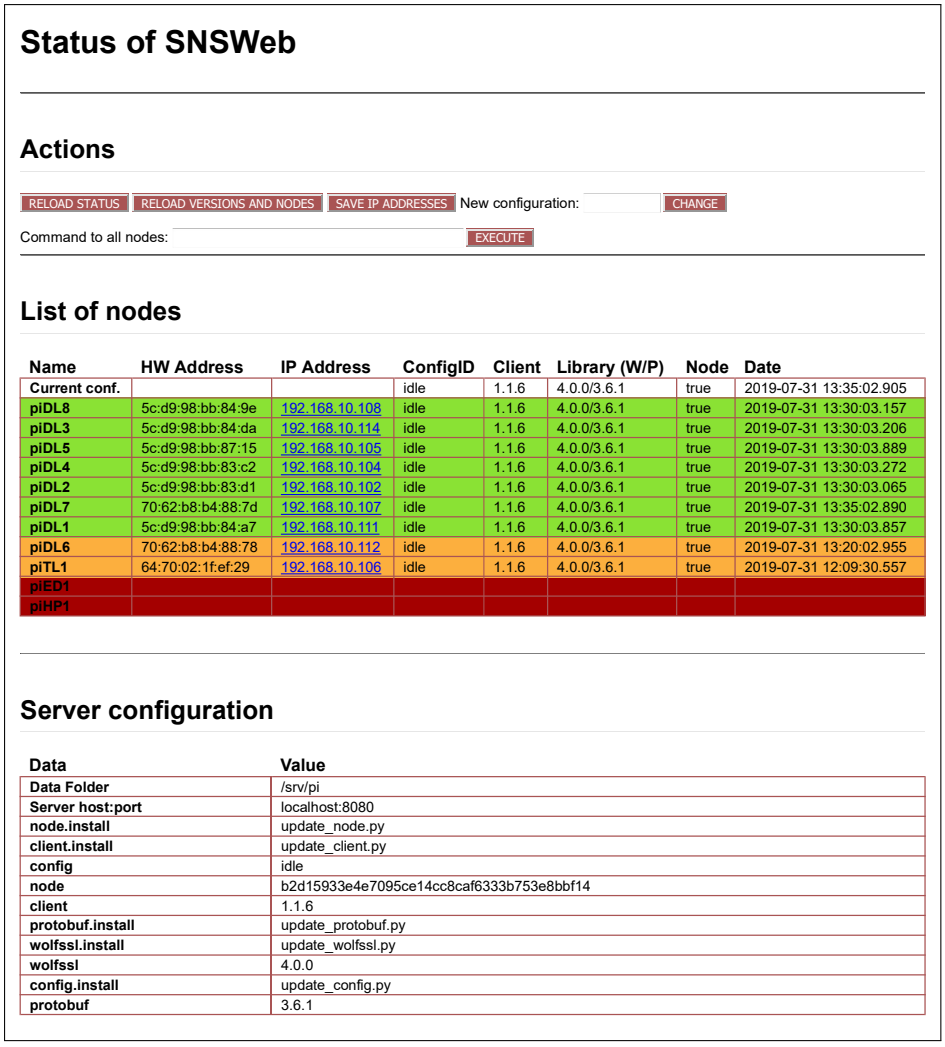
Además de las interacciones iniciadas por el agente antes comentadas, la aplicación web puede ejecutar comandos en los dispositivos utilizando esta vez el protocolo *Secure SHell* (SSH). Esto facilita la interacción directa entre el administrador y los dispositivos sin necesidad de recurrir a herramientas externas ni tener que conocer la dirección IP de los dispositivos. La conexión SSH solo permanece abierta durante la ejecución del comando. No se puede ofrecer ninguna garantía de que la conexión pueda realizarse, porque el dispositivo o el servidor pueden haber cambiado de posición y es posible que ya no se encuentren alcanzables. Entre los usos posibles, destacan la posibilidad de ejecutar comandos administrativos, ver los registros generados, forzar la ejecución del cliente o apagar el dispositivo.

La interacción con la persona administradora se realiza a través de dos páginas web:

1. La página principal donde se muestra el estado de los dispositivos e información de la configuración deseada y que permite realizar cambios y enviar comandos a todos los dispositivos.
2. Una página que permite ejecutar un comando a un único dispositivo y ver el resultado.

El aspecto que presenta la página principal se muestra en la Figura 4.19.

En la página principal la actualización de la información y la realización de las acciones se realizan de manera asíncrona, sin necesidad de recargar la página. Se divide en tres secciones que, de abajo a arriba son:



Status of SNSWeb

Actions

RELOAD STATUS | RELOAD VERSIONS AND NODES | SAVE IP ADDRESSES | New configuration: | CHANGE

Command to all nodes: | EXECUTE

List of nodes

Name	HW Address	IP Address	ConfigID	Client	Library (W/P)	Node	Date
Current conf.			idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 13:35:02.905
piDL8	5c:d9:98:bb:84:9e	192.168.10.108	idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 13:30:03.157
piDL3	5c:d9:98:bb:84:da	192.168.10.114	idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 13:30:03.206
piDL5	5c:d9:98:bb:87:15	192.168.10.105	idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 13:30:03.889
piDL4	5c:d9:98:bb:83:c2	192.168.10.104	idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 13:30:03.272
piDL2	5c:d9:98:bb:83:d1	192.168.10.102	idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 13:30:03.065
piDL7	70:62:b8:b4:88:7d	192.168.10.107	idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 13:35:02.890
piDL1	5c:d9:98:bb:84:a7	192.168.10.111	idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 13:30:03.857
piDL6	70:62:b8:b4:88:78	192.168.10.112	idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 13:20:02.955
piTL1	64:70:02:1f:ef:29	192.168.10.106	idle	1.1.6	4.0.0/3.6.1	true	2019-07-31 12:09:30.557
piED1							
piHP1							

Server configuration

Data	Value
Data Folder	/srv/pi
Server host:port	localhost:8080
node.install	update_node.py
client.install	update_client.py
config	idle
node	b2d15933e4e7095ce14cc8caf6333b753e8bbf14
client	1.1.6
protobuf.install	update_protobuf.py
wolfssl.install	update_wolfssl.py
wolfssl	4.0.0
config.install	update_config.py
protobuf	3.6.1

Figura 4.19 Aspecto de la página principal de SNSWeb.

- Configuración del servidor: directorio de datos configurados, puerto de escucha y contenido del fichero `versions.properties`.

- Lista de nodos (dispositivos): contiene una tabla con el estado de los dispositivos gestionados. La primera fila es la configuración y versiones deseadas con la fecha y hora actual. La columna «Node» indica si el programa `Node` está actualizado o no. A continuación de la primera fila aparecen en verde información de los dispositivos que han establecido contacto en los últimos 10 minutos. Después, en naranja, la de los dispositivos que han contactado alguna vez, pero llevan más de 10 minutos sin volver a establecer contacto. Por último, en rojo, los dispositivos que no han contactado

nunca. Esta lista se actualiza cada segundo. Si se hace clic en la dirección IP de un dispositivo se visualiza la página que permite ejecutar comandos individualmente.

- Acciones: muestra una serie de botones y campos de texto para:
  - Recargar la página completa.
  - Recargar los ficheros `node.json` y `versions.properties` que pueden haber sido modificados externamente. Provoca también la actualización del resto de secciones de la página.
  - Guardar en un fichero de texto la lista de las direcciones IP de los dispositivos. Esta lista puede ser utilizada en otros programas para realizar de manera automática operaciones en todos los dispositivos.
  - Cambiar el identificador del escenario que se quiere probar. Se modifica el valor almacenado en `versions.properties` y se actualiza el resto de secciones de la página.
  - Ejecutar un mismo comando en todos los dispositivos.

La segunda página simplemente permite al administrador escribir comandos que son ejecutados en un dispositivo, mostrando la salida estándar, la salida de errores estándar y el código de error devuelto.

4.6.3 Actualizadores

Los programas actualizadores de software o configuración están alojados en el directorio de datos que usa la aplicación web y deben aparecer en el fichero `versions.properties`. Son descargados por el cliente del agente a petición de la aplicación web cuando algún componente no coincida con el deseado y posteriormente son ejecutados.

Cuando se ejecutan en el dispositivo, los actualizadores pueden hacer uso de la información adicional que el cliente habrá guardado, así como de la funcionalidad del controlador de nodo. Normalmente, el actualizador depende de otros ficheros auxiliares, por lo que va a descargarlos de la aplicación web, comprobar que la descarga ha sido correcta usando el resumen SHA de cada fichero y copiarlos al directorio adecuado con los permisos necesarios. Los ficheros auxiliares suelen contener las bibliotecas o programas compilados para la arquitectura del procesador del dispositivo.

El actualizador más complejo es el que actualiza la configuración. Este debe descargar la configuración adecuada teniendo en cuenta el identificador del escenario y el nombre simbólico del nodo. Una vez descargada debe analizar la configuración para encontrar de qué otros ficheros depende y así también descargarlos, cambiar la dirección IPv6 si se utiliza el conector UDP y reiniciar `Node` enviando los registros anteriores.

4.7 Conclusiones

La implementación de SNSR ha requerido concretar parte de la flexibilidad de la arquitectura. Se ha tomado la decisión de implementar un nuevo NOS, diferente a los empleados

actualmente, resultando en la *Contribución 3* de esta Tesis Doctoral. Este NOS es independiente del uso de SNSR, pero se ha diseñado teniendo en cuenta exclusivamente sus requisitos. Este parte del funcionamiento de las WSN ya existentes y usa técnicas empleadas por otros protocolos (sobre todo el control de congestión en el transporte fiable de los datos) pero la implementación es completamente nueva y a lo largo del capítulo se han resaltado las diferencias (por ejemplo, el enrutamiento empleado). Destacan la posibilidad de usarlo tanto en equipos reales como virtualizados, el uso reducido de recursos conforme a las limitaciones de los nodos sensores y la posibilidad de adaptarlo a otros sistemas operativos gracias al uso del LLAL. Este NOS usa criptografía ECC y el protocolo DTLS y flexibiliza algunos comportamientos de cara a la realización de pruebas, incluyendo versiones maliciosas de las entidades.

Se ha adaptado SNSR para este nuevo NOS, la implementación se ha hecho para cada tipo de entidad, con un funcionamiento distinto, resultando en la *Contribución 4* de esta Tesis Doctoral. Se han tenido que concretar los algoritmos de cálculo de topologías, detalle y codificación de los mensajes del protocolo, trayectorias a realizar por el robot, etc. Se incluyen interfaces para interactuar con las entidades, destacando las que existen en el robot para comunicarse con el módulo de localización y movimiento, de tal manera que se pueda probar con robots reales o simulados.

En paralelo con la implementación de lo anterior, ha sido necesario crear una plataforma de pruebas tanto en escenarios reales como virtualizados, resultando en la *Contribución 5* de esta Tesis Doctoral. Se ha creado una PKI para generar claves y certificados, un sistema de gestión remota de dispositivos reales para facilitar las pruebas con distintas configuraciones y dos compositores/orquestadores de escenarios virtualizados, que hemos denominado *lanzadores* con dos formas diferentes de simular los canales de comunicación. En uno de ellos incluso se ha creado un módulo de localización y movimiento simulado que es capaz de calcular trayectorias. A lo anterior, hay que añadir una serie de programas auxiliares para poder analizar los registros, trazar los mensajes enviados, visualizar la información topológica generada, probar algoritmos de manera independiente, generar claves, certificados y CRL, etc.

Para poder comparar básicamente la envergadura de cada uno de los elementos anteriores, se muestra en la Tabla 4.7 las líneas de código que los componen. Por supuesto, no todas las líneas de código han requerido el mismo esfuerzo, pero representan el trabajo necesario en su conjunto. Las líneas de código generadas por *protobuf* a partir de la descripción del protocolo pertenecen a la implementación de SNSR, pero se muestran aparte para diferenciarlas del resto (codificadas manualmente). Todo el código ha sido probado multitud de veces a lo largo de esta Tesis Doctoral. Estas líneas no incluyen los escenarios definidos, pruebas de concepto, código descartado, etc.

Tabla 4.7 Tamaño en líneas de código de cada componente.

NOS	SNSR	*protobuf*	Gestión remota	*Lanzadores*	Test/Ataques	Auxiliares	Total
18.208	8.973	17.607	2.962	7.473	5.694	3.067	**63.984**

5 Experimentación y evaluación

Son vanas y están plagadas de errores las ciencias que no han nacido del experimento, madre de toda certidumbre.

Leonardo Da Vinci

En este capítulo se muestran los experimentos realizados para validar la arquitectura SNSR y la implementación desarrollada, así como un análisis comparativo de seguridad. Por tanto, en este capítulo se desarrolla la *Contribución 6* de esta Tesis Doctoral "*Experimentación y validación de todas las contribuciones anteriores*". Ha resultado difícil encontrar otros trabajos con los que hacer comparaciones cuantitativas debido a la novedad de esta Tesis Doctoral, por lo que la mayoría de los resultados numéricos tratan de caracterizar el funcionamiento de la arquitectura y comparar las distintas opciones que permite.

La implementación de la arquitectura completa contiene multitud de componentes, algoritmos, protocolos y herramientas que han debido ser probados individualmente previamente a los experimentos integradores. Muchas de las partes que componen SNSR se han creado con la intención de poder ser incorporadas posteriormente a futuros trabajos. En este capítulo se presentan solo experimentos de la arquitectura completa. También hay que destacar que SNSR ha pasado por varias versiones hasta llegar a la que se ha utilizado para obtener los resultados presentados, que se considera muy estable. No se recogen en este capítulo los muchos experimentos previos con versiones anteriores.

En la Sección 5.1, se muestra la configuración por defecto que se ha utilizado en los experimentos y las configuraciones específicas que se van a comparar. Dependiendo de la configuración específica, se proponen diferentes versiones de SNSR: M1, M2, M3, M4 y M5. A continuación, en la Sección 5.2, se hará un análisis cualitativo de la seguridad frente a los 33 ataques conocidos que se han considerado más importantes. En las Secciones 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 y 5.9 se describen experimentos y sus resultados. Los primeros se han realizado sin ataques, con distintas configuraciones, topologías y tamaños de red, distintos tipos de robot, y en escenarios reales y virtualizados. Los últimos se realizan con ataques para probar la robustez de SNSR.

5.1 Configuración de SNSR

La arquitectura SNSR permite una amplia flexibilidad como se demostró en la Sección 3.10, permitiendo adaptar los parámetros que definen su funcionamiento (ver Tabla 3.17). La implementación realizada soporta toda la anterior versatilidad. En la Sección 4.1 se describen las decisiones adoptadas para usar SNSR con una nueva torre de protocolos diseñada para la experimentación.

Evaluar el comportamiento de SNSR en las WSN ante la variabilidad de cada una de las opciones disponibles sería un trabajo muy extenso. Los experimentos que se han realizados se centran en los aspectos de seguridad y en el buen funcionamiento de la arquitectura usando solo un conjunto de configuraciones específicas que se van a emplear a lo largo de este capítulo.

Las configuraciones de SNSR usadas se diferencian en el modo de seguridad utilizado y en cómo se realiza la detección de baliza del robot. Los modos de seguridad están recogidos en la Tabla 3.2. Los tipos de detección de baliza aparecen en la Sección 3.6.1. Combinando las anteriores opciones se obtienen las diferentes configuraciones que se emplean para la validación de SNSR, que se muestran en la Tabla 5.1.

Tabla 5.1 Configuraciones de SNSR que se han experimentado.

Configuración	Modo de seguridad	Detección de baliza
M1	SEG_NO	Básica
M2	SEG_L	Básica
M3	SEG_L	Avanzada
M4	SEG_T	Avanzada
M5	SEG_LTE	Avanzada

El resto de los parámetros son iguales en todas las anteriores configuraciones y, a menos que se indiquen expresamente valores distintos en el experimento, tienen los siguientes valores de la arquitectura por defecto:

- DOMAIN = "dit.us.es."
- BS_TYPE = "bs", MS_TYPE = "ms", SN_TYPE = "sn"
- CIPHER_SUITES = TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- SIG_TYPE = ECC, HASH_TYPE = SHA256, MIN_KEY_LEN = 256 bits
- TMP_KEY_DUR = 600 s
- CRL = $\emptyset$ (vacía inicialmente)
- BVT = BVT_KNOWN_BS_SIGN
- B_DELAY = 0

- DEFAULT_HOP_LIMIT $= 48$
- DEFAULT_KAI $= 5\,\mathrm{s}$
- MAX_NB $= 512$
- MAX_ANOMALIES $= \infty$
- MIN_ANOM_INTERVAL $= 10\,\mathrm{s}$
- MAX_MS $= 1$
- TC_MAX $= 50\,\mathrm{ms}$
- BI_MAX $= 1\,\mathrm{s}$ (cuando el robot esté operativo)
- MS_FAST_HS =sí (sin reintentos)
- NL_ADDRESS$_{BS} = 1$, NL_ADDRESS$_{ROBOT} = 2$

Además, en las configuraciones experimentadas se adoptan las siguientes hipótesis y valores de los parámetros propios de la implementación:

- La información topológica inicial contiene una lista con los datos de los nodos desplegados autorizados.
- El alcance radio del robot y la BS es el doble que el de los nodos sensores.
- El intervalo de transmisión de medidas de los sensores es de $10\,\mathrm{s}$.
- Uso de DTLS con intervalo de reintento de mensajes de saludo aleatorio entre $1\,\mathrm{s}$ a $2\,\mathrm{s}$.
- Una MTU del LL de 1232 bytes. Este valor se ha obtenido de restar al valor mínimo de la MTU exigido por IPv6 [24] el tamaño de una cabecera IPv6 mínima (40 bytes) y una cabecera UDP (8 bytes). Esto permite asegurar el funcionamiento de escenarios simulados con un LL virtualizado sobre conexiones UDP en redes IPv6. Usando *Ethernet* o *Wi-Fi*, la MTU real es mayor, pero por homogeneidad se ha mantenido el anterior valor.

El valor de la MTU usado es superior al que existe en las WSN basadas en IEEE 802.15.4. Como se analiza en [97], DTLS 1.2 no está pensado para redes con MTU pequeñas por la sobrecarga añadida por el protocolo en el tamaño de los mensajes. De hecho, con el modo SEG_TL no se podrían usar esas redes. Esto se soluciona en la versión 1.3 de DTLS [127], pero se encuentra en borrador a fecha de la escritura de este documento y no se ha podido probar. Si se usaran MTU menores en los experimentos, afectaría negativamente al rendimiento al tener que segmentar más los mensajes, pero no influiría en la seguridad.

Además, cada entidad de la red debe tener instalado un certificado firmado por una Autoridad de Certificación (CA, *Certificate Authority*). Se ha creado una CA utilizando la herramienta *OpenSSL* [112] y claves privadas y certificados firmados por esta CA para:

- BS: 2 correctos y 2 expirados.
- Robots: 4 correctos y 2 expirados.

- Nodos sensores: 500 correctos y 12 expirados.

Esas claves privadas y certificados tienen las siguientes características:

- Empleo de criptografía ECC usando la curva *prime256v1*, también conocida como *secp256r1* o *NIST P-256* [107] con claves de 256 bits (32 bytes).
- Los certificados están firmados con el algoritmo ECDSA usando SHA1.

Las anteriores características se eligieron por estar basadas en ECC y ser las más habituales y recomendadas en el momento de creación. Sin embargo, podrían haberse usado otras con repercusión en el tiempo necesario para validar los certificados. De hecho, durante la escritura de este documento, SHA1 ya está desaconsejado porque se han conseguido generar colisiones [142].

A modo de ejemplo, en el Código 5.1 se muestra la versión textual del certificado de una BS.

Código 5.1 Ejemplo de certificado en formato texto.

```
Certificate:
 Data:
   Version: 3 (0x2)
   Serial Number: 509 (0x1fd)
   Signature Algorithm: ecdsa-with-SHA1
   Issuer: C = ES, ST = Sevilla, L = Sevilla, O = Universidad de
       Sevilla, OU = Departamento Ing. Telematica, CN = CA-ECC-
       FJFJ
   Validity
     Not Before: Aug 29 10:04:27 2019 GMT
     Not After : May 25 10:04:27 2022 GMT
   Subject: C = ES, ST = Sevilla, L = Sevilla, O = Universidad de
        Sevilla, OU = BS, CN = base1.bs.dit.us.es.
   Subject Public Key Info:
     Public Key Algorithm: id-ecPublicKey
       Public-Key: (256 bit)
       pub:
           04:58:41:bd:a5:99:ec:9a:80:ac:c3:3f:40:bd:db:
           78:9a:6a:15:14:45:5c:1c:78:98:eb:75:24:39:f2:
           11:45:74:86:64:a7:a5:f6:f7:03:93:a3:4d:fd:37:
           df:cb:d2:0a:16:2e:78:00:ca:b2:c4:0e:41:bd:ef:
           88:28:04:01:ec
       ASN1 OID: prime256v1
       NIST CURVE: P-256
   X509v3 extensions:
     X509v3 Subject Key Identifier:
       7D:73:30:79:58:E5:E0:67:34:FC:A5:74:40:0D:D7:
       6F:FB:F3:73:83
```

```
    X509v3 Authority Key Identifier:
      keyid:C7:1A:F2:6D:D0:D0:D8:68:93:FB:B6:CE:71:
      EC:AE:C8:03:77:B1:B2
Signature Algorithm: ecdsa-with-SHA1
    30:45:02:21:00:83:4d:52:3f:76:2f:bd:cb:14:e6:45:eb:f4:
    4e:7c:91:88:d9:a9:7e:3e:02:56:ea:fb:5c:c4:32:93:9f:b3:
    ba:02:20:18:e4:16:d1:40:c8:1d:78:83:f2:dc:12:34:23:6c:
    96:09:fd:86:ae:61:15:92:40:5f:cf:f3:3c:80:bf:20:54
```

5.2 Análisis cualitativo de seguridad

En esta sección se analiza cualitativamente la seguridad de SNSR frente a distintos ataques comparándola con la de otros 2 sistemas/estándares existentes:

1. ZigBee (versión PRO 2015) [8] usando el perfil *ZigBee Smart Energy* (ZSE) (versión 1.2a) [113], cuya descripción detalla puede encontrarse en la Sección 2.6. Esto admite muchas configuraciones distintas. Para evitar compararlas todas, este análisis se centrará en aquella más parecida a la arquitectura SNSR: uso de seguridad centralizada con un TC en red mallada sin balizas periódicas usando CBKE usando ECC, MQV, certificados *Elliptic Curve Qu-Vanstone* (ECQV) firmados por una CA y claves de enlace preconfiguradas con el TC. Para su análisis, además de la documentación oficial, se han usado estudios previos [170, 169, 10, 137]. Se han tenido en cuenta las actualizaciones introducidas en el estándar desde la publicación de esos estudios. Entre las similitudes con la arquitectura SNSR y las características interesantes que han llevado a elegir esta alternativa para la comparación están:
   - Es usado en instalaciones que requieren un alto nivel de seguridad. Un caso típico sería el de una red de contadores de electricidad para hogares o puntos de recarga.
   - Está disponible comercialmente.
   - Uso de un elemento centralizado, conocido como *Trust Center* (TC), usado para autorizar la incorporación de nuevos nodos a la red y para facilitar la conexión segura entre estos.
   - Uso de ECC y certificados para el establecimiento de claves entre nodos, con un protocolo de saludo parecido al descrito en esta arquitectura. Uso de CA para firmar los certificados únicos que tiene cada nodo. Posibilidad de usar el TC como generador de claves evitando tener que realizar operaciones ECC en el resto de nodos.
   - Autenticación inicial usando una clave simétrica precompartida entre cada nodo y el TC.
2. El protocolo INSENS (*Intrusion-Tolerant Routing in Wireless Sensor Networks*) [25, 26], cuya descripción detalla puede encontrarse en la Sección 2.7. Es un protocolo distribuido que trata de minimizar y tolerar los efectos que pueden provocar nodos

maliciosos dentro de una red de sensores. Existe una versión que admite varias BS, que no se contempla en este análisis. Las características que han llevado a su elección son:

- Aunque es distribuido, la mayor carga computacional la realiza la BS.
- Utiliza enrutamiento alternativo con 2 rutas desde cada nodo a la BS con distintos nodos intermedios.
- Los nodos usan claves simétricas diferentes pero precompartidas con la BS.

Para la arquitectura SNSR, y de la misma manera que con las otras alternativas, se ha supuesto el uso del modo de seguridad más elevado `SEG_TL` por defecto.

5.2.1 Ataques analizados

Se han analizado muchos de los ataques documentados en otros trabajos, en especial los descritos en [29, 30, 150, 133]. Estos ataques se pueden clasificar según distintos criterios:

- Según el nivel de la torre de protocolos al que afecta [29, 150, 133].
- Según si el atacante es interno o externo a la red [29, 150].
- Según el objetivo del ataque [29].
- Según si el ataque es activo o pasivo [29].
- Según la manera de realizar el ataque [30].

En la Tabla 5.2 se muestra el listado de los ataques analizados. Están clasificados según el objetivo del ataque, siguiendo un criterio similar al utilizado por [29]. Cuando un ataque puede ser clasificado en distintas categorías, se ha situado solo en la más habitual. Los nombres de los ataques se han mantenido en inglés para facilitar la consulta en la literatura y por homogeneidad. Para cada ataque se indica la seguridad que presentan las tres soluciones analizadas con los siguientes símbolos:

– El ataque no ha sido contemplado o la seguridad depende de otros elementos ajenos al sistema.

× El sistema no resiste al ataque en determinadas circunstancias.

⋈ El ataque puede tener éxito, pero el sistema limita sus efectos.

✓ El sistema es resistente al ataque.

Además, en la Tabla 5.2 se indica la sección donde está detallado el análisis. En cada una de las secciones se describirá brevemente el ataque y el comportamiento y la seguridad de cada alternativa.

Tabla 5.2 Resumen del análisis de ataques.

Ataque	SNSR	ZSE	INSENS	Sección
Disponibilidad de la red en PHY				
Jamming	–	×	⋈	5.2.2
Tampering	⋈	×	×	5.2.3
Node destruction	⋈	⋈	⋈	5.2.4
Disponibilidad de la red e integridad del servicio en LL				
LL collision	–	⋈	–	5.2.5
LL exhaustion	–	×	–	5.2.6
Unfairness	–	×	–	5.2.7
Denial-of-sleep	–	×	–	5.2.8
Service request power attack	✓	×	✓	5.2.9
Benign power attack	✓	×	✓	5.2.10
Disponibilidad de la red e integridad del servicio en NL y enrutamiento				
Spoofed routing information attack	✓	×	✓	5.2.11
Acknowledgment spoofing	✓	×	×	5.2.12
Routing table overflow	✓	×	✓	5.2.13
Routing table poisoning	✓	×	✓	5.2.14
Route cache poisoning	✓	×	✓	5.2.15
Rushing attack	✓	×	×	5.2.16
Hello flooding	✓	×	×	5.2.17
Black hole attack	✓	×	✓	5.2.18
Sink hole attack	✓	×	✓	5.2.19
Selective forwarding	⋈	⋈	⋈	5.2.20
Wormhole attack	✓	×	✓	5.2.21
Sybil	⋈	⋈	⋈	5.2.22
Byzantine attack	⋈	×	×	5.2.23
Disponibilidad de la red e integridad del servicio en TL				
TL flooding	✓	⋈	✓	5.2.24
TL desynchronization	✓	✓	✓	5.2.25
Privacidad y secreto				
Information disclosure	⋈	⋈	⋈	5.2.26
Eavesdropping	✓	⋈	⋈	5.2.27
Traffic analysis	⋈	×	⋈	5.2.28
Man-in-the-middle	✓	✓	×	5.2.29
Attacks on cryptographic techniques	✓	✓	⋈	5.2.30

Continúa en la siguiente página »

Tabla 5.2 (Continúa de la página anterior)

Ataque	SNSR	ZSE	INSENS	Sección
Integridad de los datos				
Node replication	⋈	⋈	⋈	5.2.31
Packet injection	✓	⋈	✓	5.2.32
Packet duplication	✓	✓	✓	5.2.33
Packet alteration	⋈	⋈	⋈	5.2.34

Leyenda: ✓resistente, ⋈ mitiga sus efectos, × no resistente, - no considerado

5.2.2 Análisis frente a *jamming*

Jamming es crear interferencias en PHY. Se puede hacer en una única frecuencia, en múltiples frecuencias simultáneamente o conmutando rápidamente, etc. Todo esto da lugar a multitud de variantes de ataques (*spot jamming*, *sweep jamming*, *barrage jamming*, *selective jamming*, *constant/deceptive/random/reactive jamming*, *interrupt/activity/scan/pulse jamming*, *blind/intelligent jamming*, *sniper jamming*, etc.).

SNSR No contemplado. Dependerá del nivel físico utilizado. Se detectaría una pérdida de conectividad en una zona de la red, ya que no habría conexión con vecinos y/o la BS. Se envían notificaciones de esas anomalías.

ZSE Usa el estándar IEEE 802.15.4 que es vulnerable en distintos grados a esos ataques.

INSENS No se fija un PHY concreto, pero se hicieron pruebas utilizando un dispositivo *mica2*, con *TinyOS* que usa el estándar IEEE 802.15.4. Este estándar es vulnerable a casi todos los tipos de *jamming* en diferentes grados. Sin embargo, hace experimentos para calcular cuantos nodos se quedan bloqueados debido a un ataque de X nodos con rango de acción Y (múltiplo de la distancia máxima entre nodos) para demostrar que la robustez aumenta con este protocolo.

5.2.3 Análisis frente a *tampering*

Tampering es la manipulación intencional de nodos. Esto incluye el reemplazo parcial o total del hardware o software de los nodos.

SNSR No se requieren nodos *anti-tampering*. Se podría obtener la clave privada del nodo, y las claves simétricas usadas con el resto de los vecinos. Si el proceso es tan rápido como para que no se corten las comunicaciones (según el tiempo de *keepalive*) sería difícil detectarlo hasta que el nodo hiciese algún comportamiento extraño. Si es lento, el resto de los nodos detectarían el fallo de comunicación más o menos simultáneamente gracias al proceso de *keepalive*.

ZSE No hay ninguna medida de protección estandarizada. Un nodo normal guarda la clave de red, el código de instalación (diferente al del resto de los nodos), la *Trust Center Link Key*, la *Application Link Key* y todos los parámetros asociados

a esas claves. Toda la anterior información caería en manos del atacante. El acceso a la clave de red permite descubrir servicios, enviar comandos, etc. Si el nodo afectado fuera el TC, toda la red completa se vería comprometida y prácticamente se tendría acceso a todos los demás nodos.

INSENS Igual que en el resto, no se contemplan medidas específicas para evitar la manipulación, y se considera que este paso es necesario para mucho de los demás ataques. Tampoco propone medidas para detectar el ataque. Este ataque podría obtener la clave del nodo compartida con la BS.

5.2.4 Análisis frente a *node destruction*

Es la destrucción total de un nodo.

SNSR Las comunicaciones con el nodo destruido se interrumpen. El algoritmo de enrutamiento no está definido en la arquitectura, pero en la implementación actual se están calculando todos los posibles caminos alternativos a la BS. El resto de los nodos que tienen caminos alternativos para llegar a la BS no se ven perjudicados. La elección del camino se hace a nivel local, no global. Si se realiza de nuevo una fase de *establecimiento de la topología*, la nueva versión de las tablas de rutas contemplaría la eliminación del nodo, por lo que se limitarían los efectos del ataque.

ZSE Si el nodo es el TC, la red cae completamente. Si es un *router*, los nodos finales deben buscar un nuevo nodo padre y se pierden los mensajes que estaban pendientes. En el caso de destruir un nodo final, solo afecta a los nodos que tengan comunicación con él en la capa de aplicación. Las rutas se vuelven a recalcular. Se usa una variante del protocolo de enrutamiento AODV que calcula rutas bajo demanda. Da lugar a un incremento del tráfico de la red por este último motivo.

INSENS Con una BS se generan dos rutas alternativas desde cada uno de los nodos. Hace un estudio de cuántos nodos se quedan bloqueados según el número de nodos que fallan (o son destruidos). También se pueden restablecer las rutas haciendo una nueva iteración del protocolo.

5.2.5 Análisis frente a *LL collision*

En *LL collision* un atacante transmite cuando otro nodo está también transmitiendo para ocasionar la corrupción de las tramas.

SNSR No está contemplado en la arquitectura.

ZSE No se puede evitar, pero el estándar IEEE 802.15.4 tiene asentimientos en el LL que pueden mitigarlo.

INSENS No está contemplado.

5.2.6 Análisis frente a *LL exhaustion*

LL exhaustion consiste en la manipulación de las medidas de eficiencia del protocolo para hacer que los nodos consuman más.

SNSR No se contempla ninguna medida de eficiencia en el LL, ya que solo se añaden medidas de seguridad.

ZSE Este ataque puede provocar retransmisiones, insertar tráfico redundante o inútil, y generar tráfico mal formado que obligue a hacer cálculos criptográficos para verificar su validez. También se pueden solicitar múltiples asociaciones hasta que un *router* no acepte más.

INSENS No está contemplado.

5.2.7 Análisis frente a *unfairness*

Unfairness consiste en transmitir cuando el medio está libre para evitar que otros puedan hacerlo.

SNSR No hay control sobre el medio y, por tanto, no ha sido contemplado.

ZSE Si se utilizaran balizas, el ataque tendría éxito fácilmente, porque se saben los intervalos de tiempo que cada nodo puede usar. Con el mecanismo *Guaranteed Time Slot* (GTS) que garantiza intervalos concretos a nodos es más fácil aún, ya que se podrían solicitar todos los intervalos para impedir su uso. Además, se pueden modificar los tiempos de espera (*backoff time*) para que el atacante sea el primero que transmita.

INSENS No está contemplado.

5.2.8 Análisis frente a *denial-of-sleep*

Denial-of-sleep, también conocido como *Sleep Deprivation Torture* (SDT), es un ataque donde se intenta evitar que los nodos entren en modo hibernación (*sleep*).

SNSR No hay contemplada ninguna medida de este tipo (hibernación o suspensión). Depende de los niveles PHY y LL usados.

ZSE Si se consigue la clave de red, es posible suplantar al *router* padre de un nodo final. A continuación, se puede hacer pensar al nodo final que ha recibido muchos mensajes “indirectos”. El nodo final intentará recuperarlos, consiguiendo que se mantenga despierto el mayor tiempo posible.

INSENS No está contemplado.

5.2.9 Análisis frente a *service request power attack*

Service request power attack es un subtipo de ataque SDT que consiste en enviar peticiones válidas constantemente con el consecuente consumo de recursos. Aunque se han englobado dentro de los ataques del LL, también se puede utilizar en otros niveles.

SNSR Los nodos actualmente solo se comunican con los vecinos para reenviar mensajes de/desde la BS. Para establecer la conexión con un vecino es obligatorio tener una dirección de enlace autorizada, el resto de las peticiones se descartan. Los nodos sí tratan de conectarse a las balizas de los robots. Un atacante podría transmitir balizas falsas para forzar intentos de conexión. Sin embargo, los intentos fallidos se notificarían como anomalías por lo que sería detectado y neutralizado. Se puede saturar el número máximo de conexiones soportadas por el robot, enviando múltiples peticiones de conexión, por lo que, temporalmente, otros nodos no podrían conectarse a él. El ataque es local y solo mientras el robot permanezca en la zona del atacante. Este ataque no afectaría a conexiones ya establecidas en la red, que seguirían funcionando con normalidad, y sería detectado con facilidad.

ZSE Si no se tiene la clave de red, es posible enviar intentos de asociación y todos los tipos de mensajes *join*/*rejoin*. Si se dispone de la clave de red, entonces se pueden hacer múltiples peticiones de descubrimiento, solicitudes de *Application Link Keys* constantes, solicitar el envío de múltiples mensajes indirectos de modo que no haya capacidad para los verdaderos, solicitudes de *Routing Request*, envío a grupos inexistentes para provocar muchos mensajes de difusión, etc.

INSENS Establece un límite máximo de mensajes por segundo que un nodo puede transmitir para evitar ataques de este tipo.

5.2.10 Análisis frente a *benign power attack*

Benign power attack es un subtipo de SDT y se basa en solicitar operaciones válidas con alto consumo a los nodos de manera continuada. Como en *service request power attack* se puede aplicar en distintos niveles.

SNSR Este ataque podría dirigirse a varias operaciones, pero siempre sería detectado y notificado. Primero, si la difusión por inundación con validación por firma digital está activa, un nodo podría enviar mensajes de difusión falsos a sus vecinos, obligando a verificar la firma digital. Los nodos vecinos lo detectarían como anomalía y sería fácil imponer un límite al número de fallos a partir del cual se empiecen a descartar directamente. Segundo, un nodo malicioso puede forzar la reconexión de sus vecinos, para repetir el proceso de saludo, que también se detecta como anomalía. Por último, un atacante podría simular ser un robot para que los nodos se intenten conectar a él, pero la conexión fallaría durante la autenticación y se reportaría como anomalía.

ZSE El comportamiento es similar que en *service request power attack*, pero en este ataque se eligen las operaciones de mayor consumo, por ejemplo, solicitudes de nuevas claves, envío de mensajes encriptados, etc.

INSENS Lo más dañino sería conseguir enviar mensajes de difusión, pero estos están protegidos al usar autenticación con una cadena de *hash*.

5.2.11 Análisis frente a *spoofed routing information attack*

Spoofed routing information attack, también conocido como *direct attack on routing information*, es la transmisión de información de encaminamiento falsa, alterada o repetida.

SNSR Solo un robot o la BS con claves privadas y certificados válidos, que se suponen confiables, puede transmitir información de encaminamiento. El robot se comunica directamente con cada nodo por lo que un nodo no puede dar información falsa de los demás. Por tanto el ataque no es posible.

ZSE Si no se dispone de la clave de red, no es posible. Con la clave de red, el ataque es posible ya que el encaminamiento está basado en AODV y todos los nodos colaboran para determinar la mejor ruta. Un nodo malicioso puede alterar las rutas fácilmente.

INSENS La BS calcula todas las rutas con la información obtenida de los nodos y configura los nodos. Los nodos pueden tratar de mentir sobre quiénes son sus vecinos, pero para eso deben modificar el mensaje de difusión reenviado por esos vecinos. Lo anterior es improbable, porque ese mensaje lleva un *hash* calculado usando la clave secreta que cada nodo vecino comparte solo con la BS.

5.2.12 Análisis frente a *acknowledgment spoofing*

Acknowledgment spoofing es enviar asentimientos falsificados para difundir información errónea de otros nodos. Aunque está clasificado como ataque del NL, también es aplicable a otros niveles.

SNSR Esto solo sería posible si los datos se enviaran en claro, circunstancia que no se está considerando en este análisis. En este caso un nodo intermedio podría asentir un segmento del TL sin pasar el mensaje a su verdadero origen. Con seguridad, que es el caso contemplado, el nodo malicioso debe tener las claves de sesión para crear paquetes falsificados, que es imposible si el proceso de saludo es seguro. Tampoco se pueden duplicar paquetes ya transmitidos porque se descartan automáticamente.

ZSE El estándar IEEE 802.15.4 admite asentimientos protegidos o no. En este último caso el ataque es posible. Suponiendo que todos los asentimientos van protegidos (con autenticación y/o cifrado), el éxito de este ataque depende de las claves que tenga el atacante.

INSENS No existen mensajes de asentimiento como tal, pero sí otros que tienen la misma funcionalidad. Este ataque se puede realizar durante el descubrimiento de vecinos (durante el envío del mensaje de petición de ruta) para simular vecindades inexistentes como fase previa a un ataque *Sybil*. El mensaje de *feedback* no podría utilizarse para este ataque porque está protegido al ir cifrado.

5.2.13 Análisis frente a *routing table overflow*

Routing table overflow consiste en anunciar rutas inexistentes para llenar la tabla de rutas de los nodos y evitar que se puedan añadir otras rutas reales.

SNSR Solo un robot o la BS con claves privadas y certificados válidos, que consideramos entidades seguras, pueden transmitir información de encaminamiento. No es posible hacerlo desde otra entidad, imposibilitándose el ataque.

ZSE Este ataque es posible si se dispone de la clave de red. El atacante haría múltiples peticiones de *Route Request* y simularía la respuesta, para que los nodos tengan que guardarla.

INSENS Las tablas de rutas las determina la BS, y son enviadas encriptadas a cada nodo. Por consiguiente, no es posible este ataque.

5.2.14 Análisis frente a *routing table poisoning*

El objetivo del ataque *routing table poisoning* es modificar rutas enviando actualizaciones de estas a otros nodos.

SNSR Solo un robot o la BS con claves privadas y certificados válidos, que consideramos entidades seguras, pueden transmitir información de encaminamiento. No es posible hacerlo desde otra entidad, por lo que es insensible a estos ataques.

ZSE No se pueden enviar actualizaciones de rutas directamente, pero sí se informa periódicamente a los vecinos del estado del enlace con el comando *Link Status*. En este último comando viaja información de la calidad percibida por un vecino respecto a otro, y esta información se utiliza posteriormente para calcular la mejor ruta. Todo esto requiere disponer de la clave de red.

INSENS No es posible porque las tablas de rutas las determina la BS, y son enviadas encriptadas a cada nodo.

5.2.15 Análisis frente a *route cache poisoning*

Route cache poisoning es similar a *routing table poisoning* pero para protocolos reactivos (*on-demand*).

SNSR Solo un robot o la BS con claves privadas y certificados válidos, que consideramos entidades seguras, pueden transmitir información de encaminamiento. No es posible hacerlo desde otra entidad, por lo que es insensible a este ataque.

ZSE Se usa un protocolo reactivo (*on-demand*). Si se envían *Route Reply* falsos, los receptores se lo creerán sin más. Por tanto es vulnerable.

INSENS Las tablas de rutas son calculadas por la BS, y son enviadas cifradas a cada nodo. Por tanto, no es posible este ataque porque un atacante no puede modificar las rutas que utilizan los nodos.

5.2.16 Análisis frente a *rushing attack*

En *rushing attack* un atacante transmite paquetes más rápido que los nodos legítimos, a veces suplantando la identidad de estos. Los paquetes de los nodos legítimos son identificados como duplicados y descartados (si se usa *duplicate supression*). Este ataque se puede utilizar en otros niveles.

SNSR No afecta con la implementación actual. Si se usaran otros protocolos de enrutamiento existentes y no se emplease cifrado, dependería del protocolo de encaminamiento usado. En el caso evaluado, si un atacante simula ser un robot y los nodos se han configurado para conectarse a solo un robot, el nodo malicioso puede intentar transmitir balizas antes de que llegue el robot correcto para obligar que los nodos intenten conectarse a él y no al correcto. La conexión falla durante la autenticación y el ataque es detectado. Con el método de detección de baliza básico es posible que el nodo nunca se conecte al robot correcto mientras dure el ataque. Con el método de detección de baliza avanzado, estadísticamente el ataque no evitaría la conexión con el robot correcto.

ZSE Este ataque puede tener éxito en varios contextos: a) al unirse un nuevo nodo a una red, respondiendo rápidamente; b) si el nodo malicioso está dentro de la red, puede afectar a los mensajes de difusión ya que se transmiten con un identificador de transacción; y c) se puede falsificar información de descubrimiento o balizas también.

INSENS El envío del mensaje de solicitud de ruta es vulnerable a este ataque, ya que el campo usado para verificar la autenticidad no va cifrado y los nodos no pueden saber si el mensaje es el original.

5.2.17 Análisis frente a *hello flooding*

Hello flooding consiste en transmitir mensajes de descubrimientos a nodos lejanos para que estos intenten contactar sin éxito.

SNSR Los nodos no utilizan mensajes de descubrimiento de otros nodos, pero sí del robot. Se pueden utilizar balizas para hacer este ataque. Los nodos intentan establecer comunicación con el robot simulado sin éxito, con reintentos, por lo que se desperdician recursos. Esto es detectado como anomalía y notificado, por lo que SNSR es robusto a este ataque.

ZSE Se pueden transmitir constantemente peticiones de asociación si están permitidas o intentos de reasociación (*rejoin*) segura. Si el atacante tiene las claves de red puede hacer aún más daño.

INSENS Este ataque es realizable durante el envío del mensaje de solicitud de ruta. Un nodo usa el origen del primer mensaje de solicitud de ruta recibido como nodo padre al que enviar los mensajes de *feedback*. Si el canal es asimétrico o el origen del primer mensaje está falsificado, los mensajes de respuesta se perderán.

5.2.18 Análisis frente a *black hole attack*

En un *black hole attack*, un atacante hace que otros nodos enruten su tráfico a través de otro nodo que lo descarta. Falsamente se anuncian buenas rutas.

SNSR Los nodos no anuncian rutas, por lo cual no pueden favorecer rutas que pasen por ellos. Así que, no es posible este ataque con la implementación actual.

ZSE Este ataque es posible si se consigue la clave de red.

INSENS Solo la BS envía información de encaminamiento y es la que decide las rutas, luego imposibilita este tipo de ataques.

5.2.19 Análisis frente a *sink hole*

Sink hole attack hace que los nodos encaminen su tráfico con destino al sumidero (u otro destino habitual) hacía un nodo malicioso. Es muy parecido al ataque anterior.

SNSR Es insensible a este ataque, ya que los nodos no anuncian rutas en la implementación actual.

ZSE Este ataque es posible si se consigue la clave de red.

INSENS Solo la BS envía información de encaminamiento y es la que decide, luego imposibilita este tipo de ataques.

5.2.20 Análisis frente a *selective forwarding*

Selective forwarding, también conocido como *gray hole attack*, ocurre cuando un nodo intermedio malicioso reenvía algunos mensajes y otros los descarta, incumpliendo el protocolo.

SNSR Las comunicaciones con el robot son directas, por lo que no hay intermediarios y no se puede realizar este ataque. En otras comunicaciones, esto es posible. Si se descartan pocos mensajes las comunicaciones se degradan, pero gracias a los reintentos se mantienen. Si la conexión se pierde, se detectaría la anomalía. Es difícil distinguir si hay un nodo malicioso o son problemas de comunicaciones. Habría que analizar la fiabilidad de cada camino y elegir los más fiables probando cada cierto tiempo las distintas alternativas, aunque esto no se ha implementado. Por tanto, SNSR tiene alta robustez a estos ataques, pero también se podrían integrar otros métodos para detectarlos, como por ejemplo [54].

ZSE Se utilizan reintentos, así que esto se vería mitigado, sobre todo en el AL, donde el cifrado es mayor. No utiliza rutas alternativas.

INSENS La BS establece 2 rutas alternativas para los mensajes de un nodo a la BS, intentando que estas rutas no compartan ningún nodo de paso. Al usar las dos rutas diferentes simultáneamente hay más probabilidad de que lleguen los mensajes.

5.2.21 Análisis frente a *wormhole attack*

En un *wormhole attack* se crea un túnel entre dos nodos para que otros piensen que son la mejor opción para llegar al otro extremo.

SNSR Si el robot detecta la posición de los nodos, es imposible que se dé este caso, ya que se crea un mapa global de la red de sensores. Es el robot o la BS quien decide las rutas.

ZSE Es posible si se consigue la clave de red.

INSENS Está protegido mediante códigos de autenticación de mensajes con clave anidados.

5.2.22 Análisis frente a *Sybil*

Sybil es cuando un nodo actúa como si fuera muchos nodos (identidad múltiple).

SNSR Para que esto ocurra el atacante debe hacerse con las claves privadas y certificados de varios nodos (y destruirlos para que el robot no detecte duplicados), ya que se verifica que los certificados están firmados por la autoridad certificadora. Si los nodos originales habían sido detectados previamente, el robot detectaría que estos nodos se han movido de posición, notificando esta anomalía. Si no habían sido detectados previamente, en la fase de *descubrimiento* el robot detectaría que hay varios nodos que están ocupando la misma posición y esto podría ser interpretado como indicador de ataque. El peor escenario se daría si la existencia de varios nodos en la misma posición es algo tolerado, por lo que solo podría ser detectado si más adelante el atacante comienza a comportarse maliciosamente. Los certificados comprometidos se pueden revocar y/o excluir en la siguiente configuración, eliminándose al atacante de la red. En conclusión, se limita bastante la posibilidad de éxito del ataque y se mitigan sus efectos.

ZSE Es necesario al menos conocer la clave de red. En este caso se puede utilizar el procedimiento *secure rejoin* para incorporarse a la red que no requiere de autenticación del TC. En este último caso se podría operar en el NL, pero no en AL que requiere cifrado, ya que para esto es necesario ponerse en contacto con el TC que detectaría el acceso no autorizado al no disponer de certificados válidos. Si se disponen de certificados válidos, entonces sería posible.

INSENS La BS solo acepta aquellos nodos con los cuales comparte una clave secreta. No es posible introducir nuevos nodos sin que la BS lo descubra finalmente. Pero se pueden añadir durante el inicio cuando se descubren vecinos para engañar a los nodos. Simulando múltiples identidades la memoria disponible para vecinos puede agotarse, por lo que el nodo no acepta nuevos vecinos legítimos.

5.2.23 Análisis frente a *Byzantine attack*

En un *Byzantine attack* un nodo o un grupo de nodos trabajan de manera coordinada para hacer los anteriores ataques en el NL, simultánea o aleatoriamente. Se considera uno de los ataques más difícil de detectar.

SNSR Los efectos son locales, ya que los nodos no pueden coordinarse para cambiar las rutas a nivel global y el robot comprueba sus certificados previamente. Si provocan anomalías en el sistema, estas pueden analizarse para intentar identificar la causa.

ZSE En este caso se amplifican los efectos descritos en anteriores ataques

INSENS Pueden provocar efectos globales si se colocan estratégicamente.

5.2.24 Análisis frente a *TL flooding*

TL flooding busca agotar la memoria de los nodos estableciendo muchas conexiones en el TL.

SNSR Los nodos no aceptan conexiones entrantes inicialmente (ellos inician la conexión a la BS y al robot). El robot solo establece conexiones de TL con entidades que previamente han establecido una conexión de LL. La BS solo establece conexiones con nodos previamente autorizados. Los intentos de establecer conexiones no autorizadas serían descartados. Por consiguiente, este ataque no tendría éxito.

ZSE Es necesario que varios nodos maliciosos estén ya unidos a la red y autenticados por el TC. En este caso estos pueden solicitar claves de aplicación con el nodo al que se quiere atacar para llenar sus tablas de claves. También se puede saturar la tabla de transacciones de difusión, pero esto es en el NL. Si los nodos soportan múltiples mensajes de aplicación fragmentados, se pueden saturar enviando muchos mensajes fragmentados a la vez. Por tanto, el ataque puede tener éxito, aunque con efectos limitados.

INSENS En el TL los nodos solo hablan con la BS con la cual comparten claves, luego este ataque no es posible.

5.2.25 Análisis frente a *TL desynchronization*

TL desynchronization consiste en la falsificación de mensajes de control y números de secuencia para bloquear la comunicación entre dos nodos.

SNSR Con seguridad en el TL no es posible este ataque. Además, los números de secuencia son inicialmente aleatorios.

ZSE Se utiliza cifrado que impide la falsificación.

INSENS Los nodos solo se comunican con la BS con la cual comparten claves. Los mensajes van cifrados y su integridad se verifica, así que este ataque sería complicado en el TL. Por tanto, no es vulnerable.

5.2.26 Análisis frente a *information disclosure*

Information disclosure ocurre cuando un nodo comprometido envía datos confidenciales a otros nodos no autorizados.

SNSR Un nodo tiene acceso a: claves públicas de otros nodos (esto no es problemático), conocimiento limitado de la red (tabla de rutas), todo lo que transmitan sus vecinos si el LL no estuviera encriptado (que no es el caso), mensajes reenviados si el TL no estuviera encriptado (tampoco es el caso) e información de certificados revocados. Por tanto, un nodo comprometido no puede obtener datos confidenciales de otros nodos y los efectos de este ataque son muy limitados.

ZSE Disponiendo de la clave de red se puede obtener toda la topología de la red y la configuración y servicios soportados por cada nodo. Las claves de aplicación no se podrían capturar en principio, ya que son únicas. Por tanto, los efectos de este ataque serían limitados.

INSENS Un nodo podría comunicar las claves que guarda (clave con la BS, los MAC enviados por los vecinos y sus identidades, los números de secuencia de los mensajes de petición de ruta, rutas e información de los mensajes transmitidos sin cifrar. Por tanto, los efectos de este ataque serían limitados.

5.2.27 Análisis frente a *eavesdropping*

Eavesdropping o *passive monitoring* es observar la información transmitida de manera pasiva por lo que no se puede detectar.

SNSR Usando un cifrado no vulnerable no es posible este ataque. No hay vulnerabilidades conocidas del cifrado usado en las configuraciones contempladas, por lo que este ataque no obtendría ninguna información útil.

ZSE Para que este ataque tenga éxito es necesario hacerse previamente con las claves de red y/o aplicación para poder capturar algo. Al inicio se supone que las claves iniciales se han introducido en el TC fuera de banda por lo que habría que hacerse con los códigos de instalación iniciales para poder capturar la clave de red. La clave de aplicación es imposible de capturar si se usan certificados. Con la clave de red se puede capturar toda la información a este nivel y muchos comandos del AL que no van encriptados con las claves individuales.

INSENS Se utiliza cifrado simétrico en la comunicación con la BS (clave individual compartida entre cada nodo y la BS), pero no para todos los campos de los mensajes, algunos de los cuales pueden utilizarse para realizar otros ataques. No especifica la seguridad que incorporan los mensajes de datos durante la operación normal de la red. Así que el ataque tendría éxito parcialmente.

5.2.28 Análisis frente a *traffic analysis*

En *traffic analysis* se analiza el tráfico de la red para obtener información de su topología y de las acciones que son realizadas.

SNSR Usando cifrado en el LL se puede analizar el intercambio de paquetes de red que pasen por un nodo infiltrado, las direcciones origen y destino, y el tamaño de los paquetes. Al usarse cifrado en el TL el contenido de los segmentos de este nivel no es accesible. Por tanto, solo se puede analizar información local de la red de nodos comprometidos previamente, lo que limita los efectos de este ataque.

ZSE Con la clave de red, prácticamente se puede analizar todo el tráfico, ya que lo único que no se podría ver en claro son los datos en el AL (pero sí se sabría qué servicios se están utilizando). Por tanto, es vulnerable a este ataque.

INSENS Se puede analizar el intercambio de mensajes en una zona y ver las rutas que siguen los mensajes y el tamaño de estos. Por tanto, se podría aprender parte de las tablas de enrutamiento y por consiguiente parte de la topología de la red. También se puede identificar al nodo padre escogido por cada nodo para la transmisión del mensaje de *feedback*. Los efectos de este ataque son limitados.

5.2.29 Análisis frente a *man-in-the-middle*

Man-in-the-middle es un ataque donde un nodo malicioso se sitúa entre otros dos, haciendo pasar toda la comunicación entre ellos por él sin que sean conscientes. Es muy peligroso en la fase de intercambio de claves.

SNSR Se usa DTLS que no es vulnerable a este ataque, ya que se verifican los certificados. Todas las comunicaciones están autenticadas.

ZSE Todas las claves importantes van cifradas con la clave de enlace única, que un tercero no debería conocer, por lo que no es vulnerable.

INSENS Solo la comunicación de los nodos con la BS incorpora autenticación usando la clave simétrica compartida. La comunicación entre nodos es vulnerable a este ataque, y esto podría afectar a la topología estimada por la BS.

5.2.30 Análisis frente a *attacks on cryptographic techniques*

Son ataques a los algoritmos de cifrado, funciones *hash* o protocolos de intercambios de claves.

SNSR La arquitectura es flexible a la hora de elegir los algoritmos. Si en el futuro se descubren fallos en un algoritmo, se puede utilizar otro. Por tanto, SNSR es robusto a estos ataques y evita la obsolescencia futura por este motivo.

ZSE Todos los algoritmos criptográficos son seguros a la fecha de escritura de este documento.

INSENS Para el cálculo del MAC, la cadena de *hash* unidireccional, y resto de funciones de cifrado se usa RC5 que puede ser vulnerable o no según los parámetros que se usen. En la implementación descrita en el trabajo usan una configuración vulnerable: RC5 con 12 rondas y con bloques de 64-bit. Esa configuración es susceptible a un ataque diferencial usando 2^{44} textos planos elegidos [13].

5.2.31 Análisis frente a *node replication*

Node replication, también conocido como *node cloning*, consiste en capturar un nodo y copiar toda su información para crear y desplegar réplicas en la red. Si el nodo es reemplazado por una sola réplica este ataque sería virtualmente igual al de *tampering*. En este punto analizaremos solo el caso en el que se crean múltiples copias maliciosas.

SNSR Los nodos que sean vecinos del nodo clonado verán las múltiples copias como un único nodo. Al resto de nodos no les afectaría, ya que, aunque una réplica se moviera a otra zona, los nodos próximos no se comunicarían con ella porque solo se comunican con los nodos asignados. El robot se da cuenta de este ataque durante la fase de *descubrimiento* al tener varias posiciones para un mismo nodo. Se genera una anomalía y el nodo (y sus réplicas) son excluidos en la fase de *establecimiento de la topología*. Por consiguiente, el efecto del ataque sería muy limitado.

ZSE Este ataque puede tener éxito, pero todas las réplicas se detectarían como una sola, y posiblemente solo una pueda recibir la *Trust Center Link Key*, ya que el TC solo asigna una. También se dispone de un sistema de detección de direcciones duplicadas por lo que al final para que el ataque funcione habría que asignar una dirección diferente a cada clon, que podría dar lugar a que se detectaran, porque el certificado será el mismo en todos.

INSENS Este ataque puede tener éxito, sobre todo en la fase inicial. Sin embargo, la BS recibiría varios mensajes válidos diferentes con un mismo identificador de origen y, aunque los autores no lo consideran, podría aplicar alguna política de exclusión.

5.2.32 Análisis frente a *packet injection*

Packet injection o *packet spoofing* es la inyección de paquetes falsos en la red.

SNSR Con autenticación y cifrado no es posible este ataque, los paquetes falsos son descartados.

ZSE Los mensajes en el NL y superiores están protegidos. Solo los mensajes de unión a la red, sobre todo en la fase inicial, no lo están y se pueden inyectar. Si se consigue la clave de red el ataque tendría mayores consecuencias, ya que esta clave es compartida por todos los nodos y se puede falsificar cualquier paquete. En el AL, este ataque no es posible a priori. Por tanto, es posible este ataque aunque con efectos limitados.

INSENS Todos los mensajes enviados requieren estar en posesión de una clave conocida por la BS y esta descarta cualquier mensaje incorrecto. Solo se permite iniciar difusión a la BS, que es la única capaz de generar un número de secuencia válido. Por tanto, no es posible realizar este ataque.

5.2.33 Análisis frente a *packet duplication*

En *Packet duplication* se reenvían paquetes previamente capturados.

SNSR Con autenticación y cifrado no es posible este ataque. DTLS incluye un número de secuencia en cada mensaje para evitar duplicados. Además, los mensajes de aplicación también llevan un identificador único creciente pero no consecutivo. Los mensajes duplicados son descartados.

ZSE Se utilizan contadores en todos los paquetes para evitar duplicados.

INSENS Todos los mensajes transmitidos para establecer la red llevan datos únicos, por lo que se descartan duplicados.

5.2.34 Análisis frente a *packet alteration*

Durante la ejecución de un ataque *packet alteration* se interceptan paquetes, se modifican y se envían de nuevo.

SNSR Para evitarlo, hay que utilizar una *cipher suite* en DTLS que proteja la integridad del contenido. Actualmente, la cabecera del nivel de red no va protegida por lo que un nodo intermedio puede modificarla antes de reenviarla. Sin embargo, con TL protegido, en el destino el descifrado/verificación fallaría. Un nodo infiltrado puede modificar el enrutamiento de un mensaje cambiando cualquier campo de la cabecera, provocando la pérdida de paquetes. Las comunicaciones con el robot son inmunes a este tipo de ataques al producirse entre entidades vecinas.

ZSE El ataque es posible si se tienen las claves adecuadas, de lo contrario dependerá del tipo de seguridad utilizado (si incluye código de autenticación o no).

INSENS La alteración sería detectada, ya que todos los mensajes llevan un campo MAC. Un ataque de este tipo haría que los paquetes se descartaran, pero obligaría a consumir recursos para verificarlos.

5.2.35 Conclusiones del análisis

La arquitectura SNSR no contempla medidas en los niveles PHY y LL. Hay muchos ataques de estos niveles que no se pueden valorar. Si se tuviera en cuenta los niveles utilizados durante las pruebas algunos resultados serían similares a los de ZSE y INSENS.

Hay ataques que no son evitables en ninguna de las alternativas, y la diferencia es si se limitan los daños producidos y cómo. Debido a la poca información que controlan los nodos sensores y al hecho de que ninguna clave se comparte, el análisis sugiere que SNSR es capaz de controlar mejor estos ataques.

En el NL y en el enrutamiento, ZSE se comporta peor que los otros, principalmente por usar claves compartidas de red y construir las rutas usando un protocolo distribuido. Un nodo malicioso que ya pertenezca a la red puede realizar muchos ataques. INSENS se comporta bastante bien, gracias al cálculo centralizado de 2 rutas empleando nodos intermedios diferentes. La implementación de SNSR también tiene en cuenta rutas alternativas, pero sin considerar que los nodos intermedios tengan que ser todos diferentes.

La comunicación directa del robot con los nodos, la observación real de la topología y el cálculo centralizado permiten a SNSR superar muchos de los ataques.

En ataques de privacidad, secreto e integridad, todas las alternativas resisten relativamente bien. Sin embargo, el uso de autenticación desde el LL y la no compartición de ninguna clave hacen que SNSR sea más robusto en ataques como *eavesdropping*.

Hay que destacar que ZSE e INSENS están probados en nodos sensores con muy pocos recursos, menores a los usados en SNSR. Además, el hecho de no necesitar un componente auxiliar como es el robot puede hacerlos una mejor opción para determinados escenarios.

Este análisis muestra que la arquitectura SNSR es bastante robusta. La implementación se puede mejorar utilizando técnicas específicas para evitar aquellos ataques donde la resistencia no es total, como en *selective forwarding*, *traffic analysis* o *packet alteration*. Esto se deja para futuras mejoras.

5.3 Evaluación en laboratorio en ausencia de ataques

Durante el desarrollo se han utilizado multitud de escenarios con configuraciones diferentes hasta comprobar que todo funciona correctamente. Los resultados presentados en este capítulo emplean la implementación final. Esta y las siguientes secciones muestran la evaluación de SNSR en condiciones normales sin ataques, en escenarios reales y virtualizados (usando los 2 *lanzadores* descritos en la Sección 4.5.2). En la Sección 5.9 se muestran los experimentos realizados con ataques.

El primer experimento del funcionamiento completo con dispositivos reales fue llevado a cabo en el laboratorio. Como nodos sensores se utilizan dispositivos *Raspberry Pi 1 Model B*, con el sistema operativo Linux, como se muestra en la Figura 5.1. Cada equipo incluye un adaptador Wi-Fi USB a 54 Mbps con soporte de modo *ad hoc*. Se usaron distintos modelos y hay que destacar que no todos los adaptadores Wi-Fi que están en el mercado soportan este modo. La BS se implementó con un ordenador portátil con una CPU

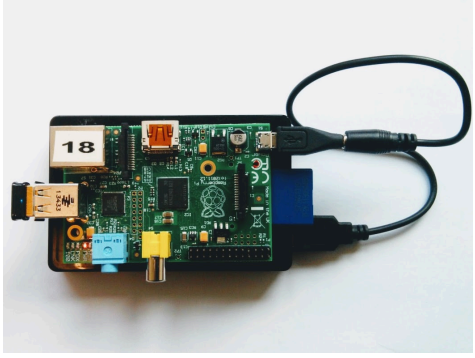


Figura 5.1 *Raspberry Pi 1 Model B* usada como nodo sensor.

i7-6500U con 2 núcleos, 4 GB de RAM y 30 GB de disco duro. Se usó un robot terrestre *Pioneer 3AT* que, además de llevar un *netbook* para realizar el seguimiento de trayectorias y localización, incorporaba otra *Raspberry Pi 1 Model B* con el software de SNSR. Este

robot se muestra en la Figura 5.2. El robot localiza a los nodos y la BS integrando medidas

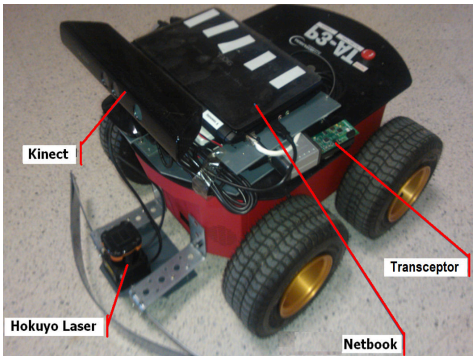


Figura 5.2 Robot *Pioneer 3-AT* empleado en experimentos.

de banda ultraancha (UWB, *Ultra-WideBand*) usando técnicas de filtros de partículas [96], que provee un error de localización suficientemente bajo tanto en interiores como exteriores. El robot se mueve siguiendo trayectorias previamente programadas a una velocidad de 0.6 m/s e incluye auto-localización LIDAR usando el algoritmo AMCL [149]. Las técnicas empleadas para la localización y el seguimiento de rutas no han sido desarrolladas en esta Tesis Doctoral, solo se ha proporcionado una interfaz para poder recibir las medidas obtenidas externamente.

Se ha utilizado el sistema de gestión remota descrito en la Sección 4.6 para configurar y reiniciar los nodos y recopilar los registros generados en cada experimento.

Aunque se han hecho pruebas con distintos modos de seguridad, aquí solo se mostrarán los datos obtenidos usando la configuración *M5*. El experimento se repitió 100 veces y los datos usados son los siguientes:

- Entidades: 23 nodos sensores, 1 robot y 1 BS.
- Conector usado: `ConnectorEthernet`, usando la dirección MAC del adaptador Wi-Fi. Se han desactivado los servicios del sistema operativo no necesarios que den lugar a tráfico no deseado.
- Disposición: los nodos se dispusieron en una cuadrícula cuadrada de 3x8, con una separación de 3 m, y la BS se situó en la posición (0,0).
- Alcance radio máximo de las antenas y criterio de separación entre vecinos máxima utilizados durante el *establecimiento de la topología*: con la disposición anterior el rango de comunicación estimado fue de 3.5 m. Se adoptó una distancia máxima entre vecinos conservativa de 5 m, de modo que los nodos que deben ser vecinos por su posición real se asignaran como tales incluso con los posibles errores de localización.
- Información de topología de la red inicial: CRL vacía y se incluye la lista de nodos autorizados, que son los 23 nodos existentes, e información de la BS.

Respecto a los parámetros comunes que se describieron en la Sección 5.1, la única diferencia es que el número de saltos máximos en los paquetes (`DEFAULT_HOP_LIMIT`) está configurado a 18 (ajustado a las dimensiones de la red).

A continuación, se describe un experimento. Se obtuvieron resultados similares en todos los experimentos realizados.

Primero, se activó la fase de *descubrimiento*. El robot tardó 75,23 s en realizar automáticamente la trayectoria de 43,33 m mostrada en la Figura 5.3. El robot descubrió todos los nodos y los localizó con un error medio de 0,83 m. En la Figura 5.3 se muestra la posición de los nodos medida, que no es exactamente la real. Después, el robot calculó las rutas y generó la nueva información topológica en 1,68 ms, que fue enviada a la BS. En promedio cada nodo necesitaba 4,09 saltos para alcanzar a la BS. La asignación de vecinos resultante se muestra con líneas (continuas y discontinuas) en la Figura 5.3. El criterio de asignación de vecinos identificó erróneamente como vecinos algunos nodos cuya distancia era en realidad superior al rango de comunicación. Estos errores serán detectados posteriormente como anomalías en la fase de *mantenimiento* y corregidos.

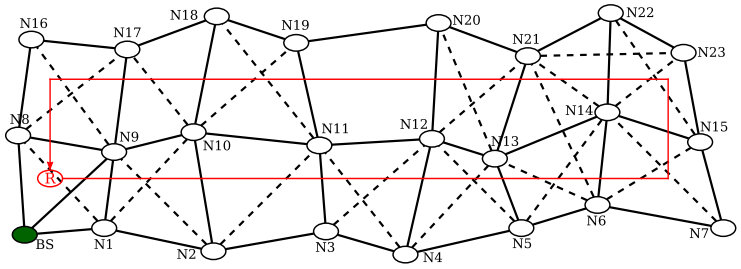


Figura 5.3 Nodos descubiertos y topología de red del experimento en laboratorio.

En la fase de *configuración*, el robot siguió la misma trayectoria. Cada nodo midió el tiempo transcurrido desde la recepción de la petición de configuración hasta que obtuvo la primera respuesta de *PING* (que se usa para simular el envío de medidas) de la BS, que denominaremos tiempo de configuración. La Figura 5.4 representa el tiempo en las 100 repeticiones del experimento (percentil 25, 75 y mediana). Los tiempos medios estuvieron comprendidos entre 1,5 y 3,8 s, con un promedio de 2,96 s. En la posición inicial del robot, el robot configuró varios nodos simultáneamente, causando colisiones y requiriendo reintentos de establecimiento de conexión, que causó tiempos mayores de configuración. En todos los experimentos realizados los tiempos de configuración fueron siempre inferiores a 10 s.

En la Figura 5.5 se representan los datos transmitidos y recibidos en el LL en bytes para cada uno de los nodos hasta que recibieron la primera respuesta de *PING*. En promedio cada nodo recibió 12,50 kB y transmitió 10,25 kB. El tráfico es mayor en los nodos cercanos a la BS debido al reenvío de paquetes. En todos los experimentos ejecutados, el tráfico BS nunca fue mayor de 46 kB. El proceso de saludo seguro fue responsable de gran parte del tráfico. Cada saludo en los nodos tardó en promedio 0,97 s y transfirió $\sim$1100 bytes en cada sentido (el 60 % de él debido a los certificados). Un nodo lleva a cabo

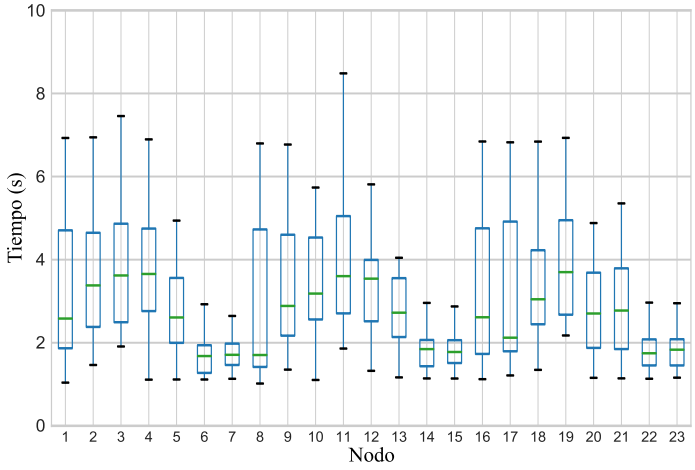


Figura 5.4 Experimento real en laboratorio: tiempo requerido para configuración de cada uno de los nodos.

saludos solo con el robot, la BS y sus vecinos, por lo que implica tiempos de configuración y tráfico debido al saludo bajos incluso en redes grandes.

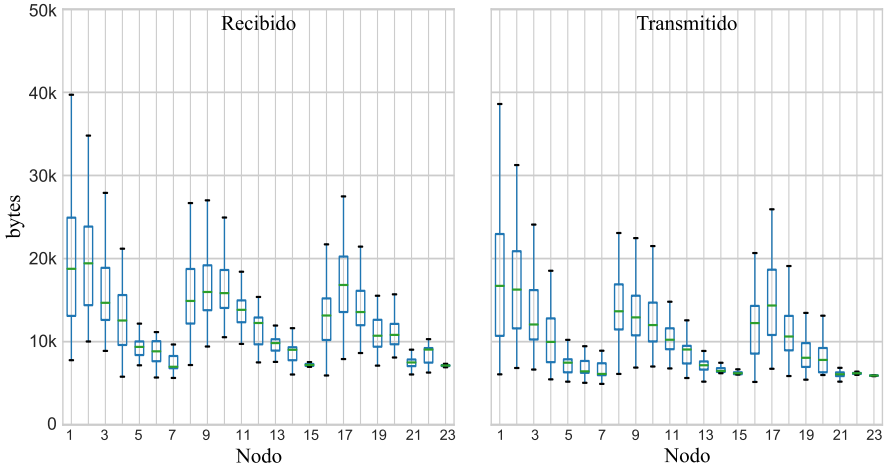


Figura 5.5 Experimento real en laboratorio: datos trasmitidos y recibidos en LL hasta que cada nodo recibe el primer asentimiento de medida de la BS.

A continuación, se pasó a la fase de *mantenimiento*. Cada nodo intentó conectarse a sus vecinos autorizados y a la BS, pero varias conexiones entre los nodos no se consiguieron establecer (mostradas con líneas discontinuas en la Figura 5.3). Estos fallos de comunicación fueron detectados como anomalías y reportados a la BS, por lo que se decidió realizar una nueva fase *descubrimiento*. En la siguiente fase de *establecimiento de la topología*, se usaron las anomalías de conectividad y el estado de los nodos para

corregir la topología. La topología actualizada, que se muestra en la Figura 5.3 con líneas continuas, ya no contenía errores. Las rutas fueron recalculadas, los nodos reconfigurados en la siguiente fase y no se detectaron más errores de conexión. Se utilizó un criterio no restrictivo para asignar vecinos para explotar la autocorrección de SNSR. Muchos métodos de seguridad obtienen información topológica a través de la interacción de nodos, que es propensa a sufrir ataques. SNSR utiliza un criterio simple basado en la ubicación de los nodos (evitando la interacción de los nodos y, por lo tanto, mejorando la seguridad en las etapas de configuración de la red), y luego corrige los errores de topología y se autoadapta, utilizando el robot.

Durante la experimentación, en varias ocasiones algunos nodos se apagaron al acabarse la batería, comprobándose como los nodos que lo usaban de intermediario detectaban su caída generando la consiguiente anomalía y utilizaban rutas alternativas. Si se repetía el establecimiento de la red, el nodo caído era eliminado de la configuración. Estos experimentos no han sido tenidos en cuenta para la elaboración de los resultados anteriores ni contabilizados, para mantener la homogeneidad. Sin embargo, nos han permitido comprobar la robustez de SNSR.

5.4 Comparación de las configuraciones de SNSR experimentadas

En este experimento, que vamos a denominar S1, se compara el funcionamiento de las distintas configuraciones de SNSR mostradas en la Tabla 5.1 en el mismo escenario. Para este experimento se emplea `launchermn` (Sección 4.5.2.2).

Todos los experimentos virtualizados, incluyendo los de las siguientes secciones, se han realizado en condiciones similares. Se han ejecutado en sistema operativo Ubuntu 18.04 corriendo en un equipo con una CPU Intel® Xeon® E5-2603 v4 con 12 núcleos a 1.70GHz. Además, suponen que los errores de localización son despreciables. Para ello, se utilizan valores de alcance radio máximo de las antenas, un criterio de separación máxima entre vecinos y factores de seguridad en el cálculo de las trayectorias del robot, de tal manera que los errores de localización obtenidos experimentalmente sean despreciables a la hora de calcular la topología y las comparaciones que se van a realizar no se vean afectadas por este componente aleatorio. En todos los experimentos virtualizados sin ataques presentados, la configuración inicial ha tenido éxito y todos los nodos sensores se han comunicado con la BS usando la primera configuración, de tal manera que no se ha necesitado realizar una nueva iteración por las diferentes fases para alcanzar un estado de red sin anomalías.

Los datos del escenario S1, son los siguientes:

- Entidades: 48 nodos sensores, 1 robot y 1 BS.
- Repeticiones del experimento: 100 veces cada configuración.
- Conector usado: `ConnectorEthernet`. Se ha asignado una dirección MAC única a cada entidad.
- Alcance radio máximo de las antenas: $50\,\mathrm{m}$ en los nodos y $100\,\mathrm{m}$ en el robot y la BS.

- Criterio de separación entre nodos máxima utilizado durante la asignación de vecindades: $35\,\mathrm{m}$.
- Disposición: los nodos y la BS se disponen en una cuadrícula cuadrada en un área de $210\,\mathrm{m}^2$ a nivel del suelo separados por $35\,\mathrm{m}$ en horizontal y vertical como se muestra en la Figura 5.6. La posición del nodo situado en la esquina inferior izquierda es $(0,0)$ y la del nodo en la esquina superior derecha la $(210,210)$. En esta figura y en las siguientes figuras donde se muestran topologías de escenarios, la BS se muestra en rojo y el color de los nodos es diferente según los saltos necesarios para comunicarse con la BS. Suponemos que el robot vuela a una altura constante de $5\,\mathrm{m}$ e inicialmente se encuentra próximo a la BS. Con esta disposición, el número de saltos medio que debe realizar un paquete desde un nodo a la BS es $3{,}5$ y el número medio de vecinos que tiene cada nodo es de $3{,}4$.

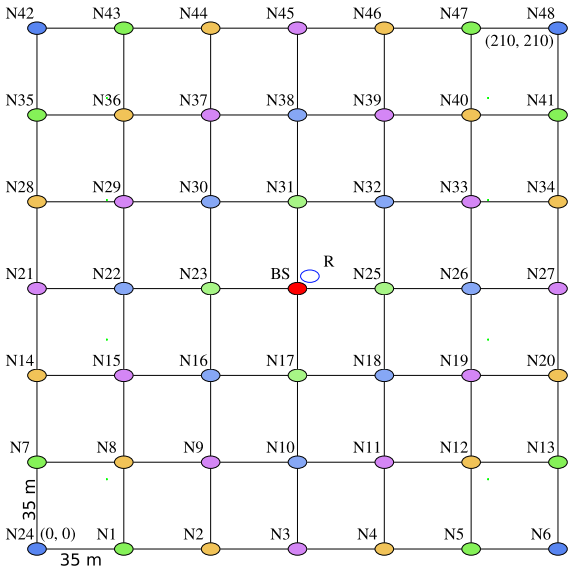


Figura 5.6 Topología del escenario del experimento S1. La BS se muestra en rojo y el color de los nodos depende de la distancia a la BS.

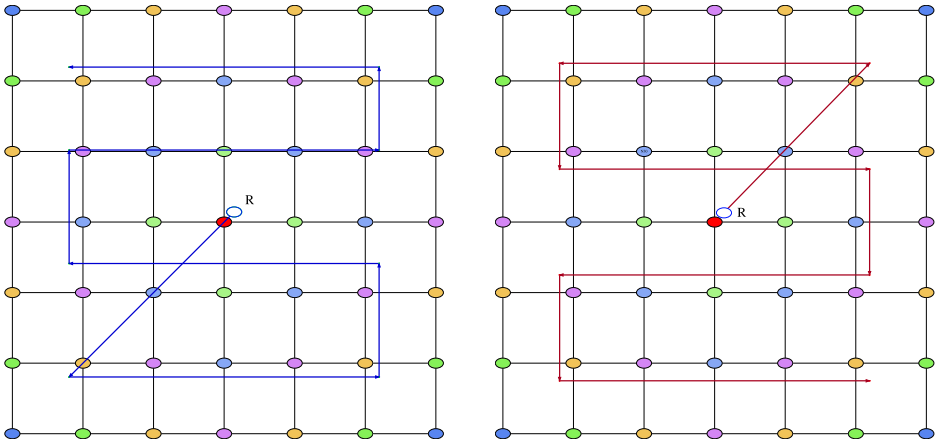
- Factor de seguridad adicional, usado para el cálculo de trayectorias del robot, σ: 0.8 (Sección 4.1.13).
- Información de topología de la red inicial: CRL vacía y se incluye la lista de nodos autorizados, que son los 48 nodos existentes, e información de la BS.

A continuación, se describe el desarrollo del experimento, que es similar para cada una de las configuraciones.

Al inicio del experimento, los nodos comienzan en modo de funcionamiento sin configuración recibida y la BS con la información de topología inicial cargada localmente. En las configuraciones *M2*, *M3* y *M5*, el robot inicia en el estado `NONE` (ver Figura 4.16). En

las configuraciones *M1* y *M3*, como no hay autenticación en el LL que permita distinguir a la BS, el robot inicia en el estado `DISCOVER` con la información de topología inicial cargada localmente.

Durante la fase de *descubrimiento*, la trayectoria seguida por el robot hasta que descubre todos los nodos se muestra en color azul en la Figura 5.7a, según el algoritmo descrito en la Sección 4.1.13. El robot inicialmente se encuentra en una posición cercana a la BS. La trayectoria tiene una longitud de $885{,}048\,\mathrm{m}$ y se recorre a una velocidad de $6\,\mathrm{m/s}$, con lo que se tarda en recorrer $147{,}508\,\mathrm{s}$. En cada cambio de dirección se verifica si se han descubierto todos los nodos y en caso afirmativo se cancela la trayectoria. Al terminar, el robot entra en la fase de *establecimiento de la topología* y vuelve a una posición cercana a la BS (ese trayecto no se representa en la figura). En todas las repeticiones del experimento, la trayectoria de descubrimiento se ha realizado por completo y al finalizar el $100\,\%$ de los nodos han sido descubiertos.



(a) Trayectoria de descubrimiento. **(b)** Trayectoria de configuración.

Figura 5.7 Trayectoria de descubrimiento (a) y configuración (b) del experimento S1. La BS se muestra en rojo y el color de los nodos depende de la distancia a la BS.

En la fase de *establecimiento de la topología*, el tiempo que necesita el robot para calcular la nueva información de topología es prácticamente similar en cada repetición, siendo su valor medio $12{,}3386\,\mathrm{ms}$. Después, el robot comparte con la BS la nueva versión de la información topológica correctamente.

Durante la fase de *configuración*, la trayectoria que sigue el robot se muestra en color rojo en la Figura 5.7b, según el algoritmo descrito en la Sección 4.1.13. Tiene una longitud de $880{,}985\,\mathrm{m}$ y se recorre a una velocidad de $6\,\mathrm{m/s}$, con lo que se tarda en recorrer $145{,}831\,\mathrm{s}$. En cada cambio de dirección se comprueba si se han configurado todos los nodos y en caso afirmativo se cancela la trayectoria. Sin embargo, en todas las repeticiones se ha realizado por completo. Al terminar la trayectoria mostrada, el robot verifica la configuración de los nodos que no han enviado la confirmación de configuración con un mensaje de tipo

`TM_CONFIGURE_RES` antes de perder la conexión con el robot. Para ello se aproxima a ellos, en orden de mayor a menor proximidad y pregunta por su estado. El robot no tuvo que verificar la configuración de ningún nodo tras terminar la trayectoria en la mayoría de los casos. Sin embargo, en la configuración *M1*, en 18 repeticiones de las 100 el robot tuvo que verificar la configuración de un único nodo (N37 en 8 repeticiones y N46 en 10), que se había configurado correctamente, pero cuyo mensaje de respuesta no llegó al robot. En la configuración *M4* también ocurrió lo anterior en 17 repeticiones (N37 en 7 repeticiones y N46 en 10). Al final, en todas las repeticiones del experimento, el 100 % de los nodos han recibido la nueva configuración correctamente.

Para finalizar el experimento, tras verificarse que todos los nodos han recibido la configuración, el robot vuelve cerca de la BS (esto no está representado en las trayectorias mostradas) y se espera como máximo 300 s a que todos los nodos reciban una respuesta de *PING* desde la BS. Los nodos envían la primera petición de *PING* a la BS justo después de finalizar el proceso de saludo con ella. En todas las repeticiones se ha comprobado que el 100 % de los nodos se configuran correctamente y establecen comunicación con los vecinos asignados y con la BS.

Para comparar las distintas configuraciones se han utilizado las siguientes métricas:

1. Datos transmitidos y recibidos en el LL por un nodo en bytes desde el comienzo hasta recibir de la BS la primera respuesta a un mensaje *PING*. Para que se reciba esta respuesta, un nodo debe haber recibido y aplicado previamente la primera configuración. Esto es una medida del coste de los protocolos usados y de la seguridad empleada.
2. Tiempo requerido por un nodo para dar por completada la primera configuración recibida, desde que se recibe la petición de configuración hasta recibir la primera respuesta de *PING* de la BS.

Se ha medido el instante de tiempo en el que cada nodo ha recibido la petición de configuración y en el que se recibe la primera respuesta de *PING* de la BS. En este último instante también se recogen las estadísticas de tramas totales enviadas y recibidas.

Para la métrica 1 (datos transmitidos y recibidos en el LL), se ha calculado el valor medio y la desviación estándar de los bytes recibidos y transmitidos por cada nodo en las 100 repeticiones para cada una de las configuraciones. Esto aparece representado en la Figura 5.8 y los resultados son los esperados.

La configuración *M1* es el caso con valores más bajos. Esto es debido a que no utiliza el saludo avanzado y los datos no van protegidos. El saludo avanzado completo supone la transmisión de unos 2230 bytes (en torno a 1090 bytes transmitidos por el servidor y 1140 por el cliente) de los cuales unos 610 bytes se utilizan para transmitir el certificado de la entidad en cada sentido. El saludo básico solo utiliza 1 byte en cada sentido. Así, a mayor número de saludos avanzados a realizar, mayor utilización de ancho de banda. La diferencia entre los datos transmitidos y recibidos se debe principalmente a la petición de configuración que reciben los nodos (que de media tiene un tamaño de 531 bytes cuando no hay seguridad en el LL y 549 bytes cuando sí hay), retransmisiones y a la recepción de balizas del robot. En todas las configuraciones se observa una desviación estándar pequeña, lo que indica que los datos son fácilmente reproducibles y el protocolo

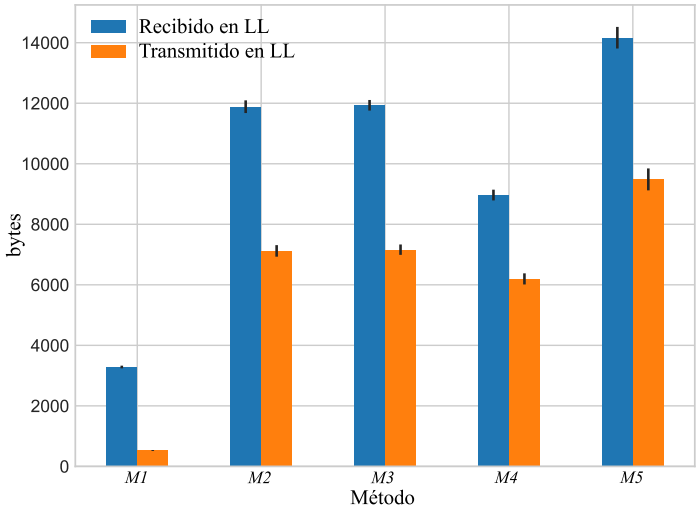


Figura 5.8 Experimento S1: datos transmitidos y recibidos en LL.

es estable. Las configuraciones *M2* y *M3* prácticamente presentan los mismos resultados, por lo que la detección de baliza empleada no afecta si no hay ataques ni congestión en la red. La configuración *M5* es la que requiere el intercambio de más datos, debido a que también tiene que realizar el saludo avanzado con la BS en el TL, cuyos datos deben ser retransmitidos en cada salto (3,5 saltos de media).

Para el segundo punto, se ha calculado el valor medio y la desviación estándar del tiempo que ha requerido completar la configuración para cada nodo en las 100 repeticiones para cada una de las configuraciones. Esto aparece representado en la Figura 5.9. Los datos

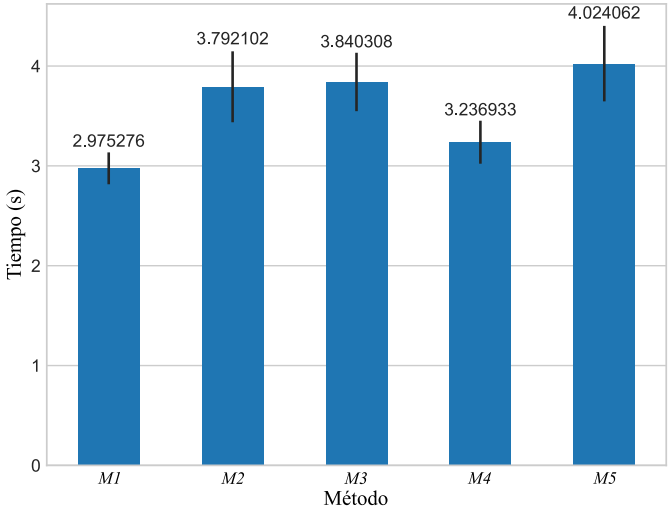


Figura 5.9 Experimento S1: tiempo requerido para configuración.

mostrados en la anterior figura muestran un patrón similar a la Figura 5.8 por los mismos motivos pero con menores diferencias entre las distintas configuraciones. La diferencia entre el más rápido y más lento es tan solo de $1{,}05\,\mathrm{s}$.

5.5 Comparación según la topología de la red

En este experimento, que denominaremos S2, se han utilizado los mismos datos del escenario del experimento S1 para la configuración *M5* a excepción de la disposición. Ahora, en la misma área de $210\,\mathrm{m}^2$, el emplazamiento de los nodos y la BS es aleatorio, aunque garantizando que el grafo de la topología resultante sea fuertemente conexo y, por tanto, que exista conectividad de red entre cualquier par de entidades. Se han realizado 100 repeticiones con disposiciones aleatorias. La trayectoria de descubrimiento es igual en todos los escenarios. La trayectoria de configuración es diferente ya que se minimiza el recorrido según la posición de los nodos y la BS. Los datos obtenidos se van a comparar con los resultados obtenidos en el experimento S1 para comprobar el efecto de la topología en los resultados.

En la Figura 5.10 y la Figura 5.11 se muestran los datos obtenidos para las mismas métricas utilizadas en la Sección 5.4 junto con los datos obtenidos en el experimento S1 para la configuración *M5*.

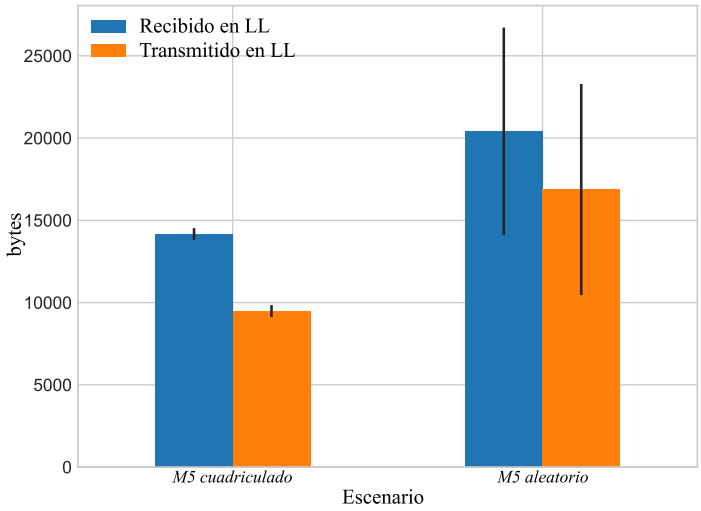


Figura 5.10 Experimento S2: comparación de los datos transmitidos y recibidos en LL entre disposiciones cuadriculadas y aleatorias.

Se observa un aumento de los valores medios y, como era de esperar, de la desviación típica. En la Figura 5.12 y la Figura 5.13 se muestran los datos desglosados por cada uno de los escenarios aleatorios (en el mismo orden secuencial).

Analizando los escenarios donde los valores divergen más de la media, se observa que los resultados dependen mucho de la trayectoria de configuración y el orden en el que se

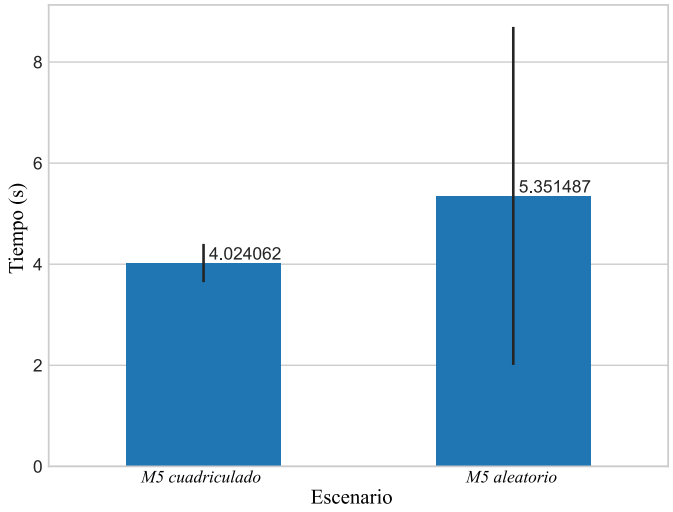


Figura 5.11 Experimento S2: comparación del tiempo requerido para configuración entre disposiciones cuadriculadas y aleatorias.

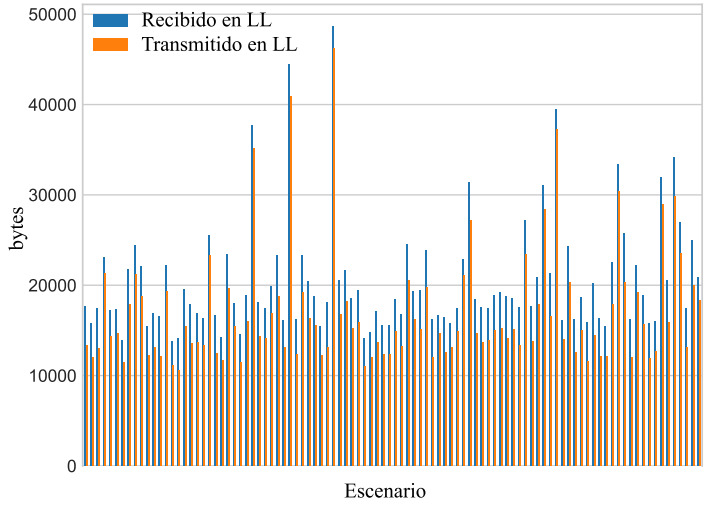


Figura 5.12 Experimento S2: datos transmitidos y recibidos en LL en cada escenario.

envía la configuración a los nodos.

Los casos donde se han transmitido más datos son debidos a la existencia de bucles no detectados por el transmisor durante el periodo transitorio. Esto da lugar a que un mismo mensaje se transmita tantas veces como el número máximo de saltos permitidos (48 por defecto). Esto se podría limitar personalizando este número máximo de saltos según la red, por ejemplo, utilizando un valor dos veces el diámetro del grafo de la topología.

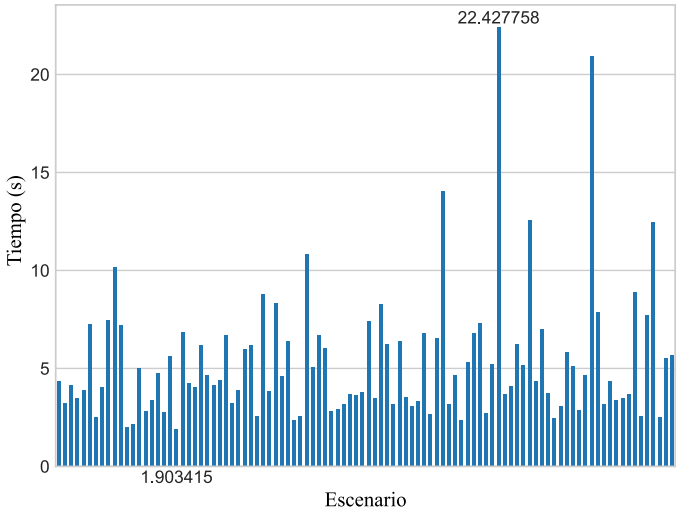


Figura 5.13 Experimento S2: tiempo requerido para configuración en cada escenario.

Los casos donde el tiempo de configuración es mayor son debidos a que se han configurado nodos alejados de la BS antes que los nodos que les sirven como intermediarios para llegar a ella. Hasta que exista una ruta para llegar a la BS la configuración no se da por terminada y se siguen haciendo reintentos. Este efecto se puede minimizar, analizando la dependencia de unos nodos respecto de otros para comunicarse con la BS y calculando una trayectoria de configuración que permita configurar aquellos nodos que sirven como intermediarios primero.

En cuanto a los datos transmitidos, se observa que ningún escenario aleatorio presenta mejoras respecto al cuadriculado. Sin embargo, en cuanto al tiempo de configuración si hay escenarios aleatorios que lo mejoran. El tiempo mínimo observado es de tan solo $1{,}9\,\mathrm{s}$.

También se ha medido el tiempo que ha necesitado el robot para calcular la topología de la red. Esto se representa en la Figura 5.14. El resultado es conforme al algoritmo usado para el cálculo de rutas descrito en la Sección 4.1.7.2. A mayor número de conexiones existentes (aristas del grafo de la red) mayor tiempo necesario para realizar todos los cálculos.

5.6 Comparación según el tamaño de la red

En este experimento, que denominaremos S3, se crea una red de sensores con 500 nodos y se comparan los datos obtenidos con los de los experimentos S1 y S2. No se puede utilizar `launchermn` debido a que el número de nodos es superior al que soporta *WmediumD* en la maquina empleada para simulaciones. Se utiliza `launcher` con la consecuente pérdida de realismo en la simulación de los canales y el movimiento del robot, como se describe en la Sección 4.5.2.1. Se usa la configuración *M5*.

A continuación, se describen otros datos del escenario del experimento S3:

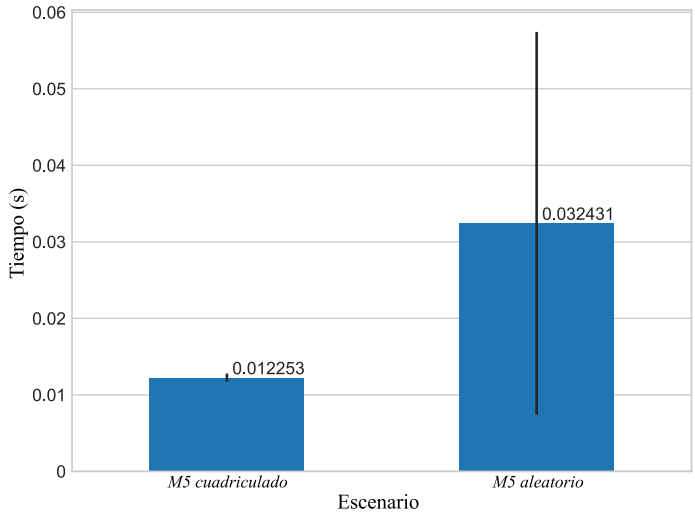


Figura 5.14 Experimento S2: tiempo requerido para calcular la topología de la red.

- Entidades: 500 nodos sensores, 1 robot y 1 BS.
- Repeticiones del experimento: 100 veces.
- Conector usado: `ConnectorUDP`. Se ha asignado un puerto UDP único a cada entidad. Se han creado 5 zonas de difusión, $Zi, 1 \leqslant i \leqslant 5$, cada una formada por 100 nodos con numeración consecutiva. La BS pertenece a la Z_3. Esto se muestra en la Figura 5.15. Se simula un retardo de transmisión para todos los mensajes de $10\,\mathrm{ms}$.
- Radio de alcance inalámbrico máximo de las antenas: igual que en los experimentos S1 y S2 ($50\,\mathrm{m}$ en los nodos y $100\,\mathrm{m}$ en el robot y la BS).
- Criterio de separación entre nodos máxima utilizado durante la asignación de vecindades: igual que en los experimentos S1 y S2 ($35\,\mathrm{m}$).
- Disposición: los nodos y la BS se disponen en una cuadrícula triangular en un área de $660\,\mathrm{m}^2$ a nivel del suelo separados por $30\,\mathrm{m}$ en horizontal y vertical como se muestra en la Figura 5.15. La BS aparece en color rojo situada en mitad del área. Con esta disposición, el número de saltos medio que debe realizar un paquete desde un nodo a la BS es $8{,}83$ y el número medio de vecinos que tiene cada nodo es de $5{,}644$.
- Trayectoria de descubrimiento: el robot descubre las entidades de una zona simultáneamente. Se simula que el robot se mueve a posiciones virtuales tales que solo se puede comunicar con las entidades pertenecientes a una zona de difusión. El robot inicia en $Z3$ y para el descubrimiento conmuta cada $30\,\mathrm{s}$ de zona en el siguiente orden: $Z1 \Rightarrow Z2 \Rightarrow Z3 \Rightarrow Z4 \Rightarrow Z5$. Al finalizar vuelve a $Z3$.
- Trayectoria de configuración: conmuta de zona cada $10\,\mathrm{s}$ o $30\,\mathrm{s}$ según haya visitado ya la zona o no respectivamente, siguiendo la secuencia: $Z3 \Rightarrow Z2 \Rightarrow Z1 \Rightarrow Z2 \Rightarrow Z3 \Rightarrow Z4 \Rightarrow Z5$. Al finalizar vuelve a $Z3$.

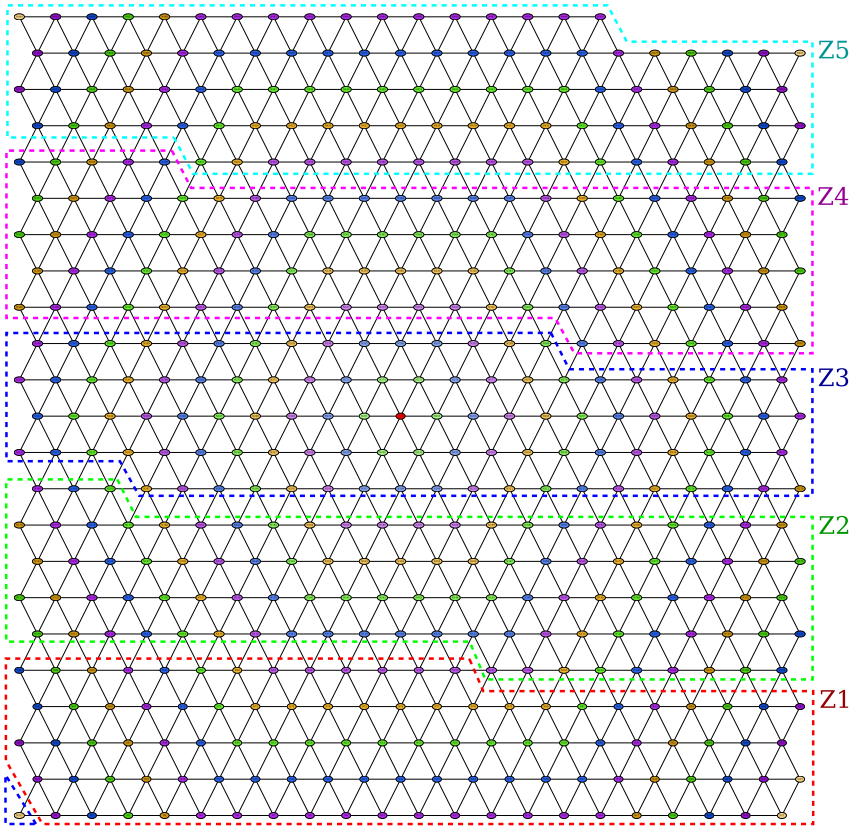


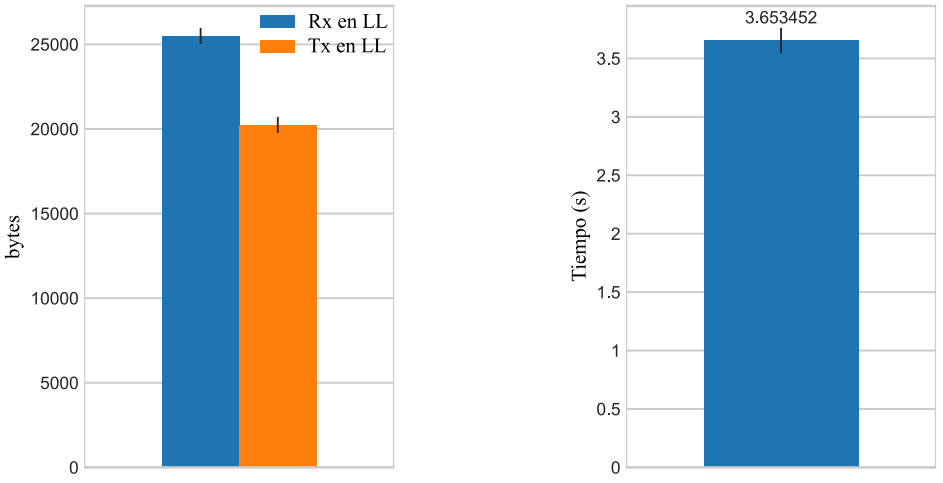
Figura 5.15 Topología del escenario del experimento S3. La BS se muestra en rojo y el color de los nodos depende de la distancia a la BS.

- Información de topología de la red inicial: CRL vacía y se incluye la lista de nodos autorizados, que son los 500 nodos existentes, e información de la BS.
- Inicio del escenario: los nodos comienzan en modo sin configuración recibida y la BS con la información de topología inicial cargada localmente. El robot inicia en el estado `NONE` (ver Figura 4.16).
- Fin del escenario: se siguen recogiendo datos del escenario hasta que transcurran 145 s del último cambio de posición del robot.

Tras la ejecución de todas las repeticiones se obtienen los siguientes resultados comunes:

1. El 100 % de los nodos han sido descubiertos, han recibido la nueva configuración y establecen comunicación con todos los vecinos asignados y con la BS. La petición de configuración tiene un tamaño medio de 2966,872 bytes. La versión calculada de la información de la topología de la red completa, `NetworkData` tienen un tamaño de 1517816 bytes cuando es serializada.

2. El valor medio y la desviación estándar de los datos transmitidos y recibidos en el LL por cada nodo hasta recibir la primera respuesta de la BS se muestra en la Figura 5.16a. El tiempo requerido se muestra en la Figura 5.16b.



(a) Datos transmitidos y recibidos en LL.

(b) Tiempo requerido para configuración.

Figura 5.16 Experimento S3: datos transmitidos y recibidos en LL y tiempo requerido para configuración.

3. El tiempo necesario para calcular la nueva información de topología es similar en cada repetición del experimento, siendo su valor medio $40{,}27\,\mathrm{s}$.

4. El robot comparte con la BS la nueva versión de la información topológica correctamente.

De los anteriores resultados hay que destacar el buen funcionamiento y que los datos son acordes al resto de experimentos. Los datos transmitidos aumentan, proporcionalmente al número de vecinos que también aumenta. El tiempo de computación se incrementa mucho debido al aumento significativo de las aristas del grafo y de la cantidad de caminos posibles que hay que calcular.

Se observa una disminución en el tiempo de configuración, comparándolo con los obtenidos en los experimentos S1 y S2. Esto se explica en la manera de simular el movimiento del robot. En una única posición el robot es capaz de configurar simultáneamente muchos más nodos que en casos anteriores. Además el orden en el que se configuran permite que todos los nodos puedan comunicarse con la BS rápidamente.

En este experimento también se observa cómo el robot es capaz de soportar un gran número de conexiones simultáneas. En este escenario, se conectan hasta 101 entidades simultáneamente, pero también se han hecho experimentos hasta el máximo de lo que está configurado (512) sin ningún problema encontrado.

5.7 Experimentos de campo con robot terrestre

Los experimentos de campo (fuera del laboratorio) iniciales han sido llevados a cabo con los mismos dispositivos usados en la Sección 5.3. Es decir, un robot terrestre *Pioneer 3-AT* en lugar de un robot aéreo y los nodos sensores son dispositivos *Raspberry Pi 1 Model B*. Cada equipo incluye un adaptador Wi-Fi USB con soporte de modo *ad hoc*.

Se utilizaron 12 nodos sensores en escenarios que emulaban una red de monitorización de puentes. En la Figura 5.17 se señala la localización de tres nodos situados en la superficie del puente. La principal diferencia respecto al escenario de la Sección 5.3 es que ahora no

Figura 5.17 Nodos sensores en un escenario de monitorización de un puente.

se contaba con los módulos de seguimiento de trayectorias y localización del robot. El movimiento del robot se realiza manualmente y no se realiza el cálculo de la topología de cada experimento durante su ejecución, sino que esta se proporciona al robot durante la fase de *establecimiento de la topología* administrativamente.

Se han diseñado 12 escenarios diferentes, donde los 12 nodos están desplegados en diferentes posiciones. La topología esperada en cada uno de estos escenarios se muestra en la Figura 5.18. En la anterior figura no se muestra el robot, la BS se muestra en rojo y el color de los nodos es diferente según los saltos necesarios para comunicarse con la BS. No se muestra la posición exacta de cada nodo, ya que en este caso no se encuentran todos en un plano a nivel del suelo, sino que algunos están colocados sobre la estructura del puente tridimensionalmente.

Se ha utilizado el sistema de gestión remota descrito en la Sección 4.6 para configurar y reiniciar los nodos y recopilar los registros generados en cada experimento.

Aunque se han hecho pruebas con distintos modos de seguridad, aquí solo se mostrarán los datos obtenidos usando la configuración *M5*. Los siguientes son otros datos de los experimentos:

- Entidades: 12 nodos sensores, 1 robot y 1 BS.
- Repeticiones del experimento: 3 veces cada topología.

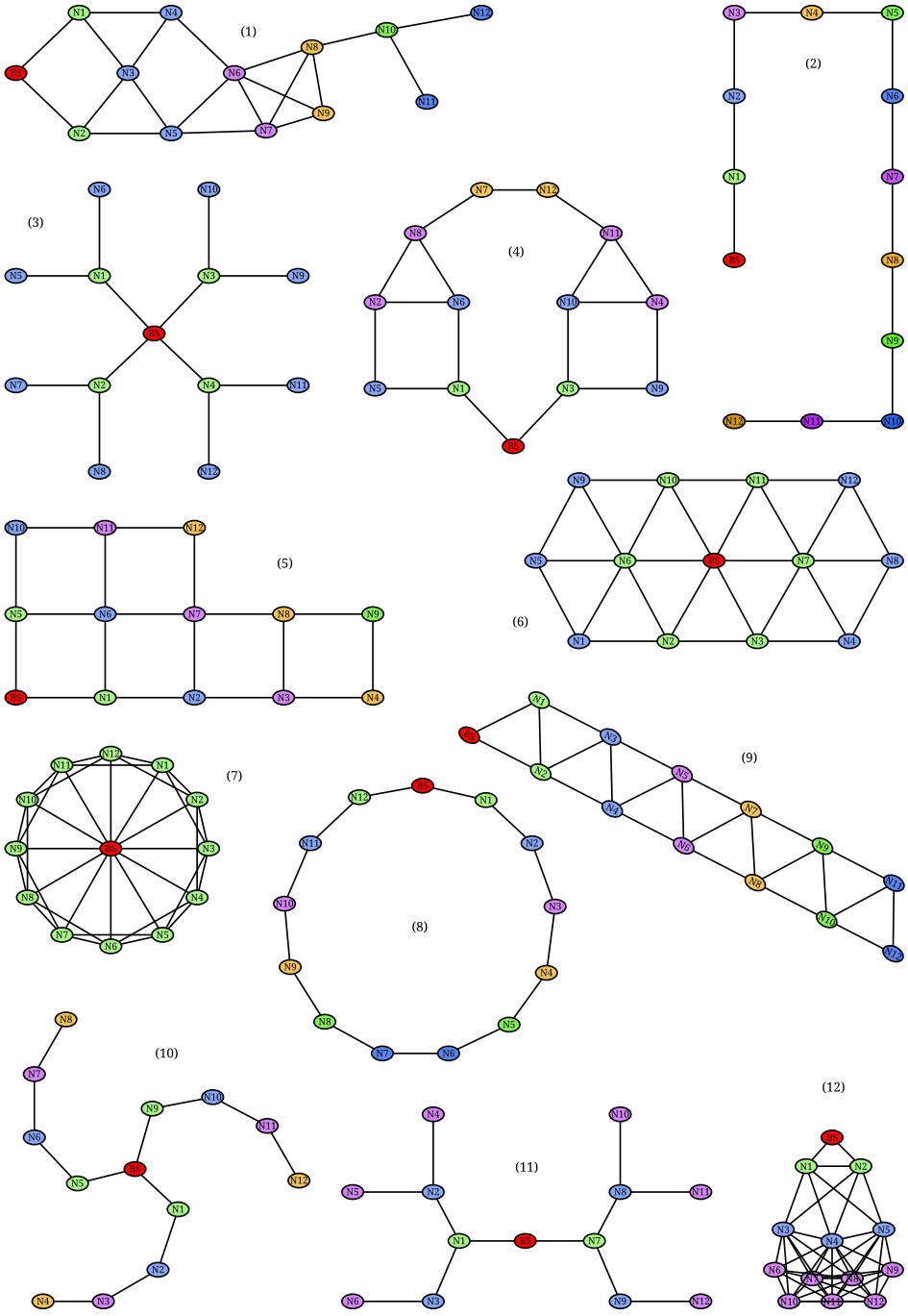


Figura 5.18 Topologías probadas con el robot terrestre en experimentos de campo. La BS se muestra en rojo y el color de los nodos depende de la distancia a la BS.

- Conector usado: `ConnectorEthernet`, usando la dirección MAC del adaptador Wi-Fi. Se han desactivado los servicios del sistema operativo no necesarios que den lugar a tráfico no deseado.
- Radio de alcance inalámbrico máximo de las antenas y separación entre vecinos máxima utilizada durante el *establecimiento de la topología*: no se ha medido el radio de alcance máximo pero si se ha hecho una estimación de la separación entre vecinos máxima para garantizar una conexión con baja probabilidad de pérdida, siendo esta de $4\,\mathrm{m}$.
- Disposición: tal que se generen las topologías mostradas en la Figura 5.18. En la Tabla 5.3 se muestran datos del número medio de vecinos y de saltos a la BS en cada una de las topologías.
- Trayectoria de descubrimiento y configuración: el robot se ha movido manualmente a nivel del suelo por el área en el que se encuentran la BS y los nodos, que es pequeña con las topologías utilizadas, siguiendo trayectorias similares a las que se calculan en escenarios virtualizados.
- Información de topología de la red inicial: CRL vacía y se incluye la lista de nodos autorizados, que son los 12 nodos existentes, e información de la BS. Se incluye información de las vecindades esperadas para que el robot no necesite las posiciones.
- Inicio del escenario: los nodos comienzan en modo sin configuración recibida y la BS con la información de topología inicial cargada localmente. El robot inicia en el estado `NONE` (ver Figura 4.16).
- Fin del escenario: tras verificarse que todos los nodos han recibido la configuración, el robot vuelve cerca de la BS y se espera a que todos los nodos reciban una respuesta de *PING* desde la BS como máximo $300\,\mathrm{s}$.

Se han utilizado las siguientes modificaciones en los parámetros comunes que se describieron en la Sección 5.1:

- `DEFAULT_HOP_LIMIT` $= 10$ en todas las topologías salvo en la (2) que vale 12.
- `DEFAULT_KAI` $= 10s$.

En todos los experimentos se ha probado la operación correcta de la arquitectura. El $100\,\%$ de los nodos conseguían conectar con la BS tras la primera fase de configuración. No ha sido necesario realizar una fase de *descubrimiento* adicional para corregir errores de comunicación.

Por comparación con el resto de experimentos, se utilizan las mismas métricas. Los datos transmitidos y recibidos en el LL en cada nodo hasta recibir la primera respuesta de la BS se muestra en la Figura 5.19. El tiempo requerido se muestra en la Figura 5.20.

Los datos obtenidos son similares a los otros experimentos. La desviación estándar es superior debido al menor número de repeticiones y de nodos. En transmisión de datos destaca el escenario 12, debido al mayor número de vecinos que tiene cada nodo. En tiempo de configuración destaca el escenario 2, debido a que en una de las repeticiones los nodos se han configurado en orden inverso a su cercanía a la BS (empezando por abajo).

Tabla 5.3 Experimentos con robot terrestre: datos de cada topología.

Grafo	Nº vecinos medio	Nº saltos a BS medio
1	$3,1\widehat{6}$	3,25
2	$1,91\widehat{6}$	6,5
3	$1,\widehat{6}$	$1,\widehat{6}$
4	$2,\widehat{6}$	2,5
5	$2,8\widehat{3}$	$2,8\widehat{3}$
6	$3,8\widehat{3}$	1,5
7	5	1
8	2	3,5
9	$3,\widehat{6}$	3,5
10	1,75	2,5
11	$1,8\widehat{3}$	$2,\widehat{3}$
12	7,5	$2,41\widehat{6}$

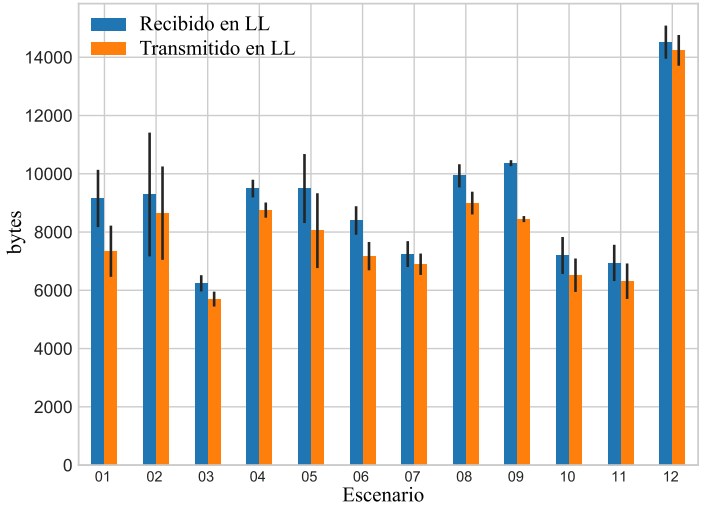


Figura 5.19 Experimentos de campo con robot terrestre: datos transmitidos y recibidos en LL.

El tiempo menor se da en la topología 7, ya que todos los nodos tienen conexión directa con la BS. En este escenario el robot no tuvo que moverse.

Respecto a los experimentos con redes virtualizadas, cabe esperar diferencias en el

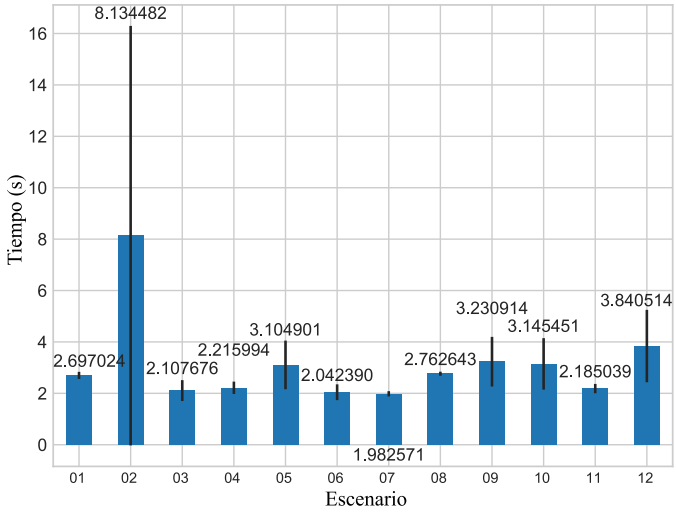


Figura 5.20 Experimentos de campo con robot terrestre: tiempos requeridos para configuración.

tiempo de computación y retardos debidas al hardware usado. Las diferencias entre las pérdidas en los canales de comunicación se han minimizado buscando una distancia máxima entre vecinos que dé lugar a valores similares. Aunque el hardware de los nodos reales tiene menor potencia (frecuencia de la CPU, juego de instrucciones), en entornos virtualizados los núcleos de la CPU son compartidos. En el saludo avanzado es dónde se realizan los cómputos más complejos. Para comparar las diferencias debido a lo anterior se ha medido el tiempo medio de realización de un saludo avanzado entre dos nodos usando la topología 1, medido en las pruebas reales y con pruebas usando `launchermn` para 3 repeticiones. Hay que recalcar que un nodo puede realizar varios saludos simultáneamente con sus diferentes vecinos. Los resultados se recogen en la Tabla 5.4. Se observa una diferencia cercana al $10\,\%$.

Tabla 5.4 Tiempo medio en realizar un saludo avanzado entre nodos en escenarios reales y virtualizados.

Escenario real	Escenario virtualizado	Diferencia
0,977209 s	0,880591 s	0,096617 s (+9,89 %)

5.8 Escenarios reales con robot aéreo

Los primeros experimentos con un robot aéreo fueron realizados con uno basado en la plataforma *DJI Flamewheel F450* equipado con una *Raspberry Pi 1 Model B* funcionando con el rol de robot. Se han utilizado 7 nodos sensores similares a los utilizados con el robot

terrestre. Se han diseñado 2 escenarios diferentes, donde los 7 nodos están desplegados en diferentes posiciones. La topología esperada en cada uno de estos escenarios se muestra en la Figura 5.21. En la anterior figura no se muestra el robot, la BS se muestra en rojo

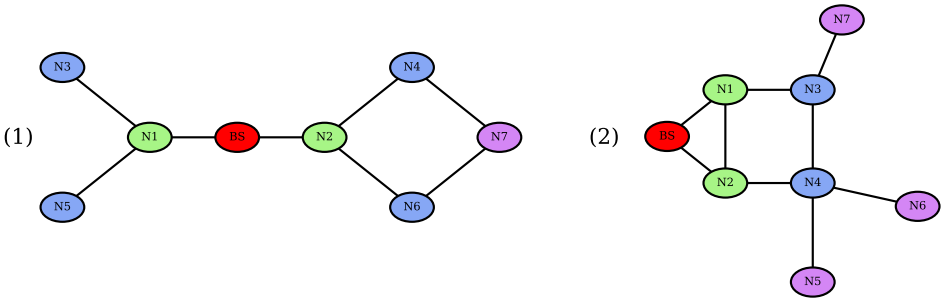


Figura 5.21 Topologías probadas con el robot aéreo. La BS se muestra en rojo y el color de los nodos depende de la distancia a la BS.

y el color de los nodos es diferente según los saltos necesarios para comunicarse con la BS. No se muestra la posición exacta de cada nodo, solo la relación de vecindad. Los nodos sensores se sitúan esta vez a nivel del suelo y el robot sobrevuela el escenario. En la Figura 5.22 se muestra una imagen captada durante la configuración de la red del escenario (1), en la que se muestra la mitad derecha del escenario y el robot sobrevolando la zona. El

Figura 5.22 Imagen del robot aéreo sobrevolando el escenario (1).

alcance radio entre nodos sensores a nivel del suelo con los adaptadores Wi-Fi empleados es muy limitado, por lo que se redujo la distancia máxima entre vecinos a $3\,\mathrm{m}$. A distancias

mayores aumentaban las pérdidas de tramas. En el LL utilizado no hay entrega fiable y esto hace que sea difícil realizar el saludo de DTLS con una elevada tasa de pérdidas. En los experimentos con la topología (2), el nodo N3 se dispuso de manera que su conectividad con el N1 fuera deficiente, forzando así la generación de anomalías y así justificar una segunda configuración. Por otro lado, el alcance radio con la BS y el robot era el normal de otros experimentos.

En general, los parámetros utilizados han sido los mismos que con los experimentos en laboratorio. Las diferencias son:

- Entidades: 7 nodos sensores usando dispositivos Raspberry Pi 1 Model B, 1 robot y 1 BS.
- Repeticiones del experimento: 4 repeticiones.
- Separación entre vecinos máxima: $3\,\mathrm{m}$ por los motivos antes comentados.
- Disposición: tal que se generen las topologías mostradas en la Figura 5.21. En la Tabla 5.5 se muestran datos del número medio de vecinos y de saltos a la BS en cada una de las topologías.

Tabla 5.5 Experimentos con robot aéreo: datos de cada topología.

Grafo	Nº vecinos medio	Nº saltos a BS medio
1	2	1,86
2	2,29	2,14

Los resultados generados en los experimentos han sido los esperados. Usar una *Raspberry Pi 1 Model B* funcionando como robot ha demostrado ser suficiente para hacer todos los cálculos de topología en redes pequeñas. Todas las repeticiones del escenario (1) se realizaron correctamente. En el escenario (2), en 3 repeticiones el N3 no consiguió establecer el enlace con el N1, generándose anomalías como era de esperar. Tras una repetición de las fases de *descubrimiento*, *establecimiento de la topología* (usando los datos aportados por los nodos) y *configuración* se llegó a una configuración sin errores.

Por comparación con el resto de experimentos, se muestran los mismos indicadores. Hay que indicar que en el escenario (2) no se consideran los datos de los nodos que deliberadamente tenían problemas. Los datos transmitidos y recibidos en el LL en cada nodo hasta recibir la primera respuesta de la BS se muestra en la Figura 5.23. El tiempo requerido se representa en la Figura 5.24.

En las figuras se observan valores ligeramente superiores a los obtenidos en otras pruebas debido a las peores condiciones de comunicación. Esto es más evidente en el escenario (2) por los motivos antes comentados.

Posteriormente, se probó SNSR en escenarios realistas de monitorización de un puente y una industria cementera. Los nodos, BS y configuración fueron similares a los anteriores experimentos. El robot en el escenario del puente estaba basado en la plataforma *DJI Flamewheel F450* dotada de un piloto automático *PixRacer* y un *Khadas VIM3* de bajo costo

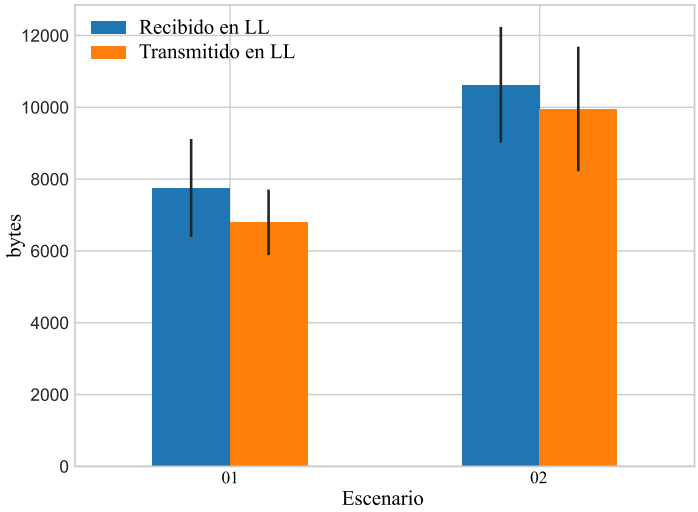


Figura 5.23 Experimentos con robot aéreo: datos transmitidos y recibidos en LL.

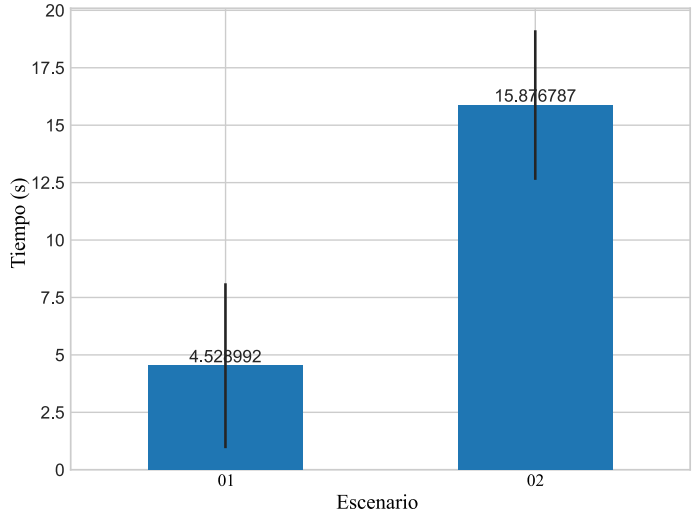


Figura 5.24 Experimentos con robot aéreo: tiempos requeridos para configuración.

para el procesamiento. El robot en la cementera era un hexarotor de diseño personalizado con un 3D-LiDAR para la auto-localización y un ordenador *Intel NUC* y se muestra en la Figura 5.25. Los datos de la topología y la información topológica en las fases *inicial*, *descubrimiento* y *establecimiento de la topología* se proporciona en [42].

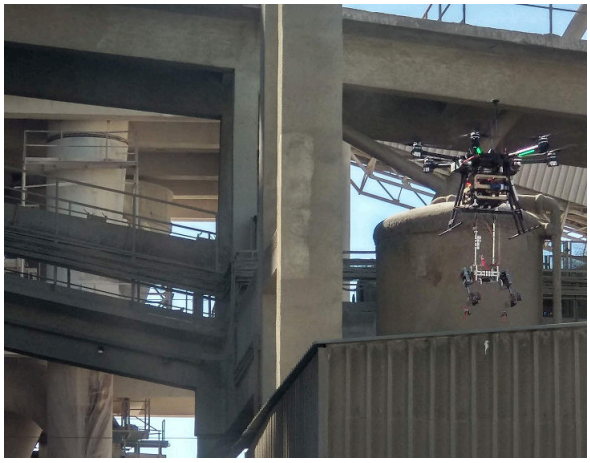


Figura 5.25 Imagen de robot aéreo sobrevolando la cementera.

5.9 Experimentos con ataques

Como quedó reflejado en la Sección 5.2, y resumido en la Tabla 5.2, algunos ataques no han sido contemplados porque no son competencia de la arquitectura de seguridad sino de la pila de red empleada. Muchos otros ataques no son posibles cuando se usan protocolos seguros como DTLS, siempre que se mantengan actualizados, la implementación sea correcta y la autenticación basada en certificados sea válida por estar seguras las claves de la CA. Gran parte de la fortaleza reside en la separación clara de funcionalidades y la poca capacidad de influir unos nodos sobre otros. Durante el desarrollo de la implementación se han hecho pruebas verificando que esta es fiel a la arquitectura SNSR, las entidades ignoran todos los mensajes no válidos y se han puesto límites máximos a todo aquello susceptible de ser abusado. También se ha monitorizado el tráfico generado comprobando el funcionamiento de los distintos protocolos y que la información transmitida en claro era la esperada.

Para evitar posibles sesgos en el análisis de seguridad también han sido llevado a cabo análisis alternativos por otras personas. Esto se conoce en seguridad informática como equipo rojo (*red team*). El *red team* se pone en el lugar de un atacante real y prueba la seguridad del sistema. Esto se describe en la Sección 5.9.1.

De los ataques posibles, en esta Tesis Doctoral se han elegido los 2 siguientes por considerar que sus resultados son menos evidentes:

- *Suplantación de robot*, donde uno o varios agentes maliciosos envían balizas con formato correcto, pero con información falsa, para forzar a los nodos sensores a intentar la conexión con ellos. Se quiere evaluar el comportamiento al usar distintos métodos de detección de baliza, parámetros de comunicación y comportamientos del agente malicioso. La BS es más inmune a estos ataques debido a sus recursos y a que podría conocer en un momento dado los robots reales activos con lo que

puede descartar balizas falsas. En función del comportamiento del agente malicioso, este ataque se puede asimilar a los ataques:

1. *Service request power attack*, cuando se obliga al nodo a realizar el proceso de saludo avanzado aunque este falle.
2. *Rushing attack*, si se consigue que las balizas del robot real sean descartadas.
3. *Hello flooding*, si se hace creer a los nodos que el robot es alcanzable sin serlo.
4. *Byzantine attack*, si varios agentes simulan distintos robots simultáneamente.

Los experimentos donde se ha ejecutado este ataque se describen en la Sección 5.9.2.

- *Node destruction* y en general cualquier fallo fatal de un nodo que provoque su desconexión de la red, ya sea por causas malintencionadas (destrucción, robo o *jamming*) o no (falta de energía, sustitución o desplazamiento temporal). En [25] se hacen pruebas similares para INSENS. Se cuenta el número de nodos que dejan de comunicarse con la BS según el número de nodos con fallos como medida de robustez de la red. Se tienen en cuenta los nodos que no pueden comunicarse por quedar aislados de la red (no hay alcance vía radio) y aquellos que no pueden debido a problemas de enrutamiento (se generan bucles) que se corregirían con una nueva configuración. Aunque el enrutamiento no está especificado por la arquitectura SNSR, se quiere comprobar si el usado en la implementación es lo suficientemente robusto para permitir reducir la urgencia con la que el robot tiene que realizar vuelos para reconfigurar la red. Los experimentos con este ataque se describen en la Sección 5.9.3.

5.9.1 Ataques de *Red Team*

El *Red Team* debe estar formado por personas ajenas a esta Tesis Doctoral. Alejandro Rosa Neupavert, estudiante del Máster en Seguridad de la Información y las Comunicaciones de la Universidad de Sevilla en el curso 2017/2018, se encargó de hacer estos ataques. Los resultados fueron publicados en su Trabajo Fin de Máster, que tituló "*Pentesting a red de sensores securizada por robot móvil*" y fue tutorado por el autor de esta Tesis Doctoral.

Los ataques se hicieron sobre escenarios virtualizados. Para ello se proporcionó al *Red Team* una máquina virtual de Ubuntu 16.04 con el compositor *launcher*, el programa `Node` y 6 escenarios ya preparados.

El programa `Node` era la versión de la implementación realizada en la fecha de ejecución de las pruebas. Solo disponía de detección básica de balizas, pero el resto de las funcionalidades estaban implementadas.

Los 6 escenarios proporcionados tienen todos la misma topología que la número 1 de la Figura 5.18. La única diferencia es que cada uno tiene un modo de seguridad diferente. En estos escenarios las entidades estáticas se dividen en 3 zonas de difusión:

Z1: N1, N2, N3 y BS.

Z2: N4, N5, N6, N7, N8 y N9.

Z3: N10, N11 y N12.

Tanto en la fase de *descubrimiento* como de *configuración* el robot sigue una trayectoria por las zonas (Z) en el orden Z1→Z2→Z3→Z2→Z1. Todo esto es modificable y se enseñó al estudiante cómo usarlo.

Además, se proporcionó documentación sobre la arquitectura y los protocolos usados. No se proporcionó el código fuente de la implementación, pero sí ejemplos básicos para generar y decodificar mensajes.

A continuación, se describen brevemente los ataques que se realizaron y sus resultados:

- **Ataque DoS a los nodos**: envío de balizas falsas a la mayor tasa de transmisión posible para evitar que los nodos se conecten al robot real. Se comprobó, como se indicó en la Sección 3.6.1, que se impedía la detección de balizas del robot real. Los nodos descartaban las balizas reales mientras intentan conectarse al robot ficticio. El robot real solo transmite entre 1 y 2 veces por segundo. Las balizas ficticias se transmiten a una frecuencia muy superior, por lo que la probabilidad de que la primera baliza vista por un nodo sea falsa es muy elevada. Ataques parecidos aunque transmitiendo balizas falsas a la misma frecuencia que el robot real se verán en la Sección 5.9.2. El ataque realizado se puede clasificar como *service request power attack* o *hello flooding*, se detecta fácilmente y se supone que no puede durar indefinidamente. Cuando se interrumpía la emisión de balizas falsas el sistema recuperaba la normalidad. Posteriormente, ya fuera del ataque de *Red Team* se repitió este ataque con el método de detección de baliza avanzada comprobando que ya no era efectivo.
- **Ataque DoS al robot**: peticiones de conexión con el robot a la mayor tasa de envío posible. Este ataque provoca que se utilicen el número máximo de enlaces permitidos por el robot, impidiendo procesar otras peticiones legítimas. Este ataque puede catalogarse como del tipo *LL exhaustion*. El robot continúa funcionando correctamente con otras conexiones previamente establecidas e inmediatamente con el resto cuando se termina el ataque.
- **Ataque DoS a la BS**: tras la configuración correcta de la red, se envían mensajes a la BS para saturarla. Todos los mensajes son ignorados, por lo que su único efecto es ocupar el canal radio, como sería en un ataque de tipo *unfairness*.
- **Intrusión con nodo con certificado duplicado**: se añade un nuevo nodo al escenario usando el mismo certificado de otro nodo existente, como si se hubieran robado sus datos privados. En la versión de la implementación proporcionada, durante el *establecimiento de la topología*, se admitía en la red al primer nodo duplicado que contactaba con el robot sin notificar esta anomalía. Sin embargo, una vez detectado el posible duplicado de los datos de seguridad, no se puede confiar en ninguno de los nodos. Esto ha sido modificado en la versión actual. Ahora se crea una anomalía para advertir de esto y revocar el certificado y se ignoran ambos nodos ante la imposibilidad de confiar en ellos.
- **Intrusión con robot con certificado duplicado**: envío de balizas desde un robot malicioso con una copia de certificados válidos una vez que el robot auténtico ya se ha conectado con los nodos. Era imposible que los nodos se intentaran conectar a él, ya que los nodos ignoran cualquier baliza una vez que están conectados al robot

real. Si los nodos no hubieran estado conectados ya, este ataque hubiera funcionado al no poderse distinguir que el robot era malicioso. Por este motivo es importante revocar rápidamente los certificados de los robots capturados.

- **Análisis de tráfico**: se captura todo el tráfico transmitido en la red para analizar su funcionamiento y la información transportada. En un escenario real este ataque requeriría la colocación de varias estaciones de escucha para cubrir todo el escenario, pero en un escenario virtualizado se realiza simplemente capturando el tráfico local. Consigue detectar los nodos que tienen más tráfico, aunque no el contenido de los paquetes.
- **Destrucción del nodo con más tránsito**: se elimina el nodo que reenvía más paquetes una vez que el sistema está funcionando, pero se detecta la anomalía y los nodos siguen funcionando con normalidad porque hay rutas alternativas.
- **Envío de información falsa desde una estación externa a la topología**: se envían mensajes aleatorios o duplicados desde otras estaciones ajenas a la red. Todos son ignorados.

Todos los ataques anteriores ya habían sido contemplados a nivel interno y los resultados eran los esperados. Sin embargo, el *Red Team* sirvió para recapacitar sobre el tratamiento que se estaba dando a los nodos clonados como se ha comentado antes. Además, muchos ataques se realizaron con herramientas diferentes a las que se habían usado, por lo que sirvió como validación externa adicional de las capacidades de la arquitectura SNSR.

5.9.2 Ataques de suplantación de robot y comparativa de métodos de detección de baliza

Se han realizado experimentos donde se han introducido robots maliciosos que transmiten balizas para que los nodos sensores se conecten a ellos. No se contemplan los modos de seguridad en el LL donde no se realiza la autenticación del robot ni tampoco se contempla que los nodos sensores puedan estar conectados simultáneamente a más de un robot. Lo primero facilitaría el ataque, lo segundo limitaría los efectos del ataque como se describe más adelante.

Se supone que los atacantes no tienen en su poder las claves privadas de robots válidos. Si lo anterior ocurriera, se deben revocar sus certificados y actualizar la CRL. En caso contrario, los nodos sensores no tienen ningún mecanismo de defensa para evitar la conexión con los atacantes. Para ello, SNSR dispone del procedimiento de actualización urgente de la CRL descrito en la Sección 3.9.2.

A diferencia de las pruebas realizadas en el *Red Team*, los atacantes van a transmitir balizas con la misma cadencia que utiliza el robot real. Es decir `BI_MAX` será 1 s, usando una componente aleatoria sin ningún control adicional sobre la emisión de balizas. Esto evita los casos límites indicados en la Sección 3.6.1 para el método básico de detección. La única ventaja permitida en este sentido será la de comenzar a emitir balizas antes de que lo haga el robot real.

Un robot malicioso con las anteriores condiciones no va a poder conectarse a ninguna otra entidad, y el único comportamiento propio de un robot que va a poder suplantar es el envío de balizas y la realización del saludo de la conexión. Es sencillo emitir balizas con formato válido que no son descartadas, ya que estás no incluyen ningún mecanismo de

seguridad más allá de la comprobación de que tienen un formato correcto y los valores de sus campos son aceptables.

Los objetivos principales del atacante van a ser:

1. Impedir que un nodo se conecte a un robot real mientras intenta conectarse a uno falso, teniendo en cuenta que los nodos sensores permiten solo una conexión a un robot (*rushing attack*)
2. Forzar a los nodos a realizar operaciones complejas (*service request power attack*) o transmisiones inútiles (*hello flooding*), provocando el consumo de energía.

Durante los experimentos se han utilizado 1, 2 o 3 robots maliciosos con 3 comportamientos diferentes, que serán identificados como `IG`, `RK` y `RD` respectivamente, y que se describen a continuación:

`IG` Envío de balizas válidas e iguales para cada robot, pero ignorando peticiones de conexión. También sería equivalente enviar balizas usando direcciones de origen falsas no alcanzables, incluyendo balizas duplicadas del robot real.

`RK` Envío de balizas válidas e iguales para cada robot, pero aceptando las peticiones. En las balizas se utiliza un número de serie falso, por lo que se fuerza al nodo a realizar el saludo avanzado. El certificado real del robot malicioso está revocado por lo que al final falla el saludo.

`RD` Envío de balizas válidas aleatorias aceptando peticiones. Es similar al caso anterior, pero para el nodo cada baliza parece provenir de un robot diferente. Los valores de los campos son aleatorios para que los nodos no puedan descartar balizas previamente vistas. El saludo falla al no coincidir la identificación del certificado con la que aparece en la baliza.

Además, se ha realizado una primera ejecución de los experimentos sin ataques para poder hacer comparaciones posteriores.

Durante el saludo avanzado, el robot actúa como servidor y, en DTLS [125], su certificado será enviado en primer lugar. En la implementación utilizada ese certificado es validado por el cliente usando diferentes criterios. Entre ellos están, en orden de ejecución, a) comprobar que el CN del certificado es el esperado; b) verificar que está firmado por la CA y la firma es correcta; y c) comprobar que el número de serie no ha sido revocado. Lo anterior implica que la duración del saludo durante un ataque `RD` será menor que en `RK`, que a su vez será bastante inferior que en `IG`. En coste computacional, los efectos de `RK` son los más costosos, seguidos de los de `RD` y en último lugar los de `IG`.

Se quiere evaluar las distintas técnicas descritas en la Sección 3.6.1 para reducir el impacto de estos ataques. Estas técnicas eran:

- Detección de baliza básica o avanzada (ambas excluyentes).
- Establecer un tiempo máximo de duración del saludo con robots.
- Establecer un tiempo mínimo de duración del saludo con robots.

Así que los anteriores ataques se han realizado sobre 4 configuraciones diferentes de los nodos. El identificador dado a cada una de ellas y sus detalles se muestran a continuación:

`M2NF` Configuración *M2* (descrita en la Tabla 5.1) con `MS_FAST_HS` desactivado (sin reducción en reintentos y temporizadores para conexiones con el robot).

`M2` Configuración *M2* con `MS_FAST_HS` activado (con reducción en reintentos y temporizadores para conexiones con el robot).

`M3NM` Configuración *M3* (descrita en la Tabla 5.1) con `MS_FAST_HS` activado.

`M3M1` Configuración *M3* con `MS_FAST_HS` activado y un tiempo mínimo de saludo de $1\,\mathrm{s}$. Las otras configuraciones no imponen un tiempo mínimo de saludo.

La activación de `MS_FAST_HS` influye en el tiempo máximo de duración del saludo. La implementación de DTLS en la biblioteca `wolfssl` espera hasta $127\,\mathrm{s}$ en cada intento de saludo si el otro extremo no responde: hace 6 reintentos, duplicando el tiempo de espera entre cada reintento partiendo de $1\,\mathrm{s}$ ($1+2+4+8+16+32+64=127$). Por las características de las redes usadas y dado que el RTT visto no suele superar $1\,\mathrm{s}$, el valor anterior es excesivo. Por ese motivo, en la implementación se ha modificado el tiempo de espera entre reintentos, a un valor aleatorio entre 1 y 2 s dado por la función $random(1,2)$. La componente aleatoria impide que sea predecible el siguiente intento de conexión. Con `MS_FAST_HS` el tiempo máximo de duración del saludo con un robot se reduce a un valor medio de $1{,}5\,\mathrm{s}$. Si no está activado, el tiempo máximo tiene un valor medio de $10{,}5\,\mathrm{s}$.

En total se han probado 40 escenarios diferentes, a los que se les ha dado un nombre simbólico en minúsculas formado por 2 caracteres iniciales que indican el número de robots, seguido por 2 caracteres que indican el tipo de ataque y el resto de los caracteres que indican la configuración de los nodos. Los escenarios se muestran en la Tabla 5.6.

Tabla 5.6 Escenarios usados en los experimentos con ataques de suplantación de robot.

Ataque	`M2NF`	`M2`	`M3NM`	`M3M1`
Sin ataque	r1nom2nf	r1nom2	r1nom3nm	r1nom3m1
`IG`, 1 atacante	r2igm2nf	r2igm2	r2igm3nm	r2igm3m1
`IG`, 2 atacantes	r3igm2nf	r3igm2	r3igm3nm	r3igm3m1
`IG`, 3 atacantes	r4igm2nf	r4igm2	r4igm3nm	r4igm3m1
`RK`, 1 atacante	r2rkm2nf	r2rkm2	r2rkm3nm	r2rkm3m1
`RK`, 2 atacantes	r3rkm3nf	r3rkm2	r3rkm3nm	r3rkm3m1
`RK`, 3 atacantes	r4rkm3nf	r4rkm2	r4rkm3nm	r4rkm3m1
`RD`, 1 atacante	r2rdm2nf	r2rdm2	r2rdm3nm	r2rdm3m1
`RD`, 2 atacantes	r3rdm2nf	r3rdm2	r3rdm3nm	r3rdm3m1
`RD`, 3 atacantes	r4rdm2nf	r4rdm2	r4rdm3nm	r4rdm3m1

Los escenarios están formados por 100 nodos al alcance radio de los robots que tratarán de conectarse simultáneamente. Para estos experimentos no es necesario incluir una BS. Se realizan 100 repeticiones de cada escenario, por lo que en total se analizarán 10.000 conexiones para cada escenario. Se mantienen el resto de los parámetros de configuración usados en otros escenarios. Los robots maliciosos comienzan a transmitir balizas desde el inicio y el robot verdadero cuando ha transcurrido 1 s.

Se hacen 2 mediciones en cada nodo para determinar el efecto de los ataques:

1. Tiempo necesario desde que un robot verdadero comienza a transmitir balizas hasta que el nodo se conecta a él. Esto mostrará cuánto tiempo han impedido la conexión los atacantes.
2. Número de intentos de conexión que ha hecho el nodo hasta conectarse al robot verdadero. Esto es proporcional al número de operaciones adicionales que los atacantes han forzado a realizar a los nodos.

En los siguientes apartados se muestran los datos obtenidos en condiciones normales y según el ataque realizado. Hay que resaltar que el 100 % de los nodos se han podido conectar al robot válido en todos los experimentos. En las figuras que se muestran en esta sección se utilizarán tonos amarillos para los escenarios `M2NF`, azules para los `M2`, rojos para los `M3NM` y verdes para los `M3M1` en ese orden de presentación de izquierda a derecha.

5.9.2.1 Conexión al robot sin ataques

Los valores medios de los resultados obtenidos en las 100 repeticiones de cada escenario se muestran en la Figura 5.26.

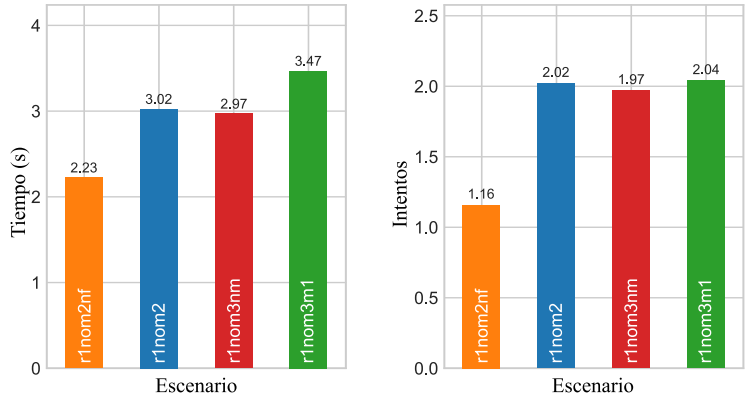


Figura 5.26 Tiempo e intentos medios necesarios para conectar al robot sin ataques.

Se observa que los tiempos necesarios son superiores a los mostrados en la Tabla 5.4. Esto es debido a la congestión inicial producida al intentar conectarse los 100 nodos casi simultáneamente al robot. A pesar de la espera de contención aleatoria realizada con `TC_MAX` de 50 ms por los nodos, el número de nodos es muy elevado. Esto implica que hay que tener en cuenta la densidad de nodos a la hora de establecer la velocidad de vuelo del robot adecuada.

Sin ataques, la configuración M2NF es la que presenta mejores resultados aun siendo la más sencilla. Hay que recordar que en este caso los intentos de conexión incluyen la retransmisión de mensajes, por lo que el menor número de intentos no implica la transmisión de menos tramas.

La activación de MS_FAST_HS provoca un aumento en el número de intentos de conexión debido a que en situaciones de congestión se descartan algunos que hubieran funcionado con un poco más de espera. Pero hay que decir que esto no implica un aumento significativo en el número de tramas transmitidas, porque en M2NF también se harán reintentos de envío dentro del mismo intento de conexión. El tiempo también se ve incrementado debido a que cada intento va a requerir esperar la recepción de una nueva baliza y un nuevo tiempo de contención.

Usar la detección de baliza básica o avanzada da resultados prácticamente similares. Pero el establecer un tiempo mínimo de saludo hace que el tiempo necesario aumente.

5.9.2.2 Comportamiento ante el ataque IG

Los valores medios de los resultados obtenidos se muestran en la Figura 5.27.

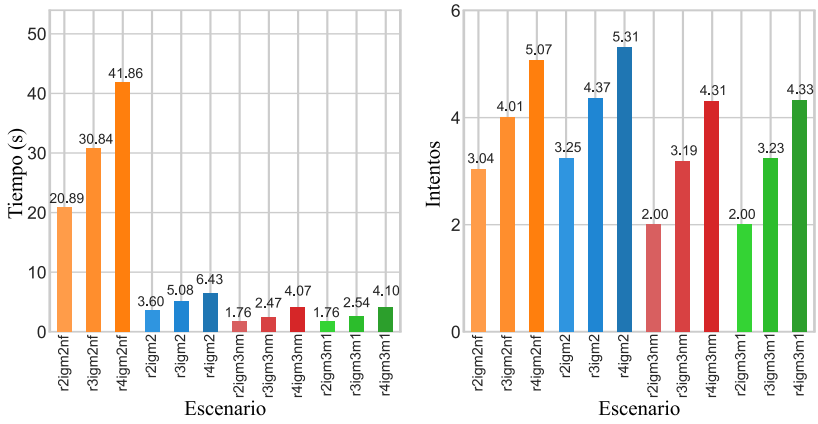


Figura 5.27 Tiempo e intentos medios necesarios para conectar al robot con ataque IG.

En este ataque, el valor del tiempo necesario para la conexión es el más importante, ya que el objetivo del atacante es evitar en lo posible la conexión con el robot real y, por tanto, que este detecte nodos sensores. Los intentos de conexión implican solo la transmisión de tramas adicionales y en general no son elevados.

En todas las configuraciones, el funcionamiento empeora respecto a los datos sin ataques a medida que aumenta el número de atacantes, a excepción de cuando hay un único atacante con detección avanzada. La explicación es que en estos casos la conexiones con el robot real se hace de manera más escalonada. Esto hace que el robot esté menos sobrecargado y responda más rápido por lo que los nodos realizan menos intentos.

Se observa un gran aumento en el tiempo necesario en M2NF. En M2, se reduce mucho el tiempo perdido cuando MS_FAST_HS está activo, aunque con un ligero aumento de intentos.

El ganador claro en este ataque es el uso de la detección avanzada, en las configuraciones `M3NM` y `M3M1`. El próximo candidato se elige de las balizas recibidas durante el intento de conexión descartando la actual, aumentando la probabilidad de elegir la del robot real. El próximo intento se realiza sin esperar a recibir una nueva baliza. En ambas configuraciones el resultado es prácticamente el mismo, ya que en este caso no es necesario aplicar el tiempo mínimo de saludo al ser el tiempo de espera de respuesta mayor.

5.9.2.3 Comportamiento ante el ataque RK

Los valores medios de los resultados obtenidos se muestran en la Figura 5.28.

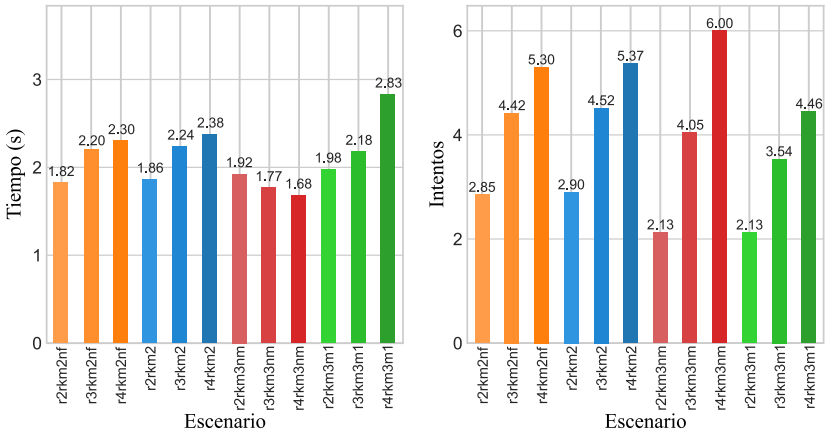


Figura 5.28 Tiempo e intentos medios necesarios para conectar al robot con ataque RK.

En comparación con el ataque `IG` y cuando no hay ataques, los tiempos de conexión son inferiores, ya que el proceso de saludo dura menos. En cuanto a los intentos, son similares al ataque `IG`, a excepción de la configuración `M3NM`.

La configuración `M3NM` presenta una disminución en el tiempo necesario conforme aumentan los atacantes, debido a la menor sobrecarga instantánea que sufre el robot correcto. Sin embargo, aumentan rápidamente el número de intentos.

Establecer un tiempo mínimo al saludo ahora sí tiene efectos observables. En `M3M1` se reducen el número de intentos comparado con `M3NM`. De hecho, es la configuración que necesita menos intentos para hacer la conexión y, por tanto, la que sufre menor coste computacional. La contrapartida está en un ligero aumento en el tiempo necesario.

5.9.2.4 Comportamiento ante el ataque RD

Los valores medios de los resultados obtenidos se muestran en la Figura 5.29. Prácticamente, los resultados son similares al ataque `RK`. Sin embargo, aquí los intentos implican un menor coste computacional que en el caso anterior, debido al proceso de verificación realizado en el saludo, que simplemente detecta que el CN del certificado enviado por el atacante no coincide con el que aparece en las balizas falsas.

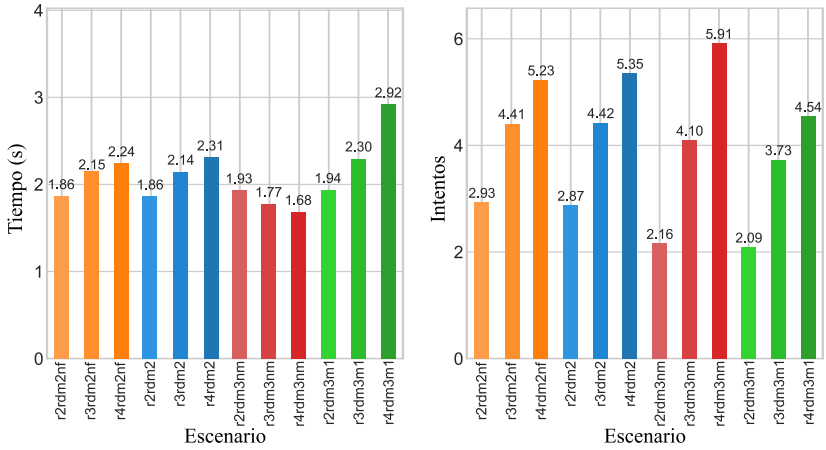


Figura 5.29 Tiempo e intentos medios necesarios para conectar al robot con ataque RD.

5.9.3 Fallos fatales en nodos aleatorios

La arquitectura SNSR no especifica el enrutamiento concreto que debe tener la red de sensores. En la implementación realizada se ha optado por usar un enrutamiento estático desde el punto de vista de los nodos cuyas rutas solo pueden ser actualizadas con un cambio de configuración. Cada nodo contiene los vecinos a emplear como siguiente salto para cada posible destino ordenados según el camino más corto y los nodos de tránsito menos utilizados. También existe un enrutamiento alternativo que se activa si se detectan bucles. Como el cálculo de las rutas se hace de manera centralizada, se podrían cambiar los criterios elegidos para calcular estas sin tener que hacer modificaciones en los nodos.

Un nodo puede tener fallos fatales por causas malintencionadas (por ejemplo, por destrucción o robo) o no (como por la falta de energía). Si el nodo sirve de intermediario para el encaminamiento de paquetes, este fallo puede causar problemas también en el resto de la red. Siempre es posible reconfigurar la red en la arquitectura SNSR para solucionar ese problema. Sin embargo, si el fallo es reparable y temporal y la red es capaz de seguir funcionando, usando rutas alternativas, se podrían evitar hasta dos reconfiguraciones que requieran el uso del robot (una temporal hasta que se arregle el problema y otra que vuelva a la configuración anterior), con lo que se reduciría su uso.

Se quiere probar la robustez del enrutamiento utilizado ante estos fallos sin necesidad de utilizar técnicas como la selección aleatoria de rutas o el envío simultáneo de tramas duplicadas por diferentes rutas (como se hace en [25]). Estas técnicas adicionales podrían ser implementas en un futuro para aumentar la protección ante ataques de *selective forwarding*.

Para medir la robustez se va a contar el número de nodos que pierden la conectividad con la BS, suponiendo que inicialmente el grafo de la red es fuertemente conexo, diferenciando 2 casos:

1. Perdida de conectividad por ausencia de posibles rutas, cuando la eliminación de nodos hace que el grafo no sea fuertemente conexo. En estos casos, una reconfigura-

ción no solucionaría el problema y habría que añadir (o reparar) nodos que sirvan de pasarela. Por consiguiente, aquí no influye el encaminamiento usado.

2. Perdida de conectividad por bucles. Existe realmente un camino posible para alcanzar la BS pero la configuración de encaminamiento no lo permite. Suponemos que el número de saltos permitido es suficientemente grande para que nunca se descarte un paquete por este motivo sin haberse producido antes un bucle. Para que esto ocurra con la implementación actual, es necesario que fallen 2 o más nodos. Una reconfiguración solucionaría el problema. En este caso influye el encaminamiento empleado.

El daño producido también va a depender de la topología que tenga la red (sobre todo de la distribución del grado de los vértices del grafo) y de si la elección de los nodos a eliminar es aleatoria o dirigida a los nodos más críticos.

En los experimentos que se describen a continuación solo se han tenido en cuenta fallos aleatorios. Se han hecho pruebas con 6 topologías diferentes, una de cuadrícula cuadrada, una de cuadrícula triangular y 4 aleatorias. Todas las topologías cuentan con 200 nodos y se han ejecutado virtualmente con `launcher`. Cada escenario ha sido configurado y una vez en fase de *mantenimiento* se han eliminado 5, 10, 15, 20, 30, 40 o 50 nodos de manera aleatoria. Por último, se ha comprobado el número de nodos activos que habían perdido la comunicación con la BS, a los que hemos denominado nodos bloqueados, diferenciando según el motivo. Se han realizado 50 repeticiones para cada número de fallos antes indicado (350 repeticiones en total para cada topología). La topología de los escenarios a tamaño reducido y sus nombres se muestran en la Figura 5.30.

En la Tabla 5.7 se muestran los datos del número medio de vecinos y de saltos a la BS en cada una de las topologías.

Tabla 5.7 Experimentos con fallos fatales: datos de cada topología.

Grafo	Nº vecinos medio	Nº saltos a BS medio
cuadrícula cuadrada	3,71	7,125
cuadrícula triangular	5,43	5,63
aleatorio1	8,995	6,24
aleatorio2	9,235	5,935
aleatorio3	12,055	4,465
aleatorio4	7,32	4,875

En la Figura 5.31 se muestran los valores medios obtenidos para el número de nodos bloqueados debido al encaminamiento en las 6 topologías. En la Figura 5.32 se muestran gráficas para cada topología con los anteriores valores además del número de nodos bloqueados debido a la topología del grafo de la red y el total, incluyendo la desviación

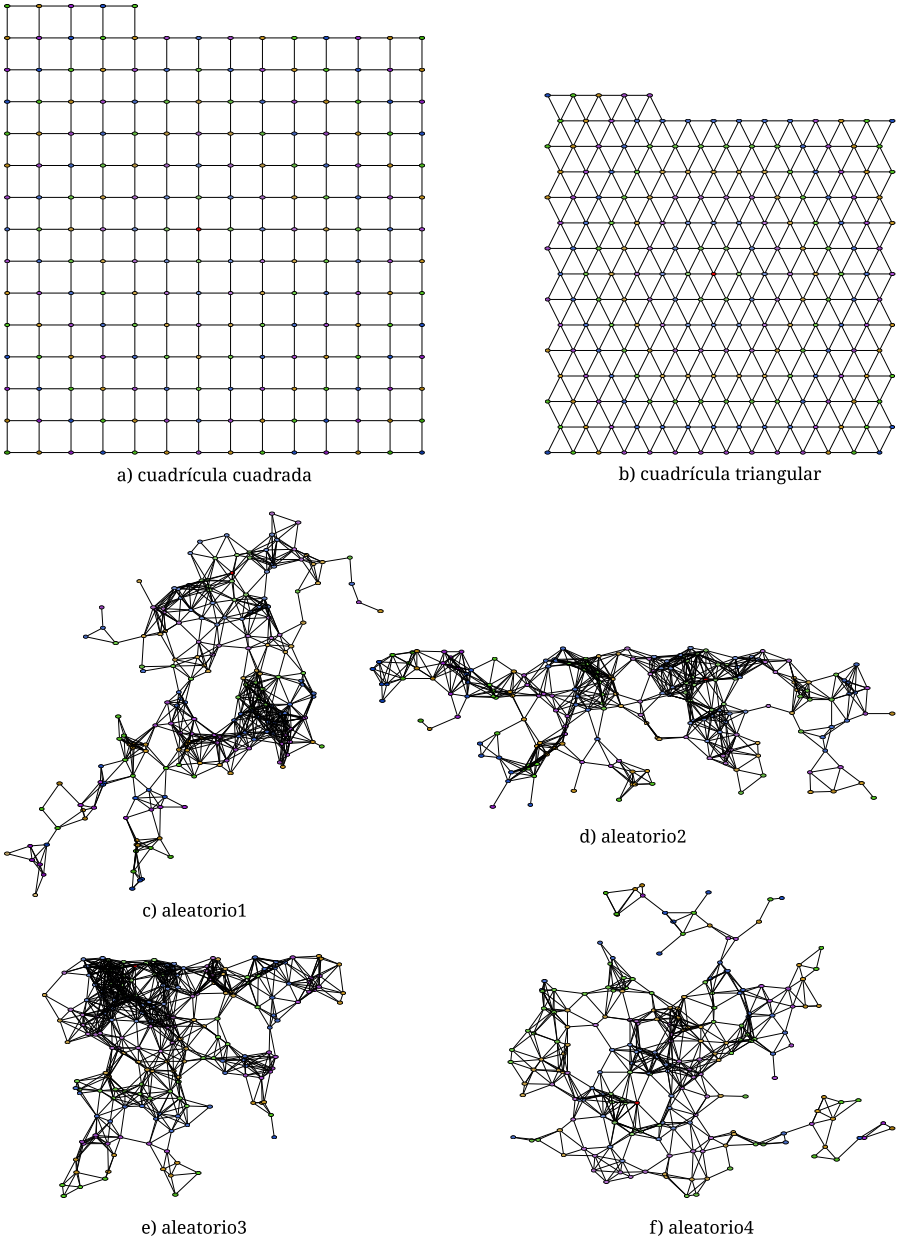


Figura 5.30 Topologías de los experimentos con fallos fatales.

estándar de cada medida. Después de la reconfiguración de la red solo permanecen los bloqueos debido a la topología (que se muestran con líneas naranja).

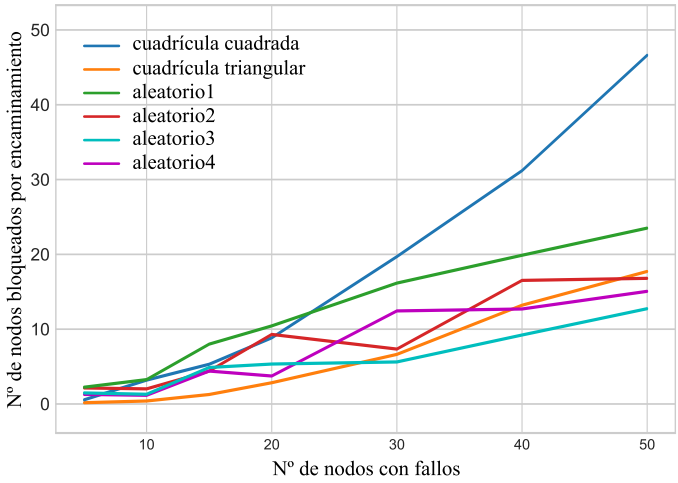


Figura 5.31 Número de nodos bloqueados debido al encaminamiento según los fallos fatales.

Se observa una relación inversa entre el número de nodos bloqueados por encaminamiento con el número medio de vecinos. A mayor número de vecinos medio, menor número de nodos bloqueados por encaminamiento. Esto es lógico, ya que existen más rutas posibles.

Existe un comportamiento irregular en escenarios aleatorios como era de esperar, ya que a veces hay menos bloqueados al aumentar el número de fallos. También se observa una gran desviación estándar, resultado que era esperable por usar elecciones aleatorias.

En resumen, con pocos fallos el número nodos de bloqueados suele ser muy bajo. Con muchos fallos (25 % de los nodos), salvo en la topología de cuadrícula cuadrada, se bloquean menos de la mitad de los nodos que han fallado. Si se compara con los resultados de INSENS [25], el número de nodos bloqueados en SNSR es bastante inferior. Se ha comprobado que haciendo una reconfiguración todos los nodos bloqueados por encaminamiento se desbloqueaban. Por tanto, SNSR reduce bastante el impacto de la destrucción de nodos en todos los escenarios incluso en el improbable caso de que se destruyan 50 nodos simultáneamente.

5.10 Conclusiones

Ante todo, los experimentos demuestran que la implementación de SNSR es estable tras hacer miles de experimentos sin errores y se ajusta a lo especificado en la arquitectura. La iteración de las fases, la nueva torre de protocolos, la recogida de anomalías, etc. funcionan correctamente y dan lugar a WSN totalmente operativas y robustas antes ataques y fallos. Se ha probado que una red de sensores puede funcionar eliminando de los nodos tareas periódicas como el descubrimiento de vecinos, la admisión de nuevos nodos y la ejecución de protocolos de encaminamiento dinámico. Estas tareas requieren recursos que con SNSR ya no hay que destinar. Sin necesitarlas, la creación de la red es posible con un coste contenido y manteniendo la capacidad de reconfiguración de una red de sensores.

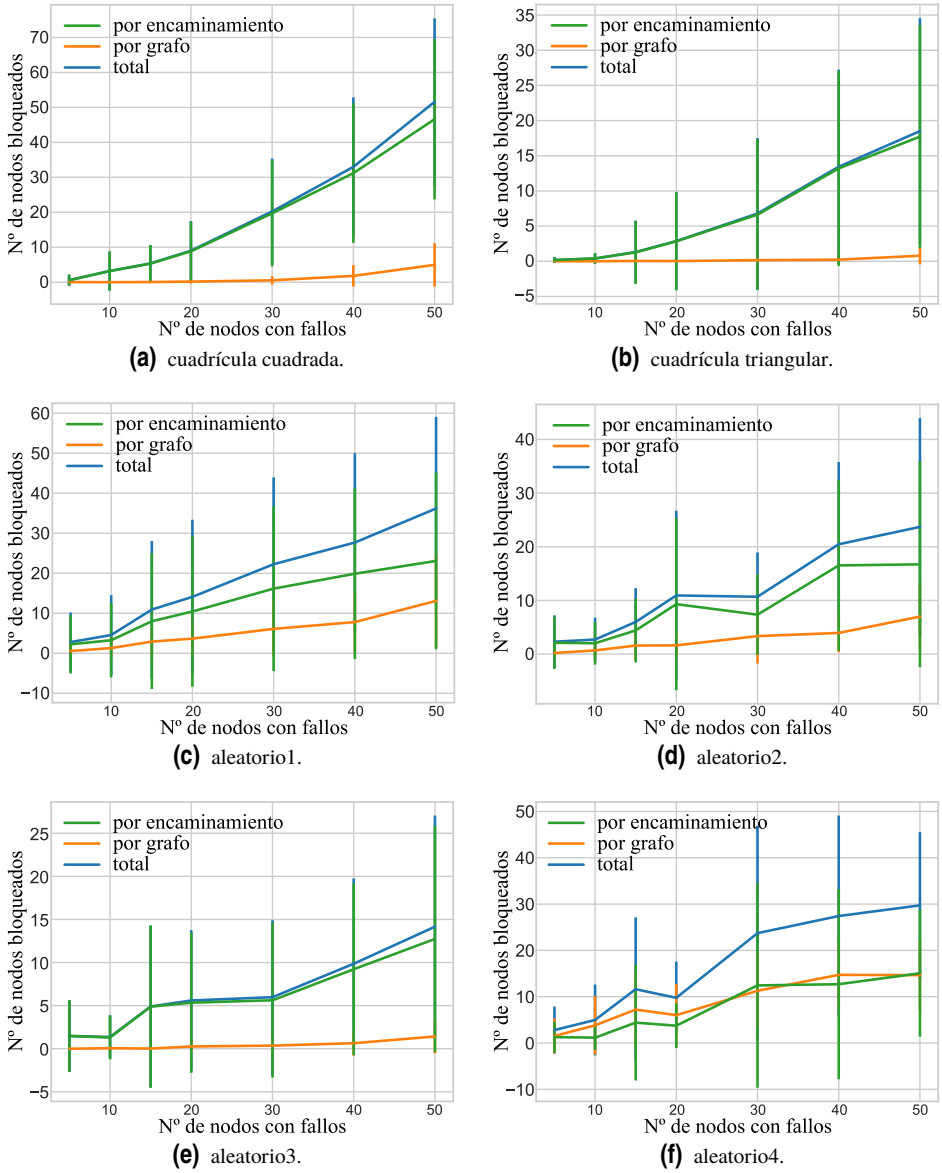


Figura 5.32 Experimentos con fallos fatales: datos de cada topología.

Desde el punto de vista funcional, el análisis de seguridad muestra como SNSR hace imposible la realización de ataques a los que otras redes son vulnerables al reducir la superficie de ataque posible. Además, limita mucho los efectos de aquellos ataques que aún serían inevitables. En comparación con ZSE e INSENS, SNSR es más segura, como se concluye en la Sección 5.2.35.

Se ha dedicado mucho esfuerzo en realizar experimentos en condiciones reales, no solo teóricos o simulados. Por eso se han utilizado robots terrestres y aéreos en algunos de ellos. Sin embargo, también se han validado los *lanzadores*, destacando su gran fidelidad a los resultados reales, ya que ejecutan el mismo software, aunque en entornos virtualizados.

Las principales métricas obtenidas en los experimentos sin ataques han buscado evaluar las fases del establecimiento de la red y el coste que implica para un nodo sensor. Se han obtenido tiempos reducidos y tráfico bajo para que un nodo esté operativo. Una vez configurada la red, el funcionamiento es similar al de otras WSN la mayor parte del tiempo, salvo que los nodos realizan menos tareas reduciendo el consumo de energía.

Se ha evaluado el coste que implica usar configuraciones más seguras en el establecimiento de la red, que es cuando se realizan las operaciones más costosas de criptografía asimétrica y el establecimiento inicial de conexiones. Este coste se amortiza con el tiempo durante la operativa normal de la red cuando no hay fallos.

SNSR ha mostrado ser robusto ante topologías diferentes y aleatorias y en redes de hasta 500 nodos virtualizados. Los resultados muestran una influencia destacable de la trayectoria del robot durante la configuración en los periodos transitorios de inestabilidad, ya que estos se producen al enviar la configuración antes a nodos que dependen de otros aún no configurados para poder comunicarse con la BS.

Por último, se ha comprobado la seguridad de SNSR, realizando simulaciones de ataques, incluso permitiendo ataques de terceras personas. Se ha prestado especial atención a los ataques de suplantación de robot, debido a la importancia de la comunicación del robot con los nodos en SNSR, y de destrucción de nodos, por sus posibles efectos en el encaminamiento implementado. Los ataques de suplantación de robot eran detectados con facilidad, pero se obtuvo que usar el método de detección de baliza avanzado, propuesto en la Sección 3.6.1, limitaba significativamente sus efectos temporales. En los ataques de destrucción de nodos, los resultados muestran que, a pesar de no usar protocolos dinámicos, el sistema es bastante robusto incluso a la destrucción de muchos nodos y que es posible recuperarse de los daños, sin necesidad de que los nodos tengan que actuar.

6 Conclusiones

La verdadera ciencia enseña, sobre todo, a dudar y a ser ignorante.

MIGUEL DE UNAMUNO

En este capítulo, primero, en la Sección 6.1, se presentan las conclusiones finales de esta Tesis Doctoral y posteriormente, en la Sección 6.2, se proponen líneas futuras de trabajo que permiten extender SNSR y explorar otras alternativas.

6.1 Conclusiones

Esta Tesis Doctoral propone SNSR, una arquitectura de seguridad novedosa, flexible y abierta que aprovecha la cooperación de robots con las WSN para lograr niveles de seguridad altos sin necesidad de usar mecanismos complejos. Su operación se estructura en fases organizadas usando un enfoque de retroalimentación, que permite repetir las fases para adaptarse a cambios en la red, ataques y corregir errores.

Para realizar este diseño ha sido necesario un análisis minucioso del estado actual de la seguridad en las WSN y de las tecnologías existentes. Por una parte, en muchas aplicaciones de monitorización basadas en WSN, sobre todo las usadas en entornos industriales o grandes infraestructuras civiles, la seguridad es un problema crítico. La gran superficie de ataque que presentan las WSN ha motivado el desarrollo de técnicas y protocolos para prevenir, detectar y mitigar los posibles ataques. Muchas de ellas se centran en ataques específicos, de tal manera que lograr una WSN segura requeriría la combinación de muchas de estas técnicas. Además, pueden ser incompatibles entre sí o suponer un consumo de recursos excesivo.

Por otra parte, los robots aéreos se están usando en muchas tareas de monitorización, como inspección, detección de fallos o mantenimiento predictivo, y esta tendencia parece ir en aumento. Sin embargo, la cooperación entre WSN y robots para mejorar la seguridad de las WSN ha sido poco investigada.

SNSR se beneficia de la capacidad de actuación del robot, que localiza y autentica nodos, interactúa con ellos directamente sin intermediarios para enviarles configuraciones y recibir informes de estado y anomalías, y puede utilizarse para obtener observaciones directas que confirmen o descarten anomalías detectadas. Estos usos del robot brindan a SNSR capacidades mejoradas de detección y mitigación de ataques, y mejoran la prevención al reemplazar tareas que en las arquitecturas tradicionales son propensas a ataques o requieren información preestablecida. Los nodos sensores son el elemento más débil de la WSN y, por tanto, más fácil de atacar. La arquitectura SNSR reduce la dependencia de unos nodos respecto a otros. Si un atacante se hace con el control de un nodo, el efecto es mucho más limitado que en otras WSN. Un nodo malicioso no puede alterar el funcionamiento de la red, como mucho puede provocar una degradación del funcionamiento a nivel local.

La arquitectura SNSR es lo suficientemente genérica y flexible para adaptarse a la infraestructura WSN concreta de cada caso de monitorización. La gran cantidad de opciones disponibles en SNSR permite su adaptación a las necesidades de distintas aplicaciones, por ejemplo, seleccionando el nivel de seguridad deseado. Incluso con requisitos bajos de seguridad, SNSR permite mejorar la eficiencia de las redes habituales al reducir la carga de trabajo de los nodos sensores. Esta Tesis Doctoral se ha centrado en el caso de una WSN con una BS y un robot. Sin embargo, los protocolos y muchas de las operaciones definidas se podrían utilizar en redes con múltiples BS o robots, sin modificación.

Para hacer realidad SNSR, se tomó la decisión de implementar un nuevo NOS, diferente a los empleados actualmente de tal manera que permita probar toda la flexibilidad disponible. Este NOS es independiente del uso de SNSR, pero se ha diseñado teniendo en cuenta exclusivamente sus requisitos. Es básico, pero fácilmente ampliable. Además, se basa en el funcionamiento de WSN ya existentes y usa técnicas empleadas por otros protocolos, pero la implementación es completamente nueva. Destacan la posibilidad de usarlo tanto en equipos reales como virtualizados, el uso reducido de recursos conforme a las limitaciones de los nodos sensores y la posibilidad de adaptarlo a otros sistemas operativos gracias al uso del LLAL. La adaptación de SNSR a este nuevo NOS ha requerido la toma de decisiones de implementación y resolver problemas como el cálculo de topologías, detalle y codificación de los mensajes del protocolo, uso de criptografía ECC, empleo del protocolo DTLS, trayectorias a realizar por el robot, etc. de manera también flexible para poder hacer experimentos posteriores con diferentes configuraciones, incluyendo versiones maliciosas de las entidades.

Se ha presentado un análisis de seguridad cualitativo que muestra cómo la reducción de la superficie de ataque de la WSN y los mecanismos empleados hacen que muchos ataques sean imposibles de realizarse en SNSR. Se ha hecho una comparación con la seguridad que tienen ZSE e INSENS, comprobándose que SNSR es más segura.

Para experimentar con la implementación de SNSR se ha creado una plataforma de pruebas nueva tanto en escenarios reales como virtualizados. Se ha creado una PKI para generar claves y certificados, un sistema de gestión remota de dispositivos reales para facilitar las pruebas con distintas configuraciones y dos compositores/orquestadores de escenarios simulados/virtualizados, que hemos denominado *lanzadores* con dos formas diferentes de simular los canales de comunicación. En uno de ellos, incluso se ha creado un módulo de movimiento simulado que es capaz de calcular las trayectorias del robot.

Muchos de los componentes desarrollados en esta Tesis Doctoral pueden ser reutilizados en otras investigaciones sobre las WSN, aunque no estén relacionadas con la seguridad o con robots. Por ejemplo, el NOS es genérico y permite crear nuevas WSN, el *lanzador* permite probar código real en entornos simulados sin requerir usar robots, el sistema de gestión remota se puede usar en pruebas de otros trabajos que requieran usar muchos dispositivos reales, y el protocolo de transporte se puede utilizar incluso en otras redes que no sean WSN como las convencionales.

Para validar la implementación y caracterizar el comportamiento de SNSR se han realizado miles de experimentos con distintos escenarios y configuraciones: experimentos virtualizados; experimentos de campo reales con robots terrestres y aéreos; con 5 configuraciones específicas; con topologías diferentes y aleatorias y en redes de distinto tamaño (hasta 500 nodos); con distintas medidas de seguridad; con distintos métodos de detección de baliza; y en condiciones normales, con ataques realizados por terceras personas y con ataques propios de destrucción de nodos y de suplantación de robots. Se extraen las siguientes conclusiones de los resultados obtenidos:

- La implementación de SNSR es estable, no se producen errores en condiciones normales y se ajusta a lo especificado en la arquitectura. La iteración de las fases, la nueva torre de protocolos y las medidas de seguridad funcionan correctamente y dan lugar a redes de sensores totalmente operativas y robustas antes ataques y fallos. El funcionamiento una vez que la red está configurada es similar al de otras WSN, pero los nodos necesitan realizar menos tareas por lo que se reduce su consumo.
- El tiempo necesario para configurar un nodo es reducido y el tráfico generado es bajo.
- Añadir criptografía asimétrica como medida de seguridad aumenta el coste del establecimiento de la red, pero este coste se amortiza con el tiempo durante la operativa normal de la red.
- Los resultados muestran una influencia destacable de la trayectoria del robot durante la configuración en los periodos transitorios de inestabilidad.
- Los ataques de suplantación de robot son detectados con facilidad. El uso del método de detección de baliza avanzado limita significativamente sus efectos temporales.
- El sistema es robusto frente ataques de destrucción de nodos, a pesar de no usar protocolos dinámicos, y puede recuperarse de los daños sin necesidad de que los nodos tengan que actuar.

6.2 Líneas futuras de trabajo

El uso de robots para mejorar la seguridad es un tema poco investigado como demuestra el pequeño número de trabajos previos. Esta Tesis Doctoral presenta un trabajo innovador que puede ser la base de futuras investigaciones. La arquitectura SNSR es muy flexible y permite explorar otras posibilidades diferentes que no se han implementado. A continuación, se describen varias líneas futuras de trabajo.

Optimización de trayectorias del robot

En los experimentos se ha mostrado que el orden en el que se configuran los nodos influye en el tiempo de configuración y el periodo transitorio de inestabilidad. Si los nodos se configuran por orden de proximidad a la BS, según el orden BFS, se reducen los tiempos y el número de paquetes enviados. En la Tesis Doctoral se ha desarrollado una trayectoria eficaz pero no eficiente en el anterior sentido. Una posible mejora sería desarrollar métodos automáticos que optimicen *la trayectoria de configuración* según el anterior criterio (u otros) y compare los resultados obtenidos.

Arquitectura con N robots y M estaciones bases

SNSR se puede generalizar para el caso de N robots y M estaciones bases, que implica una mayor coordinación y sincronización entre estos elementos, permitiendo considerar redes más extensas. Si existen varios robots simultáneamente, habría que encontrar la mejor manera de dividir las tareas entre ellos para su realización en paralelo, calculando sus trayectorias para que no colisionen, o de seleccionar el robot que pueda desempeñarlas en menos tiempo. La información recogida por los robots debería ser fusionada para obtener un mapa completo de la red y sincronizar la información de configuración entre todos los robots y las BS. Los nodos sensores podrían tener asignadas una o varias BS. La implementación actual ya permite esto, aunque solo mantiene conexión con una de ellas y solo cambia de BS si falla la conexión. Para completarlo, habría que establecer una política de asignación de BS y de envío de los datos de los sensores (solo a una BS, a varias simultáneamente, aleatoriamente…).

Actualización remota de software firmado digitalmente

Actualmente, la instalación de software nuevo en los dispositivos sensores se realiza con el agente del sistema de gestión remota (ver Sección 4.6), que se encuentra instalado en los dispositivos. Este agente es un programa independiente al que realiza las funciones de nodo sensor. Su uso se limita a un entorno de pruebas y no está preparado para entornos de producción. Usar actualizaciones OTA sería otra posible solución, como se hace en ZigBee [172]. En este caso, sería la aplicación de gestión de los nodos la que realizaría la actualización y reiniciaría el dispositivo si hiciera falta. Para ello habría que definir un nuevo mensaje de gestión, en el que se enviaría el nuevo software. Dado que una actualización permite modificar el funcionamiento del software y los parámetros de inicialización que normalmente no pueden ser modificados mediante mensajes de configuración (su identidad, sus claves y certificados, certificado de la CA…), la seguridad requerida debe ser alta. Así que la opción más segura sería firmar digitalmente estas actualizaciones para que no puedan alterarse. Podría utilizarse la firma digital de la BS u otra entidad de mayor confianza. Los nodos validarían esta firma antes de aplicar la actualización.

Incorporar un IDS y otras contramedidas específicas para determinados ataques

En el análisis cualitativo de seguridad (ver Sección 5.2) se encontraron algunos ataques que, aunque con efectos limitados, pueden seguir produciéndose. A SNSR se le podrían añadir algunas contramedidas específicas para determinados ataques. Por ejemplo, integrar otros

métodos para detectar *selective forwarding* [54], dificultar el análisis de tráfico usando caminos alternativos aleatorios, etc.

Las anomalías recolectadas por la BS también pueden ser analizadas por un IDS como los comentados en la Sección 2.3.2. La arquitectura SNSR ya proporciona los mecanismos para recolectar anomalías y ser procesadas de manera centralizada, y poder realizar acciones correctivas. Dada la diversidad de técnicas existentes en el campo de los IDS, su utilización requiere un extenso estudio para evaluar cuál es la más apropiada.

También podría ampliarse SNSR con un sistema que facilite la notificación de los ataques detectados o eventos sospechosos a operadores, mediante interfaces de usuario existentes o nuevas, y la decisión de la acción correctiva a realizar, que podría ser automática en algunos casos.

Considerar el caso de robots maliciosos con certificados válidos

En SNSR se ha supuesto que la captura de un robot sería detectada rápidamente y su certificado sería revocado para evitar que se pueda utilizar. Todas las entidades usan una CRL que normalmente contiene números de serie de certificados revocados. No es necesario almacenar en los nodos los números de secuencia revocados de la BS u otros nodos porque la comunicación con estos se puede bloquear simplemente no incluyéndolos en la configuración, por lo que normalmente su CRL contendrá pocas entradas y solo de certificados de robots.

El caso de que el robot sea capturado, modificado maliciosamente y devuelto sin que sea detectado no está contemplado. También se podrían obtener sus claves y reutilizarlas en otros dispositivos que simulen ser robots.

Un robot malicioso con un certificado válido puede modificar la configuración y la CRL de los nodos, que actualmente se aceptan sin comprobaciones adicionales porque confían en robots con certificados válidos, y generar anomalías o información de descubrimiento falsas. Esto podría afectar al funcionamiento de la red, pero nunca permitiría introducir nuevas entidades sin certificados válidos en la red. Si se detecta, la BS (si aún tiene conexión con los nodos) y cualquier otro robot podría volver a restablecer la configuración y revocar el certificado del robot malicioso. El robot malicioso para tratar de retrasar su revocación solo puede alterar las CRL de los nodos:

- Revocando antes los certificados de otros robos y de la BS suponiendo que conoce los números de serie. Pero no puede evitar que se generen nuevos certificados más adelante.
- Llenando la memoria utilizada para almacenar la CRL.

Para contemplar estos casos, se propone evaluar las siguientes medidas adicionales que habría que incluir en SNSR:

- Firmar digitalmente las CRL con certificados de la BS o CA. Esto implicaría un coste mayor de validación y complica la transmisión incremental de la CRL, pero sería suficiente para resolver el problema al evitar la falsificación.
- Añadir comprobaciones en los nodos para que no acepten CRL con números de secuencia de las BS conocidas.

- Limitar el número de entradas de las CRL, pero permitiendo añadir siempre al menos un número de serie adicional aún a costa de borrar entradas más antiguas.
- Implementar la actualización remota de software firmado digitalmente antes comentada, que permitiría cambiar los certificados de la CA y reiniciar las CRL.

Adaptación de la implementación para usar un simulador de red de eventos discretos

En esta Tesis Doctoral no se emplean simuladores de red de eventos discretos como pueden ser *ns-3* [128], *OMNeT++* [154] u otros [30]. El módulo DCE (*Direct Code Execution*) de *ns-3* [18] que permite ejecutar código real sobre el simulador se ha probado, pero no es compatible con la implementación actual del NOS. Una posible línea de continuación sería la adaptación del código para hacerlo funcionar en un simulador de este tipo, que permitiría simular redes con muchos nodos. Para ello, habría que añadir al LLAL del NOS (ver Sección 4.3.2) un nuevo conector que envíe y reciba los datos a través del simulador. También habría que modificar la gestión de eventos en tiempo real, para que en vez de usar interrupciones del sistema operativo use los temporizadores del simulador.

Ampliación de la implementación

Aunque no mejorarían la seguridad de SNSR, se propone también realizar mejoras cuya incorporación podría reducir el número de paquetes enviados, mejorar el tiempo de respuesta y aumentar la flexibilidad para poder ser utilizado en otras aplicaciones Son las siguientes:

- Cálculo del número máximo de saltos que puede dar un paquete dinámicamente, global o por nodo. Actualmente, el número máximo de saltos se configura durante la inicialización, pero se podría establecer por configuración. Después de calcular la topología de la red se puede estimar un número máximo de saltos a partir del diámetro del grafo de la red o según las comunicaciones que debería realizar cada nodo.
- Envío de configuración diferencial. La configuración de un nodo se puede enviar de manera completa o diferencial respecto a la configuración anterior, lo que permite reducir su tamaño. La implementación actual del robot siempre envía la configuración completa, pero podría calcular la diferencia respecto a la anterior y enviar solo los cambios producidos.
- Compactación y establecimiento de un número máximo de anomalías. Esto está contemplado en SNSR, pero no ha sido implementado porque durante la experimentación interesaba obtener la mayor cantidad de información de depuración posible. En una red real esta medida reduciría el tamaño de los datos transmitidos. Habría que determinar cuándo interesa hacerlo y determinar el valor máximo de anomalías a almacenar según el tipo de nodo.
- Recepción simultánea de varios mensajes en TL. El protocolo de transporte implementado solo permite la transmisión simultánea de un único mensaje fragmentado en cada sentido. Si el nodo tiene suficiente memoria esto podría mejorarse permitiendo

la multiplexión de varios mensajes. Esto no era necesario en las pruebas realizadas, pero podría ser útil si hay varias aplicaciones coexistiendo en el mismo nodo.

- Filtro de conexiones en el robot para evitar intentos de conexión de nodos que no están autorizados. La implementación actual del robot permite la conexión de todas las entidades que descubre que tengan un certificado válido. Sin embargo, si existe una lista de entidades autorizadas, el robot podría ignorar estas peticiones de conexión y simplemente registrarlas para posteriormente analizarlas como anomalías.
- Obtener datos de calidad de los enlaces y enviarlos como parte del estado del nodo. Esto podría usarse para establecer otros criterios de elección de vecinos durante el cálculo de la topología.

Realizar otras implementaciones

SNSR admite multitud de implementaciones distintas. SNSR es muy flexible y permite usarlo en diferentes escenarios o actualizarlo a las nuevas tecnologías que vayan surgiendo. En general, cualquier decisión de implementación (ver Sección 4.1) es susceptible de ser cambiada. Esto ampliaría las WSN en las que se puede aplicar SNSR y permitiría hacer posteriormente comparaciones que verifiquen las bondades de las cambios respecto a la implementación actual.

Por ejemplo, otras implementaciones podrían usar:

- Otros protocolos de encaminamiento: SDN (usando una BS o un robot como controlador), encaminamiento explícito, protocolos dinámicos... El encaminamiento estático usado requiere almacenar muchas rutas en los nodos si la densidad de la red y el número de nodos son muy altos, como se explicó en Sección 4.1.7. Por tanto, en algunas redes, habría que buscar alternativas.
- Otros tipos de WSN, por ejemplo, con clústeres. SNSR no ha sido probado en WSN con clústeres y sería interesante comprobar su funcionamiento en ellas. La configuración enviada a los nodos puede ser ampliada con información del clúster al que pertenece y datos sobre el nodo elegido como CH.
- Otros sistemas operativos y en nodos con distinto hardware. No todas las WSN emplean los mismos dispositivos que se ha usado en esta Tesis Doctoral y si se quiere adaptar la arquitectura a esas redes habrá que hacer modificaciones en la implementación.
- Diferentes niveles inferiores, cambiando el LLAL. Existen muchos niveles físicos diferentes y, como antes, si se quiere adaptar la arquitectura a esas redes habrá que hacer modificaciones en la implementación.
- Otros protocolos seguros. En especial la nueva versión de DTLS 1.3 [127] que promete mensajes más compactos. Su implementación haría que el tráfico generado disminuyera.
- Otros tipos de criptografía que no sea ECC, por ejemplo, con algunas de las variantes de criptografía postcuántica que se están desarrollando. Así, si en un futuro, la computación cuántica hiciera que ECC dejara de ser seguro, SNSR estaría preparado.

También permitiría evaluar el coste de esos tipos de criptografía respecto a ECC y su viabilidad en WSN.

- Diferentes tipos de certificados. Los certificados son muy grandes. Gran parte del coste de transmisión del saludo es debido a ellos. Los certificados se podrían optimizar y definir un formato compacto propietario en vez de usar el formato estándar X.509. Esto podría reducir los datos transmitidos. Otra tarea sería probar con certificados que usaran otra función *hash*, ya que al fin de la escritura de esta Tesis Doctoral el algoritmo SHA-1 ya está desaconsejado. Esto está soportado en la implementación actual y simplemente habría que repetir los experimentos para ver cómo afecta.
- Cambios en la forma de transmisión de certificados. El robot podría recopilar todos los certificados de los nodos y la BS. Los nodos en vez de verificar si un certificado de un vecino es válido podría compararlo con una lista que les suministre el robot, que previamente han validado. El robot podría pasar todos los certificados en la configuración, así no haría falta pasarlos en saludos posteriores. Habría que evaluar en qué situaciones esto es más ventajoso que el procedimiento actual. La asignación de vecinos no es infalible y algunos certificados se habría transmitido para nada. También se puede probar con otras alternativas de criptografía asimétrica: criptografía basada en identidad o sin certificados. Esto permitiría poder prescindir de la CA, aunque habría que evaluar posibles riesgos de seguridad.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] *ISO/IEC/IEEE International Standard - Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements–Part 15-4: Wireless medium access control (MAC) and physical layer (PHY*, ISO/IEC/IEEE 8802-15-4:2018(E) (2018), 1–712.

[2] Muhammad Ahmed, Xu Huang, and Hongyan Cui, *A novel two-stage algorithm protecting internal attack from WSNs*, International Journal of Computer Networks & Communications (IJCNC) **5** (2013), no. 1.

[3] Ohida Rufai Ahutu and Hosam El-Ocla, *Centralized routing protocol for detecting wormhole attacks in wireless sensor networks*, IEEE Access **8** (2020), 63270–63282.

[4] Petteri Aimonen, *Nanopb - protocol buffers with small code size*, https://jpa.kapsi.fi/nanopb/, [En línea. Consulta: 28-04-2019].

[5] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, *Wireless sensor networks: a survey*, Computer Networks **38** (2002), no. 4, 393–422.

[6] Jamal N. Al-Karaki and Ahmed E. Kamal, *Routing techniques in wireless sensor networks: A survey*, IEEE Wireless Communications **11** (2004), no. 6, 6–27.

[7] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan, *Homomorphic encryption security standard*, Tech. report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[8] Zigbee Alliance, *Zigbee Specification*, Zigbee Alliance website (2015), 1–542.

[9] Elaine Barker, *Recommendation for key management part 1: General*, Tech. report, National Institute of Standards and Technology, Gaithersburg, MD, jan 2016.

[10] Paolo Baronti, Prashant Pillai, Vince W.C. Chook, Stefano Chessa, Alberto Gotta, and Y. Fun Hu, *Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards*, Computer Communications **30** (2007), no. 7, 1655–1695.

[11] Deebak B.D. and Fadi Al-Turjman, *A hybrid secure routing and monitoring mechanism in IoT-based wireless sensor networks*, Ad Hoc Networks **97** (2020), 102022.

[12] Philippe Biondi and the Scapy community, *Scapy: Packet crafting for Python2 and Python3*, https://scapy.net, [En línea. Consulta: 28-04-2019].

[13] Alex Biryukov and Eyal Kushilevitz, *From differential cryptanalysis to ciphertext-only attacks*, Advances in Cryptology — CRYPTO '98 (Berlin, Heidelberg) (Hugo Krawczyk, ed.), Springer Berlin Heidelberg, 1998, pp. 72–88.

[14] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and Dave Cooper, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 5280, May 2008.

[15] Djallel Eddine Boubiche, Samir Athmani, Sabrina Boubiche, and Homero Toral-Cruz, *Cybersecurity issues in wireless sensor networks: Current challenges and solutions*, Wireless Personal Communications (2020), 1–37.

[16] Ismail Butun, Salvatore D. Morgera, and Ravi Sankar, *A survey of intrusion detection systems in wireless sensor networks*, IEEE Communications Surveys & Tutorials **16** (2014), no. 1, 266–282.

[17] Cristiano Calcagno and Dino Distefano, *Infer: An automatic program verifier for memory safety of C programs*, Springer, Berlin, Heidelberg, 2011, pp. 459–465.

[18] Daniel Camara, Hajime Tazaki, Emilio Mancini, Thierry Turletti, Walid Dabbous, and Mathieu Lacage, *DCE: Test the real code of your protocols and applications over simulated networks*, IEEE Communications Magazine **52** (2014), no. 3, 104–110.

[19] Certicom Research, *Standards for Efficient Cryptography 1 (SEC 1) : Elliptic Curve Cryptography*, Standards for Efficient Cryptography **1** (2009), no. Sec 1, 1–22.

[20] Howie Choset and Philippe Pignon, *Coverage path planning: The boustrophedon cellular decomposition*, Field and Service Robotics, Springer London, London, 1998, pp. 203–209.

[21] Gerald Combs and the WireShark team, *Wireshark · Go Deep.*, https://www.wireshark.org, [En línea. Consulta: 28-04-2019].

[22] cozybit Inc., *cozybit/wmediumd: mac80211_hwsim modifications and related stuff*, https://github.com/cozybit/wmediumd, [En línea. Consulta: 28-04-2019].

[23] Alberto De San Bernabé Clemente, José Ramiro Martínez-De Dios, Carolina Regoli, and Aníbal Ollero, *Wireless sensor network connectivity and redundancy repairing with mobile robots*, Studies in Computational Intelligence **554** (2014), 185–204.

[24] Dr. Steve E. Deering and Bob Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 8200, July 2017.

[25] Jing Deng, Richard Han, and Shivakant Mishra, *INSENS : Intrusion-Tolerant Routing in Wireless Sensor Networks ; CU-CS-939-02*, Tech. report, 2002.

[26] _____, *INSENS: Intrusion-tolerant routing for wireless sensor networks*, Computer Communications **29** (2006), no. 2, 216–230.

[27] Valgrind Developers, *Valgrind Home*, http://www.valgrind.org, [En línea. Consulta: 28-04-2019].

[28] Martin Devera, *Hierachical token bucket theory*, http://luxik.cdi.cz/ devik/qos/htb/-manual/theory.htm, 2002, [En línea. Consulta: 28-04-2019].

[29] R. Di Pietro, S. Guarino, N. V. Verde, and J. Domingo-Ferrer, *Security in wireless ad-hoc networks - a survey*, Computer Communications **51** (2014), 1–20.

[30] Alvaro Diaz and Pablo Sanchez, *Simulation of attacks for security in wireless sensor network*, Sensors **16** (2016), no. 11.

[31] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, RFC Editor, August 2008.

[32] W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), no. 6, 644–654.

[33] Ramon dos Reis Fontes, *GitHub - ramonfontes/manual-mininet-wifi*, https://github.com/ramonfontes/manual-mininet-wifi, [En línea. Consulta: 28-04-2019].

[34] Miao Du and Kun Wang, *An SDN-enabled pseudo-honeypot strategy for distributed denial of service attacks in industrial internet of things*, IEEE Transactions on Industrial Informatics **16** (2020), 648–657.

[35] A. Dunkels, B. Gronvall, and T. Voigt, *Contiki - a lightweight and flexible operating system for tiny networked sensors*, 29th Annual IEEE International Conference on Local Computer Networks, IEEE (Comput. Soc.), pp. 455–462.

[36] Adam Dunkels and Adam, *Full TCP/IP for 8-bit architectures*, Proceedings of the 1st international conference on Mobile systems, applications and services - MobiSys '03 (New York, New York, USA), ACM Press, 2003, pp. 85–98.

[37] Khalid EL Gholami, Maleh Yassine, and Imade Fahd-Eddine Fatani, *The ieee 802.15.4 standard in industrial applications: A survey*, Journal of Theoretical and Applied Information Technology **99** (2021).

[38] Pasi Eronen, Hannes Tschofenig, Hao Zhou, and Joseph A. Salowey, *Transport Layer Security (TLS) Session Resumption without Server-Side State*, RFC 5077, January 2008.

[39] Muhammad Omer Farooq and Thomas Kunz, *Operating systems for wireless sensor networks: a survey*, Sensors (Basel, Switzerland) **11** (2011), no. 6, 5900–30.

[40] Francisco J Fernández-Jiménez and J Ramiro Martínez-de Dios, *Design of a robot-sensor network security architecture for monitoring applications*, ROBOT 2017: Third Iberian Robotics Conference (Cham) (Anibal Ollero, Alberto Sanfeliu, Luis Montano, Nuno Lau, and Carlos Cardeira, eds.), Springer International Publishing, 2018, pp. 200–212.

[41] Francisco Jose Fernandez-Jimenez and Jose Ramiro Martinez De Dios, *A robot-sensor network security architecture for monitoring applications*, IEEE Internet of Things Journal **9** (2022), no. 8, 6288–6304.

[42] Francisco José Fernández-Jiménez and José Ramiro Martínez-de Dios, *Dataset of SNSR experiments*, http://dx.doi.org/10.21227/62fj-8b04, [En línea. Consulta: 17-05-2022].

[43] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson, *Scalable vector graphics (svg) 1.0 specification*, iuniverse Bloomington, 2000.

[44] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Dr. Allyn Romanow, *TCP Selective Acknowledgment Options*, RFC 2018, October 1996.

[45] Ramon R. Fontes, Samira Afzal, Samuel H. B. Brito, Mateus A. S. Santos, and Christian Esteve Rothenberg, *Mininet-WiFi: Emulating software-defined wireless networks*, 2015 11th International Conference on Network and Service Management (CNSM), IEEE, nov 2015, pp. 384–389.

[46] Enric Galceran and Marc Carreras, *A survey on coverage path planning for robotics*, Robotics and Autonomous Systems **61** (2013), no. 12, 1258–1276.

[47] Laura Galluccio, Sebastiano Milardo, Giacomo Morabito, and Sergio Palazzo, *SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks*, IEEE Conf. on Computer Communications (INFOCOM), apr 2015, pp. 513–521.

[48] L Gandhimathi and G Murugaboopathi, *Mobile malicious node detection using mobile agent in cluster-based wireless sensor networks*, Wireless Personal Communications **117** (2021), 1209–1222.

[49] Emden R Gansner and Stephen C North, *An open graph visualization system and its applications to software engineering*, Software: practice and experience **30** (2000), no. 11, 1203–1233.

[50] Gunnar Gaubatz, Jens-Peter Kaps, and Berk Sunar, *Public key cryptography in sensor networks-revisited*, Springer, Berlin, Heidelberg, 2005, pp. 2–18.

[51] Amit Kumar Gautam and Rakesh Kumar, *A comprehensive study on key management, authentication and trust management techniques in wireless sensor networks*, SN Applied Sciences **3** (2021), 50.

[52] Cyril Gavoille and Stéphane Pérennès, *Memory requirement for routing in distributed networks*, Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing - PODC '96 (New York, New York, USA), ACM Press, 1996, pp. 125–133.

[53] Google Inc., *Protocol Buffers | Google Developers*, https://developers.google.com/protocol-buffers/, [En línea. Consulta: 28-04-2019].

[54] T. H. Hai and E. Huh, *Detecting selective forwarding attacks in wireless sensor networks using two-hops neighbor knowledge*, 2008 Seventh IEEE International Symposium on Network Computing and Applications, July 2008, pp. 325–331.

[55] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick Mckeown, *Reproducible network experiments using container-based emulation*, 2012.

[56] Anne Marie Hegland, Eli Winjum, Stig F Mjolsnes, Chunming Rong, Oivind Kure, and Pal Spilling, *A survey of key management in ad hoc networks*, IEEE Communications Surveys & Tutorials **8** (2006), no. 3, 48–66 (English).

[57] Stephen Hemminger and Others, *Network emulation with NetEm*, Linux conf au, 2005, pp. 18–23.

[58] Faouzi Hidoussi, Homero Toral-Cruz, Djallel Eddine Boubiche, Kamaljit Lakhtaria, Albena Mihovska, Miroslav Voznak, and Miroslav Voznak voznak, *Centralized IDS based on misuse detection for cluster-based wireless sensors networks*, Wireless Pers Commun **85** (2015), 207–224.

[59] Chen Hongsong, Meng Caixia, Fu Zhongchuan, and Chao Hsien Lee, *Novel LDoS attack detection by spark-assisted correlation analysis approach in wireless sensor network*, IET Information Security **14** (2020), 452–458.

[60] E Horowitz and S Sahni, *Fundamentals of computer algorithms, computer science press*, Computer Software Engineering Series (1978), 526–529.

[61] Fei Hu, Qi Hao, and Ke Bao, *A survey on software-defined network and OpenFlow: From concept to implementation*, IEEE Communications Surveys & Tutorials **16** (2014), no. 4, 2181–2206.

[62] René Hummen, Jan H. Ziegeldorf, Hossein Shafagh, Shahid Raza, and Klaus Wehrle, *Towards viable certificate-based authentication for the internet of things*, HotWiSec 2013 - Proceedings of the 2013 ACM Workshop on Hot Topics on Wireless Network Security and Privacy (New York, New York, USA), ACM Press, 2013, pp. 37–41.

[63] Safwan Mawlood Hussein, Juan Antonio López Ramos, and José Antonio Álvarez Bermejo, *Distributed key management to secure IoT wireless sensor networks in smart-agro*, Sensors **20** (2020), 2242.

[64] Christophe Huygens and Wouter Joosen, *Federated and shared use of sensor networks through security middleware*, 2009 Sixth International Conference on Information Technology: New Generations, 2009, pp. 1005–1011.

[65] IANA, *Transport Layer Security (TLS) Parameters*, https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml, [En línea. Consulta: 28-04-2019].

[66] IEEE Computer Society. Portable Applications Standards Committee., IEEE Standards Board., and American National Standards Institute., *IEEE standard for information technology : portable operating system interface (POSIX) : Part 1, System application program interface (API) : Amendment 1, Realtime extension [C language]*, Institute of Electrical and Electronics Engineers, 1994.

[67] *IEEE Standard Specifications for Public-Key Cryptography*, IEEE Std 1363-2000 (2000), 1–228.

[68] *IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices*, IEEE Std 1363.1-2008 (2009), C1–69.

[69] *IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques*, IEEE Std 1363.2-2008 (2009), 1–127.

[70] *IEEE Standard for Identity-Based Cryptographic Techniques using Pairings*, IEEE Std 1363.3-2013 (2013), 1–151.

[71] *IEEE Standard Specifications for Public-Key Cryptography - Amendment 1: Additional Techniques*, IEEE Std 1363a-2004 (Amendment to IEEE Std 1363-2000) (2004), 1–167.

[72] Jana Iyengar and Ian Swett, *QUIC Loss Detection and Congestion Control*, Internet-Draft draft-ietf-quic-recovery-22, Internet Engineering Task Force, July 2019, Work in Progress.

[73] Jana Iyengar and Martin Thomson, *QUIC: A UDP-Based Multiplexed and Secure Transport*, Internet-Draft draft-ietf-quic-transport-22, Internet Engineering Task Force, July 2019, Work in Progress.

[74] Steven Jacobson, Van and Leres, Craig and McCanne, *Tcpdump/Libpcap public repository*, https://www.tcpdump.org, [En línea. Consulta: 28-04-2019].

[75] Bo Jiang, Guosheng Huang, Tian Wang, Jinsong Gui, and Xiaoyu Zhu, *Trust based energy efficient data collection with unmanned aerial vehicle in edge network*, Transactions on Emerging Telecommunications Technologies (2020).

[76] Shuai Jiang, Juan Zhao, and Xiaolong Xu, *SLGBM: An intrusion detection mechanism for wireless sensor networks in smart environments*, IEEE Access **8** (2020), 169548–169558.

[77] Gauri Kalnoor and Jayashree Agarkhed, *Pattern matching intrusion detection technique for wireless sensor networks*, Proceeding of IEEE - 2nd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics, IEEE - AEEICB 2016 (2016), 724–728.

[78] Dionisis Kandris, Christos Nakas, Dimitrios Vomvas, and Grigorios Koulouras, *Applications of wireless sensor networks: An up-to-date survey*, Applied System Innovation **3** (2020), no. 1, 14.

[79] Phil R. Karn and William A. Simpson, *Photuris: Session-Key Management Protocol*, RFC 2522, March 1999.

[80] Charlie Kaufman, Paul E. Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen, *Internet Key Exchange Protocol Version 2 (IKEv2)*, RFC 7296, October 2014.

[81] S. Kent and K. Seo, *Security architecture for the Internet Protocol*, RFC 4301, RFC Editor, December 2005.

[82] D. Kim and S. An, *PKC-Based DoS attacks-resistant scheme in wireless sensor networks*, IEEE Sensors Journal **16** (2016), no. 8, 2217–2218.

[83] Daehee Kim and Sunshin An, *Efficient and scalable public key infrastructure for wireless sensor networks*, The 2014 Intl. Symposium on Networks, Computers and Communications, IEEE, jun 2014, pp. 1–5.

[84] Neal Koblitz, *Elliptic curve cryptosystems*, Mathematics of computation **48** (1987), no. 177, 203–209.

[85] Dmitri Krioukov, k c Claffy, Kevin Fall, and Arthur Brady, *On compact routing for the internet*, ACM SIGCOMM Computer Communication Review **37** (2007), no. 3, 41.

[86] Scott Law, Laurie and Menezes, Alfred and Qu, Minghua and Solinas, Jerry and Vanstone, *An efficient protocol for authenticated key agreement*, Designs, Codes and Cryptography **28** (2003), 119–134.

[87] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, *TinyOS: An operating system for sensor networks*, Ambient Intelligence, Springer-Verlag, Berlin/Heidelberg, 2005, pp. 115–148.

[88] Fangyu Li, Rui Xie, Zengyan Wang, Lulu Guo, Jin Ye, Ping Ma, and Wenzhan Song, *Online distributed IoT security monitoring with multidimensional streaming big data*, IEEE Internet of Things Journal **7** (2020), no. 5, 4387–4394.

[89] Jinhui Liu, Lianhai Wang, and Yong Yu, *Improved security of a pairing-free certificateless aggregate signature in healthcare wireless medical sensor networks*, IEEE Internet of Things Journal **7** (2020), 5256–5266.

[90] V.J. Lumelsky, S. Mukhopadhyay, and K. Sun, *Dynamic path planning in sensor-based terrain acquisition*, IEEE Transactions on Robotics and Automation **6** (1990), no. 4, 462–472.

[91] Ivan I. Lysogor, Leonid S. Voskov, and Sergey G. Efremov, *Survey of data exchange formats for heterogeneous LPWAN-satellite IoT networks*, 2018 Moscow Workshop on Electronic and Networking Technologies (MWENT), IEEE, mar 2018, pp. 1–5.

[92] Xin Ma and Wei Luo, *The analysis of 6LowPAN technology*, 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, IEEE, dec 2008, pp. 963–966.

[93] Daniel Marjamaki, *Cppcheck - A tool for static C/C++ code analysis*, http://cppcheck.sourceforge.net, [En línea. Consulta: 28-04-2019].

[94] J. R. Martinez-De Dios, K. Lferd, A. De San Bernabé, G. Núñez, A. Torres-González, and A. Ollero, *Cooperation between UAS and wireless sensor networks for efficient data collection in large environments*, Journal of Intelligent and Robotic Systems: Theory and Applications **70** (2013), no. 1-4, 491–508.

[95] José Ramiro Martínez-de Dios, Alberto de San Bernabé, Antidio Viguria, Arturo Torres-González, and Anibal Ollero, *Combining unmanned aerial systems and sensor networks for earth observation*, Remote Sensing 2017, Vol. 9, Page 336 **9** (2017), no. 4, 336.

[96] J. R. Martínez-de Dios and et al., *Aerial robot coworkers for autonomous localization of missing tools in manufacturing plants*, Intl. Conf. on Unmanned Aircraft Systems, 2018, pp. 1063–1069.

[97] John Mattsson and Francesca Palombini, *Comparison of CoAP Security Protocols*, Internet-Draft draft-ietf-lwig-security-protocol-comparison-03, Internet Engineering Task Force, March 2019, Work in Progress.

[98] Gabi Melman, *spdlog - Very fast, header only, C++ logging library*, https://github.com/gabime/spdlog, [En línea. Consulta: 28-04-2019].

[99] Christian Miranda, Georges Kaddoum, Elias Bou-Harb, Sahil Garg, and Kuljeet Kaur, *A collaborative security framework for software-defined wireless sensor networks*, IEEE Transactions on Information Forensics and Security **15** (2020), 2602–2615.

[100] P. Mockapetris, *Domain names - concepts and facilities*, RFC 1034, November 1987.

[101] Gabriel Montenegro, Jonathan Hui, David Culler, and Nandakishore Kushalnagar, *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, RFC 4944, September 2007.

[102] Javier Munoz-Calle, Francisco J. Fernandez-Jimenez, Teresa Ariza, Antonio J. Sierra, and Juan M. Vozmediano, *Computing labs on virtual environments: A flexible, portable, and multidisciplinary model*, Revista Iberoamericana de Tecnologias del Aprendizaje **11** (2016), no. 4, 235–341.

[103] Mansoor Nasir, Khan Muhammad, Paolo Bellavista, Mi Young Lee, and Muhammad Sajjad, *Prioritization and alert fusion in distributed IoT sensors using kademlia based distributed hash tables*, IEEE Access **8** (2020), 175194–175204.

[104] Musa Ndiaye, Gerhard P Hancke, and Adnan M Abu-Mahfouz, *Software defined networking for improved wireless sensor network management: A survey*, Sensors (Basel, Switzerland) **17** (2017), no. 5.

[105] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig, *The sybil attack in sensor networks: analysis & defenses*, Proceedings of the 3rd international symposium on Information processing in sensor networks, ACM, 2004, pp. 259–268.

[106] Stefan Nilsson and Gunnar Karlsson, *Fast address lookup for internet routers*, Broadband Communications, Springer US, Boston, MA, 1998, pp. 11–22.

[107] Yoav Nir, Simon Josefsson, and Manuel Pégourié-Gonnard, *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier*, RFC 8422, August 2018.

[108] Muhammad Numan, Fazli Subhan, Wazir Zada Khan, Saqib Hakak, Sajjad Haider, G. Thippa Reddy, Alireza Jolfaei, and Mamoun Alazab, *A systematic review on clone node detection in static wireless sensor networks*, IEEE Access **8** (2020), 65450–65461.

[109] Object Management Group Inc., *OMG ® Unified Modeling Language ® (OMG UML ®)*, 2.5.1 ed., no. December, 2017.

[110] Flauzac Olivier, Gonzalez Carlos, and Nolot Florent, *SDN based architecture for clustered WSN*, 2015 9th Intl. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing, IEEE, jul 2015, pp. 342–347.

[111] Anibal Ollero, Guillermo Heredia, Antonio Franchi, Gianluca Antonelli, Konstantin Kondak, Alberto Sanfeliu, Antidio Viguria, J. Ramiro Martinez-de Dios, Francesco Pierri, Juan Cortes, Angel Santamaria-Navarro, Miguel Angel Trujillo Soto, Ribin Balachandran, Juan Andrade-Cetto, and Angel Rodriguez, *The AEROARMS project: Aerial robots with advanced manipulation capabilities for inspection and maintenance*, IEEE Robotics and Automation Magazine **25** (2018), no. 4, 12–23.

[112] OpenSSL Software Foundation, *OpenSSL: Cryptography and SSL/TLS Toolkit*, https://www.openssl.org, [En línea. Consulta: 28-04-2019].

[113] ZigBee Standards Organization, *ZigBee Smart Energy Standard*, (2014), 1–628.

[114] B. Parno, A. Perrig, and V. Gligor, *Distributed detection of node replication attacks in sensor networks*, 2005 IEEE Symposium on Security and Privacy (S&P'05), IEEE, 2005, pp. 49–63.

[115] Vladislav Perelman, *Security in IPv6-enabled wireless sensor networks: An implementation of TLS/DTLS for the Contiki operating system*, PhD diss., MSc Thesis, Jacobs University Bremen (2012).

[116] Francisco J. Perez-Grau, J. Ramiro Martinez-de Dios, Julio L. Paneque, J. Joaquin Acevedo, Arturo Torres-González, Antidio Viguria, Juan R. Astorga, and Anibal Ollero, *Introducing autonomous aerial robots in industrial manufacturing*, Journal of Manufacturing Systems **60** (2021), 312–324.

[117] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler, *Spins: Security protocols for sensor networks*, Wireless Networks **8** (2002), no. 5, 521 (English).

[118] Popescu, Stoican, Stamatescu, Chenaru, and Ichim, *A survey of collaborative UAV–WSN systems for efficient monitoring*, Sensors **19** (2019), no. 21, 4690.

[119] Srdan Popic, Drazen Pezer, Bojan Mrazovac, and Nikola Teslic, *Performance evaluation of using Protocol Buffers in the internet of things communication*, 2016 International Conference on Smart Systems and Technologies (SST), IEEE, oct 2016, pp. 261–265.

[120] Sumit Pundir, Mohammad Wazid, Devesh Pratap Singh, Ashok Kumar Das, Joel J. P. C. Rodrigues, and Youngho Park, *Intrusion detection protocols in wireless sensor networks integrated to internet of things deployment: Survey and future challenges*, IEEE Access **8** (2020), 3343–3363.

[121] Rosheen Qazi, Kashif Naseer Qureshi, Faisal Bashir, Najam Ul Islam, Saleem Iqbal, and Arsalan Arshad, *Security protocol using elliptic curve cryptography algorithm for wireless sensor networks*, Journal of Ambient Intelligence and Humanized Computing **1** (2020), 3.

[122] José Ramiro Martínez-de Dios, Anibal Ollero, Francisco José Fernández, and Carolina Regoli, *On-line RSSI-range model learning for target localization and tracking*, Journal of Sensor and Actuator Networks **6** (2017), no. 3.

[123] Theodore S Rappaport, *Wireless communications : principles and practice*, 2nd ed. ed., Prentice Hall, Upper Saddle River, NJ, 2002.

[124] Shahid Raza, Simon Duquennoy, Tony Chung, Dogan Yazar, Thiemo Voigt, and Utz Roedig, *Securing communication in 6LoWPAN with compressed IPsec*, 2011 Intl. Conf. on Distributed Computing in Sensor Systems (DCOSS), jun 2011, pp. 1–8.

[125] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2*, RFC 6347, RFC Editor, January 2012.

[126] Eric Rescorla, *TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)*, RFC 5289, August 2008.

[127] Eric Rescorla, Hannes Tschofenig, and Nagendra Modadugu, *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*, Internet-Draft draft-ietf-tls-dtls13-34, Internet Engineering Task Force, November 2019, Work in Progress.

[128] George F. Riley and Thomas R. Henderson, *The ns-3 Network Simulator*, Modeling and Tools for Network Simulation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 15–34.

[129] Tudor Rogojanu, Mihai Ghita, Valeriu Stanciu, Radu-Ioan Ciobanu, Radu-Corneliu Marin, Florin Pop, and Ciprian Dobre, *NETIoT: A versatile IoT platform integrating sensors and applications*, 2018 Global Internet of Things Summit (GIoTS), IEEE, jun 2018, pp. 1–6.

[130] Ozgur Koray Sahingoz, *Large scale wireless sensor networks with multi-level dynamic key management scheme*, Journal of Systems Architecture **59** (2013), no. 9, 801–807.

[131] Amit Sarkar and T. Senthil Murugan, *Routing protocols for wireless sensor networks: What the literature says?*, Alexandria Engineering Journal **55** (2016), no. 4, 3173–3183.

[132] Gustavo A.Nunez Segura, Sotiris Skaperas, Arsenia Chorti, Lefteris Mamatas, and Cintia Borges Margi, *Denial of service attacks detection in software-defined wireless sensor networks*, Institute of Electrical and Electronics Engineers Inc., 6 2020.

[133] Jaydip Sen, *Routing security issues in wireless sensor networks: Attacks and defenses*, Sustainable Wireless Sensor Networks (Winston Seah and Yen Kheng Tan, eds.), IntechOpen, Rijeka, 2010.

[134] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, RFC 7252, RFC Editor, June 2014.

[135] Chia-Yen Shih, Jesús Capitán, Pedro José Marrón, Antidio Viguria, Francisco Alarcón, Marc Schwarzbach, Maximilian Laiacker, Konstantin Kondak, José Ramiro Martínez-de Dios, and Aníbal Ollero, *On the cooperation between mobile robots and wireless sensor networks*, Springer Berlin Heidelberg, 2014, pp. 67–86.

[136] Chia-Yen Shih, Jesús Capitán, Pedro José Marrón, Antidio Viguria, Francisco Alarcón, Marc Schwarzbach, Maximilian Laiacker, Konstantin Kondak, José Ramiro Martínez-de Dios, and Aníbal Ollero, *On the cooperation between mobile robots and wireless sensor networks*, pp. 67–86, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[137] Devu Manikantan Shila, Xianghui Cao, Yu Cheng, Zequ Yang, Yang Zhou, and Jiming Chen, *Ghost-in-the-Wireless: Energy depletion attack on ZigBee*, (2014), no. Mic, 1–13.

[138] S. Shue and J. M. Conrad, *A survey of robotic applications in wireless sensor networks*, Proc. IEEE Southeastcon, 2013, pp. 1–5.

[139] Keith Sklower, *A tree-based packet routing table for Berkeley UNIX*, USENIX Winter, vol. 1991, 1991, pp. 93–99.

[140] Brett Smith, *A quick guide to GPLv3*, Free Software Foundation, Inc. **4** (2007), 2008, [En línea. Consulta: 18-05-2019].

[141] C Sreedhar, S Madhusudhana Verma, and N Kasiviswanath, *A survey on security issues in wireless ad hoc network routing protocols*, Intl. J. on Computer Science and Engineering (IJCSE) **2** (2010), no. 2, 224–232 (English).

[142] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov, *The first collision for full SHA-1*, Tech. report.

[143] Randall R. Stewart, *Stream Control Transmission Protocol*, RFC 4960, September 2007.

[144] Niranjan Suri, Anders Hansson, Jan Nilsson, Piotr Lubkowski, Kelvin Marcus, Mariann Hauge, King Lee, Boyd Buchin, Levent Misirhoglu, and Markus Peuhkuri, *A realistic military scenario and emulation environment for experimenting with tactical communications and heterogeneous networks*, 2016 International Conference on Military Communications and Information Systems (ICMCIS), IEEE, may 2016, pp. 1–8.

[145] Robert Tarjan, *Depth-first search and linear graph algorithms*, SIAM Journal on Computing **1** (1972), no. 2, 146–160.

[146] B. Tas and A. S. Tosun, *Mobile assisted key distribution in wireless sensor networks*, IEEE Int. Conference on Communications, 2011.

[147] Zhijun Teng, Baohe Pang, Chunqiu Du, and Zhe Li, *Malicious node identification strategy with environmental parameters*, IEEE Access **8** (2020), 149522–149530.

[148] Martin Thomson and Sean Turner, *Using TLS to Secure QUIC*, Internet-Draft draft-ietf-quic-tls-17, Internet Engineering Task Force, December 2018, Work in Progress.

[149] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, *Probabilistic Robotics*, The MIT Press, 2005.

[150] Ivana Tomic and Julie A. McCann, *A survey of potential security issues in existing wireless sensor network protocols*, IEEE Internet of Things Journal **4** (2017), no. 6, 1910–1923.

[151] A. Torres-González, J.R. Martínez-de Dios, and A. Ollero, *Range-only SLAM for robot-sensor network cooperation*, Autonomous Robots (2017).

[152] Abdelmoughni Toubal, Billel Bengherbia, Mohamed Ould Zmirli, and Abderrezak Guessoum, *FPGA implementation of a wireless sensor node with built-in security coprocessors for secured key exchange and data transfer*, Measurement: Journal of the International Measurement Confederation **153** (2020), 107429.

[153] Michal Trnka, Jan Svacina, Tomas Cerny, Eunjee Song, Jiman Hong, and Miroslav Bures, *Securing internet of things devices using the network context*, IEEE Transactions on Industrial Informatics **16** (2020), 4017–4027.

[154] Andras Varga, *OMNeT++*, Modeling and Tools for Network Simulation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 35–59.

[155] Abhishek Verma and Virender Ranga, *Security of RPL based 6LoWPAN networks in the internet of things: A review*, IEEE Sensors Journal **20** (2020), no. 11, 5666–5690.

[156] Jeffrey S. Vitter, *Random sampling with a reservoir*, ACM Transactions on Mathematical Software **11** (1985), no. 1, 37–57.

[157] G. Wang, B. Lee, J. Ahn, and G. Cho, *A UAV-assisted CH election framework for secure data collection in wireless sensor networks*, Future Generation Computer Systems **102** (2020), 152–162.

[158] Shun Sheng Wang, Kuo Qin Yan, Shu Ching Wang, and Chia Wei Liu, *An integrated intrusion detection system for cluster-based wireless sensor networks*, Expert Systems with Applications **38** (2011), 15234–15243.

[159] Harald Welte and Pablo Neira Ayuso, *The netfilter.org project*, https://www.netfilter.org, [En línea. Consulta: 28-04-2019].

[160] David A. Wheeler, *Flawfinder home page*, https://dwheeler.com/flawfinder, [En línea. Consulta: 28-04-2019].

[161] Andrew Wichmann, Burcu Demirelli Okkalioglu, and Turgay Korkmaz, *The integration of mobile (tele) robotics and wireless sensor networks: A survey*, Computer Communications **51** (2014), 21–35.

[162] wolfSSL Inc., *wolfSSL Embedded SSL/TLS Library | wolfSSL Products*, https://www.wolfssl.com/products/wolfssl/, [En línea. Consulta: 27-04-2019].

[163] A.D. Wood, J.A. Stankovic, and S.H. Son, *JAM: A jammed-area mapping service for sensor networks*, Proc. Intl. Symposium on System-on-Chip (IEEE Cat. No.03EX748), IEEE Comput. Soc, 2003, pp. 286–297.

[164] Hu Yih-Chun and Adrian Perrig, *A survey of secure wireless ad hoc routing*, IEEE Security & Privacy **2** (2004), no. 3, 28–39.

[165] P.A. Yilmaz, A. Belenkiy, N. Uzun, N. Gogate, and M. Toy, *A trie-based algorithm for IP lookup problem*, Globecom '00 - IEEE. Global Telecommunications Conference. Conference Record (Cat. No.00CH37137), vol. 1, IEEE, pp. 593–598.

[166] Yongsheng Liu, Jie Li, and M. Guizani, *PKC based broadcast authentication using signature amortization for WSNs*, IEEE Transactions on Wireless Communications **11** (2012), no. 6, 2106–2115.

[167] Jin Yong Yu, Euijong Lee, Se Ra Oh, Young Duk Seo, and Young Gab Kim, *A survey on security requirements for WSNs: Focusing on the characteristics related to security*, IEEE Access **8** (2020), 45304–45324.

[168] Erdong Yuan, Liejun Wang, Shuli Cheng, Naixiang Ao, and Qingrui Guo, *A key management scheme based on pairing-free identity based digital signature algorithm for heterogeneous wireless sensor networks*, Sensors **20** (2020), no. 6, 1543.

[169] Ender Yüksel, *Analysing ZigBee key establishment protocols*, 2012.

[170] Ender Yuksel, Hanne Riis Nielson, and Flemming Nielson, *ZigBee-2007 security essentials*, NordSec 2008 (2008).

[171] Jun Zheng and Abbas Jamalipour, *Wireless sensor networks: A networking perspective*, John Wiley & Sons, 2009.

[172] ZigBee Alliance, *ZigBee Over-the-Air Upgrading Cluster Version 1.1 Revision 23*, Tech. report, ZigBee Alliance, 2014.