Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería Informática

Redes Inalámbricas de Sensores: Sistema Integral de monitorización de estructuras metálicas

Autor: Jose Vicente Escamilla Lopez
Autor: Albert Esteve Garcia
Director: Dr. Alberto Miguel Bonastre Pina
Codir.: Dr. Jose Vicente Capella Hernández

Valencia, Diciembre de 2010

Agradecimientos

El proyecto que presentamos aquí es tan solo el final de un ciclo que comenzó hace más tiempo del que pudiera parecer. Y durante todo ese tiempo he acumulado tantos conocidos a los que me gustaría y debería agradecer que espero que no se enfaden si olvido a alguien.

Pero creo que el primero de mis agradecimientos debe ir a mi padre, Ramón. Me consiguió inculcar muchas cosas, una de ellas el ansia por aprender y poder superarme cada día. Sin él no estaría seguramente donde estoy y a él le debo gran parte de mis éxitos, ahora y siempre.

También le debo mucho, por supuesto, a mi madre Lola. Ella me ha animado a seguir siempre adelante y me ha ayudado lo indecible para que consiguiera llegar a donde estoy. Le debo más de lo que puedo escribir en un párrafo así que ni siquiera lo voy a intentar.

Por supuesto a mi hermano Victor, que ha sido mi modelo a seguir durante mucho tiempo, y espero que lo siga siendo durante mucho tiempo más.

A mi novia, Carla, que me ha empujado a seguir adelante incontables veces y ha creído en mí siempre. Le quiero con toda mi alma y espero que no deje de creer en mí nunca. Prometo no fallarte.

Por supuesto a mi compañero de proyecto, a Vicente. He pasado más horas con él en estos últimos tiempos que con ningún otro, para lo bueno y para lo malo. Durante la carrera, de principio a fin, ha sido mi compañero de estudios, de sueños y de frustaciones. Sin él seguramente el camino entero hubiera sido mucho más cuesta arriba.

Al resto de mis compañeros durante la carrera: Salva, Nuria, Héctor, Eduardo. A ellos les debo las partidas de cartas, las fiesta, las siestas entre clases y de vez en cuando, las jornadas de estudio en grupo. De ellos me llevo varios de los mejores recuerdos y compañeros que uno se puede encontrar.

Y por último a mis amigos: Manolo, Edu y Samuel. Por muchos motivos ellos han sido mi válvula de escape y me han apoyado en muchas de las etapas de mi vida. Sin ellos seguramente tampoco estaría donde estoy ahora y les debo mucho más de lo que les puedo devolver.

Albert Esteve

En primer lugar quiero darle las gracias a mis padres. Ellos me han educado, me han apoyado siempre en todas las decisiones que he tomado en la vida. Sin ellos obviamente no estaría donde estoy.

A mis compañeros de carrera Héctor, Nuria, Eduardo y Salva. Ellos han sido mis cómplices de estudio, mis compañeros en las horas muertas y mis amigos fuera de las aulas. Sin ellos la carrera sin duda habría sido mucho menos llevadera.

A Raul por echarme una mano con la parte de la electrónica del proyecto.

A mis amigos Samuel, Estibaliz, Natalia, Rafa, David, Héctor, Manuel, Eduardo. Les debo todas las tardes y noches de fiesta y las carcajadas que conseguían hacer que desconectara del estrés y los problemas.

A mis compañeros de club de tiro con arco de la UPV: Vicente (todos los del club), Maite, Maria, Api, Cristobal, Mery, Juan Ramón, Ramón. Junto al deporte en sí han sido mi válvula de escape del día a día.

A mi compañero de proyecto Albert. Gracias a él el desarrollo de éste proyecto ha sido mucho más llevadero.

Por supuesto una mención especial a Ana por su inmenso apoyo. Por estar siempre ahí para ayudarme en lo que hiciera falta y para escucharme cuando necesitaba hablar. Sin duda le debo buena parte de lo que soy.

José Vicente Escamilla

En el apartado técnico ambos tenemos que agradecer a aquellos profesores y personal del GSTF que prestó su ayuda desinteresadamente en mayor o menos medida y sin los que seguramente el proyecto hubiese sido si cabe más costoso.

Entre esas personas se encuentran: Pascual Pérez Blasco, Andres Lapuebla, Juan José Serrano y en general a todos los que se han interesado y ayudado en el transcurso del largo camino que nos ha llevado a la finalización con éxito del presente proyecto.

Albert Esteve y José Vicente Escamilla

Índice general

1. Introducción **1**
- 1.1. Preámbulo . . . 1
- 1.2. Justificación . . . 1
- 1.3. Descripción . . . 2
- 1.4. Distribución de tareas . . . 5

2. Fundamentos Teóricos **7**
- 2.1. Redes Inalámbricas de Sensores . . . 7
  - 2.1.1. Características de una red sensora . . . 8
  - 2.1.2. Estándares y especificaciones . . . 9
  - 2.1.3. Aplicaciones . . . 12
  - 2.1.4. Tendencias de Procesador . . . 14
  - 2.1.5. Sistemas Operativos . . . 14
  - 2.1.6. Implementaciones . . . 15
  - 2.1.7. Algoritmos . . . 15
- 2.2. Galgas extensiométricas . . . 16
  - 2.2.1. Fundamentos físicos de funcionamiento . . . 16
  - 2.2.2. Efectos adversos . . . 17
  - 2.2.3. Factor de galga . . . 18
  - 2.2.4. Elección de la galga extensiométrica . . . 19
- 2.3. Fundamentos del protocolo de comunicación inalámbrica . . . 23
  - 2.3.1. Formación de la red . . . 24
  - 2.3.2. Función estándar . . . 25
- 2.4. Aplicación puente USB-IGU . . . 27
  - 2.4.1. WebSocket . . . 30

3. Protocolo de Comunicación Propuesto **43**
- 3.1. Supuestos Básicos . . . 43
- 3.2. Fundamentos . . . 44
  - 3.2.1. Formato de trama de Nível Físico . . . 44
  - 3.2.2. Formato de paquetes del protocolo . . . 46
  - 3.2.3. Desarrollo por fases . . . 52
  - 3.2.4. Formación del grafo . . . 56
  - 3.2.5. Cálculo de los caminos mínimos . . . 57

3.2.6. Establecimiento de la programación de *macro-frames* . 58
3.2.7. Toma de medidas 62
3.2.8. Solucionando errores en la red 62

4. Entorno de trabajo **65**
4.1. Microcontrolador 65
4.1.1. CPU y Memoria 67
4.1.2. Interfaz de depuración 68
4.1.3. Control de Potencia 68
4.1.4. Temporizadores 70
4.1.5. Radio 71
4.1.6. USART 73
4.2. Tarjeta de adaptación de sensores 73
4.3. Entorno IAR Embedded Workbench 74
4.3.1. Interfaz Gráfica 75
4.3.2. Depuración 76
4.3.3. Perfiles de ejecución 78
4.4. SimpliciTI 81
4.4.1. Componentes modulares 81
4.4.2. Resumen de la arquitectura 83
4.4.3. Estructura del paquete 84
4.5. SmartRF Packet Sniffer 86
4.5.1. Flujo de datos 87
4.5.2. Detalles de la aplicación 87

5. Implementación **95**
5.1. Tarjeta de adaptación de sensores 95
5.1.1. Galgas extensiométricas 95
5.1.2. Puente Wheatstone 95
5.1.3. Cálculo de la variación de resistencia de las galgas extensiométricas 99
5.1.4. Cálculo de la salida del puente 102
5.1.5. Amplificador 103
5.1.6. CAD 109
5.1.7. Tensiones de alimentación 114
5.2. Protocolo de comunicación de la red de sensores 116
5.2.1. Implementación de la comunicación en los nodos 116
5.2.2. Algoritmos representativos 117
5.2.3. Estrategias de programación 119
5.3. Aplicación puente USB-IGU 120
5.3.1. Descripción del funcionamiento 120
5.3.2. Comunicación USB-puente 124
5.3.3. Comunicación puente-IGU 132
5.4. Interfaz Gráfica de Usuario 135

5.4.1. Conexión WebSocket . . . 136
5.4.2. Interacción con el script PHP . . . 138
5.4.3. Órdenes sobre la red de sensores . . . 139
5.4.4. Gestión de los parámetros de configuración . . . 140
5.4.5. Listado de nodos . . . 141
5.4.6. Posicionamiento de los nodos en el mapa de nodos . . . 143
5.4.7. Cambio de la imagen del mapa de nodos . . . 144

6. Resultados **147**
6.1. Red de sensores . . . 147
6.1.1. Captura del funcionamiento de la Red de Sensores . . . 150
6.2. Tarjeta de adaptación . . . 154
6.3. Interfaz gráfica de usuario . . . 156

7. Conclusiones y futuras ampliaciones **165**
7.1. Conclusiones . . . 165
7.1.1. Red de sensores . . . 165
7.1.2. Tarjeta de adaptación de sensores . . . 166
7.1.3. Aplicación puente y la IGU . . . 166
7.2. Futuras ampliaciones . . . 167
7.2.1. Red de sensores . . . 167
7.2.2. Tarjeta de adaptación de sensores . . . 168
7.2.3. Aplicación puente USB-IGU . . . 169
7.2.4. Interfaz gráfica de usuario . . . 169

Apéndices **171**

A. Manual de Usuario **173**
A.1. Instalación de los nodos . . . 173
A.1.1. Adhesión de los sensores a la pieza . . . 173
A.1.2. Preparando la BBDD . . . 180
A.2. Preparación del terminal base . . . 182
A.3. Instalando la interfaz gráfica . . . 182
A.4. Accediendo a la interfaz gráfica . . . 184
A.4.1. Elementos de la IGU . . . 185
A.4.2. Parámetros de la red . . . 185
A.4.3. Iniciar la red . . . 186
A.4.4. Reiniciar la red . . . 187
A.4.5. Forzar medición bajo demanda . . . 187
A.4.6. Mapa de nodos . . . 187
A.4.7. Cambiar la imagen del mapa de nodos . . . 189
A.4.8. Recibiendo medidas de la red . . . 189

B. Manual del programador **193**
B.1. Manual de la implementación del protocolo de red 193
B.1.1. Introducción 193
B.1.2. Estructuras de datos 193
B.1.3. Bibliotecas Software 205
B.2. Manual de la implementación del puente USB-IGU 213
B.2.1. Instruducción 213
B.2.2. Flujo de ejecución 213
B.2.3. Estructuras de datos 213
B.2.4. Referencia de funciones 217
B.2.5. Libreria para el manejo de la UART 225
B.3. Manual de la implementación de la interfaz gráfica / Script PHP 227
B.3.1. Interacción con el script PHP 228
B.3.2. Estructuras de datos 228
B.3.3. Referencia de funciones 230
B.4. Manual de implementación de la libreria para la lectura de los sensores 235
B.4.1. Descripción 235
B.4.2. Referencia de funciones 235

Bibliografía **237**

Lista de Figuras

1.1. Detalle de un envío a través del nodo sumidero 4

2.1. Esquema de componentes de un nodo sensor típico 8
2.2. Comaparación entre la pila de protocolos OSI y ZigBee 11
2.3. Detalle de una galga extensiométrica 16
2.4. Galga sometida a tracción 17
2.5. Galga sometida a compresión 18
2.6. Galga sometida a tracción 18
2.7. Galga sometida a compresión 18
2.8. Galga sometida a tracción 19
2.9. Detalle de las partes de una galga extensiométrica 20
2.10. Gráfica de la deformación de galga a lo largo del material . . . 23
2.11. Diagrama de la arquitectura de comunicaciones de la red con el IGU 31
2.12. Diagrama de una comunicación WebSocket 32

3.1. Formato de la Capa Física del CC1110 44
3.2. Formato de la Capa Física del SimpliciTI 45
3.3. Formato del paquete de descubrimiento 47
3.4. Formato del paquete de Orden de descubrimiento 48
3.5. Formato del paquete de Contestación de Tablas de intensidades 49
3.6. Formato del paquete de Programación del macro-frame 50
3.7. Formato del paquete de Envío de mediciones 51
3.8. Formato del paquete de Actualización de configuración 51
3.9. Esquema de comunicaciones de un nodo sensor durante la formación de la red 54
3.10. Esquema de comunicaciones del sumidero durante la formación de la red 55
3.11. Ejemplo de grafo con caminos mínimos 58
3.12. Ejemplo de recorrido por niveles en árbol 60
3.13. Ejemplo de recorrido por niveles en árbol con identificadores desordenados 61
3.14. Ejemplo de recorrido por ramas en árbol 61

3.15. Ejemplo programación de *macro-frame* de emergencia con recorrido por ramas 63
3.16. Ilustración de error durante la formación de la red 64

4.1. Arquitectura y asignación del patillaje del CC1110 66
4.2. Espacio de memoria XDATA del CC1110 68
4.3. Modo *Free-running* del temporizador 1 70
4.4. Modo *Up/down* del temporizador 1 71
4.5. Módulo de Radio del CC1110 72
4.6. Proceso de adaptación de la señal al microcontrolador 74
4.7. Espacio de trabajo del IEW 75
4.8. IEW cargando en el microcontrolador el código compilado . . 76
4.9. IEW en modo depuración 77
4.10. Barra de controles de depuración 77
4.11. Ventana de registros del microcontrolador 78
4.12. Selección del perfil de ejecución activo 79
4.13. Desactivar la compilación de un archivo 80
4.14. Componentes modulares de SimpliciTI 82
4.15. Arquitectura de SimpliciTI 83
4.16. Detalle de las secciones lógicas de un paquete del SimpliciTI . 85
4.17. Flujo de datos del *Packet Sniffer* 87
4.18. Ventana de lanzamiento del *Packet Sniffer* 88
4.19. Ventana principal del *Packet Sniffer* 89
4.20. Barra de tareas del *Packet Sniffer* 89
4.21. Selección de dispositivos de captura 90
4.22. Parámetros de configuración del dispositivo de captura 91
4.23. Archivo de configuración por defecto para el CC1110 en el *Packet Sniffer* 92
4.24. Pestaña de selección de campos para SimpliciTI 92
4.25. Formato de un paquete de un archivo PSD 93

5.1. Esquema de conexiones de un puente Wheatstone 96
5.2. Esquema de conexiones de un cuarto de puente Wheatstone . 97
5.3. Esquema de conexiones de medio puente Wheatstone 98
5.4. Esquema de conexiones de puente Wheatstone completo . . . 99
5.5. Gráfica tensión/deformación del acero 100
5.6. Amplificador con tensión de entrada positiva 103
5.7. Amplificador con tensión de entrada negativa 104
5.8. Esquema de una fuente simétrica fabricada con dos baterías idénticas 104
5.9. Amplificador con tensión de entrada 0V y V_{ref} 105
5.10. Amplificador con tensión de entrada positiva y V_{ref} 106
5.11. Amplificador con tensión de entrada negativa y V_{ref} 106

5.12. Diagrama de correspondencia entre salida del puente y del amplificador . . . 107
5.13. Diagrama de conexión básico SPI . . . 111
5.14. Cronograma de las señales del CAD . . . 113
5.15. Trama enviada por la UART . . . 128
5.16. Trama enviada por la aplicación puente . . . 128
5.17. Formato de un paquete de listado de nodos . . . 130
5.18. Formato de un paquete de recepción de medidas . . . 131
5.19. Formato de un paquete de envio de parámetros de configuración 131
5.20. Formato de un paquete de envio solicitud de medición bajo demanda . . . 131
5.21. Formato de un paquete de envio solicitud de reinicio de la red 132
5.22. Formato de un paquete WebSocket de inicio de la red . . . 134
5.23. Formato de un paquete WebSocket de envío de parámetros de configuración . . . 134
5.24. Formato de un paquete WebSocket de solicitud de medición bajo demanda . . . 134
5.25. Formato de un paquete WebSocket de reinicio de la red . . . 135
5.26. Formato de un paquete WebSocket de notificación de listado de nodos . . . 135
5.27. Formato de un paquete WebSocket de notificación de actualización de medidas . . . 136

6.1. Etapa de formación de la red . . . 148
6.2. Árbol tras la etapa de formación de la red . . . 148
6.3. Ejemplo de *macro-frame* sin errores . . . 149
6.4. Ejemplo de *macro-frame* con un error . . . 150
6.5. Captura del proceso de formación de la red . . . 151
6.6. Captura del proceso de formación de la red . . . 152
6.7. Captura del proceso de formación de la red . . . 152
6.8. Captura del proceso de formación de la red . . . 153
6.9. Captura de *macro-frame* . . . 153
6.10. Árbol resultante de las capturas mostradas . . . 154
6.11. Salida del CAD con las galgas extensiométricas en reposo . . . 155
6.12. Salida del CAD con las galgas extensiométricas deformadas . . . 155
6.13. Salida del CAD con las galgas extensiométricas deformadas en sentido contrario . . . 156
6.14. IGU contactando con el servidor WebSocket (cabeceras de petición de conexión WebSocket) . . . 156
6.15. La aplicación puente responde con las cabeceras de respuesta WebSocket . . . 157
6.16. La IGU se identifica como una interfaz gráfica válida . . . 158
6.17. La aplicación puente informa del estado de la red . . . 158
6.18. La IGU solicita al script PHP el listado de nodos y sus medidas 159

6.19. La IGU envía la señal para que se inicialice la red 160
6.20. La IGU envía los parámetros de la red 160
6.21. La aplicación puente notifica que ha finalizado la etapa de formación de la red 161
6.22. La aplicación puente notifica que ha recibido medidas nuevas de los nodos 162
6.23. La IGU solicita al script PHP la configuración actual 162
6.24. La IGU solicita al script PHP la configuración actual 163

A.1. Nodo NK01 de WSNVAL 173
A.2. Tarjeta de adaptación de las galgas extensiométricas (cara superior) 174
A.3. Tarjeta de adaptación de las galgas extensiométricas (cara inferior) 174
A.4. Nodo conectado a la tarjeta de adaptación de las galgas extensiométricas 175
A.5. Listón de metal sin preparar 176
A.6. Listón después de lijar con papel de grano grueso 176
A.7. Listón después de lijar con papel de grano fino 177
A.8. Listón limpiado con acetona 177
A.9. Listón limpiado con acetona 177
A.10. Recogiendo la galga extensiométrica con celo 178
A.11. Pegando la galga con celo provisionalmente 179
A.12. Aplicando el pegamento con base de cianoacrilato 179
A.13. Adhiriendo la galga definitivamente 180
A.14. IGU con la red sin inicializar 184
A.15. Elementos del IGU 185
A.16. Botón de posicionado del nodo 187
A.17. Posicionando un nodo en la imagen 188
A.18. Nodo posicionado en la imagen 189
A.19. Popup para cambiar la imagen 190
A.20. IGU recibiendo medidas 191
A.21. IGU recibiendo medidas nuevas 191

B.1. Diagrama de una Lista de Nodos 195
B.2. Diagrama del estructura del grafo 197
B.3. Diagrama de las Tablas de Intensidades 204
B.4. Prototipos de las funciones de *controlList.h* 205
B.5. Prototipos de las funciones de *timerLayer.h* 206
B.6. Prototipos de las funciones de *packetFormat.h* 208
B.7. Diagrama de flujo de datos de la aplicación puente USB-IGU . 214

Lista de Tablas

3.1. Tabla de IDs de paquete 46
3.2. Tabla de Flags de mediciones 51
3.3. Tabla de Flags de umbrales 52

4.1. Sumario de Campos relevantes del paquete del SimpliciTI . . . 85

5.1. Relación de los puertos de E/S usados para el CAD 113
5.2. Tramas recibidas por la IGU 137
5.3. Tipos de peticiones GET al script PHP 138
5.4. Parámetros de las peticiones GET al script PHP 139

A.1. Parámetros del ejecutable de la aplicación puente 183
A.2. Variables correspondientes a los datos de la BBDD en el script PHP 184
A.3. Elementos de la interfaz gráfica 186

B.1. Descripción de las variables de un nodo_t 194
B.2. Multiplicadores disponibles para el periodo del temporizador 2 206
B.3. Modos disponibles para el temporizador 2 207
B.4. Divisores disponibles para los temporizadores 3 y 4 207
B.5. Modos disponibles para los temporizadores 3 y 4 207
B.6. Divisores disponibles para reloj del sistema 208
B.7. Pines del Puerto 0 a los que están conectados los LEDs 209

Capítulo 1

Introducción

1.1. Preámbulo

Este proyecto se engloba dentro de la informática dedicada a las Redes Inalámbricas de Sensores (RIS). Se pretende diseñar un sistema integral de monitorización para estructuras metálicas que supervisa la deformación que sufren las estructuras metálicas de una construcción.

1.2. Justificación

En el sector de la construcción llevan a cabo proyectos cada vez de mas envergadura, más arriesgados. Las necesidades destinadas a cubrir por éstas construcciones, la necesidad de reducir el coste de ésta, o la simple busca de originalidad y/o notoriedad de la construcción empuja a los arquitectos a diseñar estructuras que muchas veces desafían los limites de los materiales empleados.

Estos factores provocan que en ciertas ocasiones (por errores de cálculo, factores ambientales, etcétera) los materiales utilizados en la construcción de la estructura superen el límite para el que fueron concebidos y éstos reaccionen de forma que comprometan la integridad de la construcción.

Igualmente, en construcciones considerablemente antiguas sucede algo similar. Sin embargo, en éste caso el factor que pone en peligro la estructura es la propia antigüedad de ésta. Los materiales con el tiempo ven mermadas sus capacidades debido al desgaste, el óxido, la fatiga, etc. Debido a ésto las construcciones de cierta edad también son un conjunto de riesgo a tener en cuenta.

Dada la potencialmente peligrosa situación de éstas estructuras, una de las soluciones que suelen tomarse es monitorizar ciertos parámetros que de-

finen la consistencia de la estructura. De ésta forma, en el momento en el que se registren valores anormales, se pueden tomar medidas que eviten daños mayores o incluso pérdidas personales. Uno de los parámetros más interesantes que reflejan la estabilidad de una estructura es la deformación que ésta sufre. Una estructura inestable o débil tenderá a deformar más de lo debido los materiales con los que está fabricada.

Por ejemplo, en el caso de un rascacielos, los pilares sobre los que está asentado tienden a torcerse por el viento que impacta sobre él, inclinando así la totalidad de la estructura, ésto fenómeno pone en peligro la integridad estructural del rascacielos cuando éste llega a un punto de inclinación crítico que según la climatología de la zona donde resida y la altura del mismo (entre otros factores), puede llegar a ocurrir con relativa facilidad. Tal es la problemática en éste tipo de edificios que algunos de los más importantes rascacielos (como el Taipei 101, Taiwán) del mundo incorporan un contrapeso de varios cientos de toneladas en la parte más alta de éstos. Éste contrapeso se mueve accionado por cilindros hidráulicos en la dirección opuesta a la que es empujado el rascacielos. De ésta forma se evita que el edificio se incline en exceso y los materiales cedan ante la excesiva deformación a la que están sometidos.

1.3. Descripción

El presente proyecto trata sobre la creación de un sistema integral de monitorización de estructuras metálicas de construcciones. A pesar de que el sistema está orientado a la monitorización de dichas estructuras, puede ser usado para monitorizar otro tipo de metales sin modificaciones considerables dependiendo de las necesidades de dicha aplicación.

Dicho sistema de monitorización medirá las deformaciones en distintos puntos de la estructura y mostrará los resultados a través de un terminal con el que el usuario podrá interactuar.

El sistema de monitorización constará de una red inalámbrica de sensores instalados en los puntos en donde sea necesario medir. Dicha red inalámbrica transmitirá los datos un ordenador en donde el usuario podrá visualizarlos e interactuar con la red de sensores.

La red de sensores estará formada por nodos inalámbricos modelos NK01 y SUM-USB fabricados por **WSNVAL**. Los nodos NK01 (en adelante nodo convencional o sensor) son los dispositivos que serán utilizados para realizar las mediciones. Éstos constan de un microcontrolador compatible con la serie 8051[1] y una etapa de radio que le permite enviar datos por radiofrecuencia.

[1]Microcontrolador desarrollado por Intel ampliamente extendido en sistemas embebidos

Dichos nodos no están provistos de sensores para medir deformaciones en metales, por lo que se emplearán sensores externos al nodo con éste fin. Dichos sensores constarán de cuatro galgas extensiométricas[2] que serán adheridas al metal del cual se quiere obtener la lectura de su deformación. Éstas cuatro galgas estarán conectadas a su vez a un circuito que amplificará y adaptará la señal para adecuarla a la entrada del microcontrolador.

El diseño del protocolo de comunicación esta específicamente desarrollado para la aplicación. El objetivo es que el protocolo permita la comunicación de un número a priori ilimitado de sensores y que se mantenga una base de datos histórica de las medidas, los sensores que conforman la red y otra información relevante y además que todo el proceso se complete de la forma más eficiente posible teniendo en cuenta las necesidades propias del supuesto para el que se diseña dicho protocolo.

El diseño se ha realizado teniendo en mente un entorno en el que el tiempo no es una necesidad crítica y sí lo es, sin embargo, el ahorro de energía y el tiempo de vida de los sensores. Y bajo estos dos objetivos se propuso el protocolo que se describe en este proyecto.

Para conseguirlo el protocolo se basa en divisiones de tiempo que se agrupan en *macro-frames*[3]. Cada división o marco supone una comunicación punto a punto entre dos sensores adyacentes. Así se descartan las colisiones y se aumenta la eficiencia de las emisiones. Un *macro-frame* completo implica la recepción total o parcial de datos desde los sensores al nodo sumidero.

La red inalámbrica de nodos se conformará como una red multisalto con topología en árbol con enlaces ponderados. Los enlaces se ponderan según el RSSI[4] para conseguir que las emisiones se produzcan encaminando los paquetes punto a punto y permitiendo emisiones de menor potencia.

Se conoce por nodo recolector o sumidero (*sink node* en inglés) al nodo no sensor que actúa como puerta de enlace entre la red y la estación de monitorización. El sumidero arbitra la comunicación del resto de nodos, forma los enlaces entre nodos y debe conocer a todos los componentes de la red. Este nodo esta conectado a la red eléctrica y envía los datos por puerto USB a un ordenador desde el que se monitoriza toda la información. En el terminal éstos datos serán recogidos por una aplicación diseñada ex profeso para éste fin. Dicha aplicación (en adelante «puente USB-IGU») volcará los datos en una base de datos para que un interfaz gráfico haga uso de los mismos.

El puente USB-IGU además se encargará de transmitir a la red inalámbrica las ordenes provenientes del IGU[5] amén de implementar algunos mecan-

[2] Sensor basado en el efecto piezo-resistivo que reacciona ante una deformación

[3] Agrupaciones de divisiones o marcos temporales

[4] *Received Signal Strength Indicator*. Es un parámetro que caracteriza el nivel de potencia de una señal recibida en redes inalámbricas.

[5] Interfaz Gráfico de Usuario

ismos de sincronización con el mismo que se explicaran con mas detalle más adelante. El sumidero además carece de las restricciones energéticas a las que están sujetos el resto de nodos de la red, con lo que podemos aumentar su carga computacional y comunicacional para permitir al resto de nodos ahorrar tiempo de cómputo.

Tenemos pues un protocolo centralizado y basado en divisiones de tiempo en el que los sensores pasarán la mayor parte del tiempo ociosos y solo utilizarán unos pocos *frames* (marcos temporales) para comunicarse. El tiempo que pasan ociosos lo harán en un estado de ahorro de energía o *sleep-time* del que se pueden despertar ellos mismo mediante un temporizador (*sleep-timer*) o mediante una señal de radio emitida desde el sumidero mediante un circuito de RF externo al microcontrolador que provoca que los nodos se despierten[6] (señal conocida como «bocinazo»). Esto permite aumentar significativamente el tiempo de vida de los sensores y mantener un control de la red a través del nodo sumidero.

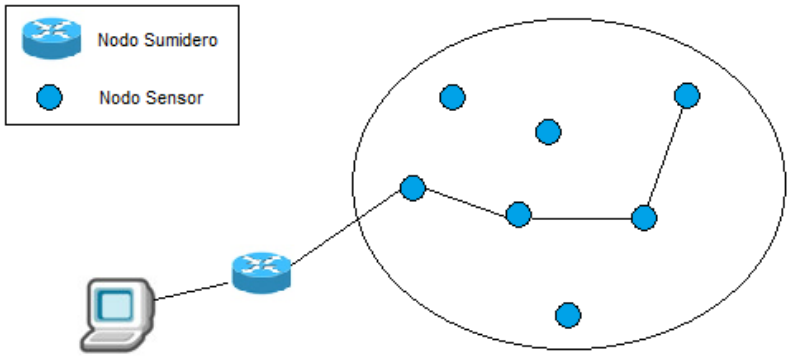


Figura 1.1: Detalle de un envío a través del nodo sumidero

Valga como introducción, veamos la descripción de los capítulos del proyecto: en el capítulo 2 se describirán los fundamentos teóricos de las RIS, las galgas y del protocolo, para pasar, en el capítulo 3 a describir los fundamentos del protocolo con más detalle y detenimiento. El capítulo 4 estará dedicado al entorno de trabajo utilizado para el desarrollo del proyecto y las características que motivaron su utilización. Los capítulos 5 y 6 estarán dedicados a la descripción de distintos detalles de la implementación del protocolo y los resultados obtenidos del mismo respectivamente. Por último en el capítulo 7 se indicarán las conclusiones extraídas y se propondrán posibles futuras ampliaciones que se pudieran implementar al protocolo.

[6]El circuito no se ha diseñado en el proyecto, pero se propone como ampliación

1.4. Distribución de tareas

El proyecto engloba el proceso entero de diseño e implementación de la aplicación de redes de sensores presentada. El proceso completo se puede dividir a su vez en varias partes diferenciadas y dependientes entre sí que son: adaptación de las galgas extensiométricas a la electrónica de los nodos de WSNVAL, diseño e implementación del protocolo de comunicación de la red de sensores, implementación del puente de comunicación del nodo sumidero con la estación base, diseño e implementación de la IGU.

En lo que a distribución de tareas la de diseño e implementación del protocolo de comunicación de la red de sensores recayó en Albert Esteve, mientras que el resto de partes del proyecto fueron desarrolladas por José Vicente Escamilla.

Capítulo 2

Fundamentos Teóricos

2.1. Redes Inalámbricas de Sensores

Una Red Inalámbrica de Sensores (RIS o *Wireless Sensor Networks WSN* en inglés) consiste en sensores autónomos espacialmente distribuidos para monitorizar de forma cooperativa condiciones físicas o medioambientales, tales como la temperatura, el sonido, la vibración, presión, movimiento, voltaje e incluso oxígeno disuelto. El desarrollo de las RIS fue motivado, como en muchos otros caso, por el interés en aplicaciones militares. Hoy en día, sin embargo, tiene numerosas aplicaciones en áreas industriales y civiles, como la monitorización de procesos industriales, monitorización en áreas de salud o en entornos domésticos (domótica), control de tráfico, etc.

Como predecesor de las RIS se tiene la *Sound Surveillance System* (SOSUS), una red de boyas sumergidas instaladas en los Estados Unidos durante la Guerra Fría para detectar submarinos usando sensores de sonido.

Además de uno o dos sensores, cada nodo en una RIS está típicamente equipado con un trasceptor de radio u otro sistema de transmisión inalámbrico, un pequeño microcontrolador (en el caso que nos ocupa los nodos funcionan con el 8051), un circuito analógico, y una fuente de energía, generalmente una batería. Un nodo sensor puede variar en tamaño desde el equivalente a una caja de zapatos hasta un grano de maíz. El coste es igualmente variable, dependiendo de las capacidades sensoras de los nodos además de su tamaño. Las limitaciones en tamaño y precio en nodos sensores corresponden con limitaciones en otro recursos, tales como la energía, la memoria, la velocidad de computación o el ancho de banda.

Las redes de sensores forman típicamente redes *ad-hoc* sin infraestructura física preestablecida ni administración central. Así cada nodo soporta algoritmos en los que la información se encamina mediante saltos punto a punto entre distintos nodos cercanos físicamente hasta la estación base.

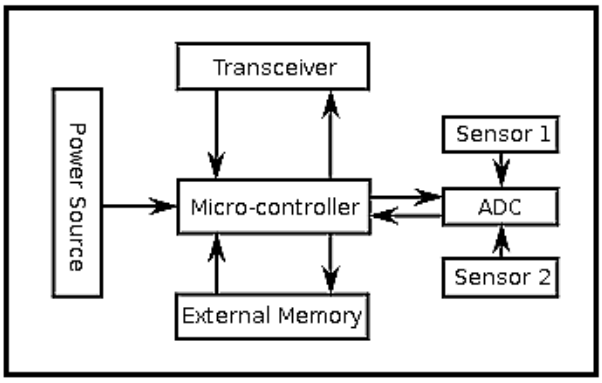


Figura 2.1: Esquema de componentes de un nodo sensor típico

Este tipo de redes se caracterizan por su facilidad de despliegue y por ser autoconfigurables, pudiendo actuar indistintamente como emisor o receptor y ofrecer servicios de encaminamiento entre nodos sin visión directa, registrando datos de cada uno de los sensores locales. Otra de sus características es su gestión eficiente de la energía lo que les permite ser altamente autónomos y plenamente operativos. La idea es repartir aleatoriamente estos nodos en un territorio grande, en el cual los nodos «observan» hasta que sus recursos energéticos se agoten.

Actualmente, las redes de sensores es un tema muy activo de investigación en varias universidades, aunque ya empiezan a existir aplicaciones comerciales basadas en este tipo de redes. La red de sensores hasta la fecha más grande consistió de 800 nodos y fue puesta en servicio el 27 de agosto de 2001 para una duración breve en la universidad de Berkeley con el fin de demostrar la potencia de esa técnica en una presentación.

2.1.1. Características de una red sensora

Las RIS tienen una serie de características propias, mientras que otras vienen heredadas como características de las redes ad-hoc:

- Topología Dinámica: en una red de sensores la topología es altamente variable y los nodos sensores deben ser capaces de adaptarse a los cambios para seguir comunicándose.
- Variabilidad del canal: el canal de radio en el que transmiten los nodos es variable y puede ser objeto de atenuaciones, de svanecimientos o interferencias que resulten en errores en la recepción de los datos.
- No se utiliza infraestructura de red: no es necesaria infraestructura en una RIS ya que los nodos son capaces de actuar tanto como emisores,

como receptores o como enrutadores indistintamente. Sin embargo cabe destacar la figura del nodo recolector o sumidero que es el encargado de recoger información y medidas del resto de nodos en tiempo discreto generalmente. Esta información es transmitida a un ordenador que será el encargado de transmitir esa información por redes cableadas o inalámbricas según el caso.

- Tolerancia a errores: un nodo sensor en una RIS debe ser capaz de seguir funcionando aún pese a la existencia de errores en el sistema propio.

- Comunicaciones multisalto o bradcast: en aplicaciones sensores es común la utilización de protocolos que permitan el multi-hop (véase AODV, DSDV, EWMA u otras), aunque también es común encontrarse con protocolos basados en emisiones broadcast.

- Consumo energético: es un factor altamente sensible en general, como lo es para el caso de estudio en concreto. En general los sensores vienen equipados con un microcontrolador de consumo ultra bajo así como un trasceptor de radio con la misma característica. El consumo es restrictivo incluso con el software, que se debe crear conjugando la misma característica.

- Limitaciones hardware: debido a la limitación del consumo energético, tenemos como resultado un hardware sencillo que limita la capacidad de proceso del nodo.

- Costes de producción: puesto que las RIS suelen estar pensadas para ser compuestas por gran número de nodos, éstos, una vez definida su aplicación, pueden ser baratos de producir si se hacen en grandes cantidades.

2.1.2. Estándares y especificaciones

En las aplicaciones RIS con frecuencia tres años de duración de las baterías es un requisito de las mismas. Por este motivo hoy en día varios de los sistemas RIS están basados en protocolos ZigBee o IEEE 802.15.4. Pero hay muchas estandarizaciones que siguen en desarrollo para RIS. Mientras IEEE se centra en las capas física y de acceso al medio, la IETF (*Internet Engineering Task Force*) trabaja a partir de la capa 3 y superiores. Además de las mentadas, otras iniciativas como ISA (*International Society for Automation*) y la fundación HART proveen soluciones verticales que incluyen todas las capas de protocolos. Por último encontramos también cierto número de especificaciones o mecanismos no estándares y propietarios.

Muchos de estos estándares sin embargo se basan en el estándar de radio IEEE 802.15.4-2006.

IEEE 802.15.4

Como ya hemos comentado, este estándar define las capas de Control de Acceso al Medio y Físico del modelo de red (las capas superiores están definidas por ZigBee), proporcionando comunicación en las bandas ISM de 868 a 915 Mhz y 2,4 Ghz y razones de datos de hasta 250 kbps. Algunas de sus características de mercado incluyen: poco consumo de potencia, bajo coste, soporta mayores ordenes de red (<= 65k nodos), distintos niveles de seguridad seleccionables (usando AES-128, incluyendo privacidad, autenticación de emisor e integridad de mensaje) y flexibilidad en el protocolo para que se adapte a distintas aplicaciones. Además soporta topologías en estrella o punto a punto.

Características de la reducción de potencia

La reducción de potencia se consigue mediante la reducción del sistema de ciclo de trabajo consistentes en picos de corriente bajos. En la capa física esto recomienda la utilización de una tasa de datos elevada, pero una baja tasa de símbolos, lo que a su vez implica el uso de señalización multinivel.

En el nivel de acceso al medio se emplea el protocolo CSMA-CA. Con un CSMA-CA convencional, la mayor parte del consumo de potencia se debe a la recepción, debido a los largos periodos de monitorización requeridos para soportar la operación durante largos períodos de tráfico a tasas altas.

El estándar IEEE 802.15.4 suporta un modo de extensión del tiempo de vida de la batería (*Battery Life Extension BLE*), en el cual el retroceso exponencial del CSMA-CA esta limitado a un rango entre 0 y 2. Esto reduce radicalmente el ciclo de trabajo durante la recepción.

Tanto el protocolo de comunicación IEEE 802.15.4 como ZigBee ofrecen una basta variedad de características para el ahorro de energía en todas las capas de la pila de protocolos.

ISA100

Este es un estándar novedoso que salió de fase de desarrollo en 2009 (en septiembre de 2009 ISA100.11a vió la luz). Hace uso de 6LoWPAN (*IPv6 over Low power Personal Area Networks*) y provee acuerdos adicionales para aplicaciones de control industrial.

WirelessHART

Este estándar es una extensión del protocolo HART y se diseñó específicamente para aplicaciones industriales como el control y la monitorización de procesos. WirelessHART fue incluido en la cima de la suite del protocolo HART como parte de la *HART 7 Specification* que fue aprobada en junio de 2007. En 2010 WirelessHART fue aprobado por ANSI y la IEC (*International Electrotechnical Comission*) como un estándar internacional (IEC 62591).

IETF RPL y 6LoWPAN

ROLL es el grupo que, junto con IETF define RPL (*IPv6 Routing Protocol for Low Power and Lossy Networks*), mientras que 6LoWPAN junto con IETF produjo una serie de estándares para la transmisión de paquetes IPv6 sobre IEEE 802.15.4.

IEEE 1451

También importante para las RIS es el IEEE 1451 que pretende crear estándares para el mercado de sensores inteligentes (*smart sensors*). La idea principal de estos sensores es integrar el procesamiento inteligente al dispositivo sensor, o lo que es lo mismo, desarrollar toda una familia de estándares para interfaces para transductores inteligentes.

ZigBee

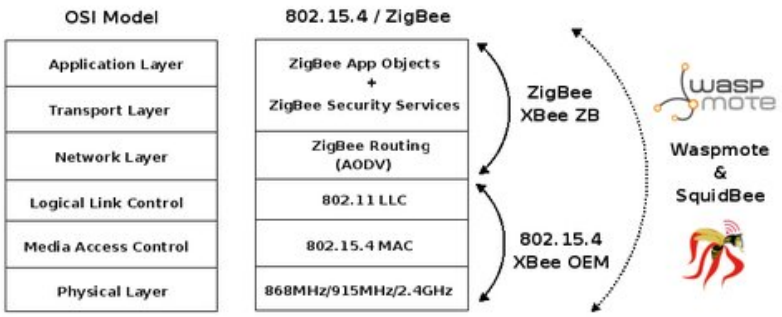


Figura 2.2: Comaparación entre la pila de protocolos OSI y ZigBee

ZigBee se construye en las capas 802.15.4 para brindar seguridad, fiabilidad a través de las topologías de red en malla e inter operatividad con otros dispositivos y estándares. ZigBee también permite objetos de aplicación o

perfiles definidos por el usuario, lo cual ofrece personalización y flexibilidad en el protocolo. Usado en un gran abanico de aplicaciones domóticas, médicas e industriales.

Algoritmo de encaminamiento

El algoritmo de encaminamiento basado en el coste del ZigBee 1.0 emplea calidad de enlace y cuenta de saltos. Esto es un intento para maximizar la calidad de las comunicación entre dispositivos finales, minimizando el número de salto y evitando el uso de enlaces de baja calidad. Además reduce el consumo de energía minimizando las retransmisiones de mensajes y las repeticiones de las rutinas de descubrimiento de ruta.

EnOcean

EnOcean es un sistema para la comunicación inalámbrica orientada al mundo de la automatización de edificios. No se ajusta a ninguno de los estándares habituales.

EnviroNet

EnviroNet es una RIS asociada generalmente a control industrial y aplicaciones de automatización ambiental. Posee algunas versiones sin estandarizar, y otras versiones que se ajustan al estándar ISA100.

2.1.3. Aplicaciones

Las aplicaciones que se le puede dar a una RIS como hemos visto son altamente variadas, aunque generalmente implican conceptos de monitorización, seguimiento o control remoto. En una aplicación típica los nodos sensores se esparcen en una zona en la que recogerán datos de sus sensores hasta que se les agote la fuente de energía.

Monitorización de un área

La monitorización de un área podría considerarse como una aplicación común dentro de las RIS, en el que los sensores se sitúan en un área en la que algún tipo de fenómeno necesita ser monitorizado. Por ejemplo, en una aplicación militar podrían situarse sensores a lo largo de una zona para detectar el acercamiento de un posible enemigo. Si se disparase algún sensor de algún nodo de la red (presión, temperatura, etc.) éste enviaría una señal

a una estación de monitorización encaminándose a través de la red, desde donde se podrá dar una respuesta adecuada al evento. Dependiendo de qué aplicación estemos usando se requerirán distintas estrategias para la propagación de datos, incluyendo entre otros respuesta en tiempo real, redundacia de los datos o la necesidad de cierto grado de seguridad.

Monitorización ambiental

Algunas RIS han sido pensadas para la monitorización ambiental. Muchas de ellas no han pasado de ser meros prototipos debido a la dificultad del tipo de aplicación. Un ejemplo de una aplicación de monitorización de este tipo con un largo periodo de vida útil es el de la monitorización del estado de la nieve en los Alpes Suizos (*The PermaSense Project o Glacier Monitoring*).

Monitorización de maquinaria médica o Mantenimiento basado en las condiciones

Las RIS han sido ideadas en muchos casos para el mantenimiento de la maquinaria basada en sus condiciones (*Condition-Based Maintenance CBM*) como un sistema eficaz para ahorrar dinero y evitar fallos humanos. Para maquinaria situada en sitios inaccesibles, en zonas de riesgo biológico o áreas restringidas entre otras pueden ser ahora monitorizadas eficazmente mediante una RIS de este tipo. Lo cual además de ser más barato también salva de errores humanos en el mantenimiento y observación de dicha maquinaria.

Monitorización Industrial - Monitorización del agua

Existen distintas situaciones en las una RIS puede ser una buena solución para la Industria del Agua. En instalaciones que no estén cableadas para la transmisión de energía o datos una RIS puede ser la solución ideal para su monitorización, incluso pudiendo instalar paneles solares para la alimentación de los nodos.

Además puede ser una solución eficaz para la comprobación y medición del nivel del agua en pozos, reservas o tanques de agua, mediante el uso de sensores de presión sumergibles. Soluciones que pueden ser aplicables también a la agricultura.

Detección de vehículos

Adosados y rodeando a los vehículos, se pueden usar RIS para sentir la presencia de otros vehículos u obstáculos acercándose al vehículo desde cualquier ángulo. Un paso más hacia la conducción automática.

2.1.4. Tendencias de Procesador

Para extender la duración de la fuente de energía (usualmente una batería), un nodo RIS se enciende periódicamente para adquirir y transmitir datos encendiendo la radio, para posteriormente apagarse a fin de ahorrar energía. La radio RIS debe transmitir los datos eficientemente y permitir el apagado con un mínimo uso de energía. Así como la radio, el procesador también debe ser capaz de apagarse y encenderse eficientemente. Las tecnologías para microprocesadores RIS incluyen la reducción del consumo de potencia y la posibilidad de mantener o aumentar la velocidad del procesador. Esto hace a las arquitecturas PowerPC y las basadas en ARM una opción poco aconsejable para dispositivos alimentados con baterías. Por contra, una de las opciones más comunes como arquitectura RIS son las que incluyen el TI MSP430 MCU, diseñado específicamente para operar con baja potencia. Dependiendo el procesador específico, el consumo de potencia en estado de reposo puede variar desde 1 a 50 μW, mientras que en modo de encendido el consumo puede variar de 8 a 500 mW.

2.1.5. Sistemas Operativos

Los sistemas operativos para RIS son en general menos complejos que los sistemas operativos de propósito general, ya sea por las necesidades específicas de las aplicaciones para redes de sensores, ya sea por las limitaciones en recursos en las plataformas hardware de las redes de sensores. Por ejemplo, los sistemas operativos para redes de sensores no son interactivos de la misma forma que los son los sistemas operativos para PCs. Es por esto que los sistemas operativos para redes de sensores no incluyen soporte para interfaces de usuario. Además, las limitaciones de recursos en términos de memoria y soporte hardware de mapeo de memoria hace de algunos mecanismos como la memoria virtual bien innecesarios o bien imposibles de implementar.

El hardware para RIS no es muy distinto de los sistemas embebidos tradicionales, siendo por tanto posible la utilización sistemas operativos para sistemas embebidos tales como eCos o uC/OS. Sin embargo este tipo de sistemas operativos son diseñados habitualmente con propiedades de tiempo real, que los sistemas operativos diseñados específicamente para redes de sensores no ofrecen.

TinyOS es, quizá, el primer sistema operativo específicamente diseñado para las RIS. A diferencia de la mayoría del resto de sistemas operativos, TinyOS se basa en un modelo de programación por eventos en lugar de uno basado en hilos (*multithreading*). Cuando un evento externo ocurre, como un paquete de datos entrantes o una lectura del sensor, TinyOS llama al manjedor de eventos adecuado para manejarlo. Los manejadores pueden enviar

tareas programadas por el kernel del sistema para ejecutarse posteriormente. Tanto los programas para TinyOS como el sistema operativo en sí mismo están programados en nesC, una extensión del lenguaje de programación C. NesC está diseñado para detectar condiciones de carrera entre las tareas y los manejadores de eventos.

Aunque también hay algunos sistemas operativos que permiten la programación en C. Algunos ejemplos de dichos sistemas operativos incluyen Contiki, MANTIS, Btnut y NanoRK, que ofrecen algunas diferencias de diseño especificas según las necesidades de uso (como la carga de módulos en el Centiki por ejemplo).

Tal como TinyOs o Contiki, SOS también es un sistema operativo basado en eventos. La característica principal de SOS es su soporte para módulos cargables. Un sistema completo se construye a partir de módulos más pequeños, generalmente en tiempo de ejecución. Para dar el soporte adecuado al dinamismo inherente de su interfaz de módulos, SOS también se centra en el soporte para memoria dinámica.

LiteOS es un sistema operativo recientemente desarrollado para RIS, que ofrece una abstracción similar a la que ofrece UNIX y soporte para C.

ERIKA Enterprise es uno de los más nuevos en el mercado de sistemas para redes de sensores. Posee un kernel de tiempo real en código abierto y un API compatible con el estándar 802.15.4.

2.1.6. Implementaciones

En ocasiones junto con un sistema operativo especifico existen implementaciones de ciertos protocolos RIS.

- OpenWSN es una implementación de código abierto de una pila de protocolos completa basada en estándares (IEEE 802.15.4E, 6LoWPAN, RPL, TCP/UDP, OpenADR) en TinyOS. Se centra en la capa MAC.
- La pila IP de bajo consumo de Berkeley (*Berkeley Low-Power IP stack o BLIP*) viene junto con la distribución de TinyOS y se centra en la integración de Internet en una RIS mediante el uso de 6LoWPAN.
- uIPv6 es una pequeña pila IPv6 para RIS. Maneja el sistema operativo Contiki.

2.1.7. Algoritmos

Las RIS se componen de un elevado número de nodos sensores, además, un algoritmo para RIS es implícitamente un algoritmo distribuido. En una

RIS el recurso más escaso es la energía, siendo una de las operaciones que consumen más energía las de transmisión de datos y la escucha ociosa. Por esto, la investigación de algoritmos en las RIS se centran en su mayoría en el estudio y diseño de algoritmos dedicados al ahorro de energía, disminuyendo la cantidad de datos siendo transmitidos (usando técnicas como la agregación de datos), cambiando la potencia de transmisión de los nodos sensores o apagándolos preservando conectividad y cobertura.

Otra característica a tener en cuenta es que debido al rango de transmisión de radio limitado y el crecimiento polinomial en el coste energético de la transmisión de radio respecto a la distancia de transmisión, es improbable que todos los nodos alcancen la estación base, así que habitualmente la transmisión de datos salta de nodo a nodo hasta el sumidero.

2.2. Galgas extensiométricas

Para medir las deformaciones que sufre el metal se van a usar unos sensores llamados galgas extensiométricas. Éstos sensores están formados por una lamina de un material no conductor con ciertas propiedades elásticas sobre el que se adhiere una pista de una aleación conductora con un patrón determinado en función del tipo de tensiones a los que va a estar sometido el material. Ésta pista conductora suele dibujarse siguiendo un patrón zigzag formando un conjunto de lineas largas y paralelas de forma que la mayor parte del conductor está distribuido paralelamente en una única dirección tal y como se muestra en la figura 2.3.

Figura 2.3: Detalle de una galga extensiométrica

2.2.1. Fundamentos físicos de funcionamiento

La galga extensiométrica basa su funcionamiento en el efecto piezo-resistivo[1] sobre el material conductor del que está formado la misma. Al aplicar una fuerza de tracción de forma longitudinal sobre la galga (de forma paralela a

[1]Cambio en la resistencia de un material al aplicarle un estrés mecánico.

la las pistas de mayor longitud), el material conductor se deformará aumentando significativamente la longitud del mismo. Éste aumento de longitud debido a la tracción se traduce además en una disminución de sección en el mismo debido al efecto de Poisson. Tanto el factor de longitud como el de la sección del conductor determinan la resistencia de un material conductor como se deduce de la ecuación:

$$R = P * \frac{L}{S}$$

Dado que la resistencia es directamente proporcional a la longitud L del conductor e inversamente proporcional a la sección S del mismo y siendo P la constante de resistividad del material, resulta ahora evidente que al someter un conductor a tracción (aumenta su longitud y disminuye su sección) la resistencia eléctrica del mismo aumentará proporcionalmente.

Figura 2.4: Galga sometida a tracción

El efecto contrario sucede cuando se somete el conductor a compresión. Un conductor sometido a compresión disminuye su longitud L y aumenta su sección S, por lo que, como se deduce de la ecuación anterior, su resistencia disminuirá de forma proporcional.

Gracias a éste efecto piezo-resistivo es posible medir la deformación que sufre un material adhiriendo cuidadosamente y de forma precisa la galga extensiométrica al material del cual se desea medir su deformación.

2.2.2. Efectos adversos

Dado que la mayor parte del conductor se distribuye en lineas paralelas la mayor variación de resistencia se registra cuando la deformación (ya sea tracción o compresión) se produce en dirección paralela a las mismas. Sin embargo, al aplicar una fuerza a éstas, los tramos de conductor que conforman

Figura 2.5: Galga sometida a compresión

los cambios de sentido de las pistas están dispuestos de forma perpendicular a la fuerza de deformación, por lo que físicamente experimentan el efecto contrario a las lineas paralelas (Figuras 2.6 y 2.7). Ésto, a su vez, produce el efecto contrario en la resistencia al producido en las lineas paralelas. No obstante, dada la gran anchura y la poca longitud de éstos tramos en comparación con los tramos paralelos, la variación de resistencia en éstos tramos minoritarios de conductor se considera despreciable.

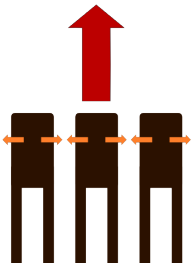


Figura 2.6: Galga sometida a tracción

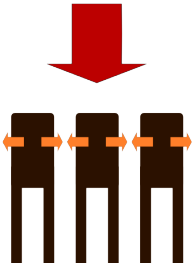


Figura 2.7: Galga sometida a compresión

Éste tipo de galgas extensiométricas están diseñadas para cuantificar el elongamiento producido en sentido paralelo a las pistas de conductor. En el caso de que el elongamiento se produzca de forma perpendicular a éstas (Figura 6), las galgas experimentan una variación de resistencia. Sin embargo, éste caso no será contemplado en nuestro sistema ya que éste tipo de sensores no están diseñados para medir deformaciones en dirección perpendicular a éstos y estas fuerzas de deformación no deberían registrarse si se ha llevado a cabo una correcta instalación de los mismos.

2.2.3. Factor de galga

Como se ha explicado anteriormente, las galgas extensiométrica varían su resistencia eléctrica en función de la deformación que sufren. Ésta variación

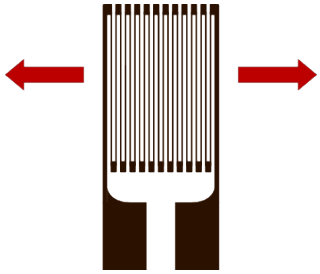


Figura 2.8: Galga sometida a tracción

de resistencia proporcional a la deformación se cuantifica mediante el factor de galga y se define con la siguiente fórmula:

$$K = \frac{dR/R}{dL/L}$$

El factor de galga (K) es una constante definida por diversos parámetros referentes a la fabricación del conductor. Es por ello que ésta constante viene dada por el fabricante para cada tipo de galga, e incluso ésta suele venir especificada en el embalaje al adquirirlas ya que suele ser calculada por el fabricante en cada lote que éste pone a la venta con el fin de proporcionar al usuario la constante K exacta del lote que ha adquirido.

2.2.4. Elección de la galga extensiométrica

A la hora de elegir una galga extensiométrica que se ajuste a nuestras necesidades tenemos que tener en cuenta diversos factores que afectarán a la precisión y vida útil de la misma. Diferentes factores concernientes al entorno en el que se encontrará la galga y al tipo de estrés al que estará sometida la misma condicionará el tipo de conductor a usar así como la base sobre la que deberá estar impreso. Con éste propósito los fabricantes ofrecen una gran variedad de tipos de galgas, cada una orientada a unas necesidades concretas de temperaturas, precisión y tipo de estrés.

Básicamente existen dos grandes conjuntos de galgas extensiométricas: las metálicas y las semiconductoras. Fundamentalmente la gran diferencia entre las galgas metálicas y las semiconductoras es el factor de galga. Mientras las galgas metálicas ofrecen factores de galga que oscilan entre 1,5 y 3.2, las galgas semiconductoras presentan factores de galga entre 50 y 200. Las galgas extensiométricas semiconductoras tienen pues una sensibilidad notablemente mayor, lo cual las hace atractivas ya que simplifican la electrónica de amplificación. Sin embargo éstas galgas sufren ciertos problemas que las hacen inadecuadas para algunas aplicaciones: alta sensibilidad a la temperatura, muy frágiles y difícil instalación. Es por ello que para nuestra aplicación

hemos elegido galgas de tipo metálicas ya que son mucho mas inmunes a la temperatura y su instalación no es tan crítica.

Antes de valorar distintas características de las galgas extensiométrica con el fin de acotarlas a nuestras necesidades, es conveniente repasar algunos de los parámetros físicos de éstas con la ayuda de la siguiente figura:

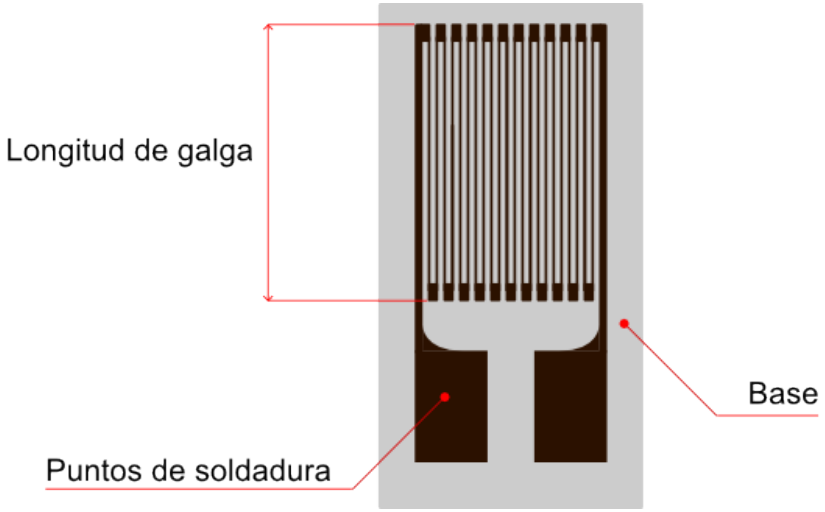


Figura 2.9: Detalle de las partes de una galga extensiométrica

Tipo de conductor

El tipo de conductor de la galga extensiométrica va a determinar el factor de galga, la precisión, la resistencia a la fatiga y la temperatura de trabajo entre otros parámetros. A continuación se muestra un listado de las características más importantes de cada material disponible.

- Constantán[2]

  Buen factor de galga

  Relativamente insensible a las variaciones de temperatura

  Buena resistencia a la fatiga

  Alta capacidad de deformación (5 %)

  Auto-compensado de temperatura

- Constantán templado

  Mismas características que el Constantán a excepción de:

  Muy alta capacidad de deformación (>20 %)

  Baja resistencia a la fatiga

[2]Aleación formada por un 55 % de cobre y un 45 % de níquel

- Aleación Isoelástica

  Muy buena resistencia a la fatiga

  Alto factor de galga (3.2)

  No adecuado para deformaciones estáticas

  No lineal

- Aleación Karma

  Buena resistencia a la fatiga

  Excelente estabilidad

  Idóneo para deformaciones estáticas

  Rangos de temperatura:

    Régimen normal: -269ºC a 260ºC

    Régimen puntual: hasta 400ºC

  Auto-compensado de temperatura (más preciso que el Constantán)

Nuestra aplicación no requiere grandes exigencias de temperatura, y las deformaciones van a ser estáticas, no muy pronunciadas y durante largos periodos de tiempo. Además, se requiere que la galga sea lo más lineal posible y tenga cierta compensación de temperatura. Por ésta razón optaremos por el Constantán como conductor ya que ofrece una combinación de características óptimas para nuestra aplicación.

Tipo de base

El tipo de base que se escoja para la galga extensiométrica condicionará, al igual que el tipo de conductor, el rango de temperatura que soportará la misma, la máxima elongación del sensor, así como la precisión de la medida que nos proporcione el mismo. Fundamentalmente tenemos a nuestra disposición dos tipos de base: poliamida, fibra de vidrio reforzada y epoxy-fenólico. A continuación se muestra un listado de las características más importantes de cada tipo de base:

- Poliamida

  Rango de temperatura de -195ºC a 175ºC

  Ideal para estrés dinámico y estático

  Permite elongaciones $<20\,\%$

- Fibra de vidrio reforzada y epoxy-fenólico

  Rango de temperatura:

  Régimen normal: -269ºC a 290ºC

  Régimen puntual: hasta 400ºC

  Permite estrés dinámico y estático

  Permite elongaciones de 1 % a 2 %

Valor de la resistencia

Cada modelo de galga extensiométrica está disponible en distintos valores de resistencia nominal. Ésta resistencia nominal es la que sufrirá las variaciones en su valor en función de la elongación producida por las fuerzas de tracción y compresión a las que se va a ver sometida. La elección de éste valor representa un punto importante en nuestro proceso de elección de la galga idónea para nuestra aplicación ya que cuanto mayor sea el valor de la resistencia mayor será la variación de la misma en valor absoluto para una misma elongación. Además, un mayor valor de resistencia disminuye el calor disipado por la galga a causa del voltaje aplicado a la misma.

Longitud de la galga

La longitud de la galga es una característica muy importante en tanto en cuanto puede condicionar seriamente la precisión de la medida que realice.

Los materiales sometidos a una fuerte deformación tienden a concentrar la máxima deformación en regiones muy concentradas creando un alto gradiente de deformación en la zona sometida a estrés. Las galga extensiométricas tienden a promediar la deformación que sufre la región del material cubierta por la zona sensible de la galga (la rejilla). Dado que el promedio de una deformación no uniforme siempre será menor que la deformación máxima, una galga que sea más grande que la región sometida a la máxima deformación indicará una medida inferior a la deformación máxima que está sufriendo el material, lo cual sería erróneo.

Sin embargo, escoger galgas de menos de 3mm implica ciertas problemáticas ya que éstas galgas permiten una menor elongación, son inestables para deformaciones estáticas, gozan de menor vida útil ante deformaciones alternadas cíclicas y disipan peor el calor.

Para nuestra aplicación necesitaremos una galga estable y precisa para deformaciones estáticas, no requeriremos que la galga soporte altas temperaturas ya que supuestamente estarán instaladas en entornos a temperatura

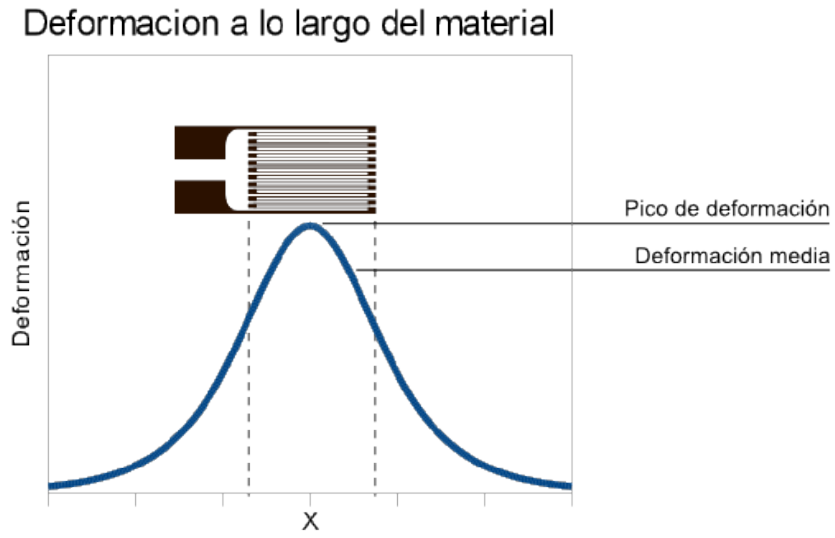


Figura 2.10: Gráfica de la deformación de galga a lo largo del material

ambiente y dado que las piezas serán relativamente voluminosas (vigas, pilares, etc), la región de deformación máxima será relativamente amplia, por lo que no podremos permitirnos galgas de un tamaño medio-alto ganando así estabilidad en las medidas entre otras ventajas añadidas. Además se valorará el factor de galga para simplificar en la medida de lo posible la electrónica añadida.

Teniendo en cuenta todas las necesidades y las características de cada tipo de galga vamos a decantarnos por una galga extensiométrica metálica de Constantán con base de Poliamida de 6.35mm de larga y 350 Ohmios ya que es la que mejor se ajusta a nuestras necesidades.

2.3. Fundamentos del protocolo de comunicación inalámbrica

El protocolo de comunicación diseñado para la interacción de los nodos hereda multitud de conceptos e ideas de las RIS. Pero se busca, a su vez, introducir cierto grado de innovación a la idea de red de sensores partiendo de los mismos principios. Se pretende pues darle una vuelta de tuerca más al objetivo de ahorro de energía combinándolo con la idea de un protocolo basado en divisiones de tiempo.

Uno de los recursos más importantes de las RIS en general y de la aplicación para la que esta diseñado el protocolo en particular es la energía. En este caso, además, debido a la dificultad añadida al sustituir alguno de los sensores, alargar la vida útil de cada nodo es una de las metas más vitales.

Es fácil darse cuenta de que las etapas de radio son una de las operaciones que más coste energético suponen para un nodo. De este modo el protocolo de comunicación ejecuta varias estrategias de ahorro energético que lo fundamentan.

El protocolo esta basado en conjuntos de divisiones o marcos de tiempo cuya finalidad es hacer llegar las lecturas de los sensores de la red a la estación base a través del nodo sumidero. En cada ranura temporal se utiliza un enlace que une virtualmente a dos nodos sensores y se pretende que tan solo dichos sensores participen en la comunicación durante dicha ranura temporal. De este modo se eliminan las colisiones y se evitan reenvíos innecesarios.

Además los enlaces que se utilizan vienen determinados por la programación enviada por el nodo sumidero, que debe utilizar para ello un grafo ponderado con rutas de caminos mínimos que una cada sensor en la red. El grafo se ponderará según la intensidad (RSSI) con la que los nodos perciben a los sensores que le rodean. Así las comunicaciones siempre se realizan al nodo que requiera menor consumo energético o lo que es lo mismo, el enlace del grafo de menor coste.

Es por esto que la formación del grafo, que es lo mismo que hablar del establecimiento de los enlaces de la red y el censo de todos los nodos que la componen es algo tan vital. Pero tan importante como la formación de la red es el mantenerla actualizada para evitar consumos desiguales de los nodos que la componen o para reencaminar los paquetes en caso de que un nodo falle. Por esto en el envío de las mediciones incluye, además del nivel de intensidad del enlace el nivel de batería, siempre que alguno de estos valores varíe significativamente superando un umbral configurable.

Tener el grafo y los caminos mínimos actualizados ante cambios en el mismo permite que el desgaste de la red sea uniforme. Además se incluye la capacidad de poder recuperar parte de la red reencaminando sus mensajes a otros nodos del grafo y recalculando los caminos mínimos después de haber eliminado el enlace que produjo el fallo.

2.3.1. Formación de la red

La formación de la red es una etapa fundamental, ya que determina los enlaces de la red y el funcionamiento del protocolo en su totalidad. La formación de la red debe por tanto ser fiable y es, de hecho, definitiva: si se forma mal la red está destinada a funcionar mal. El objetivo es conseguir que se registren todos los enlaces, que se envíe toda la información de los mismos al sumidero y que el nodo sumidero forme el grafo con toda la información en el menor tiempo posible.

En este periodo de formación también se siguen ciertos protocolos de

ahorro de energía. Durante esta fase el nodo sumidero no conoce a ningún nodo sensor, por lo tanto no puede arbitrar las comunicaciones. Se arbitran por sí mismas para conseguir el mismo objetivo que se persigue durante todo el protocolo, evitar reenvíos innecesarios y el ahorro del uso de etapas de radiofrecuencia.

La etapa consiste en una serie de envíos por difusión de un paquete de descubrimiento que genera respuestas en los nodos circundantes. Ésta emisión se genera inicialmente en el nodo sumidero para posteriormente ser enviada ordenadamente por todos los nodos que formarán la red. Cuando un nodo recibe respuestas anotan las direcciones que reciben de ellas en tablas que envían al sumidero junto con los niveles de intensidad con las que los perciben y su propio nivel de batería.

Los envíos se generan desde el sumidero y se alejan progresivamente como una ola hasta cubrir todos los nodos de la red. Cuando esto suceda el sumidero poseerá todas las tablas con nodos e intensidades de señal de recepción para crear el grafo ponderado y posteriormente calcular los caminos mínimos sobre él.

Otra de las características que permiten un ahorro energético por parte de los nodos sensores es la regulación del nivel de emisión a bajos niveles de potencia. Esto hace que las emisiones de los nodos sensores no alcancen al nodo sumidero, lo que obliga a encaminar los mensajes de los nodos sensores al nodo sumidero durante esta fase. Sin embargo la topología entera de la red esta destinada a que las emisiones atraviesen distintos nodos para amortizar el coste de una emisión entre distintos nodos sensores.

En la fase de formación de la red, que el árbol no esta aún formado es el propio sumidero el que le debe indicar al nodo sensor qué ruta debe seguir para enviar las tablas de intensidades y que el sumidero pueda registrar sus enlaces.

2.3.2. Función estándar

Esta fase engloba todo lo demás. Depende estrechamente de las estructuras de datos creadas y de su correcta actualización para que su funcionamiento sea fiable.

En esta fase se realizan las lecturas de los sensores de los nodos y se envían las mediciones tomadas al nodo sumidero para que las reciba la estación de monitorización correspondiente. Para ello hace uso de los *macro-frames* para dividir los envíos en espacios de tiempo a fin de evitar las colisiones.

Cada *macro-frame* empieza con un «bocinazo». Los envíos son programados por el nodo sumidero, ya que es quien posee el árbol de caminos mínimos. Se producen desde las hojas al sumidero para que se acumulen las medidas

progresivamente a medida que vayan subiendo de nivel en el árbol, hasta un límite de 6[3]. Superado este límite el nodo enviará 6 medidas en una división de tiempo y el resto en una segunda división de tiempo (y así cada vez que se acumule una cantidad múltiplo de seis medidas). Esta ranura temporal extra deberá ser prevista por el nodo sumidero durante el envío de la programación del *macro-frame*.

Habrá al menos tantas divisiones de tiempo como enlaces haya en el árbol de caminos mínimos, más una división extra para la lectura de las galga al principio de cada *macro-frame*. Cada división de tiempo supone el uso de uno de los enlaces del árbol, que a su vez supone una comunicación punto a punto entre dos nodos cercanos.

En los envíos de medidas se adjunta también el nivel de intensidad del enlace (que se actualiza en el paquete durante la recepción) y el nivel de batería del nodo emisor. Datos que, por otra parte, solo aparecerán en el paquete si se superó un umbral (configurable desde el interfaz de usuario) con respecto al último dato que se envió. Así los bytes enviados se acortan para acortar a su vez el tiempo que se consume en la etapa de radio (una de las operaciones que supone más consumo energético para los nodos).

En el envío de la programación los nodos deben saber en qué división de tiempo empezarán a participar y durante cuántas ranuras temporales lo harán (toda esa información incluida en un paquete de programación que envía el nodo sumidero por difusión). El «bocinazo» será la señal de sincronización para todos los sensores, a partir de la cual calcularán cuándo deben participar en la comunicación y hasta cuándo deberán hacerlo. Cuando un nodo sensor acabe su participación en el *macro-frame* pasará de nuevo al modo de ahorro de energía.

Al finalizar el *macro-frame* el nodo sumidero actualiza los enlaces del grafo y recalcula los caminos mínimos si fuese necesario en ambos casos, según las medidas recibidas. Así si un enlace (y por extensión un nodo) sufre un desgaste mayor que el resto se podrá actuar a tiempo y reencaminar los paquetes en sucesivos *macro-frames* para poder paliarlo. Además deberá envíar toda la información a la estación de monitorización para que actualice los datos haciendo uso del puerto USB.

Este mecanismo protege a su vez de errores en algún nodo de la red. Si no se recibe uno de los paquetes de mediciones, bien porque el nodo emisor ha fallado, bien porque se han agotado el número de reenvíos permitidos en un marco temporal sin obtener respuesta, el nodo receptor marca el paquete que reenviará con el fallo para que lo detecte el sumidero. Si el problema no se solventa en un segundo *macro-frame* (programado específicamente para tal propósito y que incumbe solo a los nodos implicados en el fallo) puede

[3]Debido al máximo de bytes en la emisión que impone la capa física

requerir que se redirija todo el tráfico del sub-árbol que colgaba del nodo que falló por otro enlace del grafo. Si por alguna razón no existiera otro enlace el sumidero avisaría del problema a la estación de monitorización y si no se ejecuta ninguna acción los nodos que fallaran no participarían de nuevo en los *macro-frames*.

2.4. Aplicación puente USB-IGU

La finalidad de la red inalámbrica de sensores es la de medir los parámetros que definen la deformación en los elementos que están sometidos a una tensión física. No obstante, la red no tiene ninguna utilidad si ésos datos sobre la deformación no se muestran de alguna forma al operario encargado de dicha red para que éste controle en todo momento el estado de los elementos sujetos a deformaciones y pueda tomar las medidas oportunas en caso de riesgo. Es por ello que la red necesita algún tipo de mecanismo que permita que los datos provenientes de la red sean visualizados en algún tipo de terminal al que el operario tenga fácil acceso. Además, dada la implementación concreta de nuestra red de sensores, se hace necesario que existan mecanismos mediante los cuales el operario pueda iniciar la red de sensores y éste pueda actuar sobre parámetros de la configuración de la red de forma que consiga el comportamiento deseado de la misma.

A la hora de elegir la arquitectura de la interfaz gráfica se han barajado básicamente dos vertientes. Como primera posiblidad se planteó la opción de hacer la interfaz gráfica en forma de aplicación que funcionara sobre Windows y/o Linux, pero ésta opción resultaba poco práctica ya que debía estar instalada en el terminal que tuviera directamente conectado el nodo sumidero al puerto USB y además resultaba costosa de implementar ya que para hacerlo debía hacer uso de bibliotecas de acceso al puerto USB y librerias de acceso a bases de datos, lo cual no es algo trivial en la mayoria de lenguajes. Además, realizar el interfaz gráfico también es un proceso relativamente costoso y muchas veces dependiente de la plataforma del sistema operativo. La segunda opción consiste en realizar un interfaz gráfico basado en Web en lugar de una aplicación nativa de un sistema operativo. Ésta opción tiene las ventajas de que no es dependiente del sistema operativo, no confina el acceso a los dato de la red al terminal al cual está conectado el nodo sumidero, permite acceder a la interfaz desde cualquier dispositivo que tenga acceso por red al terminal que dispone de nodo sumidero, y es mucho mas fácil de implementar. Por éstas razones nos hemos decantado por la opción de la interfaz via web ya que ofrece muchas ventajas y posibilidades de escalabilidad del sistema con un coste reducido.

Para poder llevar a cabo la tarea de presentar los datos en una pantalla

al usuario éstos tienen que pasar por una serie de etapas que dependerán en gran medida de la arquitectura que deseemos emplear. El objetivo es que el usuario que quiere visualizar los datos de la red pueda acceder con un navegador a una URL determinada y se le presenten en el navegador en tiempo real los datos que los sensores están enviando al sumidero. Sin embargo, el terminal que usa el usuario no será necesariamente el mismo terminal que tiene conectado el sumidero. Lo habitual será que el terminal que disponga del nodo sumidero esté conectado a una red o a Internet, y el terminal del usuario acceda mediante HTTP a un servidor que le ofrezca éstos datos.

En éste punto resulta evidente que lo lógico es que el propio terminal que dispone del nodo USB haga de servidor HTTP para ofrecerle de primera mano éstos datos al navegador del usuario. No obstante hay que recordar que el sumidero está conectado al terminal mediante el puerto USB y el servidor Web no dispone de mecanismos para acceder al puerto y formatear los datos del flujo del puerto para posteriormente transmitirlos al navegador del cliente. Además, éstos datos deben ser previamente volcados en una BBDD para poder extraer estadísticas de las anteriores lecturas registradas por los sensores además de ser necesarios para algunas operaciones internas de la interfaz gráfica. Por todo ésto se ha optado por desarrollar una aplicación ex profeso que se dedicará principalmente a crear una conexión permanente entre el nodo sumidero y la misma a través del puerto USB y actuar de intermediaria entre el nodo y la IGU propiamente dicha creando un puente entre ambos. Ésta aplicación (en adelante .ªplicación puente") recogerá los datos que le envíe el nodo sumidero a través del puerto USB, formateará convenientemente dichos datos y los volcará en la BBDD. De ésta forma, los datos quedarán almacenados listos para que la IGU los lea de la BBDD. Para que la IGU pueda leer los datos debe crear una conexión con la BBDD y ejecutar las sentencias oportunas para extraer los datos que la IGU requiere. Obviamente, el lenguaje HTML carece de los mecanismos que permiten interactuar con una BBDD, por ello, emplearemos un script PHP que será el encargado de interactuar dinámicamente con la BBDD para extraer los datos necesarios para la IGU.

En éste punto ya tenemos las herramientas necesarias para leer los datos provenientes de la red, volcarlos en una base de datos, y mostrarlos en un navegador a través de un script PHP. Sin embargo existe un problema de sincronismo entre el flujo de datos que proviente del sumidero y la visualización en el navegador de éstos datos. HTML no es un lenguaje dinámico, cuando se realiza una petición HTTP el servidor responde con los datos solicitados y el navegador los muestra por pantalla, pero si ésos datos son modificados en el servidor el navegador no reflejará el cambio a no ser que el navegador realice otra petición sobre ésos datos (el usuario actualiza la página). Extrapolandolo a nuestro caso concreto, los nuevos datos que la aplicación puente vaya

volcando en la BBDD el usuario no las visualizará a no ser que manualmente se actualice la página en el navegador. Ésta problemática tradicionalmente siempre se ha resuelto mediante etiquetas HTTP especiales que actualizaban la página pasado un periodo de tiempo determinado. Ésto es una solución a medias pues nunca se visualizan los datos en tiempo real, si el periodo de actualización automática es X, en el peor de los casos el usuario puede pasar X tiempo visualizando unas medidas que no se corresponden con la realidad ya que en el servidor esos datos fueron actualizados X tiempo atrás, pero el navegador todavia no ha realizado una nueva petición que actualice esos datos en pantalla. Además, ésta solución presenta la problemática de que cada X tiempo la página entera es recargada y ésto puede afectar a elementos interactivos de la página, lo cual es una molestia añadida y además una sobrecarga innecesaria en la red ya que cada X tiempo la página entera es retransmitida en su totalidad. Sin embargo, con la versión 5 de HTML éste problema queda resuelto de forma eficiente con los llamados WebSocket.

WebSocket es un mecanismo por el cual mediante unas pocas instrucciones JavaScript es posible crear una conexión sobre TCP entre un servidor y la página HTML (a través de JavaScript). WebSocket incluye su propio protocolo y está diseñado principalmente para que sea el propio servidor HTTP el que implemente éste protocolo y sea éste el encargado de gestionar las conexiones WebSocket. Sin embargo, a nosotros nos interesa usar éste mecanismo que nos permite comunicarnos con el código JavaScript de una página HTML principalmente para notificar a ésta la existencia de mediciones nuevas en el preciso instante en el que la aplicación puente las recibe, y de ésta forma que sea la propia página la que solicite los datos a la BBDD de forma inmediata mediante el script PHP. De éste modo no existirá ningun lapso de tiempo durante el cual el usuario esté visualizando mediciones antiguas en pantalla mientras en realidad existen mediciones más actualizadas en la BBDD. No obstante, para que ésta arquitectura de la interfaz funcione necesita una entidad que actue de servidor WebSocket y se encargue de notificar a la interfaz gráfica inmediatamente después de recibir medidas por parte del nodo sumidero. Ésta tarea recaerá en la aplicación puente que, si bien su función principal hasta ahora era la de actuar de intermediario entre el nodo sumidero y la BBDD, lo más lógico, simple y eficiente es que sea la misma la que implemente el servidor WebSocket para notificar de primera mano a la interfaz gráfica la existencia de nuevas mediciones y actuando de intermediario entre la IGU y el nodo sumidero. Ésta arquitectura nos brinda un gran número de posibilidades ya que, además de poder notificar a la interfaz gráfica en el momento en el que se actualicen medidas, podemos gestionar a través de WebSocket todo tipo de información de control sobre la red.

Con todo ésto ya tenemos una solución para la arquitectura que conforma comunicación de la red con la interfaz gráfica a través de la aplicación puente. La aplicación está constantemente a la escucha en el puerto USB a la espera

de mediciones por parte del nodo sumidero. Cuando llegan nuevas mediciones la aplicación puente vuelca éstos datos en la BBDD e inmediatamente notifica a la interfaz gráfica mediante WebSocket para que ésta extraiga los datos de la BBDD. Todo éste proceso se puede ver en la figura 2.11

2.4.1. WebSocket

WebSocket es un mecanismo que forma parte del conjunto del nuevo estándar HTML5 por el cual se permite que una página HTML inicie una conexión mediante sockets hacia un servidor, creando así un flujo bidireccional de datos entre la página y el servidor. WebSocket funciona sobre el protocolo TCP[4] y además incluye su propio protocolo el cual hace uso de unas cabeceras muy similares a las de HTTP.

Dado que HTML por definición es un lenguaje estático, por sí mismo no es capaz de hacer uso de WebSocket, ya que es incompatible con la naturaleza dinámica de un flujo de información proveniente de una conexión como la de un socket. Es por ello que el mecanismo de WebSocket se encuentra en el dominio de JavaScript, el cual sí es un lenguaje dinámico capaz de hacer uso de una conexión. Por lo tanto, para hacer uso de WebSockets hay que emplear el API disponible para JavaScript y realizar todas las acciones sobre el socket en el contexto de dicho lenguaje. Obviamente JavaScript también es capaz de alterar el contenido HTML de la página en la que está incluido, por lo que el uso lógico de JavaScript en el contexto de los WebSockets es el de hacer de intermediario entre la página HTML y el flujo de información que proviene del servidor WebSocket.

Para hacer uso de WebSocket tan sólo hay que conocer el nombre del servidor de WebSocket y el puerto en el que sirve dicho servicio. Una vez se establezca la conexión, JavaScript nos proporcionará un objeto que dispondrá de varias funciones relacionadas con el envío y la recepción de datos. A partir de ése momento tanto el servidor como el código JavaScript podrán hacer uso de la conexión para intercambiarse datos de forma totalmente bidireccional, ya sea en modo binario o ASCII, aunque para nuestro caso haremos uso únicamente del modo ASCII, por lo que será éste el modo en el que nos centraremos en éste texto.

WebSocket utilizada una arquitectura de comunicación muy simple. En primer lugar el cliente (el navegador) realiza una petición de conexión al puerto donde el servidor WebSocket está escuchando. Junto a la petición se adjuntan una serie de cabeceras muy similares a las de una petición HTTP. El servidor analiza las cabeceras y, si todo es correcto, responde al cliente aceptando la conexión y a su vez adjuntado otra serie de cabeceras. En el

[4]Transmission Control Protocol. Protocolo orientado a conexión perteneciente a la capa de Sesión de la pila TCP/IP.

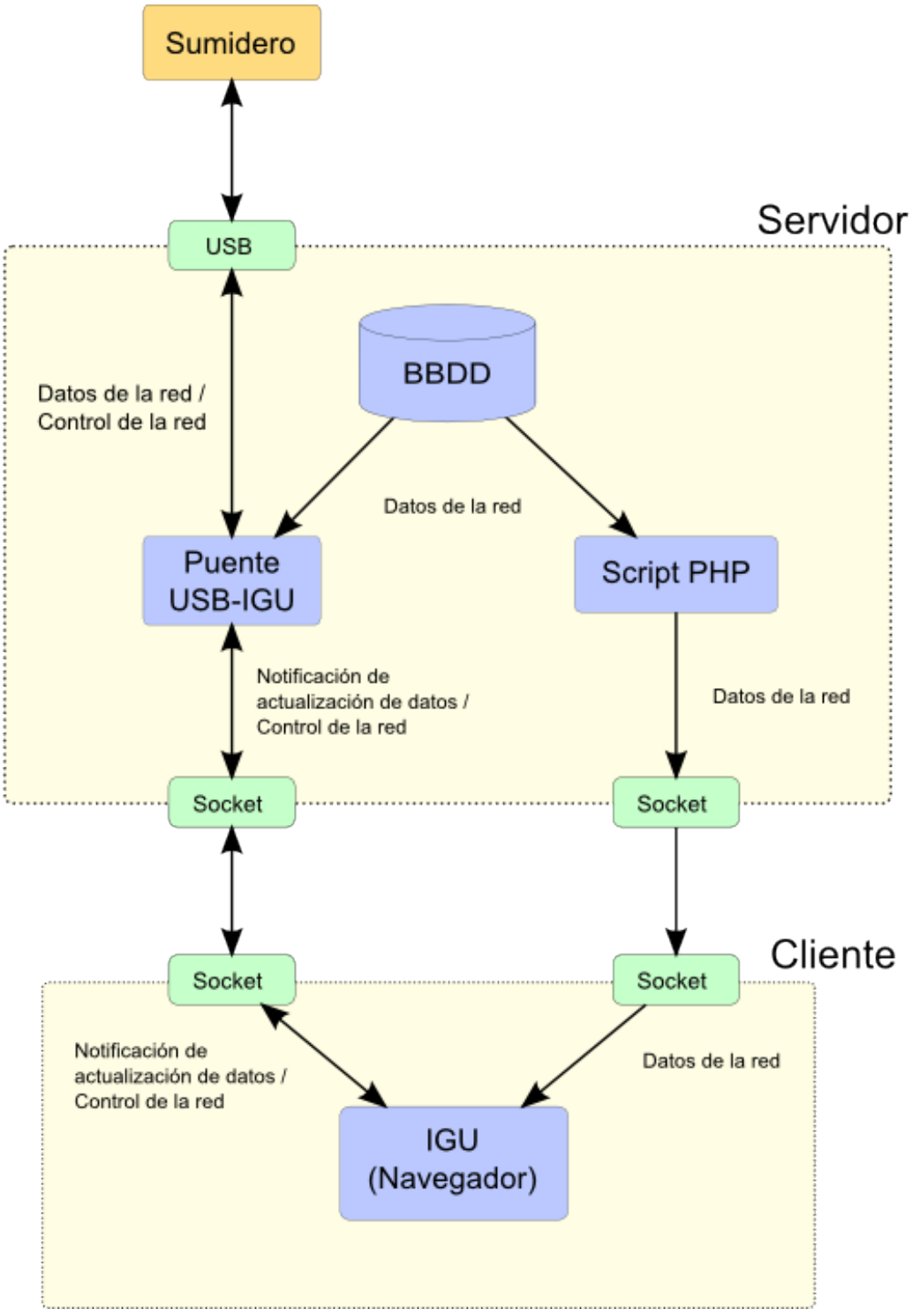


Figura 2.11: Diagrama de la arquitectura de comunicaciones de la red con el IGU

momento en el que el cliente recibe las cabeceras por parte del servidor y comprueba su válidez la conexión queda establecida y ambos pueden enviarse mutuamente flujos de datos de forma bidireccional. En la figura 2.12 se muestra un diagrama que muestra de forma simple éste proceso.

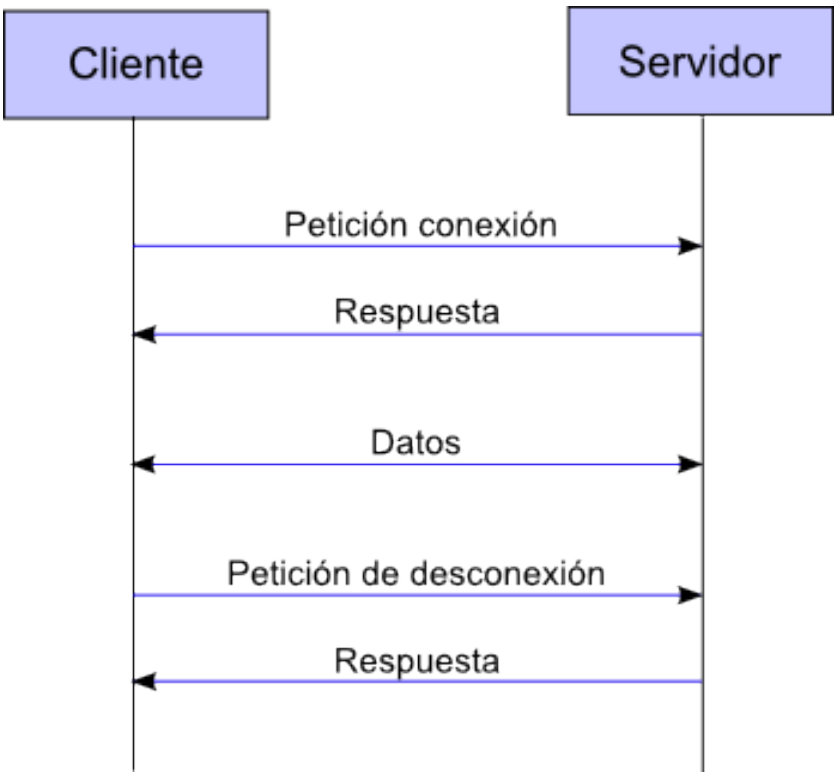


Figura 2.12: Diagrama de una comunicación WebSocket

Antes de pasar a describir el protocolo WebSocket cabe aclarar que la aplicación puente implementa la versión 76 del borrador de RFC correspondiente a WebSocket del 7 de Noviembre de 2010. Ésta especificación del protoclo WebSocket ofrece distintas posibilidades de conexión, sin embargo nuestro servidor WebSocket no implementa todas las posibilidades dado que no son de utilidad para nuestra aplicación. Por ello en el presente texto tan sólo explicaremos las funcionalidades implementadas en nuestro servidor. Éstas funcionalidades cumplen con la especificación WebSocket a pesar de que los mecanismos que no son de utilidad de la especificación no están contemplados.

WebSocket ofrece dos tipos de conexión, la conexión segura y la no segura. Nuestro servidor tan sólo implementa el tipo de conexión segura, por lo que si el cliente intenta conectar en modo no seguro la negociación de la conexión no tendrá éxito y la conexión será abortada. Por otro lado WebSocket en el momento de transmitir los datos soporta dos modos de transmisión: ASCII y binario. La aplicación puente tan sólo soporta el modo ASCII por lo que

si un cliente intenta enviar datos en modo binario el comportamiento de la aplicación puente puede ser imprevisible. En la versión actual del borrador de RFC citada ántes se advierte que el modo binario actualmente no se puede usar, por lo que a dia de hoy es innecesario implementar dicha funcionalidad. Hay que destacar que WebSocket actualmente es un borrador por lo que todavia se están realizando cambios en la especificación del protocolo y no todos los navegadores implementan la misma versión del borrador de WebSocket y algunos ni siquiera lo implementan en absoluto.

Realizadas las pertinentes aclaraciones pasaremos a explicar el proceso de conexión, transmisión y desconexión de una conexión WebSocket teniendo en cuenta las restricciones de la implementación.

Protocolo del lado del cliente

Establecimiento de conexión

En primer lugar el cliente establece una conexión con el servidor WebSocket. Cuando el servidor acepta la petición de conexión TCP el cliente envía envía las cabeceras. Éstas cabeceras se transmiten como texto plano en formato UTF-8 de forma muy similar a como lo hace HTTP. Las cabeceras se dividen en lineas separadas por los caracteres CR (0x0D) LF (0x0A) correspondiendo cada una a un campo de las cabeceras de la petición de conexión.

Cabe destacar que la aplicación puente tan sólo reconoce las cabeceras obligatorias de la especificación WebSocket, por lo que tan sólo se describirán dichas cabeceras o campos.

Cabeceras enviadas por el cliente

La primera cabecera o campo siempre tiene el formato de una petición GET HTTP contra el servidor. Ejemplo:

GET / HTTP/1.1

El resto de cabeceras que le siguen pueden tener un orden aleatorio. Por lo que el servidor no puede asumir un orden concreto a la hora de procesarlas. Dichas cabeceras se describen a continuación. Éstas cabeceras tendrán un formato clave-valor separando la clave del valor con los caracteres 0x3A (:) 0x00 (espacio). Cada par clave-valor terminará con 0x0D 0x0A (CR LF).

La cabecera *Host* contiene el nombre y puerto del servidor al que va dirigida la petición de conexión. Éste campo se usa para evitar el ataque de

DNS Rebinding[5]. Ejemplo:

Host: localhost:9321

La cabecera *Origin* contiene la URL del cliente que solicitó la conexión. Éste campo es usado para evitar ataques de origen cruzado no autorizado. Ejemplo:

Origin: http://localhost

Las cabeceras *Upgrade* y *Connection* son cabeceras propias de HTML. Dado que WebSocket es un protocolo pensado para que sea implementado por servidores HTTP se incluyen éstas cabeceras con el fin de que las peticiones WebSocket sean compatibles con HTTP y sean entendidas por el servidor HTTP además de identificar la petición como una petición de conexión WebSocket. Éstas cabeceras siempre tienen el mismo contenido. Ejemplo:

Upgrade: WebSocket

Connection: Upgrade

Las cabeceras anteriormente descritas son las cabeceras obligatorias que debe enviar el cliente WebSocket para ajustarse a la especificación WebSocket. Sin embargo, como se explicó anteriormente WebSocket contempla dos tipos de conexión: segura y no segura. Si se desea establecer una conexión no segura sería suficiente con enviar las cabeceras descritas hasta ahora. No obstante, en nuestra aplicación haremos uso del modo de conexión seguro. Éste modo exige enviar tres campos adicionales que detallaremos a continuación.

Los campos *Sec-WebSocket-Key1* y *Sec-WebSocket-Key2* contienen cada uno un entero aleatorio generado conforme al algoritmo que se describe a continuación.

Generación de claves desafio-respuesta de Secure WebSocket

- En primer lugar generamos dos enteros aleatorios no mayores a 4,294,967,295 a los que llamaremos *max1* y *max2* respectivamente. Ejemplo:

  858,993,459 y 477,218,588

- Generamos un entero aleatorio comprendido entre 0 y *max1* al que llamaremos *num1*. Ejemplo:

  777,007,543

[5]Es un ataque basado en DNS de código embebido en páginas web aprovechándose de la política del mismo origen de los navegadores.

- Generamos otro entero aleatorio comprendido entre 0 y *max2* al que llamaremos *num2*. Ejemplo:

  114,997,259

- Luego generamos otro par de enteros aleatorios comprendidos entre 1 y 12 ambos inclusive a los que llamaremos *spc1* y *spc2*. Ejemplo:

  5 y 9

- Multiplicamos *num1* y lo multiplicamos por *spc1*. Al resultado lo llamaremos *producto1*. En nuestro ejemplo obtendremos 3,885,037,715.

- Multiplicamos *num2* y lo multiplicamos por *spc2*. Al resultado lo llamaremos *producto2*. En nuestro ejemplo obtendremos 1,034,975,331.

- Convertimos los enteros *num1* y *num2* en sendas cadenas de caracteres que contengan dichos números expresados con los caracteres UTF-8 desde el 0x0x30 al 0x39. En nuestro ejemplo obtendríamos "3885037715ƨ "1034975331ƨ los llamaremos *clave1* y *clave2* respectivamente.

- Insertamos en la *clave1* un número aleatorio comprendido entre 1 y 12 de caracteres UTF-8 contenidos en los rangos 0x21-0x2F y 0x3A-0x7E en posiciones aleatorias de *clave1*. Obtenemos:

  «P388O503D&ul7{K %gX(%715»

- Insertamos en la *clave2* otro número aleatorio comprendido entre 1 y 12 de caracteres UTF-8 contenidos en los rangos 0x21-0x2F y 0x3A-0x7E en posiciones aleatorias de *clave2*. Obtenemos:

  «1N?|kUT0or3o4I97N5-S3O31»

- Insertamos *spc1* espacios en posiciones aleatorias de *clave1*. Obtendremos:

  «P388 O503D&ul7 {K %gX(%7 15»

- Insertamos *spc2* espacios en posiciones aleatorias de *clave2*. Obtendremos:

  «1 N ?|k UT0or 3o 4 I97N 5-S3O 31»

  clave1 y *clave2* serán las claves que se enviarán en los campos *Sec-WebSocket-Key1* y *Sec-WebSocket-Key2* respectivamente.

- Seguidamente a éstas cabeceras se insertará una linea en blanco añadiendo los caracteres 0x0D y 0x0A, y una tercera clave que no tendrá nombre de campo. Es decir, la clave ocupará la linea entera. Ésta

clave estará formada por 8 bytes aleatorios (codificados en orden Big-Endian[6]). Ésta clave no terminará con 0x0D y 0x0A como el resto de cabeceras.Por ejemplo:

0x47 0x30 0x22 0x2D 0x5A 0x3F 0x47 0x58

Y con ésto acabarían las cabeceras adicionales que el cliente enviará en el caso de que desee establecer una conexión en modo seguro de WebSocket. Al enviar al servidor el paquete con la petición de conexión el servidor responderá a la petición con otro paquete con sus cabeceras. Si todo ésta correcto en el paquete de respuesta del servidor, el proceso de negociación de la conexión se da por finalizada. En éste punto tanto el cliente como el servidor ya estarán en posición de enviar datos a través de la conexión.

Transmisión de datos

El protocolo de transmisión de datos mediante una conexión WebSocket es muy simple. Los datos que se deseen enviar deben ir precedidos por el valor 0x00 y deben finalizar con 0xFF. Entre éstos dos valores deben ir los valores de los caracteres codificados en UTF-8.

Cierre de conexión

Si el cliente desea cerrar la conexión enviará los caracteres 0xFF 0x00. El servidor responderá 0xFF confirmando la desconexión.

Protocolo del lado del servidor

El servidor siempre estará escuchando el puerto correspondiente a WebSocket. Cuando el servidor reciba una conexión en el puerto WebSocket recibirá automáticamente las cabeceras enviadas por el cliente que pretende establecer una conexión WebSocket.

Establecimiento de conexión

Al recibir las cabeceras del cliente el servidor deberá procesarlas y devolver otro conjunto de cabeceras, algunas de las cuales dependerán de las cabeceras recibidas por parte del cliente.

El conjunto de cabeceras de respuesta del servidor tiene exactamente el mismo formato que las enviadas por el cliente, sólo que el servidor enviará

[6]El orden Big-Endian establece el orden de los pesos de bits de forma que el bit más significativo de cada byte es el que está situado más a la derecha

otras cabeceras distintas a las enviadas por el cliente. Al igual que sucede con el cliente todas las cabeceras estarán separadas por los caracteres 0x0D 0x0A (CR LF).

La primera cabecera será la respuesta a la petición del cliente. Estará formada siempre por la cadena de caracteres «HTTP/1.1 101 WebSocket Protocol Handshake» terminada con 0x0D 0x0A.

El resto de cabeceras serán una serie de pares clave-valor separando la clave del valor con los caracteres 0x3A (:) 0x20 (espacio). Cada cabecera terminará con 0x0D 0x0A (CR LF). A su vez se deberán añadir un par de caracteres 0x0D 0x0A adicionales para delimitar el final de la lista de cabeceras y el comienzo de una cadena de 16 bytes que serán generados en función de los valores de los campos *Sec-WebSocket-Key1*, *Sec-WebSocket-Key2* y los últimos 8 bytes aleatorios enviados por el cliente en el paquete de negociación de la conexión. El orden de éstas cabeceras debe ser aleatorio.

Definiremos *host* como el valor obtenido de la cabecera *Host* enviada por el cliente. Si nuestro servidor no implementa VirtualHosts[7] éste campo puede ser fijo y se corresponde con la URL del servidor WebSocket.

Enviamos la cabecera *Upgrade* cuyo contenido debe ser *WebSocket*.

Enviamos la cabecera *Connection* cuyo contenido debe ser *Upgrade*

Definimos *wsURL* como la concatenación de «wss://» con *host*. Si nuestro servidor implementara la conexión no segura se contemplaria la posibilidad de concatenar «ws://» en lugar de «wss://». Además, si nuestro servidor soportara VirtualHosts habría que tener en cuenta que después de *host* deberiamos concatenar el nombre del recurso proporcionado en la primera cabecera de la petición del cliente (el contenido existente entre el segundo y tercer caracter 0x20). Sin embargo, dado que no es nuestro caso podemos permitirnos ésta simplificación de la cabecera.

Enviamos la cabecera *Sec-WebSocket-Location* con *wsURL* como valor.

Definimos *origen* como el valor contenido en la cabecera *Origin* enviada en la petición del cliente.

Enviamos la cabecera *Sec-WebSocket-Origin* con *origen* como valor.

Finalmente enviamos los caracteres 0x0D 0x0A para marcar el fin del conjunto de cabeceras.

No obstante, dado que estamos manejando una conexión WebSocket en modo seguro debemos calcular una cadena de 16 bytes en base a ciertos datos contenidos en la petición del cliente y devolver éstos 16 bytes en la respuesta a la petición. Éstos 16 bytes se concatenan justo después de los dos últimos

[7] Virtual hosting es un mecanismo para hospedar distintos nombres de dominio en un servidor HTTP usando una sola IP

caracteres 0x0D 0x0A.

Cálculo de la solución al desafio-respuesta

- Definimos *clave1* como el contenido del campo *Sec-WebSocket-Key1* enviado por el cliente.
- Definimos *clave2* como el contenido del campo *Sec-WebSocket-Key2* enviado por el cliente.
- Definimos *numerosClave1* como los caracteres comprendidos entre 0x30 y 0x39 que contiene *clave1* ordenados por orden de aparición. Ejemplo:
- Sec-WebSocket-Key1: 3e6b263 4 17 80 $\rightarrow 3,626,341,780$
- Definimos *numerosClave2* como los caracteres comprendidos entre 0x30 y 0x39 que contiene *clave2* ordenados por orden de aparición. Ejemplo:
- Sec-WebSocket-Key2: 17 9 G‘ZD9 2 2b 7X 3 /r90 4 17 80 $\rightarrow 1,799,227,390$
- Definimos *clave3* como la cadena de 8 bytes aleatorios enviada por el cliente. Ésta cadena está formada por los 8 bytes siguientes a la única aparición de los caracteres 0x0D 0x0A 0x0D 0x0A de forma consecutiva. Ejemplo:

  Cabeceras enviadas por el cliente:

  ```
  GET / HTTP/1.1
  Connection: Upgrade
  Host: example.com
  Upgrade: WebSocket
  Sec-WebSocket-Key1: 3e6b263  4 17 80
  Origin: http://example.com
  Sec-WebSocket-Key2: 17  9 G‘ZD9   2 2b 7X 3 /r90

  WjN}|M(6
  ```

  clave3 = WjN}|M(6

- Definimos *spc1* como el número de caracteres 0x20 que aparecen en *clave1*. En nuestro ejemplo éste valor sería 4.
- Definimos *spc2* como el número de caracteres 0x20 que aparecen en *clave2*. En nuestro ejemplo éste valor sería 10.
- Definimos como *parte1* el resultado de dividir *numerosClave1* entre *spc1*. En el ejemplo el resultado sería 906,585,445.

- Definimos como *parte2* el resultado de dividir *numerosClave2* entre *spc2*. En el ejemplo el resultado sería 179,922,739.

- Definimos como *desafio* la concatenación de *parte1*, expresado como un entero de 32 bits Big-Endian, *parte2*, expresado igualmente como un entero de 32 bits Big-Endian, y *clave3*. El orden de la concatenación debe ser el mismo orden en el que se enviaron dentro del conjunto de las cabeceras de la petición. En nuestro ejemplo *desafio* contendría los siguientes 16 bytes: 0x36 0x09 0x65 0x65 0x0A 0xB9 0x67 0x33 0x57 0x6A 0x4E 0x7D 0x7C 0x4D 0x28 0x36.

- Finalmente definimos como *respuesta* la huella generada al calcular el algoritmo MD5 sobre *desafio* como una cadena de caracteres de 128 bits Big-Endian. En nuestro ejemplo la huella MD5 de *desafio* se corresponderia con los bytes: 0x6E 0x60 0x39 0x65 0x42 0x6B 0x39 0x7A 0x24 0x52 0x38 0x70 0x4F 0x74 0x56 0x62.

  Ésta huella de 16 bytes será la que finalmente se inserte al final de la respuesta a la petición WebSocket. El cliente la analizará y si coincide con la que él mismo calculó y el resto de cabeceras son correctas, se establecerá la conexión y tanto el servidor como el cliente podrán enviar datos de forma bidireccional a través de la conexión WebSocket.

Transmisión de datos

Al igual que se explicó en el apartado dedicado al algoritmo del lado del cliente, que se desee enviar el servidor deben ir precedidos por el valor 0x00 y deben finalizar con 0xFF. Entre éstos dos valores deben ir los valores de los caracteres codificados en UTF-8.

Cierre de conexión

Al igual que sucede en el caso de que el cliente desee iniciar el proceso de desconexión, si es el servidor el que desea cerrar la conexión enviará los caracteres 0xFF 0x00. El servidor responderá 0xFF confirmando la desconexión.

Ejemplo de captura del intercambio de cabeceras

Aqui se muestra un ejemplo de la petición de conexión WebSocket de un cliente (Firefox 4.0 Beta 7) y la respuesta por parte del servidor. Éstas capturas están extraidas con WireShark de una comunicación real de nuestro servidor implementado.

Petición por parte del cliente

```
GET / HTTP/1.1
Sec-WebSocket-Key1: & 3  '91  4 k  423  g.450
Upgrade: WebSocket
Sec-WebSocket-Key2: # 1 4    5   04  7066 0
Host: localhost:9321
Connection: Upgrade
Origin: http://localhost

C>}m
```

```
0000   47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a  GET / HTTP/1.1..
0010   53 65 63 2d 57 65 62 53 6f 63 6b 65 74 2d 4b 65  Sec-WebSocket-Ke
0020   79 31 3a 20 26 20 33 20 20 27 39 31 20 20 34 20  y1: & 3  '91  4
0030   6b 20 20 34 32 33 20 20 67 2e 34 35 30 0d 0a 55  k  423  g.450..U
0040   70 67 72 61 64 65 3a 20 57 65 62 53 6f 63 6b 65  pgrade: WebSocke
0050   74 0d 0a 53 65 63 2d 57 65 62 53 6f 63 6b 65 74  t..Sec-WebSocket
0060   2d 4b 65 79 32 3a 20 23 20 31 20 34 20 20 20 20  -Key2: # 1 4
0070   35 20 20 20 30 34 20 20 37 30 36 36 20 30 0d 0a  5   04  7066 0..
0080   48 6f 73 74 3a 20 6c 6f 63 61 6c 68 6f 73 74 3a  Host: localhost:
0090   39 33 32 31 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e  9321..Connection
00a0   3a 20 55 70 67 72 61 64 65 0d 0a 4f 72 69 67 69  : Upgrade..Origi
00b0   6e 3a 20 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c 68  n: http://localh
00c0   6f 73 74 0d 0a 0d 0a b6 b8 43 3e 7d 6d 1f d3     ost......C>}m..
```

Respuesta por parte del servidor

```
HTTP/1.1 101 Web Socket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://localhost
Sec-WebSocket-Location: ws://localhost:9321/

o7s!
```

```
0000   00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  ..............E.
0010   00 f1 d2 15 40 00 40 06 69 ef 7f 00 00 01 7f 00  ....@.@.i.......
0020   00 01 24 69 c7 65 9d f1 02 e9 9e 88 84 9d 80 18  ..$i.e..........
0030   02 11 fe e5 00 00 01 01 08 0a 00 20 d0 fd 00 20  ........... ... 
0040   d0 fd 48 54 54 50 2f 31 2e 31 20 31 30 31 20 57  ..HTTP/1.1 101 W
0050   65 62 20 53 6f 63 6b 65 74 20 50 72 6f 74 6f 63  eb Socket Protoc
```

```
0060   6f 6c 20 48 61 6e 64 73 68 61 6b 65 0d 0a 55 70  ol Handshake..Up
0070   67 72 61 64 65 3a 20 57 65 62 53 6f 63 6b 65 74  grade: WebSocket
0080   0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 55 70  ..Connection: Up
0090   67 72 61 64 65 0d 0a 53 65 63 2d 57 65 62 53 6f  grade..Sec-WebSo
00a0   63 6b 65 74 2d 4f 72 69 67 69 6e 3a 20 68 74 74  cket-Origin: htt
00b0   70 3a 2f 2f 6c 6f 63 61 6c 68 6f 73 74 0d 0a 53  p://localhost..S
00c0   65 63 2d 57 65 62 53 6f 63 6b 65 74 2d 4c 6f 63  ec-WebSocket-Loc
00d0   61 74 69 6f 6e 3a 20 77 73 3a 2f 2f 6c 6f 63 61  ation: ws://loca
00e0   6c 68 6f 73 74 3a 39 33 32 31 2f 0d 0a 0d 0a 6f  lhost:9321/....o
00f0   e1 ac 97 37 04 df 73 9d c8 9e db da cc d2 21     ...7..s.......!
```

Capítulo 3

Protocolo de Comunicación Propuesto

3.1. Supuestos Básicos

El protocolo de comunicación inalámbrica propuesto atiende a unos supuestos que derivan directamente del tipo de aplicación para la que esta pensado. Como ya se ha indicado, los nodos que componen la red irán equipados con galgas extensiométricas adheridas a pilares metálicos en estructuras de hormigón. Las galgas capturarán las microdeformaciones que se puedan producir en los pilares en caso de la acción de esfuerzos axiales que produzcan el pandeo de los mismos. Una lectura anormal podría avisar a tiempo de una deformación que derivara en un derrumbe.

Sin embargo las lecturas en microdeformaciones no varían con frecuencia. Pueden pasar meses antes de que una lectura pueda considerarse crítica, y las lecturas se realizarán en plazos configurables de horas e incluso días. Además las sensores no son reemplazables fácilmente ya que las galgas extensiométricas están adheridas a la superficie metálica. Así el protocolo parte del supuesto de que el tiempo no es un factor importante, mientras sí lo es alargar la vida útil de los sensores lo máximo posible. Como se ha indicado anteriormente la energía es un recurso escaso en las RIS, y todos sus protocolos están orientados al ahorro máximo del mismo. Pero como hemos visto en la aplicación que nos ocupa es, si cabe, más importante (además de poder aprovechar los largos periodos de tiempo en los que los nodos pasan ociosos para ejecutar estrategias adicionales de ahorro de energía).

Otra de las características de la red es la diferenciación de dos tipos de nodo que ocurre en la gran mayoría de las RIS: nodo sensor y nodo sumidero. La red estará compuesta de nodos sensores, mientras que el nodo sumidero será quien centralice la información, controle al resto de nodos y actúe como

puerta de enlace entre los nodos sensores y la estación de monitorización. El protocolo propuesto es centralizado, delegando toda la responsabilidad sobre el nodo sumidero. A cambio el nodo sumidero no estará sujeto a las mismas restricciones energética o de memoria que el resto de nodos. Además contará con una antena más potente que le permitirá alcanzar al resto de nodos sin necesidad de encaminar paquetes.

Los nodos sensores por otra parte sí que deberán encaminar sus paquetes hacia el sumidero. Es por esto que resulta necesaria una estructura de árbol para que los nodos sepan hacia donde encaminar sus emisiones a fin de que los paquetes de lecturas alcancen al sumidero. Durante todo el proceso de formación del mencionado árbol los nodos sensores permanecen en escucha ociosa y se van dando a conocer al nodo sumidero mediante unos paquetes especiales.

A fin de mantener el árbol actualizado, a las lecturas de sensores se adjuntan la intensidad del enlace y el nivel de batería del nodo emisor para ponderar de nuevo un arco del árbol si fuese necesario. Esto permite ajustar un posible desgaste desigual de los nodos cercanos al sumidero, optimiza las emisiones entre nodos sensores y ofrece cierta protección frente al fallo de algún nodo de la red.

3.2. Fundamentos

3.2.1. Formato de trama de Nível Físico

Capa Física del CC1110

Las emisiones se realizan sobre la capa física que ofrece el propio integrado CC1110 sobre el que se ha desarrollado el diseño de los sensores y las pruebas de protocolo.

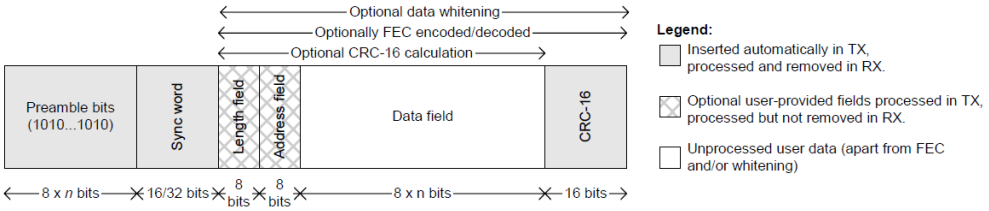


Figura 3.1: Formato de la Capa Física del CC1110

El formato del paquete que ofrece el CC1110 consiste en los siguientes campos:

- Preámbulo: es una secuencia alterna de ceros y unos que marcan el inicio del paquete.
- Palabra de sincronización: un valor configurable de dos bytes que ofrece sincronización de bytes en la recepción.
- Longitud del paquete: longitud del campo de datos en bytes. El valor máximo del campo es de 255 bytes. Si se necesitaran paquetes de mayor longitud se debe activar el modo de longitud de paquete infinito modificando un registro de la placa. El valor mínimo, por contra, es de 1 byte.
- Byte de dirección: dirección física del nodo de un byte.
- Campo de datos: éste es el campo sobre el que se encapsulará la información del protocolo.
- CRC Opcional de 2 bytes: la capa ofrece un mecanismo de detección de errores en la transmisión mediante dos bytes de CRC.

Formato de paquete del SimpliciTI

Dentro del campo de datos del CC1110 se encapsula parte de la estructura de paquete de la capa física del SimpliciTI[1], que se abordará más detalladamente en el siguiente capítulo.

Hay partes del paquete que se solapan con la capa física que ofrezca en su caso el dispositivo sobre el que se este desarrollando.

PREAMBLE	SYNC	LENGTH	MISC	DSTADDR	SRCADDR	PORT	DEVICE INFO	TRACTID	Security	App Payload	FCS
RD*	RD*	1	RD*	4	4	1	1	1	CTR (1) MAC (2)	*n*	RD*

Figura 3.2: Formato de la Capa Física del SimpliciTI

RD en la figura superior se refiere a Radio Dependiente, es decir, que depende de la capa física del CC1110, campos que a su vez son procesados por el hardware asociado a la antena. El campo de FCS es también opcional y dependiente del campo CRC del CC1110 y de su circuitería.

Como hemos indicado veremos los detalles más adelante, pero cabe destacar que SimpliciTI usa direcciones de cuatro bytes y un puerto que denota en este caso la aplicación para la que esta destinado el paquete. Estos campos podrían considerarse de nivel de red (usualmente el direccionamiento incumbe

[1]Librerías que implementan un sencillo protocolo para productos productos basados en la comunicación por radiofrecuencia de baja potencia, consultar sección 4.4

a dicha capa), pero esto es una cuestión que responde a pura semántica. En este caso se encarga una librería propia del SimpliciTI que sería ubicada entre la capas de enlace y física del estándar OSI.

3.2.2. Formato de paquetes del protocolo

Sobre el campo de datos del paquete del nivel físico que ofrece SimpliciTI se encapsulan los paquetes del protocolo con un único campo añadido de cabecera de cuatro bits que indicará el ID del paquete. Como se ha indicado anteriormente el valor mínimo de la longitud de paquete es de un byte, por lo que en algunos paquetes habrá que extender los cuatro bits con relleno para que la longitud sea un múltiplo de octeto, codificando la ID en la parte alta del byte correspondiente.

ID	Paquete
0x00	ACK
0x10	Descubrimiento
0x20	Contestación al descubrimiento
0x30	Orden de descubrimiento
0x40	Contestación de tablas de intensidades
0x50	Programación del *macro-frame*
0x60	Envío de mediciones
0x70	Actualización de configuración

Tabla 3.1: Tabla de IDs de paquete

De entre los paquetes descritos, los identificadores del 1 al 4 corresponden a paquetes que se generan durante la etapa de formación de la red; los identificadores 5 y 6 son los paquetes que se generan durante el funcionamiento estándar para, respectivamente, enviar la configuración del macro-frame a los nodos sensores y enviar las lecturas de los nodos al sumidero; y por último el identificador 7 corresponde a un paquete especial en caso de que se quisiera modificar algunos de los parámetros configurables del protocolo. El identificador 0 es el de acuse de recibo básico que simplemente se usa como confirmación de la recepción de un paquete, es de uso general en todas las fases del protocolo en las comunicaciones punto a punto entre nodos.

Paquete de descubrimiento

Este paquete (ID: 1) será enviado por todos los nodos en la etapa de formación de la red, empezando por el nodo sumidero. Marca el inicio de la etapa de formación de la red y el sumidero realiza esta emisión con el nivel de

potencia mínimo (-30 dBm[2]) para que le escuchen tan solo los nodos sensores más cercanos (ya que la emisión se realiza a la dirección de difusión).

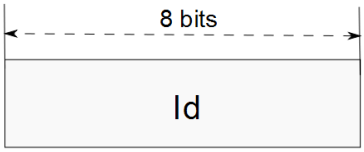


Figura 3.3: Formato del paquete de descubrimiento

El paquete no consta más que de 8 bits, 4 bits para el ID y otros 4 de relleno. El nodo emisor se queda a la escucha de contestación, la cual llega de sus nodos más cercanos motivada por la recepción de dicho paquete. El emisor espera contestación durante un *timeout* renovable para poder recibir todas las contestaciones que pudieran producirse.

Paquete de Contestación al descubrimiento

El paquete de Contestación al descubrimiento (ID: 2) se envía como contestación al paquete anterior. Este paquete se envía a la dirección de la que se recibió el paquete de descubrimiento y el nodo sumidero también deberá contestar si lo escuchara.

El envío se realiza mediante un algoritmo similar al retroceso exponencial binario[3] para evitar colisiones, incluyendo la detección de portadora antes de emitir. Así, antes de emitir, los nodos esperarán un tiempo aleatorio proporcional al número de intentos realizados y con un umbral máximo correspondiente al *timeout* del emisor y a diez reintentos para posteriormente escuchar el medio antes de enviar.

Con las medidas relatadas se espera que el número de colisiones sea el mínimo posible teniendo en cuenta

El formato es idéntico al del descubrimiento solo que con la ID correspondiente.

Este paquete simboliza un enlace entre dos nodos (o medio enlace, ya que éste debe ser bidireccional para que se utilice en el grafo) y el receptor anotará el RSSI con que escucha el paquete.

[2]El dBm se define como el nivel de potencia en decibelios en relación a un nivel de referencia de 1 mW.

[3]Algoritmo desarrollado para Ethernet con CSMA/CD, que consiste en reducir gradualmente la «agresividad» del emisor aumentando el espacio de espera para el reenvío si se encuentra con una situación de medio congestionado

El emisor de este paquete esperará un ACK[4] del receptor para salvar posibles colisiones.

Paquete de Orden de descubrimiento

La Orden de descubrimiento (ID: 3) es un paquete que envía el nodo sumidero a los nodos que tenga registrados, ya sea directamente a través de otros nodos. En este segundo caso al paquete se le adjuntará la ruta a través de la cual ha sido descubierto para que pueda encaminar la contestación hacia el sumidero. La ruta incluye la dirección del sumidero, de forma que cuando un nodo intermedio tenga que encaminar eliminará al nodo receptor de la ruta y adjuntará de nuevo al resto de direcciones hasta que llegue al sumidero (que al ser el último nodo la ruta estará vacía). Este mecanismo se basa en la premisa de que el nodo sumidero alcanza a todos los nodos de la red.

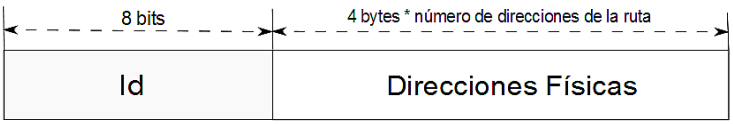


Figura 3.4: Formato del paquete de Orden de descubrimiento

Este paquete motiva que un nodo sensor envíe el paquete de descubrimiento para registrar a los nodos cercanos tal como hemos visto en los paquetes anteriores. Una vez vencido el *timeout* del descubrimiento el nodo receptor de la orden contestará con las tablas de intensidades y su nivel de batería al siguiente nodo en la ruta adjuntando el resto de direcciones físicas de la misma.

El formato incluye 4 bits de relleno entre el ID y la ruta.

Paquete de Contestación de Tablas de intensidades

Después de anotar las intensidades de señal con las que recibe las contestaciones al descubrimiento, un nodo sensor envía su batería y dichas intensidades al nodo sumidero en un paquete (ID: 4) de este tipo. Este paquete es el más importante de la fase de formación de la red, ya que es la que recoge la información de intensidades de señal y baterías que permitirá al nodo sumidero formar el grafo ponderado y calcular los caminos mínimos, clave del protocolo propuesto.

[4]Acuse de recibo, explicado más adelante.

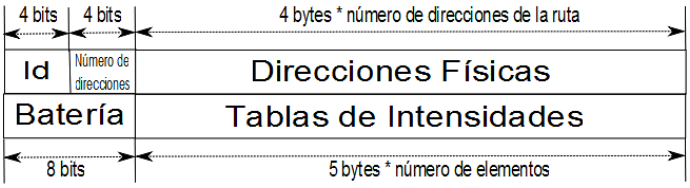


Figura 3.5: Formato del paquete de Contestación de Tablas de intensidades

En dicho paquete se deberá adjuntar la ruta que recibió del sumidero en la orden de descubrimiento, utilizando los 4 bits de relleno entre la ID y la ruta para indicar el número de direcciones que la compone. Posteriormente se adjuntará el nivel de batería del nodo en los 8 siguientes bits seguido de la tabla de intensidades.

La tabla de intensidades consta de todas las direcciones físicas de todos los nodos circundantes (4 bytes) del nodo emisor junto con la intensidad de señal (dBm) con que éste los percibe (1 byte). Por lo tanto cada nodo circundante supone 5 bytes más en la longitud del paquete, que se corresponde con una fila de la tabla de intensidades.

Esto además permite que el nodo sumidero registre aquellos nodos que estén alejados de él y que no los hubiera descubierto éste directamente. Así seguirá la fase de formación de la red hasta que no se registren nuevos nodos y a todos les haya enviado el sumidero una orden de descubrimiento y recibido éste a su vez un paquete de tablas de intensidades. La ruta para estos nodos nuevos pasará a través de aquellos nodos sensores que los hayan descubierto.

El envío de este paquete se resuelve con un acuse de recibo que se enviará punto a punto entre los nodos que conforman la ruta hasta el sumidero.

Paquete de Programación del macro-frame

Una vez terminada la etapa de formación de la red, formado el grafo y calculados los caminos mínimos se pasa a la etapa estándar. El nodo sumidero enviará periódicamente una señal de *wake-up* para sacar a los nodos sensores del estado de reposo y para sincronizarlos. Posteriormente se enviará un paquete de Programación del macro-frame (ID: 5) que se compone de marcos de dos direcciones físicas (emisor y receptor) que indican los nodos que participarán en cada frame.

El orden en que se utilizan los enlaces para cada marco temporal del macro-frame es algo que se abordará más adelante. Pero cabe decir que puesto que los paquetes de envío de mediciones son variables, debemos ponernos en el peor de los casos para calcular el máximo de mediciones que un nodo puede acumular por cada envío (frente al máximo de 255 bytes de longitud

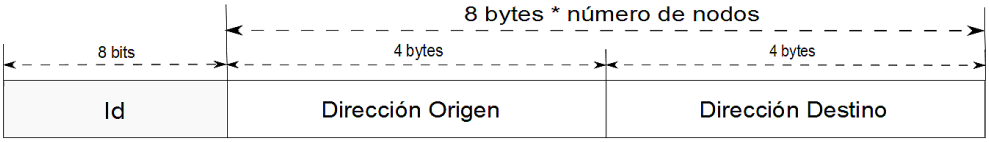


Figura 3.6: Formato del paquete de Programación del macro-frame

de paquete impuesto por las librerías del SimpliciTI para el CC1110). Ya que la programación del macro-frame se realiza desde los nodos de niveles inferiores hacia el sumidero, acumulando medidas de hijos a padres en el árbol.

Esto quiere decir que un nodo puede que no sea capaz de enviar todas las mediciones de los nodos que cuelgan de él directa o indirectamente en un solo paquete, por lo que habrá que utilizar varios marcos temporales para emitir repetidamente hasta que pueda enviar todas las mediciones que ha acumulado.

Por tanto habrán casos en los que haya más marcos temporales que nodos, porque alguno tendrá que ocupar más de uno para enviar sus medidas.

La duración de cada marco temporal es invariable y viene fijado por el equivalente a tres emisiones (que suponen el vencimiento de sus respectivos *timeouts*, y que a su vez depende de la velocidad de transmisión) de paquetes de medidas. Como ya hemos indicado el tiempo no es importante y el objetivo es que un error en una transmisión suponga la caída de un nodo y no un error puntual de comunicación.

Paquete de Envío de mediciones

Cada marco temporal programado por el sumidero supone el envío de uno de estos paquetes (ID: 6) entre dos nodos sensores utilizando un enlace ascendente del árbol. El envío del paquete se realiza siempre en cada marco, pero no siempre se enviará ninguna medición. Las mediciones, tanto la lectura de las galgas como las lecturas de control del RSSI o de la batería se enviarán siempre que se supere un umbral configurable. Por ello se hacen necesarios los flags que indicarán qué mediciones se adjuntan en el paquete.

El receptor acumulará el paquete, comprobará si es necesario actualizar el nivel de intensidad de la señal respecto al umbral (y adjuntarlo si se diera el caso) y le responderá con un acuse de recibo para el control de errores punto a punto.

El campo de flags de 4 bits se estructura como sigue:

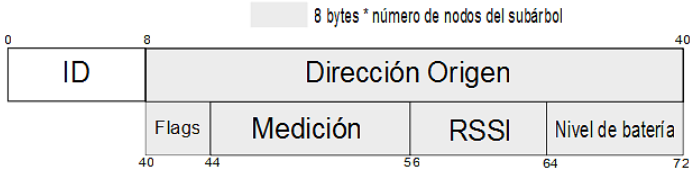


Figura 3.7: Formato del paquete de Envío de mediciones

Bit	Descripción
1	Flag de Error
2	Flag de medición (lectura de galgas)
3	Flag de nivel RSSI
4	Flag de nivel de batería

Tabla 3.2: Tabla de Flags de mediciones

En caso de que un nodo, durante uno de los marcos temporales del *macro-frame* no recibiera un paquete de mediciones que tuviera programado, anotaría la dirección física del nodo que no ha emitido su paquete y lo marcaría con el flag de error. Los flags de medición y batería los marca el emisor y el de nivel de RSSI lo marca el receptor si procediera. Por este motivo los nodos guardan las tablas de intensidades de la etapa de formación de la red donde tienen anotado el RSSI de todos los nodos adyacentes y lo actualizan si procediera.

Paquete de Actualización de configuración

La etapa estándar debe comenzar con este paquete (ID: 7) si desde la aplicación de monitorización se especifican alguno de los valores de umbral (en caso contrario se toman valores por defecto). A su vez, si durante el desarrollo de la etapa estándar se desea cambiar alguno de los umbrales que los nodos toman como referencia para enviar sus mediciones, se deberá enviar un paquete de este tipo.

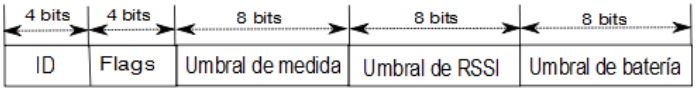


Figura 3.8: Formato del paquete de Actualización de configuración

Consta de tres campos opcionales dependiendo de qué parámetros se quiera modificar. Para indicar qué campos son los que contiene el paquete se incluye un campo de flags de 4 bits entre el id y los umbrales de estructura equivalente a los flags del paquete de envío de mediciones.

Bit	Descripción
1	Sin uso
2	Flag de umbral de medición
3	Flag de umbral de nivel RSSI
4	Flag de umbral de nivel de batería

Tabla 3.3: Tabla de Flags de umbrales

Los umbrales son en valor absoluto y representan la variación de la susodicha medición que se considera suficientemente relevante para enviar el dato al sumidero. El umbral de medición se toma en $\mu\epsilon$ (microdeformaciones[5]), el de la intensidad de señal es en dBm (decibelios por miliWatio) y el umbral de batería en mAh[6] (miliamperios hora).

Este paquete, como el de programación del *macro-frame*, viene precedido por una señal de *wake-up* para levantar a los nodos del modo reposo y prepararlos para la recepción del mensaje (excepto si el envío del paquete se produce antes del primer *macro-frame*). Una vez procesado los nodos volverán al estado de reposo.

Paquete de Acuse de Recibo

También conocido como *Acknoledgement* en inglés o abreviado como *ACK*, este mensaje se utiliza en muchos protocolos para confirmar que un mensaje ha llegado correctamente. En este caso se utiliza en varios mensajes durante las fases del protocolo en los enlaces punto a punto como se ha indicado en los paquetes anteriores.

El paquete de acuse de recibo consiste en un solo byte con todos sus bits a cero (ID: 0 + relleno).

3.2.3. Desarrollo por fases

Fase de formación de la red

El objetivo final de esta fase es la formación de un grafo que contenga todos los nodos de la red y sus respectivos enlaces entre ellos que indiquen la

[5]La deformación es el cambio en el tamaño o forma de un cuerpo debido a esfuerzos internos producidos por una o más fuerzas aplicadas sobre el mismo o la ocurrencia de dilatación térmica.

[6]Un amperio hora es una unidad de carga eléctrica y se abrevia como Ah. Indica la cantidad de carga eléctrica que pasa por los terminales de una batería, si esta proporciona una corriente eléctrica de 1 amperio durante 1 hora.

conectividad de la red. Sobre estos enlaces se calcularán los caminos mínimos que formarán el árbol sobre el que se programarán los *macro-frames*.

El nodo sumidero poseerá una estructura de datos que le permitirán alojar a los nodos sensores que descubra durante el desarrollo de la fase. Esta estructura no es más que una lista enlazada que contiene la dirección física del nodo y un puntero al nodo que descubrió al actual. Este puntero se utiliza para poder encaminar los paquetes hasta el sumidero mientras el árbol no este formado. La lista se inicia conteniendo al propio nodo sumidero con el puntero a su *descubridor* a nulo. A medida que descubra nuevos nodos los añadirá a la lista y la recorrerá demandando tablas de intensidades de las que a su vez podrá descubrir e insertar nuevos nodos en la lista. Así la fase termina cuando haya recorrido la lista entera sin descubrir nuevos nodos, lo que querrá decir que posee las tablas de intensidades de todos los nodos de la red y podrá formar el grafo.

Las tablas de intensidades son una estructura de datos que contiene las direcciones e intensidades de señal de todos los nodos adyacentes que un nodo haya percibido. Contiene la dirección del nodo que creó la tabla, su nivel de batería actual (que enviará al nodo sumidero) y la lista de filas de intensidades, que no son más que las direcciones de los nodos adyacentes con sus respectivas intensidades de señal.

Se darán más detalles de implementación más adelante en los anexos, pero sirva este adelanto para entender cómo registra el sumidero los nodos y las rutas durante la fase de formación de la red y qué son las tablas de intensidades que se envían al sumidero por parte de los nodos sensores.

Intercambio de paquetes durante la formación de la red

El registro de nuevos nodos se realiza por medio de los paquetes de descubrimiento. Un nodo que ya esté registrado en la lista (empezando por el propio sumidero) envía un paquete de descubrimiento. Este paquete es contestado por todos los nodos cercanos con el paquete de contestación al descubrimiento como explicamos en el desarrollo de los paquetes (retroceso exponencial binario y detección de portadora antes de enviar).

Con cada contestación que reciba el nodo emisor renovará el temporizador para que puedan registrarse nuevos nodos. Éste a su vez anota todos los nodos en su tabla local y la envía al sumidero encaminando el mensaje en caso necesario a través de la ruta que indicara la orden. Los nodos realizan esta acción a instancias del propio sumidero al recibir una orden de descubrimiento y deberán contestar con sus tablas de intensidades.

Como vemos durante esta fase los nodos sensores deberán estar perpetuamente en escucha ociosa del medio y contestando a todos los paquetes que

escuchen y que fueran dirigidos a ellos o sean de difusión. Los nodos se irán descubriendo desde los más cercanos al sumidero hasta los más alejados a medida que que se vayan registrando.

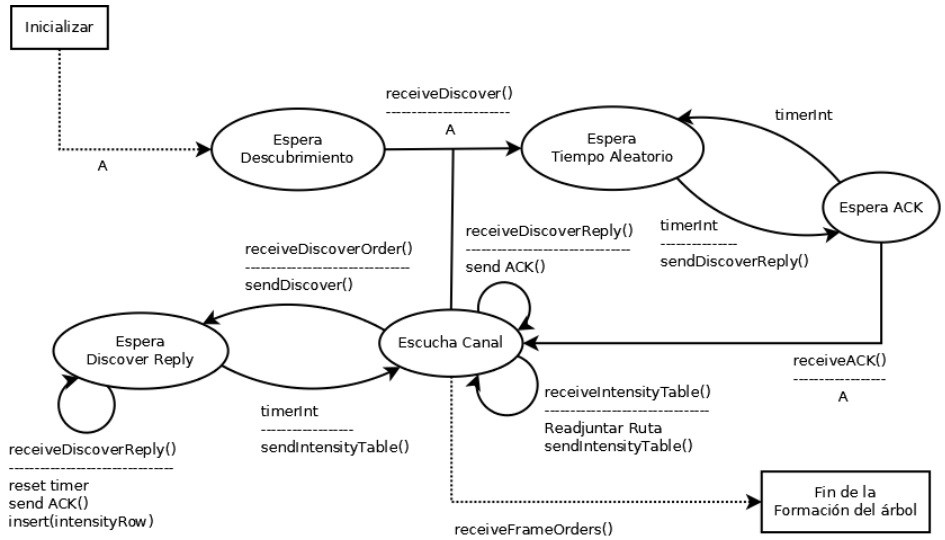


Figura 3.9: Esquema de comunicaciones de un nodo sensor durante la formación de la red

El nodo sumidero en primera instancia tanteará él mismo la red para registrar los primeros nodos cercanos para, posteriormente recorrer su lista buscando nuevos nodos mientras les envía ordenes de descubrimiento. Cuando posea las tablas de intensidades de todos los nodos registrados pasará a formar el grafo con todas las tablas de intensidades que ha recibido.

Fase de funcionamiento estándar

Una vez finalizada la formación de la red los nodos pueden empezar el modo normal de operación, que consiste en este caso en la programación periódica de los *macro-frames* para que las lecturas de las galgas de los nodos sensores lleguen a través del nodo sumidero a la estación de monitorización. El tiempo transcurrido entre un *macro-frame* y el siguiente será configurable para permitir que la aplicación se adapte a las necesidades de monitorización del entorno actual.

Durante un *macro-frame*, el primero de los marcos temporales se reserva para la toma de lecturas de las galgas por parte de todos los nodos de la red. Las lecturas por otra parte, como ya se ha indicado anteriormente, solo se enviarán si se ha superado un umbral absoluto también configurable.

Los nodos sensores dejan la escucha ociosa del medio propia de la fase anterior en el momento en el que reciben alguno de los dos tipo de paquete

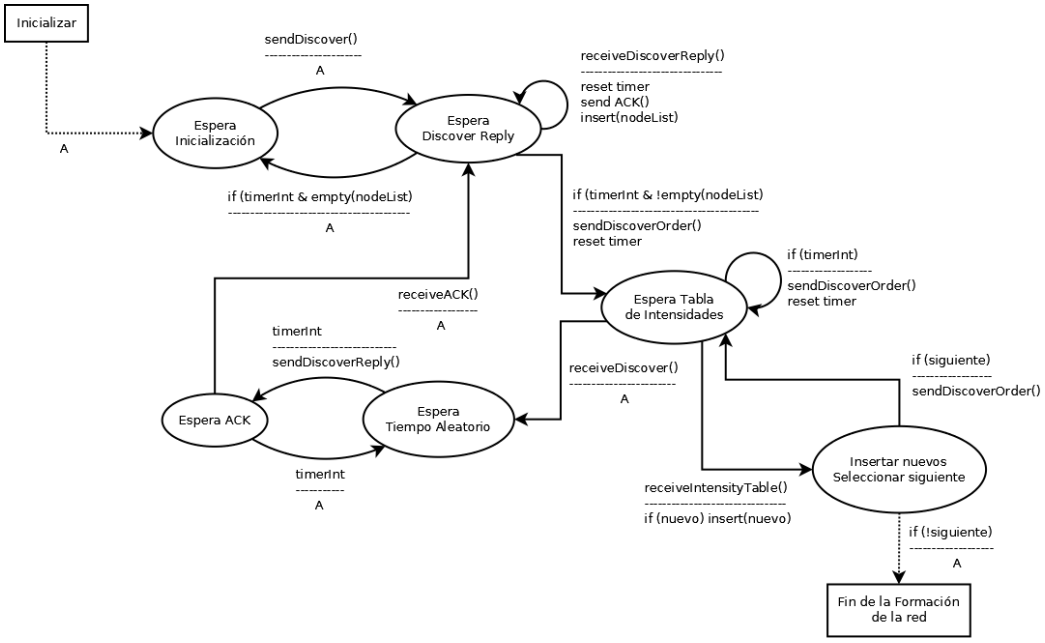


Figura 3.10: Esquema de comunicaciones del sumidero durante la formación de la red

pertenecientes a esta fase del protocolo. Éstos serán enviados por el sumidero inmediatamente después de haber formado el árbol de caminos mínimos.

Intercambio de paquetes durante el funcionamiento estándar

En esta fase solo participan dos paquetes, el de programación del *macro-frame* y el de envío de mediciones. En este caso el nodo sumidero tan solo envía periódicamente la programación y espera un tiempo proporcional al número de marcos programados[7] (contando con el reservado para la toma de medidas de los sensores), y cercionándose de que ha recibido todos los paquetes.

El resto de nodos recibirán un «bocinazo» por cada *macro-frame* que se vaya a programar. Recibirán la programación que es enviada por difusión a todos los nodos de la red y procesarán los marcos en los que participan, recibiendo o enviando.

Cuando tengan que recibir escucharan el medio durante el espacio de tiempo de un marco temporal completo y si lo reciben lo guardaran y contestarán con un acuse de recibo. Si no recibieran ningún paquete durante todo el marco temporal, anotarían la dirección física del nodo del que debían

[7] $T_{espera} = (2 + n_{marcos}) * T_{marco}$

haber recibido el paquete de medidas y marcan para él el flag de error. Al contrario, al enviar adjuntan sus medidas a las que hubiera podido recibir y las envían en un paquete al nodo que tengan programado como receptor. Cada marco temporal esta programado para tres reenvíos en caso de que venciera el *timeout* sin recibir su respectivo acuse de recibo.

El nodo sumidero enviará también los acuses de recibo correspondientes. Una vez recibidos todos los paquetes de medidas el sumidero las recorrerá en busca de nuevas medidas o paquetes con el flag de error, actuando en cada caso de forma adecuada al problema si debiera.

3.2.4. Formación del grafo

Una vez recogidas todas las tablas de intensidades el nodo sumidero pasa a formar el grafo con los datos que contienen. El grafo es otra de las estructuras que poseerá el sumidero en la que cada vértice representará un nodo terminal y los arcos o aristas representan las conexiones inalámbricas presentes. El grafo será ponderado y dirigido (con un par de aristas por enlace).

El grafo lo formará utilizando las tablas de intensidades de todos los nodos que debe haber recibido tras la etapa anterior. Éstas tablas, como se ha indicado, contienen el nivel de batería de todos los nodos en el momento del envío de sus tablas y las intensidades de señal con que perciben a los nodos cercanos asociadas a sus respectivas direcciones físicas. Datos necesarios todos ellos para formar el grafo y ponderar sus vértices.

Los nodos se distinguen en el grafo por unos identificadores ascendentes que empiezan en el 1, que corresponderá al vértice que representa al sumidero. Este identificador se utilizará entre otras cosas para evitar que se dupliquen vértices.

Al crear un enlace debe crearse a su vez el vértice al que apunta en caso de que este no existiera. Los enlaces se ponderan al crearse y un enlace entre dos nodos supone dos aristas uniendo a ambos nodos en ambos sentidos. Es importante procesar los dos vértices al crear un enlace ya que al ponderarlo se tiene en cuenta la batería de ambos nodos y las respectivas intensidades con que se perciben respectivamente. La fórmula utilizada para ponderar un enlace es la siguiente:

$$IndiceP = \alpha + \beta$$

$$Siendo \begin{cases} \alpha = Bateria_1 \cdot Bateria_2 - |Bateria_1 - Bateria_2| \\ \beta = RSSI_1 \cdot RSSI_2 - |RSSI_1 - RSSI_2| \end{cases}$$

Al final ambos arcos del enlace deben ponderarse igual (el resultado en ambos sentidos es el mismo ya que la resta es en valor absoluto) aunque en

instantes del recorrido distintos. Tanto los operandos como el resultado de la operación se consideran sin signo (el de la batería y el del RSSI) ya que el RSSI suele verse con valores negativos de dBm y se almacena como un entero con signo. Se busca una concordancia entre los índices α y β.

El cálculo se formula para penalizar los valores desiguales en los niveles de batería y RSSI, considerando por igual ambas medidas (α y β). Como podemos ver si los valores son similares la resta tenderá a cero y entonces la ponderación dependerá del resultado de la multiplicación.

También es fácil de observar en este caso que ante niveles que resulten en un mismo valor al multiplicarlos, se ponderará más positivamente aquella combinación que presente números más similares. Por ejemplo, si $RSSI_{11}$=2 y $RSSI_{12}$=4 por un lado, y $RSSI_{21}$=1 y $RSSI_{22}$=8 por otro (valores válidos para ilustrar el ejemplo solamente y que no se ajustan en ningún caso a la realidad), tenemos:

$$\beta_1 = 2 \cdot 4 - |2 - 4| = 8 - 2 = 6$$

$$\beta_2 = 1 \cdot 8 - |1 - 8| = 8 - 7 = 1$$

Con lo que queda claramente ilustrada la penalización ante valores desiguales que era el objetivo de la fórmula para la ponderación de arcos presentada.

3.2.5. Cálculo de los caminos mínimos

El cálculo de caminos mínimos una vez formado el grafo se realiza aplicando el algoritmo de Dijkstra[8] y formando un árbol de expansión[9] sobre el mismo grafo. Como vértice origen se toma al nodo sumidero y se calculan los caminos de todos lo vértices a éste usando un puntero al nodo al que deben realizan el envío.

El cálculo determina la forma del árbol y los niveles de los que constará. Cada vértice posee su nivel en el grafo como atributo y el nodo sumidero se guarda el nivel máximo del árbol para poder recorrerlo desde las hojas de los niveles inferiores. Además cada nodo, una vez terminado el proceso poseerá un puntero al siguiente nodo del mismo nivel[10] en el árbol.

[8] Algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo dirigido y con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

[9] Un árbol de expansión de un grafo G es una selección de aristas de G que forman un árbol que cubre todos los vértices. Esto es, cada vértice está en el árbol, pero no hay ciclos.

[10] Corresponde a detalles del algoritmo utilizado para la programación del *macro-frame* que se explicarán a continuación

Estas operaciones se realizan cada vez que se actualice alguno de los enlaces del árbol. En el caso en que dos nodos sufran un desgaste desigual (los nodos de niveles superiores sufren un mayor desgaste energético que los nodos de niveles inferiores) quizá proceda reencaminar alguno de los enlaces del árbol por otro de los enlaces presentes en el grafo y se deban determinar nuevamente los niveles del árbol y las listas de nodos por niveles.

En cualquier caso es imposible evitar por completo el desgaste desigual mencionado, pero el objetivo de la actualización dinámica del árbol pretende paliar en la medida de lo posible dicho efecto. De esta forma un nodo cercano al sumidero pero que haya sufrido un desgaste superior al resto en su nivel de batería puede ser sustituido por un nodo más alejado pero que posea un índice de batería significativamente superior a la diferencia entre el índice de RSSI de ambos nodos.

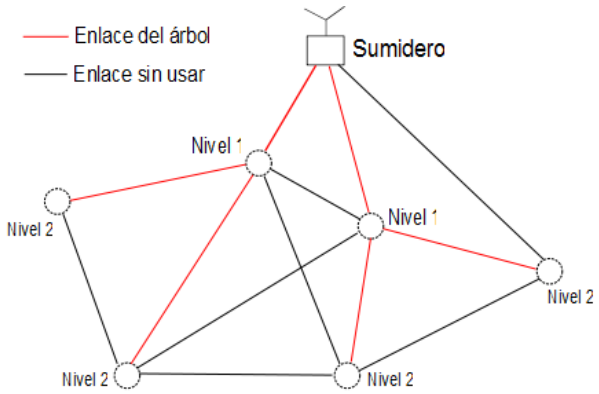


Figura 3.11: Ejemplo de grafo con caminos mínimos

3.2.6. Establecimiento de la programación de *macro-frames*

Condiciones previas

La programación del *macro-frame* debe entenderse como el establecimiento del orden en que serán utilizados los enlaces del árbol en los diversos espacios o marcos temporales dispuestos para hacer llegar las medidas al sumidero.

Para saber cómo se establece dicho orden se debe conocer las condiciones sobre las que se asientan el envío de las lecturas de los sensores. Condiciones establecidas para un uso eficiente de la red atendiendo a la topología en árbol de la misma.

Para simplificar el problema se da por supuesto el uso de un solo enlace en cada marco temporal. Se podría dar como ampliación algoritmos para la

planificación del uso de varios enlaces en nodos que no tengan visibilidad directa entre ellos o incluso del uso de varios canales de emisión para evitar solapamientos.

Los nodos sensores acumulan los paquetes de mediciones recibidos de sus hijos, y por ende de los subárboles que cuelgan de cada uno de ellos, y no podrán enviar información hasta haber recibido la información de sus hijos. Por tanto los nodos de niveles superiores serán más susceptibles de acumular medidas y realizar un uso mayor de su etapa de radio en el envío de éstas. Además, el uso de los enlaces deberá hacerse desde los nodos de niveles inferiores (las hojas del árbol) hacia los de niveles superiores, siendo estos los último en participar en el *macro-frame*.

El paquete de envío de mediciones tiene una longitud máxima de:

$$L = 8 * n + 1 \; bytes$$

Donde n es el número de nodos que acumula el nodo actual (ver la descripción del paquete en la sección 3.2.2, Id 6). Como ya hemos indicado la capa física que nos ofrece el CC1110 puede contener un máximo de 255 bytes, por lo que despejando 'n' de la ecuación anterior tenemos que podríamos acumular hasta 30 nodos en un paquete de envío de mediciones en el peor de los casos.

En la práctica la librería de SimpliciTI usada para las pruebas del proyecto y sobre la que en realidad se encapsulan los datos del protocolo da por defecto tamaños de paquete no superiores a 50 bytes.

Además hay que llegar a un compromiso con el tiempo de cada marco temporal, teniendo en cuenta la tasa de datos[11] y la cantidad de datos máximos que fijamos por emisión, multiplicado por el número de reenvíos de cada paquete por marco (se pretende que haya tiempo para tres reenvíos). En este caso para las pruebas se ha llegado a una solución de compromiso entre los factores que hemos descrito y se opta por 6 medidas acumuladas que dará como máximo tamaños de paquete de hasta 49 bytes.

Estrategias para la programación de macro-frames

Para el recorrido se pueden optar por varias estrategias, quedando por determinar cuál es la más eficiente. Las consideraciones iniciales y la forma en que se ha formado el árbol dejan abierta la opción del recorrido por niveles o por ramas como primeros acercamientos a la solución del problema.

El recorrido por niveles se inicia en los nodos del nivel más inferior (es decir, todas las hojas de ese nivel) y, como su propio nombre indica, realiza

[11] Velocidad de transferencia de datos

el recorrido por todos los nodos de ese nivel haciéndoles envíar su medida al nodo padre (aquel al que apunta después del cálculo de caminos mínimos). Una vez terminado el recorrido con los de ese nivel se seguiría con los nodos del siguiente nivel, que acumularán si procede sus medidas a las recibidas y la enviarán a su vez a su padre, nodo de nivel superior en el árbol. Así sucesivamente hasta que no queden nodos.

El recorrido en este caso se hace desde el primer nodo de ese nivel que se encuentre en el árbol, que resultará ser el nodo del nivel actual con el identificador más bajo. Aunque no es imprescindible, se supone que los nodos contarán con punteros al siguiente nodo de nivel para ahorrar tiempo de procesamiento (cargando así a la función de cálculo de caminos mínimos con la tarea de dar valor a éstos punteros).

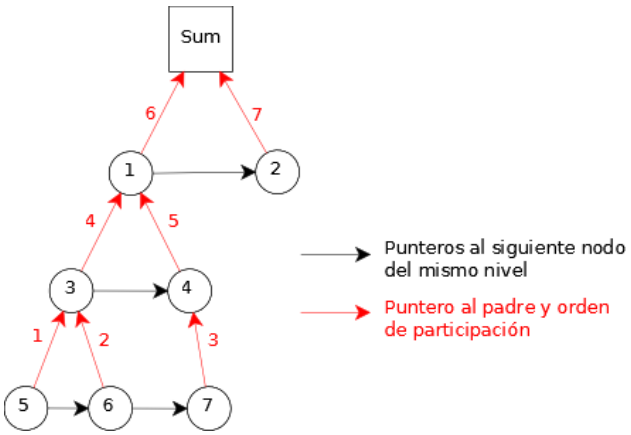


Figura 3.12: Ejemplo de recorrido por niveles en árbol

En ocasiones los punteros del mismo nivel puede que no estén ordenados con respecto a su identificador y en la programación no se procesen a los hijos de un nodo por orden como en la figura 3.11. En este caso los nodos anotarán en qué marcos deben recibir paquetes de medidas y en caso de que no sean consecutivos esperar ociosos hasta la recepción de todos los paquetes programados. El nodo solo esperará en modo de ahorro de energía antes y/o después de su participación en el *macro-frame* (Fig. 3.12).

Por otra parte el recorrido por ramas empieza, en lo referente a los envíos de mediciones, en las hojas y sigue hacia arriba en el árbol hasta llegar al sumidero. Para que la acumulación se produzca se debe comprobar que antes de enviar un paquete de medidas se hayan recibido todos los paquetes de medidas de los hijos. Así, al procesar un nodo se recorre su lista de nodos adyacentes para comprobar si alguno de ellos es su hijo y aún no ha sido procesado, y así recursivamente para procesar a todo el subárbol por cada nodo que se encuentre.

Este proceso recursivo se inicia desde el sumidero. La estrategia en este

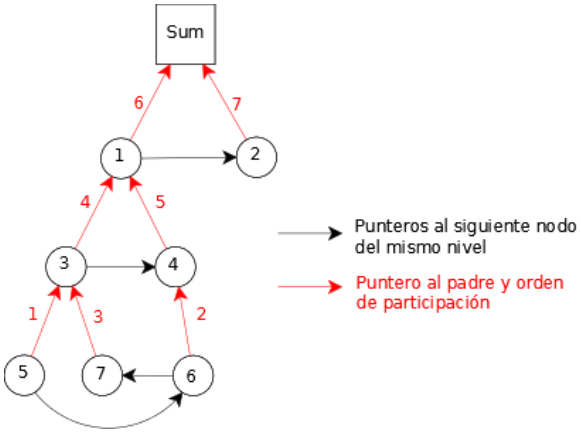


Figura 3.13: Ejemplo de recorrido por niveles en árbol con identificadores desordenados

caso para el recorrido por ramas es que el caso base sea encontrar una hoja o que los nodos hijos que cuelgan del actual ya estén procesados. El recorrido busca el caso base, y cuando lo encuentra lo procesa, lo añade a la programación, sube de nivel y vuelve a comprobar si el nodo actual cumple el caso base. Así, como en el caso anterior, sucesivamente hasta que no quedan nodos sin procesar y se sale de la función.

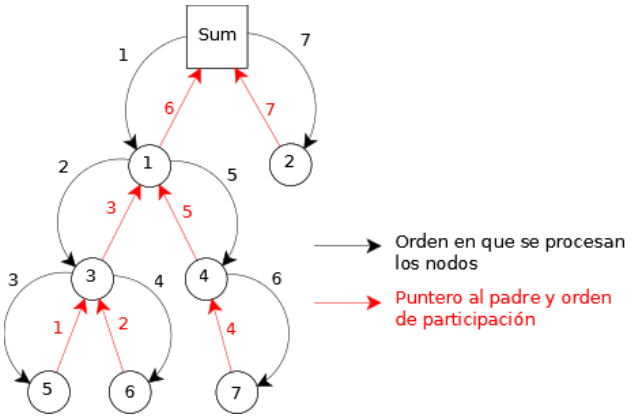


Figura 3.14: Ejemplo de recorrido por ramas en árbol

Una de las ventajas de este sistema es la programación de *macro-frames* de emergencia (como veremos más adelante) en la que tan solo se debe indicar el nuevo nodo raíz para recorrer el subárbol que cuelga de aquel que produjo el error.

En ambos casos se debe llevar una cuenta del número de nodos acumulados para programar la cantidad de marcos temporales que necesitará para enviar todas las medidas al padre. El resultado, sea cual sea la estrategia seguida, se guardará en un array para no tener que programar el *macro-*

frame cada vez y hacerlo solo cuando se reprogramen los caminos mínimos.

La estrategia usada para programar el *macro-frame* determina levemente la estrategia de los nodos al procesar la programación. En general un nodo dormirá hasta su primera participación. Entre recepciones puede haber un tiempo de espera de uno o varios marcos temporales, el espacio de tiempo indicado lo pasará en espera ociosa hasta que vuelva a participar. Entre la última recepción y el primer (y sucesivos si los hubiere) envíos puede haber o no una parada dependiendo de la estrategia. Después del último envío volverá a dormir hasta el siguiente «bocinazo».

Estos son los dos tipos de algoritmo diseñados para la programación de *macro-frames*, teniendo en cuenta que se usa un solo enlace por marco temporal.

3.2.7. Toma de medidas

El primero de los marcos temporales de un *macro-frame* se reserva para la toma de lecturas de las galgas extensiométricas de cada nodo. Se toman cinco muestras descartando la mayor y la menor de ellas. De las tres medidas restantes se calcula la media aritmética[12] siendo ésta la medida que se tomará como medida de las galgas para todo el *macro-frame*.

De esta forma eliminamos picos o falsas medidas producidas por cambios ambientales o errores de lectura puntuales debido a problemas mecánicos.

La toma de medida del nivel de batería se hace en el mismo marco temporal. Esta toma de medidas no se ha implementado en el desarrollo del proyecto, se utilizaban en su lugar medidas de prueba. Los nodos, durante todas las pruebas en laboratorio se han realizado alimentando los nodos a través del puerto USB (con lo que su nivel de batería era siempre el máximo).

El indicador de potencia de señal de recepción se toma en cada recepción que lo requiera.

Las medidas se discriminarán en el momento de adjuntarlas al paquete para enviarlo, teniendo en cuenta en cada caso el valor de umbral actual para cada una de ellas.

3.2.8. Solucionando errores en la red

El sumidero detecta un error si al finalizar un *macro-frame* y repasar paquetes de mediciones detecta un campo de flags con el flag de error activo.

[12]La media aritmética de un conjunto finito de números es igual a la suma de todos sus valores dividida entre el número de sumandos

Esto quiere decir que un nodo esperaba un paquete de mediciones en uno de los marcos temporales que nunca se produjo y lo marco con el flag de error. Así faltarían no solo las medidas del nodo que falló en su emisión sino todas las medidas del subárbol que cuelga de dicho nodo.

En este caso la primera medida que se debe tomar es programar un segundo *macro-frame* en el que solo participen los nodos que no han podido enviar sus medidas. Es inevitable, por otra parte, que los nodos que hay en la ruta entre el subárbol que falló y el nodo sumidero adjunten de nuevo sus medidas. Este *macro-frame* especial se lanzará después de haber procesado las medidas del resto de nodos.

La estrategia para establecer estos conjuntos de marcos para el subárbol debe ser la misma que la utilizada para el *macro-frame*. En el caso de que el recorrido sea por ramas la estrategia es la misma solo que en lugar de partir desde el nodo sumidero, se parte desde el nodo que produjo el fallo y luego se une este nodo directamente al sumidero por la ruta de caminos mínimos.

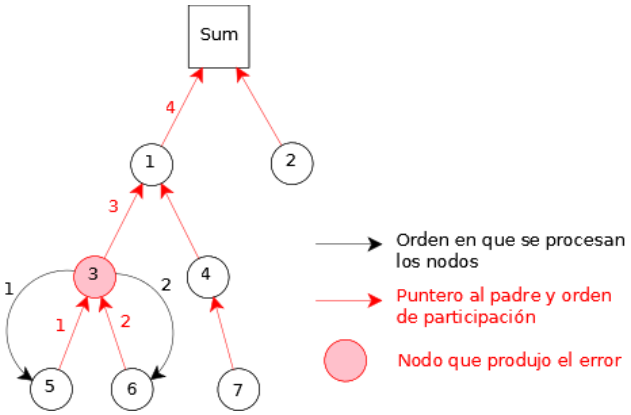


Figura 3.15: Ejemplo programación de *macro-frame* de emergencia con recorrido por ramas

En el caso del recorrido por niveles hay que comprobar para cada nodo si pertenece o no al subárbol del que se pretenden recoger las medidas. El recorrido de cada nivel se hará completo excepto en el nivel del nodo que falló. Desde el nodo que falló hasta el sumidero, como en el caso anterior, simplemente se sigue la ruta de caminos mínimos hacia el sumidero.

Si con este segundo *macro-frame* se solventa el problema y se reciben las medidas que faltaban, en ese caso simplemente se procesan y se actualizan los caminos mínimos si procediera y se espera al siguiente *macro-frame*. Por contra, si el problema persiste se debe descartar el nodo de la red eliminando el vértice del grafo y con él todos los enlaces (aquellos arcos que vayan hacia o provengan de otros nodos) adyacentes. Una vez descartados los enlaces al nodo que falla se calculan de nuevo los caminos mínimos para intentar encaminar las medidas del subárbol que cuelga del nodo descartado por otro

de sus enlaces y se programa un nuevo *macro-frame* completo para intentar hacer llegar sus medidas al sumidero.

Si alguno de los nodos sensores deja de poder alcanzar al sumidero, es decir que se queda sin enlaces utilizables que le unan al resto de la red, la estación de monitorización avisará al usuario de la existencia del nodo huérfano. El sumidero en este caso desmontará la red para formar nuevos enlaces y permitir que el nuevo nodo se registre a través de alguno de ellos. En caso de que ni siquiera esta solución sirva y el nodo huérfano no se registre en la nueva red simplemente se asumirá la pérdida del nodo huérfano, debiendo recaer en el usuario la responsabilidad de reemplazar algún nodo sensor para que deje de estar huérfano.

Por otro lado cabe añadir una posible fuente de errores en la formación del árbol. En principio no se ha establecido un límite al número de reenvíos en ningún paquete de la formación de la red. Esto requeriría un estudio más a fondo de qué paquetes pueden resultar en un bucle infinito de envíos a un nodo que ha dejado de escuchar. No es una situación habitual y por eso no se ha contemplado excepto en el caso de la contestación al descubrimiento.

Un nodo se puede quedar atascado en el envío de contestaciones al descubrimiento a un nodo si en algún error puntual su recepción no ha sido posible y el nodo receptor ya ha enviado sus tablas de intensidades. Es por esto que se ha añadido una guarda por la que si un nodo recibe un paquete de contestación al descubrimiento y no esta ya formando su tabla de intensidades simplemente enviará un acuse de recibo para sacar al nodo del bucle de envíos (contemplado en la figura 3.9).

A continuación podemos ver la imagen 3.16 que intenta aclarar la situación relatada anteriormente:

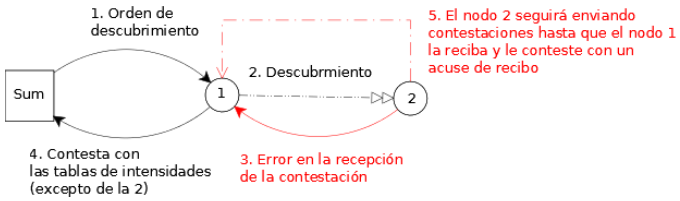


Figura 3.16: Ilustración de error durante la formación de la red

El resto de posibles errores en la red se asumen y solucionan en su caso con los *timeouts*, los acuses de recibo y los CRCs[13] que ofrecen las librerías del SimpliciTI (y el propio microcontrolador), es decir, las técnicas típicas de control de errores y de flujo de la capa de acceso al medio.

[13]Comprobación de redundancia cíclica, pueden ser usados como suma de verificación para detectar la alteración de datos durante su transmisión o almacenamiento

Capítulo 4

Entorno de trabajo

4.1. Microcontrolador

El nodo **WSNVAL** utilizado para el desarrollo y las pruebas del proyecto cuenta con un Microcontrolador CC1110. Entre las características principales podemos destacar:

- Núcleo 8051 que ofrece un elevado rendimiento a cambio de un bajo consumo
- Funcionamiento en las bandas de frecuencia de: 300-348 MHz, 400-464 MHz u 800-928 MHz.
- Tasa de envío de datos programable de hasta 500 kbps.
- 32KB de memoria flash programable.
- 4KB de memoria SRAM.
- Dos interfacez USART (*Universal Synchronous/Asynchronous Receiver/Transmitter*) para la comunicación con dispositivos serie.
- Un temporizador de 16 bits, tres de 8 bits y uno de 31 bits que funciona en varios modo de bajo consumo (*sleep timer*).
- Bajo consumo energético (22 mA en la recepción y 31mA en la transmisión con el MCU funcionando a 26 MHz).
- Potencia de emisión programable de hasta 10 dBm para todas las frecuencias soportadas.
- Hasta 0,6 μA de consumo en su modo de mínimo consumo (PM3).

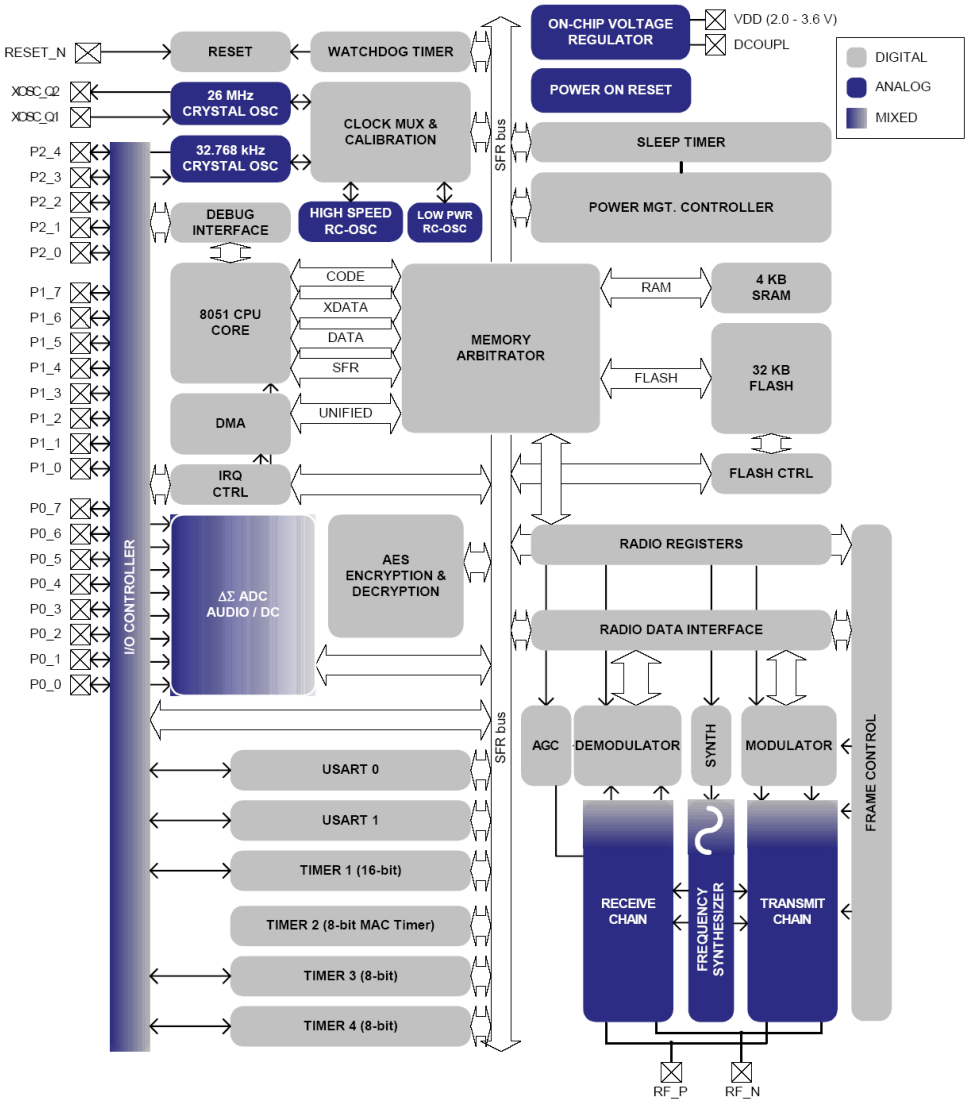


Figura 4.1: Arquitectura y asignación del patillaje del CC1110

De entre estas destacan las que recibieron un trato más de cerca en el ámbito del proyecto. En la fase de desarrollo del protocolo algunas características se mostraron de gran utilidad para ajustar el funcionamiento al descrito en la etapa de diseño teórico del mismo. Algunas incluso son indispensables, como los temporizadores o los modos de bajo consumo para permitir reducir el gasto energético en los periodos que el nodo sensor pasa ocioso.

4.1.1. CPU y Memoria

El núcleo de la CPU es una versión mejorada del estándar 8051, con el mismo juego de instrucciones que el estándar, pero se ejecutan más velozmente debido entre otros al decremento de los ciclos por instrucción de ésta versión del núcleo.

Además de las mejoras en velocidad también posee algunas mejoras arquitectales, sin perder compatibilidad con el estándar industrial del 8051, excepto en los casos en los que en el código existan bucles de temporización que pueden requerir cierta modificación.

Con lo que respecta a la memoria, la CPU del 8051 posee cuatro espacios bien diferenciados:

- Código: un espacio de memoria de solo lectura de 16 bits para memoria de programa.

- Datos: un espacio de memoria de lectura y escritura de 8 bits, que puede ser directamente o indirectamente accesible por una instrucción de CPU. Los 128 bytes de menor peso del espacio de memoria pueden ser direccionados de ambas formas, mientras que los restantes 128 bytes solo puede ser direccionados indirectamente.

- XDATA: un espacio de memoria de lectura y escritura de 16 bits a la cual acceder requiere de 4 a 5 ciclos de instrucción de la CPU. El acceso a este espacio de la memoria es más lento que al espacio de datos.

- Registros de Funciones Especiales (SFR - *Special Function Registers*): un espacio de memoria de lectura y escritura de 7 bits que puede ser direccionada por medio de una simple instrucción de la CPU. Para los registro cuya dirección sea múltiplo de ocho, cada bit es direccionable también individualmente.

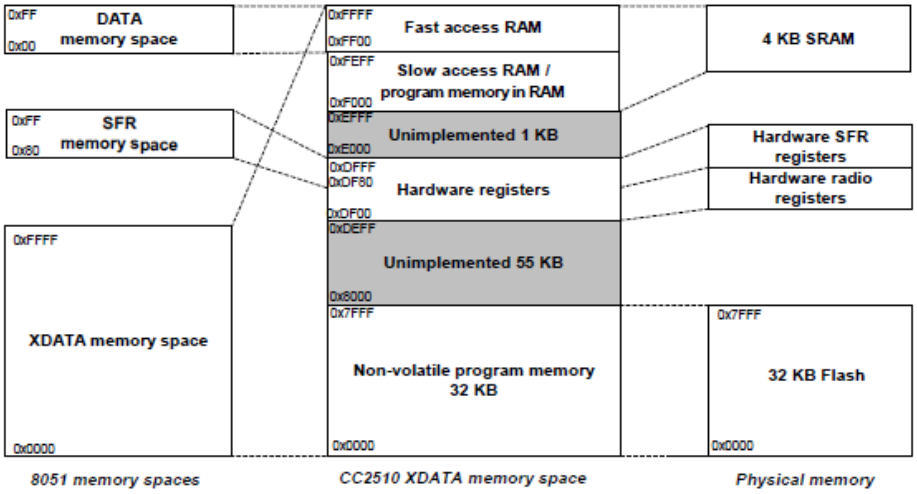


Figura 4.2: Espacio de memoria XDATA del CC1110

4.1.2. Interfaz de depuración

La interfaz de depuración ofrece dicha depuración por medio de un sistema propietario consistente en dos cables de transmisión serie. A través de esta interfaz es posible realizar un borrado completo de la memoria flash, controlar qué osciladores están habilitados, iniciar y detener la ejecución del programa que carga el usuario, ejecutar instrucciones suministradas por el núcleo del 8051, establecer puntos de interrupción y ejecución paso a paso de las instrucciones.

Mediante el uso de éstas técnicas resulta posible llevar a cabo la depuración del código que se cargue sobre el dispositivo.

4.1.3. Control de Potencia

El CC1110 posee cuantro modos de potencia con distintas características que permiten asegurar un mínimo consumo adaptado a las exigencias de cada situación. Los modos son: PM0, PM1, PM2 y PM3. PM0 es el modo activo mientras que el resto son modos de bajo consumo, siendo PM3 el de menor consumo de potencia.

Los modos de operación a (ultra) bajo consumo se consiguen cortando el suministro de energía a los módulos para evitar consumo estático y bloqueando los relojes para reducir el consumo dinámico.

PM0

Este modo es el modo activo, en el que todas las funciones de operación y los periféricos del MCU están activos.

El consumo en este modo con la etapa de radio activa puede llegar a los 31 mA cuando la radio está en modo de transmisión.

PM1

En este modo los osciladores de alta velocidad están apagados, deteniendo a su vez la CPU y los periféricos. El regulador de voltaje permanece encendido, asi como el oscilador activo es el de 32,768/34 kHz, las interrupciones externas o los periféricos del *sleep timer*.

Cuando el dispositivo pasa de PM1 a PM0 los osciladores de alta velocidad se inician.

Este modo de ahorro de energía es indicado cuando el tiempo hasta la señal que lo despierte sea supuesto corto ya que se usa una secuencia de entrada y salida del modo rápida.

El consumo ronda tipicamente los 300 μA en este modo.

PM2

PM2 es el segundo modo con la menor consumición energética de las disponibles. Se usa el mismo oscilador que el PM1 y los periféricos del *sleep timer* siguen activos. El resto de circuitos internos se apagan, incluido el regulador de voltaje.

Este modo es indicado cuando se espera un tiempo relativamente largo hasta el momento en que se espere la señal de despertado, ya que la secuencia de entrada y salida del modo, a diferencia del PM1, es costosa y lleva un tiempo relativamente elevado. Este modo se usa típicamente con el *sleep timer* como evento de despertado.

El consumo ronda tipicamente los 0,8 μA en este modo.

PM3

PM3 es indicado cuando se busca el modo con el menor consumo energético. Este modo detiene todos los circuitos internos junto con el regulador de voltaje y todos los osciladores.

El encendido al reiniciar y las interrupciones externas son las unicas funciones operativas. Así que solo una interrupción externa puede despertar a un

nodo de este modo. Los contenidos de la RAM y los registros son preservados. Se usa la misma secuencia de entrada y salida que en el PM2.

El consumo ronda tipicamente los 0,6 μA en este modo.

4.1.4. Temporizadores

Sleep Timer

Este temporizador funciona a baja potencia usando el oscilador de cristal de 32,768 kHz o el orcilador RC de la misma frecuencia como periodo. Este temporizador funciona de forma continuada en todos los modos de potencia excepto en PM3. Se puede configurar dentro de una amplia gama de resoluciones para llegar a un compromiso ajustado entre la resolución y el periodo.

Su uso típico es como señal de despertado en ciertos modos de ahorro de energía.

Temporizador 1

Este temporizador posee un contador de 16 bits con tres canales con un valor de comparación de 16 bits. Se puede usar para capturar la cadencia de los flancos del reloj como contador.

El prescalar permite dividir el periodo entre 1, 8, 32 o 128.

Los modos de funcionamiento son:

Free-running

En este modo de operación el contador empeiza en 0x0000 y se incrementa en cada flanco de subida. Cuando el contado alcanza 0xFFFF el contador se carga de nuevo con el valor 0x0000 y continúa incrementando su valor. Cada vez que esto ocurre se activa el flag de interrupción.

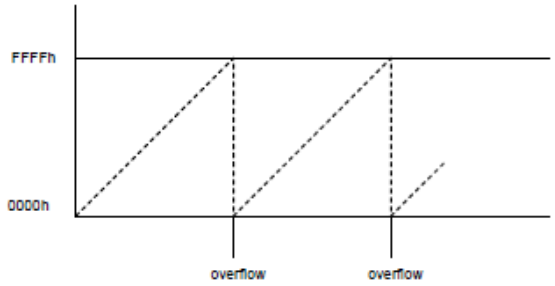


Figura 4.3: Modo *Free-running* del temporizador 1

Módulo

Este modo es similar al modo *free-running*, pero la interrupción y la vuelta del contador a 0x0000 no se producen en 0xFFFF sino cuando el valor coincide con el contenido en un registro del microcontrolador.

Up/Down

Este modo empieza con el contador a 0x0000 y se incrementa hasta su valor coincide, como en el caso anterior, con un registro del microcontrolador, y entonces se decrementa de nuevo hasta 0x0000.

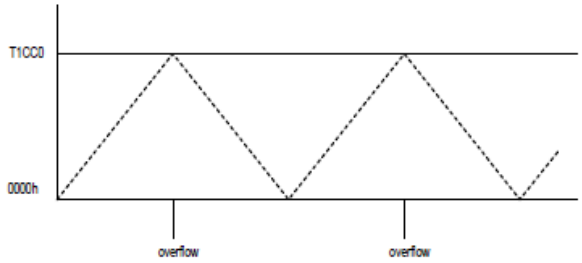


Figura 4.4: Modo *Up/down* del temporizador 1

Temporizador 2

También conocido como temporizador MAC, diseñado especialmente para soportar protocolos software temporizados. El periodo es configurable y posee un rango de prescalar de 18 bits.

Temporizadores 3 y 4

Estos temporizadores son de 8 bits de periodo, con prescalares configurables y un canal configurable con un comparador de 8 bits. Su uso es similar al del temporizador 1, incluidos los modos de funcionamiento.

Los prescalares dividen la frecuencia entre 1, 2, 4, 8, 16, 32, 64 o 128 (permitiendo ajusta el periodo deseado).

4.1.5. Radio

En la figura 4.5 vemos el diagrama del módulo de radio del microcontrolador.

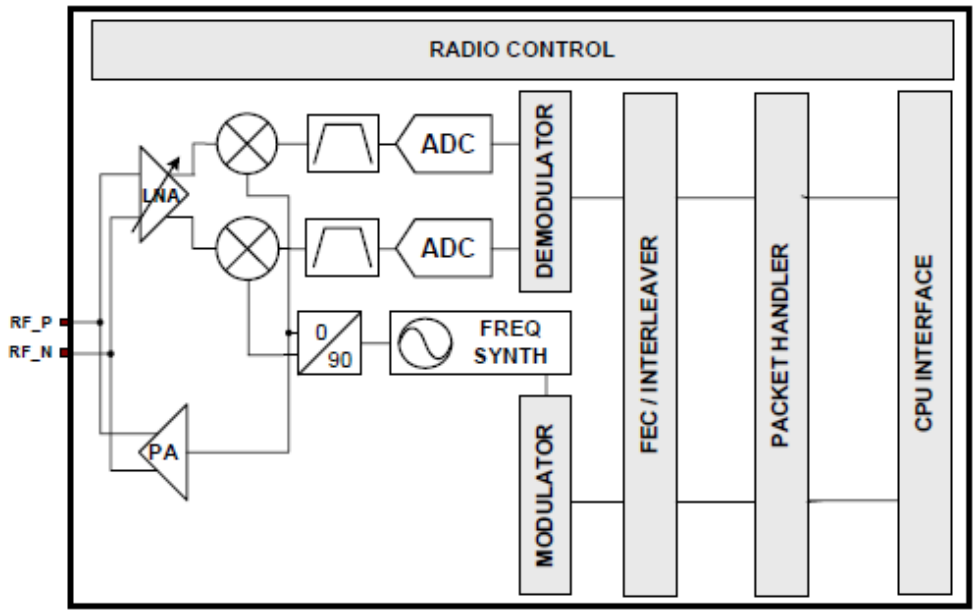


Figura 4.5: Módulo de Radio del CC1110

El microcontrolador cuenta con un receptor de frecuencias intermedias y bajas. La señal de radio-frecuencia es amplificada y convertida en cuadratura a frecuencia intermedia. En dicha frecuencia, las señales son digitalizadas por el convertidor analógico-digital. El resto del proceso de filtrado y demodulación es realizado digitalmente.

La parte transmisora del CC1110 se basa en la síntesis directa de la radio frecuencia. El sintetizador de frecuencia incluye un integrado en chip que realiza la modulación (en fase o cuadratura).

Los parámetros de la radio son ampliamente configurables, en frecuencia, canal, incluso el método de modulación. La configuración se realiza a través de ciertos registros del propio microcontrolador.

Las interrupciones también se manejan a través de registros del microcontrolador. Permite establecer una máscara y capturar distintos tipos de interrupciones para ajustar al máximo el comportamiento del dispositivo ante acciones específicas de la radio.

El cálculo del CRC, la estimación del RSSI o la detección de portadora son funciones que las realiza el módulo de radio y se comprueban a través de registros del CC1110.

El formato del paquete del CC1110 ya se detalló en el Capítulo 3, Sección 2.1.1.

4.1.6. USART

El microcontrolador cuenta de dos USART para la transmisión de datos serie mediante los puerto de E/S de propósito general. Las USART permiten transmitir en modo UART o SPI. En el modo UART permiten transmitir datos en serie de forma asíncrona mientras que en el modo SPI permiten hacer uso del estandar SPI para la transmisión de datos.

Las USART permiten configurar diversos parámetros de transmisión como la paridad par o impar, el orden de los bits (MSB o LSB), la velocidad de transmision, los bits de parada e inicio, etc.

Para configurar y manejar las USART se utilizan los registros UxGCR, UxUCR y UxCSR, donde x puede ser 0 o 1 dependiendo de la USART que se desee utilizar, mientras que para transmitir y recibir datos se utiliza el registro U0DBUF.

Las USART permiten dos alternativas para el conjunto de puertos de E/S usados para el modo UART y otros dos para el modo SPI de forma que se usa un juego distinto de puertos dependiendo de la USART utilizada, el modo en el que funciones y la alternativa de puertos elegida. Las alternativas de puertos elegida se configura en el registro PERCFG.

Las USART además incorporan mecanismos de interrupciones de forma que éstas se pueden usar para activar rutinas ante la recepción de datos.

4.2. Tarjeta de adaptación de sensores

Como hemos dicho anteriormente, nuestra red de sensores tiene como objetivo medir las deformaciones que sufre el material sobre el que se situará el nodo. Para medir éstas deformaciones nos hemos decantado por el uso de galgas extensiométricas. La señal de éstos sensores debemos llevarla al microcontrolador para que éste la procese y la envíe posteriormente al nodo sumidero a través de la red de nodos. Sin embargo, éstos sensores son unas simples resistencias que varian su valor en función de la deformación que sufren, por lo que hay que diseñar la electrónica necesaria para generar una señal a partir de la variación de resistencias de las galgas que sea adecuada para aplicarla en las entradas de E/S del microcontrolador

Para realizar todo ése proceso de adaptación que transformará la escasa variación de resistencia de las galgas en una señal digital apta para los puertos de E/S del microcontrolador se diseñará un circuito electrónico de adaptación. Dicho proceso queda resumido en la figura 4.6.

En primer lugar emplearemos un puente Wheatstone (explicado en el capitulo 5) que convertirá la variación de resistencia en una pequeña tensión

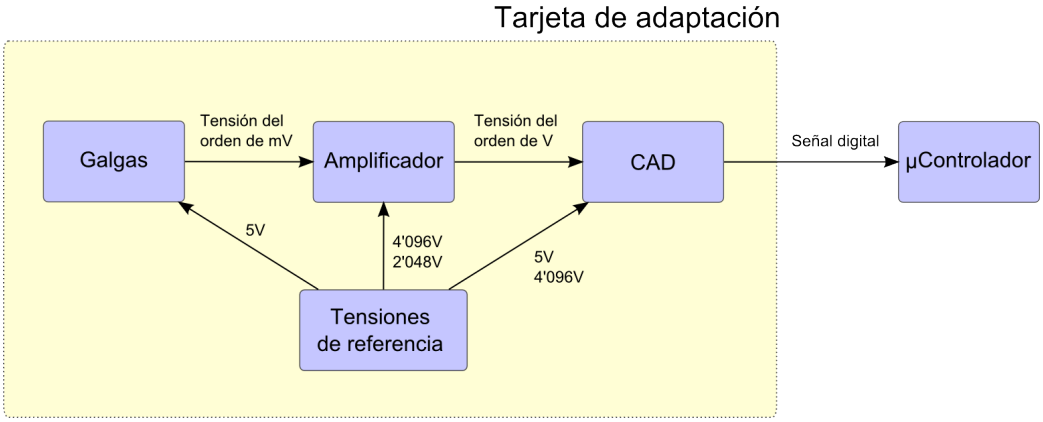


Figura 4.6: Proceso de adaptación de la señal al microcontrolador

diferencial del orden de milivoltios. Posteriormente ésta señal se aplicará a un dispositivo que amplificará dicha señal a tensiones del orden de pocos voltios. Ésta señal amplificada la inyectaremos a un conversor analógico-digital que convertirá la señal analógica proveniente de la salida del amplificador a una señal digital apta ya para aplicarla directamente a la entrada del microcontrolador.

Ésta tarjeta fisicamente será una placa de circuito impreso independiente a la que implementa el nodo la cual se conectará a éste a través del conector IDC 9x2 del que dispone la PCB[1] del nodo. A través de éste conector circularán las señales necesarias para controlar la electrónica de la tarjeta de adaptación (señal CLOCK y CS del conversor analógico digital) y la señal digital correpondiente a las medidas que realizan las galgas extensiométricas. De ésta forma se establecerá físicamente el flujo de información entre el microcontrolador y la tarjeta de adaptación.

4.3. Entorno IAR Embedded Workbench

Para dotar a los nodos del comportamiento que se espera de ellos debemos programar los microcontroladores para definir mediante código dicho comportamiento. Para facilitarnos la tarea haremos uso de un entorno de desarrollo que nos provea de las herramientas necesarias para hacerlo. En nuestro caso nos decantamos por el IDE[2] de IAR Embedded Workbench (en adelante IEW) de IAR Systems.

IEW es un entorno de desarrollo creado por IAR Systems. IEW propor-

[1]Printed Circuit Board (Placa de circuito impreso)
[2]Integrated Depelopment Environment

ciona un entorno completo para la programación y desarrollo en diversos tipos de microcontroladores. IEW dispone de diferentes versiones para diferentes plataformas de sistemas empotrados. En concreto nosotros usaremos la versión para la arquitectura 8051 ya que es la arquitectura con la que está implementado el microcontrolador que incorpora el CC1110 que está montado en los sensores de WSNVAL.

4.3.1. Interfaz Gráfica

Una vez instalado IEW, al ejecutarlo nos aparece la interfaz gráfica sobre la que trabajaremos.

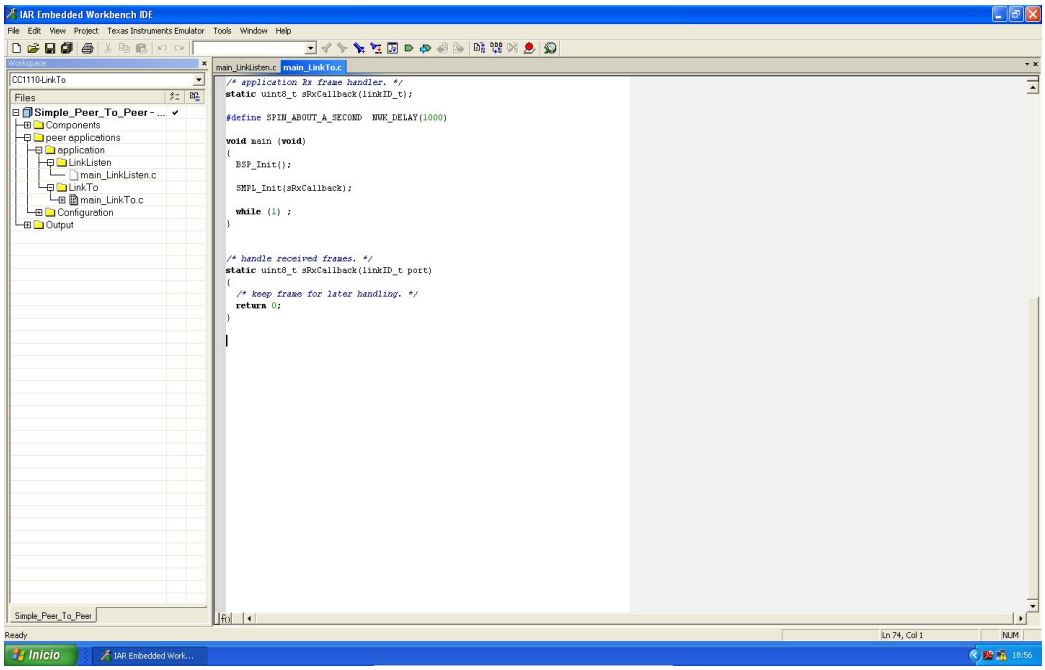


Figura 4.7: Espacio de trabajo del IEW

En ella distinguimos varias zonas del espacio de trabajo.

- En la parte izquierda de la ventana está la estructura de carpetas del proyecto. No se corresponde necesariamente con la estructura de carpetas del proyecto en disco. Al crear un proyecto éste se encuentra vacío, es el propio usuario el que crea éstas carpetas dentro del proyecto sin relación alguna con las carpetas que puedan haber en disco, es decir, ésas carpetas solo existen dentro del proyecto del IAR.

- En la parte central se encuentra el editor de código fuente

- En la parte superior tenemos la barra de menús y de herramientas. Cabe destacar la utilidad del último botón. Éste boton es el que inicia la compilación del proyecto y posterior y automáticamente la programación del microcontrolador.

Una vez tenemos un código listo para compilar y cargar en el microcontrolador ordenamos dicha acción mediante el último botón de la derecha de la barra de herramientas. Automáticamente IEW compilará todo el proyecto y realizará la carga del código ejecutable en el microcontrolador del nodo que tengamos conectado al PC como muestra la figura 4.8.

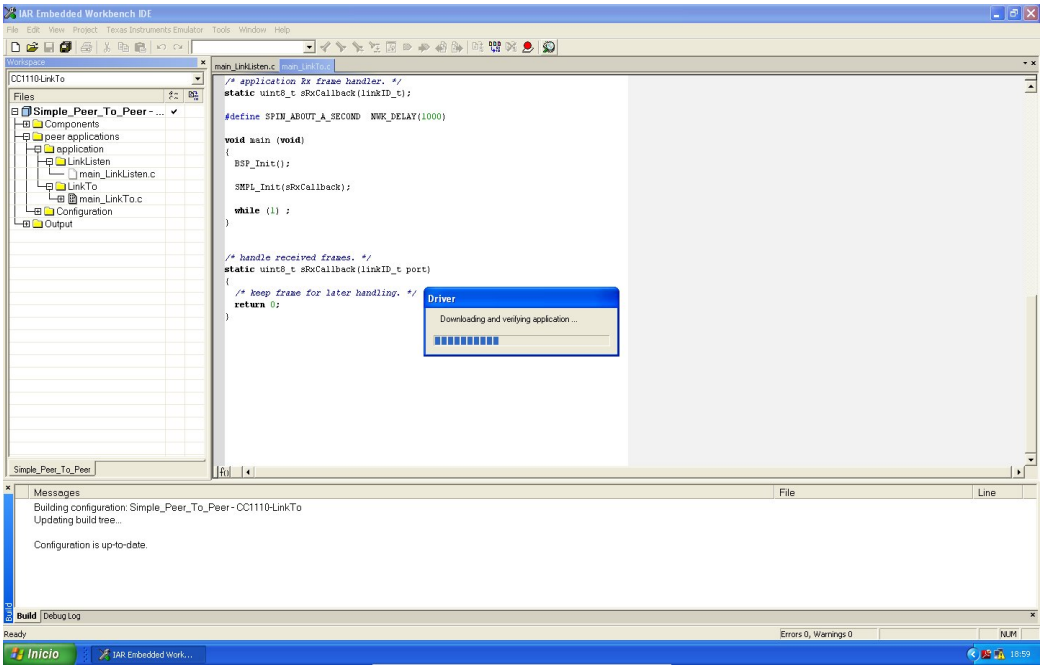


Figura 4.8: IEW cargando en el microcontrolador el código compilado

4.3.2. Depuración

Una vez terminado éste proceso el IDE automáticamente entra en modo de depuración indicando en verde la linea del código fuente que se está ejecutando. Además aparecerá una ventana con el código ensamblador generado por el compilador en la cual también se indica con el color verde la linea de código ensamblador que se está ejecutando. De forma adicional también se mostrará encima de la ventana de la estructura del proyecto una barra con los controles del modo de depuración. Con ellos podremos controlar el flujo de ejecución del código.

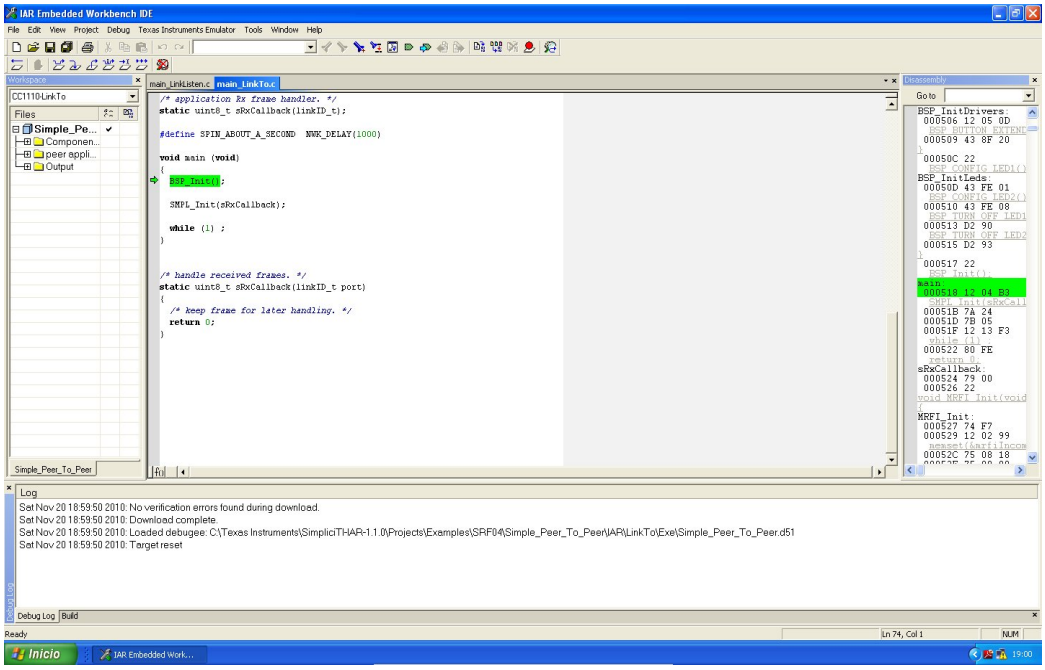


Figura 4.9: IEW en modo depuración

Los controles de depuración son los siguientes (descritos de izquierda a derecha):

Figura 4.10: Barra de controles de depuración

- Reset (reinicia la ejecución)
- Break (detiene la ejecución)
- Step over
- Step into (si ejecuta una función penetra dentro del código de la misma)
- Step out (ejecuta el resto del código de la función actual y regresa al código que la llamó)
- Next Statment (ejecuta la siguiente instrucción)
- Run to cursor (ejecuta el código hasta donde se encuentre el cursor)

Una de las herramientas más útiles que nos proporciona IEW en el modo de depuración es la posibilidad de visualizar los registros del microcontrolador en tiempo de ejecución (figura 4.11). Ésto es de gran ayuda a la hora de depurar aplicaciones que hacen uso de modulos internos del microcontrolador como el controlador de los puertos de E/S, la UART o bien para visualizar registros de configuración del microcontrolador. Para visualizar ésta ventana tenemos que ir al menú *View* → *Registers*.

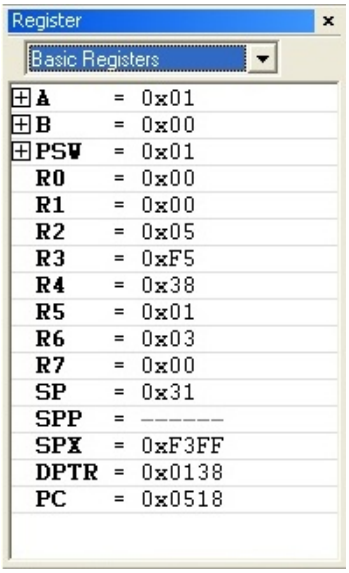


Figura 4.11: Ventana de registros del microcontrolador

En el mismo menú *View* podemos encontrar además diversas ventanas para visualizar contenido que nos puede ser útil a la hora de depurar el código de nuestro microcontrolador.

4.3.3. Perfiles de ejecución

Una de las funcionalidad más útiles que nos proporciona IEW son los perfiles de ejecución. IEW permite configurar el proyecto con distintos parametros relacionados con la compilación, enlace, parámetros de la arquitectura sobre la que se va a programar, librerias, etc. Además, IEW permite definir otros tipos de parámetros a nivel de cada archivo del proyecto individual que afectan sólo a ése archivo. Ambas configuraciones son guardadas y relacionadas con el perfil de ejecución activo en ése momento. De ésta forma, cuando se selecciona ése perfil se activará las configuraciones que estan relacionadas con ese perfil (4.12). Por defecto existen dos perfiles de ejecución «debug» y «release» los cuales se pueden utilizar para definir diferentes configuraciones del proyecto para depurar nuestro código o para realizar ejecuciones desatendidas.

Figura 4.12: Selección del perfil de ejecución activo

Nuestra red de sensores está formada por los nodos que conforman la red y un nodo sumidero el cual estará conectado mediante USB a un terminal. Los nodos convencionales que conforman la red contendrán el mismo código, pero el nodo sumidero debe ejecutar un código totalmente distinto ya que su comportamiento es radicalmente distinto al resto de nodos. En circunstancias normales lo lógico sería crear un proyecto para el código que ejecutarán los nodos convencionales, y otro proyecto que se correspondería con el código del nodo sumidero. Sin embargo, a pesar de que ambos tipos de nodos ejecutan un código principal diametralmente distinto, ambos tipos de nodos comparten gran parte de las librerías de funciones que controlan ciertas funciones del microcontrolador. Por lo que, además de la molestia que resulta manejar dos proyectos distintos, no es lo más adecuado en tanto en cuanto comparten muchas librerias.

La solución a éste problema viene de la mano de los perfiles de ejecución. Para nuestra aplicación crearemos un único proyecto que albergue todas las librerias correspondientes a ambos nodos y sus códigos principales. El problema es que si, al incluir todo el código en un mismo proyecto, intentaramos compilar el IEW nos mostraría un error ya que tenemos dos funciones *main* (una del código correspondiente a los nodos convencionales y otro que se corresponde con el del nodo sumidero). Aquí es donde entran en juego los perfiles de ejecución. Los parámetros de configuración a nivel de archivo permiten entre otras muchas cosas deshabilitar el archivo, es decir, que ése archivo no será tenido en cuenta para la compilación del proyecto. Si deshabilitamos uno de los archivos que contiene una función *main* (figura 4.13), automáticamente IEW conseguirá compilar ya que sólo tendremos una función *main* activa. Ésa configuración de archivo sólo quedará guardada en el perfil de ejecución activo, por lo que si cambiamos a otro perfil, ése archivo seguirá activo. Si en el perfil en el que todavia están activos las dos funciones *main*

realizamos la misma acción pero con la función *main* que no fue deshabilitada en el primer perfil, conseguimos que con el perfil actual podamos compilar la función *main*. De ésta forma, ya tenemos dos perfiles de ejecución que compilan correctamente cada una de las dos funciones *main*, por lo que, a efectos prácticos, es como tener dos proyectos en uno ya que según que perfil de ejecución tengamos activo IEW compilará una función *main* u otra.

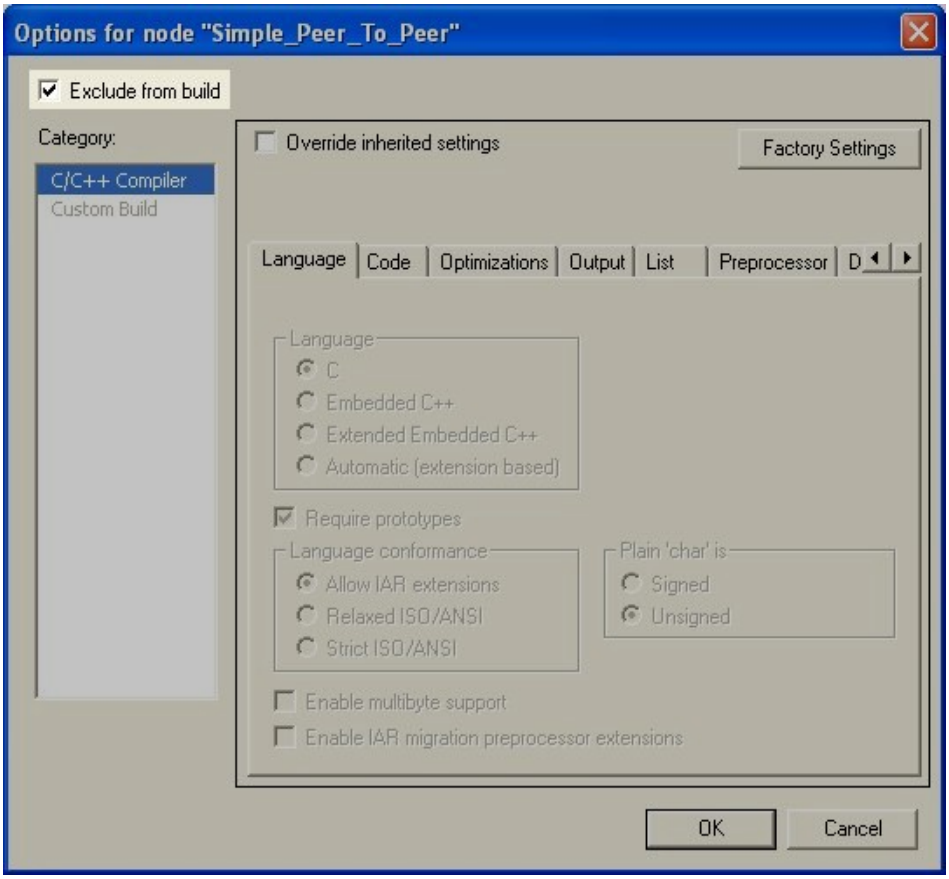


Figura 4.13: Desactivar la compilación de un archivo

Extrapolando ésto a nuestra casuística particular tendremos dos perfiles de ejecución, «Node» y «Sink», el primero compilará el código principal de los nodos convencionales y el segundo el código correspondiente al nodo sumidero. En el perfil de ejecución «Node» deshabilitaremos la compilación del archivo que contiene la función *main* del nodo sumidero y con el perfil «Sink» activo desactivaremos la compilación para el archivo que contiene la función *main* que corresponde a los nodos convencionales. Ambos perfiles de ejecución compilarán a partir de su función *main* correspondiente y nosotros activaremos el perfil de ejecución correspondiente dependiendo de si queremos compilar y programar un nodo convencional o un nodo sumidero.

Los perfiles de ejecución además pueden ser usado para otros propósitos

como generar diferentes versiones de los binarios del mismo código fuente para distintas arquitecturas de microcontrolador simplemente cambiando el perfil de ejecución. En general es un mecanismo que permite tener guardades y a mano diferentes configuraciones del mismo proyecto.

4.4. SimpliciTI

SimpliciTI^{TM} es un protocolo dirigido a pequeñas redes de bajo consumo de energía y uso de radiofrecuencia[3] de baja potencia. Éste fue desarrollado por Texas Instruments (en adelante TI) con la finalidad de facilitar la implementación y el despliegue de aplicaciones usando algunas de las plataformas de TI basadas en radiofrecuencia, entre las que se encuentra por supuesto la familia de sistemas en chip[4] (*System-on-Chip* - SoC) del CC1XXX.

Esencialmente el protocolo esta dirigido para aquellos que buscan una solución inalámbrica para redes de baja potencia, bajo coste y baja velocidad de transmisión como un gran caja negra que les permita enviar mensajes por dicha red inalámbrica de la forma más simple posible (por ejemplo sistemas de alarma y seguridad, redes de detectores de humo o sistemas domóticos entre otros).

El protocolo esta orientado principalmente para aplicaciones en los que la comunicación se realiza punto a punto entre nodos que están enlazados explicitamente. Sin embargo SimpliciTI soporta otro tipo de arquitecturas.

Desde el punto de vista del desarrollo, el API de SimpliciTI provee de medios para inicializar la red, enlazar dispositivos y enviar mensajes por la red de forma transparente. De entre las funciones de las librerías del SimpliciTI se hace uso en el proyecto sobretodo del manejo transparente de tramas y de las opciones de configuración de la radio y los registros del microcontrolador y no del protocolo propuesto por SimpliciTI específicamente.

4.4.1. Componentes modulares

La pila básica del SimpliciTI soporta comunicación inalámbrica sin ninguna caractesristica adicional. Como vemos en la figura 4.14, hay algunos módulos cuyas funcionalidades pueden ser añadidas a la pila básica combinándolas para conseguir un entorno adaptado a la aplicación del consumidor.

En el ámbito del desarrollo del proyecto solo se ha utilizado el módulo básico del SimpliciTI, aunque el uso de alguno de los módulos presentes

[3] Porción menos energética del espectro electromagnético

[4] Sistemas en los que se integran los componentes de un computador o cualquier otro sistema electrónico en un único circuito integrado (chip)

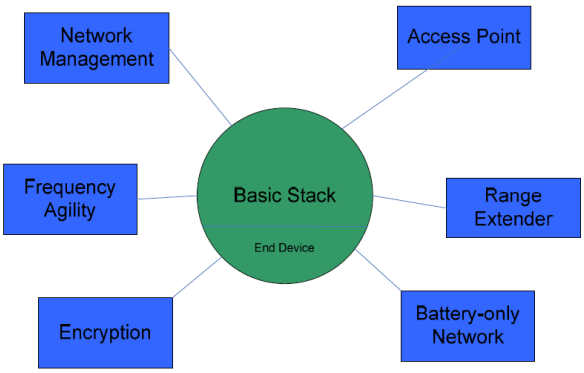


Figura 4.14: Componentes modulares de SimpliciTI

podría ser una opción de ampliación de las funcionalidades de los nodos de la red propuesta. Por ello se ofrece a continuación una breve descripción de algunos de ellos que podrían resultar interesantes.

Encriptación

Cuando se habilita este módulo todos los campos excepto el de dirección se encriptan. Funcionalidad que aumenta la seguridad aun con el contra de que las estaciones puente deben desencriptar el mensaje para saber a qué dirección deben reenviarlo.

Frecuency agility

Este módulo representa la capacidad de la red para migrar de una frecuencia a otra si detecta que la red presenta ruido o se ha visto comprometida la comunicación.

La frecuencia se centraliza a todos los nodos de la red a través de una tabla de frecuencias. Los nodos detectan los cambios si no reciben acuse de recibo ante dos reenvíos. El emisor cambia de frecuencia recorriendo la tabla hasta que recibe el acuse de recibo.

Depende de la arquitectura el cambio de frecuencia puede ser impuesto por un nodo actuando como punto de acceso.

Por otro lado los dispositivos que únicamente transmitan sin escuchar el medio (*Tx-only devices*) deberán enviar dos veces en todas las frecuencias de la tabla (transmiten bien en una sola o bien en una secuencia de frecuencias o canales).

Extensores de Rango

Éstos nodos reenviarán todos los paquetes excepto si son los destinatarios de los mismos (como un ping). El número de saltos se decrementa al atravesar un nodo extensor de rango.

4.4.2. Resumen de la arquitectura

El software del SimpliciTI soporta conceptualmente tres capas de la pila de protocolos: nivel físico, nivel de red y nivel de aplicación. De estas tres el nivel de aplicación es el único que el cliente necesita desarrollar. La comunicación se realiza a través de algunas funciones propias del API[5] para inicializar la red y enviar o recibir los mensajes inalámbricos.

La arquitectura no sigue estrictamente el modelo de referencia OSI, de modo que si se necesita funcionalidad extra de alguna de las capas del modelo OSI ésta debe ser asumida por alguna de las capas presentes.

La capa física por otra parte no posee las funcionalidades típicas de una capa física ya que los datos se reciben directamente de la radio que es quién realiza dichas funcionalidades (el nivel físico del CC1110).

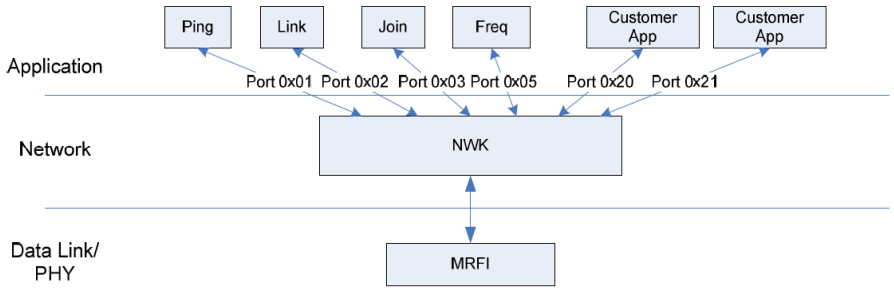


Figura 4.15: Arquitectura de SimpliciTI

Capa de Aplicación

El usuario desarrolla la aplicación y las interacciones del sensor con el entorno. El uso del API de SimpliciTI aporta a la aplicación la posibilidad de enviar o recibir mensajes hacia o desde otro dispositivo.

[5]Interfaz de programación de aplicaciones, un conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción

Capa de Red

La capa de red del SimpliciTI extiende la funcionalidad típica del estándar OSI.

La configuración de la red se da en tiempo de ejecución por medio de una interfaz del tipo *ioctl()*. De esta forma se pueden configurar diversos parámetros, desde la frecuencia base, número de frecuencias soportadas, método de modulación, velocidad de transmisión, entre otros.

El espacio de nombres del SimpliciTI consiste en direcciones de cuatro octetos, excluyendo los direcciones 0x00000000 y 0xFFFFFFFF que se reservan para difusiones.

MRFI

MRFI (del inglés *Minimal RF Interface*) es una capa del SimpliciTI que ofrece un nivel de abstracción ante envío y recepción de paquetes a través del interfaz de radio. Es decir, se sitúa en el nivel físico del estándar OSI.

BSP

Este paquete (BSP, del inglés *Board Support Package*) ofrece a su vez un nivel de abstracción para distintas funciones del microcontrolador, como son el interfaz SPI con la radio y el manejo de interrupciones o LEDs por medio de las conexiones GPIO[6].

4.4.3. Estructura del paquete

Los paquetes de SimpliciTI se separan en tres secciones lógicas:

- Una parte que se procesa por los niveles físico y de acceso al medio, consistente en los bits de preámbulo y de sincronización.
- Una parte que soporta la gestión de la red, utilizados para su control y especificar distintos parámetros, como el tipo de paquete (a nivel del SimpliciTI), el número de secuencia, contador de saltos, etc.
- Una parte que representa la carga útil de la capa de aplicación creada por el usuario, en nuestro caso los paquetes del protocolo de comunicación diseñado que se procesa como parte de la trama recibida.

[6]Entrada/Salida de propósito general (*General Purpose Input/Output*), una interfaz de comunicación presente en el microcontrolador que ofrece interconexión con distintos periféricos

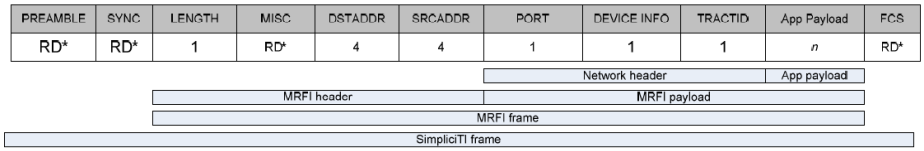


Figura 4.16: Detalle de las secciones lógicas de un paquete del SimpliciTI

Campo	Descripción	Comentarios
Preámbulo y Sync	Sincronización de la radio	Dependiente y manejado por la radio del CC1110
Longitud	Longitud del paquete en bytes	Cuenta a partir del campo siguiente
DSTADDR y SRCADDR	Campos reservados para las direcciones de fuente y destino	Parcialmente dependiente de la radio, pero manejados por SimpliciTI
Puerto	Solo los seis bits de menor peso se refieren al puerto de la aplicación, el resto se refieren al nivel de seguridad	El espacio de nombres para los puertos reservan las direcciones de la 0x20 a la 0x3F para aplicaciones del cliente
TRACTID	Identificador de transacción	Identificador secuencial para añadir robustez en la comunicación
APP PAYLOAD	Datos de la aplicación	Espacio en el que se encapsula el protocolo propuesto, el número máximo de bytes es entre 50 o 111 dependiendo del tipo de radio
FCS	Frame Check Sequence	Corresponde al CRC calculado por el hardware de la radio

Tabla 4.1: Sumario de Campos relevantes del paquete del SimpliciTI

Puertos

Los puertos son abstracciones conceptuales que especifican la aplicación objetivo que manejará el paquete. Los puertos del 0x00 al 0x1F son reservados como puertos *bien conocidos* con servicios destinados a la gestión de la red.

Los puertos del 0x20 al 0x3F se mapean a manejadores del usuario. En concreto el puerto 0x3F se reserva para emisiones por difusión en nodos que no han sido enlazados[7] entre sí.

Los puertos se mapean a nivel de red respecto a los identificadores de los enlaces. Se asocian a manejadores que se lanzan cuando se envían o reciben paquetes.

Campos omitidos

En la tabla 4.1 se han omitido intencionadamente los campos MISC y DEVICE INFO, que no se usan en el protocolo propuesto y cuya finalidad esta relacionada con cuestiones del protocolo definido por SimpliciTI del cual no se hace uso.

4.5. SmartRF Packet Sniffer

Packet Sniffer es una aplicación desarrollada por SmartRF^{TM} que se usa para mostrar por pantalla en tiempo real (y con posibilidad de almacenar) los paquete capturados por un nodo esuchando en el espectro de radiofrecuencia. La aplicación da opción de filtrar la captura y decodifica el paquete mostrando cada campo de forma visual y clarificadora.

Packet Sniffer soporta una amplia gama de protocolos que se seleccionan desde una ventana que se lanza antes de acceder al interfaz principal de la aplicación. Entre los protocolos soportados se encuentran Bluetooth (especificación 4.0), ZigBee, RF4CE [8] y SimpliciTI, además de un modo de captura genérica.

El uso de esta aplicación ha sido de una utilidad decisiba para las pruebas del protocolo de comunicación, ya que permitía observar y detectar en tiempo real errores en el intercambio de mensajes entres los nodos de los que se disponía sin la necesidad de depurar los dispositivos. Además ofrece muchas posibilidades de filtrado e información adicional interesante que facilitaba el desarrollo de las comunicaciones inalámbricas entre nodos.

[7] El enlazado de nodos forma parte del protocolo del SimpliciTI, parte que no se usa en el protocolo propuesto y que por tanto obliga a usar el puerto 0x3F de difusión en todas las emisiones

[8] Protocolo que forma parte de la ZigBee Alliance

4.5.1. Flujo de datos

Desde el punto de vista del PC los paquetes se almacenan en un búffer, y pasan por una caché en la RAM para mejorar el tiempo de acceso cuando un paquete debe ser mostrado en el entorno gráfico de usuario.

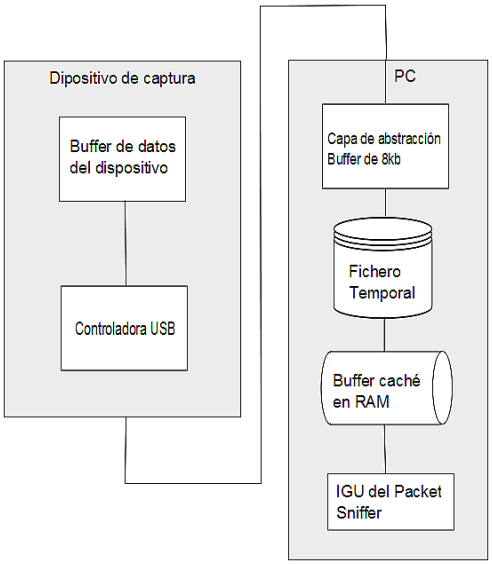


Figura 4.17: Flujo de datos del *Packet Sniffer*

4.5.2. Detalles de la aplicación

El firmware se descarga automáticamente sobre el dispositivo que actuará de sniffer si se requiere al inicializar la captura. Se comprueba desde la interfaz gráfica.

La aplicación es compatible únicamente con Windows.

Interfaz gráfica de usuario

Ventana de Lanzamiento

Cuando se ejecuta la aplicación se da la opción de elegir entre distintos protocolos y configuraciones desde una ventana lunar (o ventana de lanzamiento) que se muestra al incio.

Para iniciar una sesión del programa se selecciona el protocolo, que en nuestro caso es el SimpliciTI, y seleccionamos el botón *start*. La ventana abre la sesión y cerrar o no la ventana de lanzamiento no afecta la ejecución

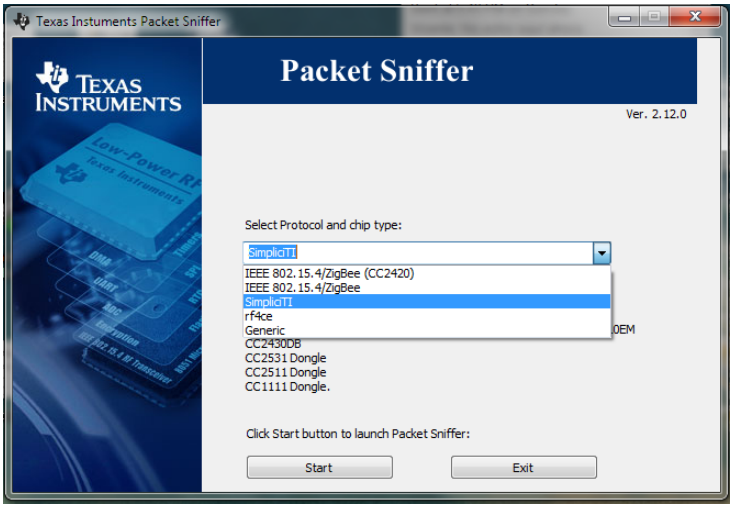


Figura 4.18: Ventana de lanzamiento del *Packet Sniffer*

de la aplicación. En la ventana de lanzamiento además se especifican algunos de los dispositivos compatibles con los distintos protocolos y con la aplicación.

Ventana de sesión del *Packet Sniffer*

Una vez lanzada la aplicación se ve la ventana principal dividida en dos secciones, una en la que aparecen los paquetes capturados y otra en la que se selecciona y configuran distintos parámetros de los dispositivos conectados y de los filtros.

En la captura de la ventana principal mostrada podemos ver un ejemplo con el protocolo SimpliciTI y los datos del protocolo de comunicación desarrollado en medio de la etapa de formación de la red.

En la barra de estado (parte inferior izquierda de la ventana) se encuentra un contador de paquetes capturados, otro para el numero de paquetes con errores (por desbordamiento del búffer o por errores de CRC) y el estado actual del filtro.

Menus y barra de tareas

En la siguiente figura (figura 4.20) mostramos los botones de la barra de la parte superior de la ventana principal:

Los botones y sus funciones son, de izquierda a derecha:

1. Vacía el buffer y la lista de paquetes (también accesible desde el menú File→Reset)

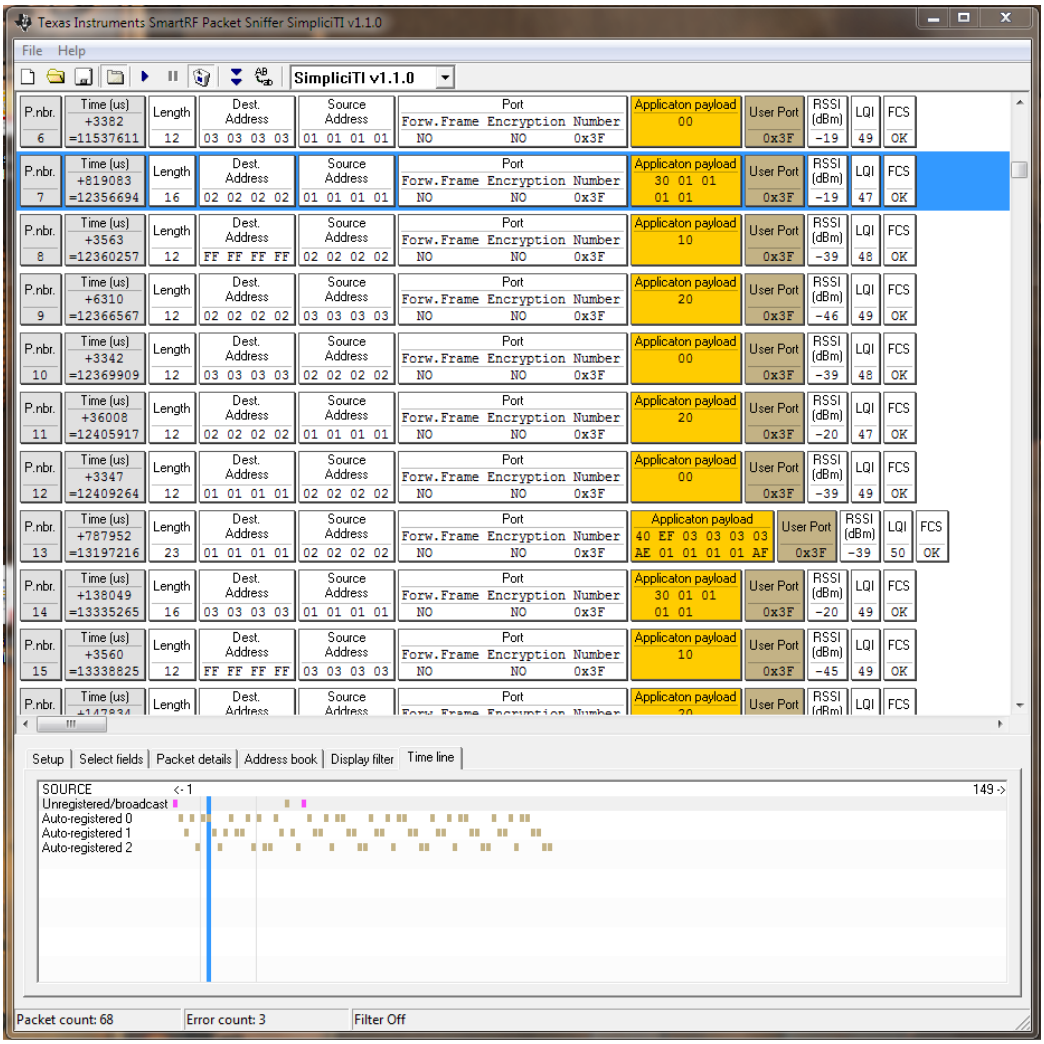


Figura 4.19: Ventana principal del *Packet Sniffer*

Figura 4.20: Barra de tareas del *Packet Sniffer*

2. Carga un buffer de paquetes desde un archivo (PSD) (también accesible desde el menú File→Open data...)
3. Guarda un buffer de paquetes a un archivo (PSD) (también accesible desde File→Save data...)
4. Muestra las etiquetas de la parte inferior de la ventana
5. Inicia una nueva captura (o bien usando F5)
6. Pausa la captura actual (o bien usando F6)
7. Al inicia una nueva captura borra los paquetes de la captura anterior
8. Activa o desactiva el desplazamiento automático de la captura
9. Cambia entre tamaños de fuente en la vista de paquetes
10. Selección de la versión del protocolo (solo para ZigBee y SimpliciTI)

Otras opciones interesantes que se pueden configurar desde el menú son el tamaño del búffer[9] y el multiplicador del reloj[10].

Dispositivo de captura

En la parte inferior de la pantalla principal existen varias pestañas para seleccionar y configurar distintas opciones extra, entre ellas esta la opción de seleccionar el dispositivo objetivo sobre el que se descargará el programa y se realizará la captura. Debe haber un dispositivo seleccionado antes de iniciar la captura.

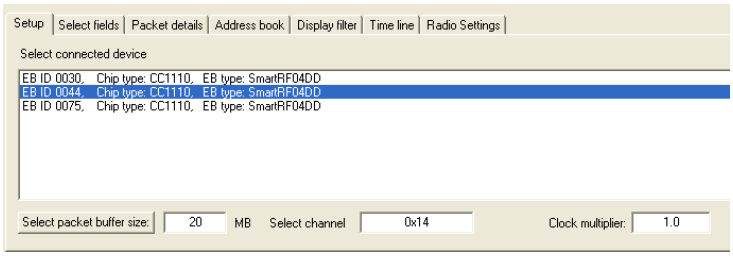


Figura 4.21: Selección de dispositivos de captura

En la captura vemos los distintos dispositivos identificados su USB ID y su chip ID. Si el dispositivo tiene interfaz de comunicación en serie se puede seleccionar para monitorizar los datos recibidos por dicho interfaz.

[9]Tamaño del búffer RAM en megabytes que se reserva para contener los paquetes capturados, para mejorar el tiempo de acceso de la aplicación gráfica

[10]El multiplicador de reloj ayuda a compensar las diferencias en la frecuencia de reloj entre el dispositivo conectado que actuará como sniffer y el resto de dispositivos en la red

Configuración de la radio

Otra de las pestañas de la sección de la parte inferior es la de la configuración de la radio del sniffer. En él se puede configurar toda la gama de parámetros de la radio del dispositivo, el canal, la frecuencia, etcétera.

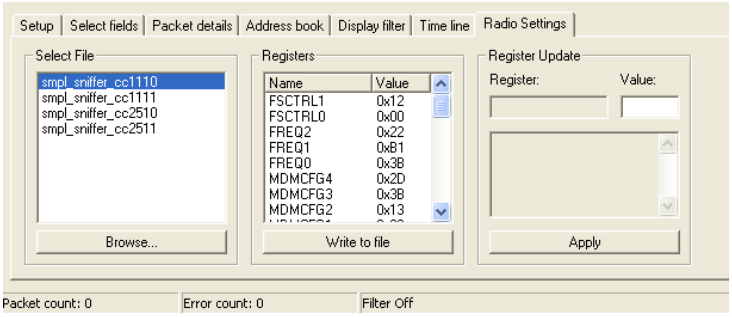


Figura 4.22: Parámetros de configuración del dispositivo de captura

La configuración se almacena o se carga desde un fichero de texto (se puede generar a través del SmartRF© Studio[11]), aunque cuenta ya con varios archivos de distintas configuraciones por defecto (configuración utilizada en el banco de pruebas del proyecto). El formato de cada fila del fichero es:
`«Nombre del registro | Dirección de memoria | Valor | Breve Descripción»`

Selección de campos en la lectura

La pestaña de selección de campos permite elegir qué campos mostrar y cuales no mostrar en la pantalla durante la captura. Por ejemplo, capturando paquetes del protocolo de SimpliciTI permite mostrar u ocultar los campos de la cabecera del SimpliciTI, su carga útil, etc.

Los campos se agrupan en categorías que se colorean en la captura para distinguirlas fácilmente.

El *timestamp* se puede mostrar en microsegundos o milisegundos. La carga últil (*App Payload*) se puede mostrar en bytes hexadecimales o bien en texto plano (en texto plano aquellos caracteres que no se puedan mostrar serán reemplazados por «∗»).

Configuración de filtros durante la captura

Los filtros permiten ocultar de la captura todos los paquetes que no cumplan los requisitos, pudiendo discriminarlos por todos los campo definidos en la ventana de selección de campos.

[11] Aplicación desarrollada por Chipcon para configurar los distintos dispositivos con circuitos integrados que trabajan en radiofrecuencia (entre los que se encuentra el CC1110)

```
1  FSCTRL1   |0xDF07|0x12|Frequency synthesizer control.
2  FSCTRL0   |0xDF08|0x00|Frequency synthesizer control.
3  FREQ2     |0xDF09|0x22|Frequency control word, high byte.
4  FREQ1     |0xDF0A|0xB1|Frequency control word, middle byte.
5  FREQ0     |0xDF0B|0x3B|Frequency control word, low byte.
6  MDMCFG4   |0xDF0C|0x2D|Modem configuration.
7  MDMCFG3   |0xDF0D|0x3B|Modem configuration.
8  MDMCFG2   |0xDF0E|0x13|Modem configuration.
9  MDMCFG1   |0xDF0F|0x22|Modem configuration.
10 MDMCFG0   |0xDF10|0xF8|Modem configuration.
11 CHANNR    |0xDF06|0x14|Channel number.
12 DEVIATN   |0xDF11|0x62|Modem deviation setting (when FSK modulation is enabled).
13 FREND1    |0xDF1A|0xB6|Front end RX configuration.
14 FREND0    |0xDF1B|0x10|Front end RX configuration.
15 MCSM0     |0xDF14|0x18|Main Radio Control State Machine configuration.
16 FOCCFG    |0xDF15|0x1D|Frequency Offset Compensation Configuration.
17 BSCFG     |0xDF16|0x1C|Bit synchronization Configuration.
18 AGCCTRL2  |0xDF17|0xC7|AGC control.
19 AGCCTRL1  |0xDF18|0x00|AGC control.
20 AGCCTRL0  |0xDF19|0xB0|AGC control.
21 FSCAL3    |0xDF1C|0xEA|Frequency synthesizer calibration.
22 FSCAL2    |0xDF1D|0x2A|Frequency synthesizer calibration.
23 FSCAL1    |0xDF1E|0x00|Frequency synthesizer calibration.
24 FSCAL0    |0xDF1F|0x1F|Frequency synthesizer calibration.
25 TEST2     |0xDF23|0x88|Various test settings.
26 TEST1     |0xDF24|0x31|Various test settings.
27 TEST0     |0xDF25|0x09|Various test settings.
28 PA_TABLE0 |0xDF2E|0x8E|PA output power setting.
29 PKTCTRL1  |0xDF03|0x04|Packet automation control.
30 PKTCTRL0  |0xDF04|0x05|Packet automation control.
31 ADDR      |0xDF05|0x00|Device address.
32 PKTLEN    |0xDF02|0xFF|Packet length.
```

Figura 4.23: Archivo de configuración por defecto para el CC1110 en el *Packet Sniffer*

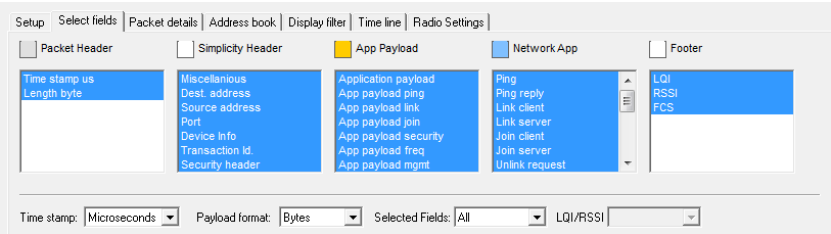


Figura 4.24: Pestaña de selección de campos para SimpliciTI

Se ofrece una plantilla para cada tipo de filtro todos ellos mostrando el campo con su nombre corto correspondiente, un signo de igualdad y el valor sobre el cual se desea restringir la captura(representado por una «x»). Si el campo tuviera subcampos éstos se muestran entre paréntesis en la plantilla.

Se pueden configurar condiciones para la aplicación del filtro con el operador AND (operador conjunción). Los filtros se puede añadir y eliminar en tiempo de captura, éstos solo impiden que se muestre, no que el dispositivo lo capture. Así si se elimina un filtro se mostrarán los paquetes filtrados de nuevo.

Formato de los paquetes en los ficheros PSD

Los ficheros PSD (*Packet Sniffer Data*) guardan toda la información de una captura, el formato de los paquetes en dicho fichero es el siguiente:

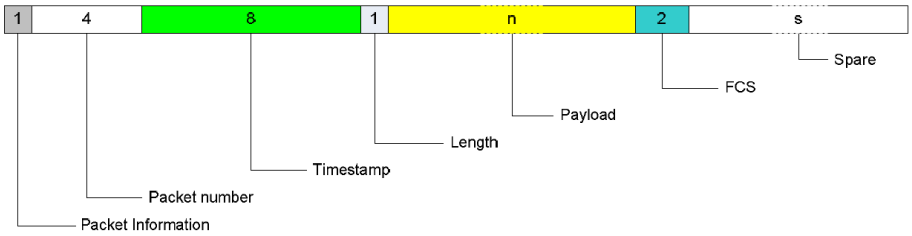


Figura 4.25: Formato de un paquete de un archivo PSD

- Información de paquete: campo introducido por Packet Sniffer desde la versión 2.3.0. y que contiene información para leer correctamente los datos.
- Timestamp: contador de 64 bits a partir del primer paquete recibido (que tendrá timestamp = 0).
- Longitud: longitud del paquete a partir de dicho campo. Puede o no incluir el campos FCS dependiendo de la información del paquete.
- Carga útil: la información empaquetada sobre el protocolo usado.
- FCS: indica si la suma de verificación del paquete es correcta o no.
- Relleno: depende de la cantidad de bytes usados por el programa para guardar cada paquete. Ésta cantidad depende a su vez del protocolo usado en la captura.

Capítulo 5

Implementación

5.1. Tarjeta de adaptación de sensores

En éste apartado explicaremos con detalle los cálculos y técnicas usadas a lo largo del proceso de implementación de la electrónica necesaria para adaptar la minúscula variación de resistencia de las galgas extensiométricas al ser deformadas solidarias al material sometido a tensión de forma que ésta señal adquiera las características necesarias para ser inyectada en una de las entradas del microcontrolador. En el texto explicaremos los pormenores de cada etapa que atraviesa la señal siguiendo el mismo orden que seguiría la misma al propagarse desde el punto en el que es generada en las galgas extensiométricas hasta que finalmente es admitida y procesada en el microcontrolador.

5.1.1. Galgas extensiométricas

Las galgas extensiométricas son un elemento con una relativa escasa variación de resistencia. Éste problema puede ser parcialmente paliado usando más de uno de éstos sensores para medir la deformación en el mismo punto. Usando un puente *Wheatstone* para interconectar varias galgas se puede ampliar la variación de resistencia dada una misma deformación.

5.1.2. Puente Wheatstone

El puente Wheatstone está formado por 4 elementos resistivos (en nuestro caso serán galgas extensiométricas) interconectados como muestra la figura 5.1:

Éste puente se caracteriza por que si todos los elementos resistivos del

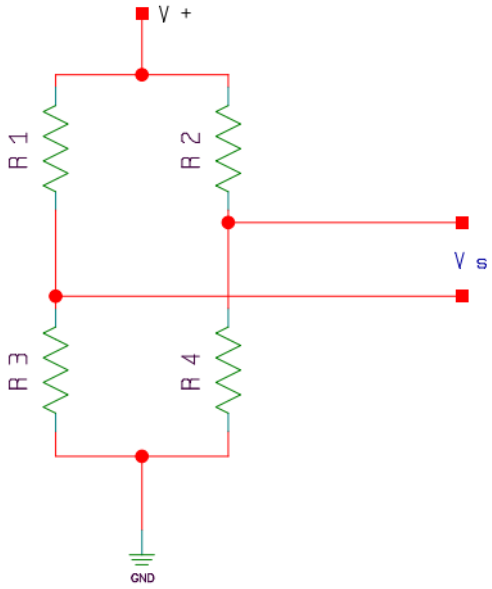


Figura 5.1: Esquema de conexiones de un puente Wheatstone

puente son exactamente iguales la salida Vs será 0V. Sin embargo, si cualquiera de las resistencias varía mínimamente Vs pasará a ser distinto de 0V. Éste comportamiento se modela con la siguiente ecuación:

$$Vs = \frac{R_4}{R_2 + R_4} - \frac{R_3}{R_1 R_3} * V$$

Donde se puede ver claramente que si todas las R son iguales la ecuación se resolverá:

$$Vs = (\frac{R}{2R} - \frac{R}{2R}) * V = 0 * V = 0V$$

Sin embargo, si R1 y R4 aumentan de forma solidaria al doble del valor de R2 y R3 la ecuación se despeja:

$$Vs = (\frac{2R_{23}}{R_{23} + 2R_{23}} * \frac{2R_{23}}{R_{23} + 2R_{23}}) * V = \frac{R_{23}}{3R_{23}} * V = 2R_{23} * V$$

Obviamente, el resultado sería el mismo si en lugar de aumentar R1 y R4, fueran R2 y R3 las que disminuyeran. Se ve claramente que en ésta situación, Vs aumentará a valor positivos.

De forma análoga, si R2 y R3 aumentan (o R1 y R4 disminuyen) Vs se obtiene de la siguiente forma:

$$Vs = (\frac{R_{14}}{2R_{14} + R_{14}} - \frac{2R_{14}}{R_{14} + 2R_{14}}) * V = \frac{R_{14}}{3R_{14}} * V = 2R_{14} * V$$

Por lo que claramente se deduce que para valores de R2 y R3 superiores a R1 y R4 en la salida obtendremos tensiones negativas.

El puente puede ser usado de distintos modos. Usando tres resistencias fijas y una activa (cuarto de puente) que será la que varíe su valor en función de algún factor externo (en nuestro caso será la galga que variará su resistencia en función de la deformación), usando dos resistencias fijas y dos activas (medio puente), o usando las cuatro resistencias como elementos activos (puente completo).

Cuarto de puente

Con una configuración en cuarto de puente obtenemos la mínima sensibilidad ya que sólo disponemos de un elemento activo que varía su resistencia, las otras tres resistencia son fijas. Ésta configuración es la más económica ya que los elementos activos son notablemente más caros que una simple resistencia, pero no ofrecen una gran sensibilidad como demostraremos a continuación.

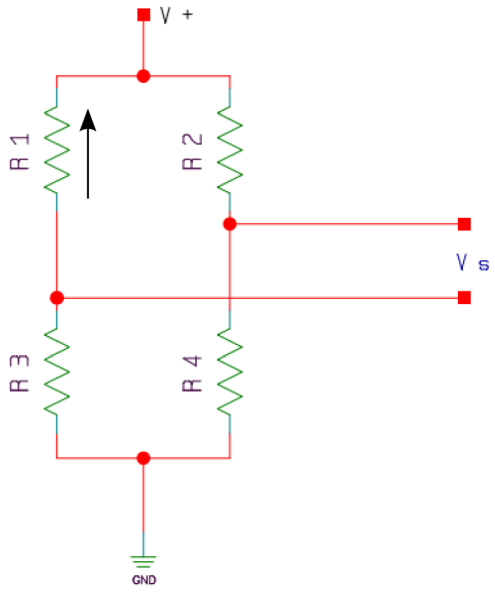


Figura 5.2: Esquema de conexiones de un cuarto de puente Wheatstone

Suponiendo que R1 es el elemento activo y R2=R3=R4=Rf, y que el valor máximo que adquiere R1= 2Rf, se deduce:

$$Vs = (\frac{R_f}{R_f + R_f} - \frac{R_f}{2R_f + R_f}) * V = \frac{1}{6} * V = 0{,}166666\ldots * V$$

Medio puente

En éste caso disponemos de dos elementos activos y dos resistencias fijas.

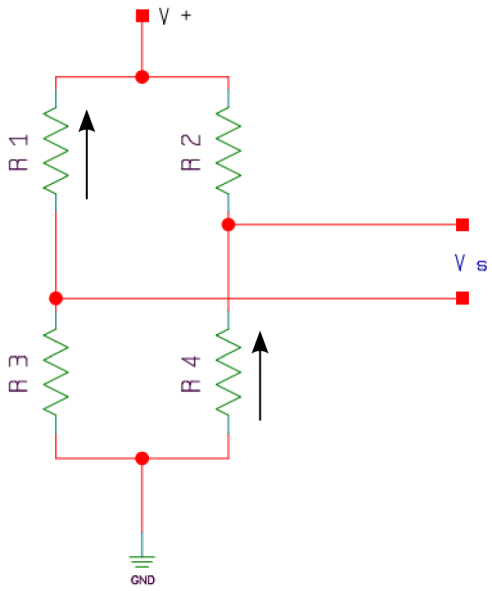


Figura 5.3: Esquema de conexiones de medio puente Wheatstone

Los elementos activos varían sus resistencias de forma solidaria, es decir, R1 siempre es igual a R4, R2 y R3 son resistencias fijas de igual valor.

$$Vs = (\frac{2R_f}{R_f + 2R_f} - \frac{R_f}{2R_f + R_f}) * V = \frac{1}{3} * V = 0{,}33333\ldots * V$$

Puente completo

Para obtener el máximo valor de salida del puente para éste caso hay que tener en cuenta que las galgas tienen un valor nominal en reposo y que pueden variar para aumentar respecto de ese valor nominal o disminuir. Así pues, el caso más extremo en ésta configuración es que R1 y R4 aumenten al máximo su valor respecto al nominal y R2 y R3 a su vez disminuyan al mínimo su valor respecto del nominal. De éste modo suponiendo que R1, R2, R3 y R4 son los elementos activos, R2=R3=Rnom/2, y que R1=R4=2Rnom, se deduce:

$$Vs = (\frac{2R_{nom}}{R_{nom}/2 + 2R_{nom}} - \frac{R_{nom}/2}{2R_{nom} + R_{nom}/2}) * V = \frac{3}{5} * V = 0{,}6 * V$$

Al observar todas las configuraciones resulta ya evidente que el puente completo ofrece una sensibilidad mucho mayor que las otras configuraciones a cambio de usar cuatro elementos activos, que, obviamente, encarecen el sensor.

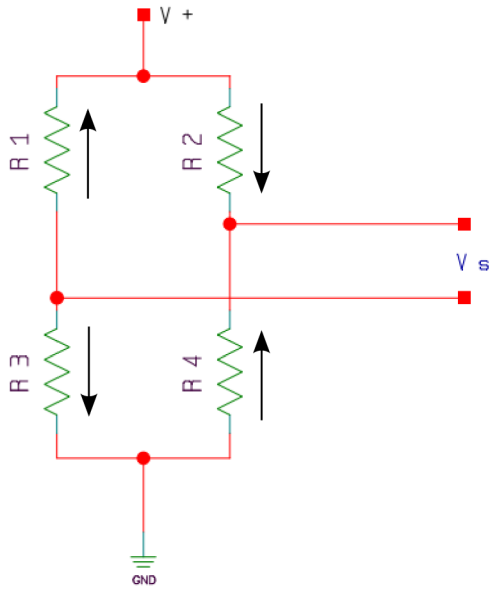


Figura 5.4: Esquema de conexiones de puente Wheatstone completo

Además, el puente completo ofrece una ventaja añadida, y es que al ser las cuatro resistencias activas e idénticas, todas ellas son influenciadas del mismo modo por el calor, de éste modo ésta configuración se auto-compensa por temperatura.

Por éstas razones, para nuestra aplicación vamos a optar por conectar cuatro galgas en un puente Wheatstone completo.

5.1.3. Cálculo de la variación de resistencia de las galgas extensiométricas

Para diseñar el circuito amplificador primero hay que conocer de antemano entre qué valores oscilará su entrada. Para conocer éstos valores antes debemos saber cuánto se deformaran las galgas como máximo, ya que la elongación de éstas es directamente proporcional a la variación de resistencia de las mismas. Dado que las galgas se deforman solidarias al material sobre el

que están adheridas, la deformación máxima que sufrirán éstas será la deformación máxima del material sobre el que están midiendo. Sin embargo, la deformación máxima que sufrirá el material no es algo trivial de hallar, ya que cada material tiene unas características distintas, y éstas a su vez cambian cuando se alea con otro material, y varían además según las proporciones de la aleación.

Dado que nuestra aplicación está orientada a medir deformaciones en estructuras metálicas empleadas en la construcción podemos acotar considerablemente la variedad de metales a tener en cuenta dado que en la construcción se emplean determinados tipos de aleaciones de acero estándar. Es por ello que para simplificar el diseño de la electrónica entre otras cosas, asumiremos que mediremos deformaciones sobre acero de tipo S275JR ya que es el tipo de acero mas común en la construcción.

Fundamentos básicos de mecánica de materiales

Todos los materiales lineales sometidos a una deformación se caracterizan por presentar dos comportamientos distintos ante la misma. Para tensiones hasta un limite σ, los materiales se deforman, pero una vez desaparece la tensión el material vuelve a su forma inicial. Para tensiones mayores a σ, una vez desaparece la tensión, el material queda deformado de forma permanente. A éstos dos comportamientos se les denomina *comportamiento elástico* y *comportamiento plástico* respectivamente, y a la tensión límite σ que separa la zona elástica de la zona plástica se le denomina *límite elástico* del material. Dicho límite elástico es una característica propia de cada tipo de material. En la figura 5.5 podemos ver ambas zonas bien diferenciadas.

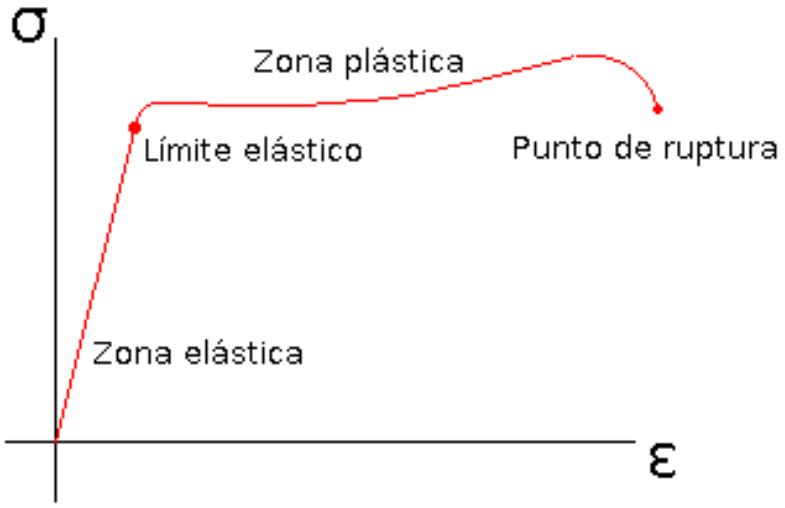


Figura 5.5: Gráfica tensión/deformación del acero

Obviamente, para nuestra aplicación nos interesa medir los materiales en la zona elástica de los mismos. Si una viga o pilar sufriera una tensión suficiente y éste entrara en su zona plástica, dicho material sufriría deformaciones importantes y permanentes por lo que probablemente la estructura de la que dependa la integridad del pilar o viga deformados se vería gravemente comprometida de forma inmediata e irreversible. Dado que en éste caso extremo, medir la deformación carecería de sentido en una aplicación real por razones obvias, nos centraremos en medir las deformaciones de la zona elástica del material ya que éstas son las más comunes y las que conviene vigilar dado su peligro potencial y la capacidad de reacción ante una situación en la que se presente dicho peligro.

Para caracterizar el comportamiento de los materiales elástico sometido a una fuerza se usa el modulo de Young. Dicha constante es distinta para cada material y es la misma para tracción y para compresión para materiales elástico lineales isótropos[1] como el acero. Además puede ser calculada mediante la siguiente ecuación:

$$E = \frac{\epsilon}{\sigma}$$

Donde E = Modulo de Young, σ= Tensión aplicada al material (N/mm^2) y = deformaciones[2].

Conociendo dichos datos ya podemos calcular la deformación máxima del material. La deformación que consideraremos como máxima para nuestra aplicación será la deformación sufrida por el material en el límite elástico por las razones antes citadas. Dado que el acero tiene una constante de Young de 210000 N/mm^2 y el límite elástico de dicho material en el rango de temperatura 16 <t <= 40 es de 265 N/mm^2, calculamos:

$$\epsilon = \frac{\sigma}{E} = \frac{265}{210000} = 1'261904762\mu\epsilon$$

Anteriormente definimos el factor de galga como:

$$K = \frac{\Delta R/R}{\Delta L/L}$$

Dado que $\frac{\Delta L}{L} = \epsilon$, siendo ϵ el valor antes calculado para el material ya que la galga se deformará de forma solidaria al material, redefinimos el factor de galga como:

[1] Es la característica de los cuerpos cuyas propiedades físicas no dependen de la dirección

[2] Unidad de medida adimensional que representa la variación de elongación respecto a la longitud inicial

$$K = \frac{\Delta R/R}{\epsilon}$$

Siendo K el factor de galga que nos proporciona el fabricante en cada lote, ϵ la deformación calculada y R el valor nominal de resistencia de la galga que también es conocido, ya podemos saber cual será la variación de R que sufrirá la galga ante la deformación sufrida en el límite elástico del material.

$$K = \frac{\Delta R/R}{\epsilon} \rightarrow \frac{\Delta R}{R} = K * \epsilon \rightarrow \frac{\Delta R}{350} = 2'105 * 1'262 * 10^{-3} \rightarrow$$
$$\frac{\Delta R}{350} = 2'656 * 10^{-3} \rightarrow \Delta R = 2'656 * 10^{-3} * 350 = 0'92971\Omega$$

Tenemos que, para la deformación en el límite elástico, cada galga variará $\pm+-0'92971\Omega \approx 0'93\Omega$, ya que la deformación puede ser positiva (a tracción) o negativa (a compresión).

5.1.4. Cálculo de la salida del puente

Una vez tenemos el rango de valores entre los que oscila la variación de resistencia de las galgas respecto del valor nominal, es trivial calcular los valores entre los que oscilará la salida del puente Wheatstone. Así pues, solo tenemos que aplicar la ecuación del puente Wheatstone.

$$Vs = (\frac{R_4}{R_2 + R_4} - \frac{R_3}{R_1 + R_3}) * V$$

Valor mínimo de salida (máximo negativo)

El valor máximo negativo se dará cuando R2 y R3 alcancen su máximo valor mientras que R1 y R4 alcancen su mínimo valor. Sabiendo que el valor nominal de las galgas en reposo es de 350 Ohm y la tensión de alimentación del puente será de 5V, calculamos:

$$Vs = (\frac{350 - 0,93}{350 + 0,93 + 340 - 0,93} - \frac{350 + 0,93}{350 - 0,93 + 350 + 0,93}) * 5 = -0{,}01328571428V$$

Valor máximo de salida (máximo positivo)

El valor máximo positivo del puente se dará cuando R1 y R4 lleguen a su máximo valor de resistencia al tiempo que R2 y R3 llegan a su valor mínimo. Así pues, con los mismos valores que en el punto anterior calculamos:

$$Vs = (\frac{350 + 0,93}{350 - 0,93 + 340 + 0,93} - \frac{350 - 0,93}{350 + 0,93 + 350 - 0,93}) * 5 = 0{,}01328571428V$$

5.1.5. Amplificador

Una vez tenemos el valor mínimo y máximo que nos entregará el puente Wheatstone ante la máxima deformación que puede sufrir el material sobre el que vamos a medir, ya podemos realizar los cálculos oportunos concernientes al amplificador que nos proporcionará unos valores de tensión adecuados para trabajar en las posteriores etapas de tratamiento de la señal.

El amplificador va a jugar un papel esencial en nuestro circuito de adaptación de la señal proveniente de las galgas extensiométricas. La salida del puente Wheatstone irá directamente conectada a la entrada de nuestro amplificador, éste amplificará dicha señal y la salida se la aplicaremos a un conversor analógico-digital para que convierta la señal amplificada a una señal digital. Sin embargo, el puente Wheatstone nos puede entregar tanto tensiones positivas como negativas en función de si las galgas son deformadas a tracción o a compresión (el perfil flexa en un sentido o en otro). Si conectaramos el puente de galgas directamente al amplificador, cuando el puente entregara tensiones positivas el amplificador entregaría a su salida tensiones positivas.

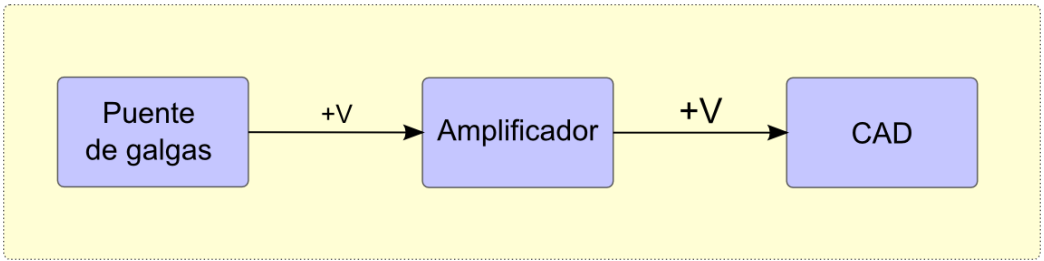


Figura 5.6: Amplificador con tensión de entrada positiva

Sin embargo, dado que el amplificador en reposo nos entrega 0V, si a su entrada le inyectáramos una señal negativa, a la salida seguiríamos teniendo 0V ya que el amplificador no es capaz de entregar tensiones negativas estando alimentado a una tensión de V_{cc}. Ésto supone un problema ya que sólo seríamos capaces de medir deformaciones en un sentido. Es decir, no podriamos las deformaciones que se produjeran en el sentido para el cual el puente Wheatstone entregara tensiones negativas.

Una forma de solucionar dicho problema sería alimentando el amplificador con una tensión simétrica en el rango de $\pm$ V_{cc}. De éste modo el amplificador sería capaz de entregar tensiones negativas amplificadas en la salida.

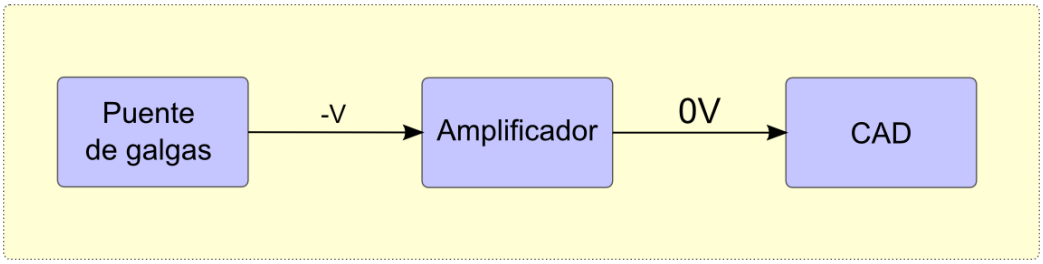


Figura 5.7: Amplificador con tensión de entrada negativa

Sin embargo, ésta solución plantea otros problemas, y es que para generar una tensión simétrica a partir de baterías sería necesario el montaje de dos baterías independientes como muestra la siguiente figura.

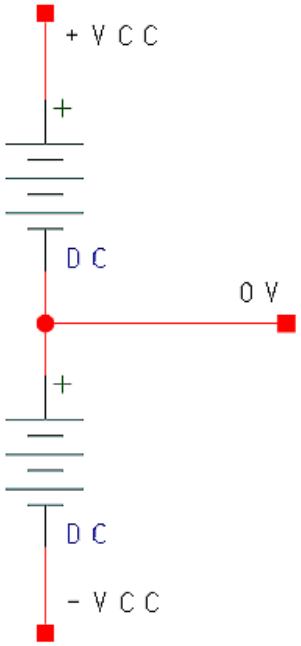


Figura 5.8: Esquema de una fuente simétrica fabricada con dos baterías idénticas

Ésto plantea serios problemas en tanto en cuanto las baterías nunca serán exactamente iguales ni se desgastaran por igual, lo que provocaría una asimetría en la alimentación que afectaría a las mediciones y sería difícil de corregir. Además implicaría el tener que montar dos baterías en cada nodos, lo cual aumentaría de volumen cada uno de los mismos así como los costes.

La solución que se ha adoptado consiste en desplazar el valor de salida del amplificador en reposo de 0V a $V_{cc}/2$. De éste modo, la salida del amplificador en reposo será la mitad de la alimentación. Cuando se le inyecte una tensión

positiva a la entrada del amplificador éste amplificará hasta un valor máximo de V_cc y, análogamente, cuando se le inyecte una tensión negativa a la entrada éste amplificará pero en términos negativos hasta 0V. Ésta solución tiene la desventaja de que perdemos la mitad del rango de valores en la salida del amplificador y, por lo tanto perdemos también resolución, pero es una perdida asumible a cambio de ganar enormemente en simplicidad.

Para obtener el comportamiento descrito anteriormente se aprovecha uno de los terminales del CI[3] del amplificador llamado V_{ref}. Éste terminal tiene la finalidad de fijar la tensión de salida en reposo del amplificador. Es decir, la tensión que se le suministre a éste terminal será la tensión que usara como punto de partida para amplificar.

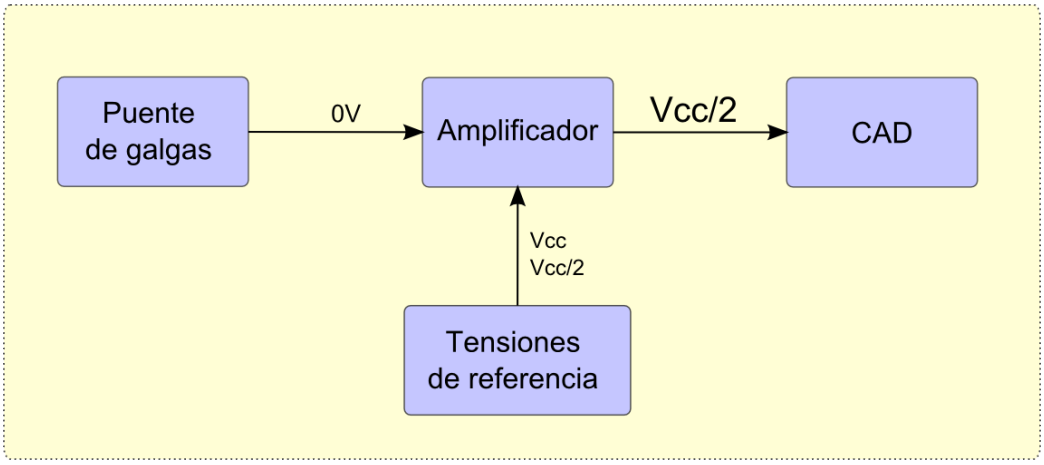


Figura 5.9: Amplificador con tensión de entrada 0V y V_{ref}

Si fijamos la tensión de alimentación a V_{cc}, V_{ref} a $V_{cc}/2$ y le suministramos a la entrada tensiones positivas, amplificara la tensión de entrada en un factor G (configurable), a ésta le sumará V_{ref} y la tensión resultante la entregará a la salida.

Análogamente, si le suministramos a la entrada tensiones negativas llevará a cabo la misma operación, con la diferencia de que, al ser una tensión negativa, una vez amplificada G veces la restará a V_{ref}, por lo que obtenemos el comportamiento deseado sin necesidad de tensiones de alimentación simétricas.

De ésta forma obtenemos un amplificador que, de forma simple y precisa amplifica la débil señal del puente Wheatstone en el rango de $\approx$ 0V hasta $<V_{cc}/2$ para tensiones positivas, es decir, cuando el material se deforma en un sentido dado, y en el rango de tensiones desde $>V_{cc}/2$ hasta $\approx V_{cc}$ cuando

[3]Circuito Integrado

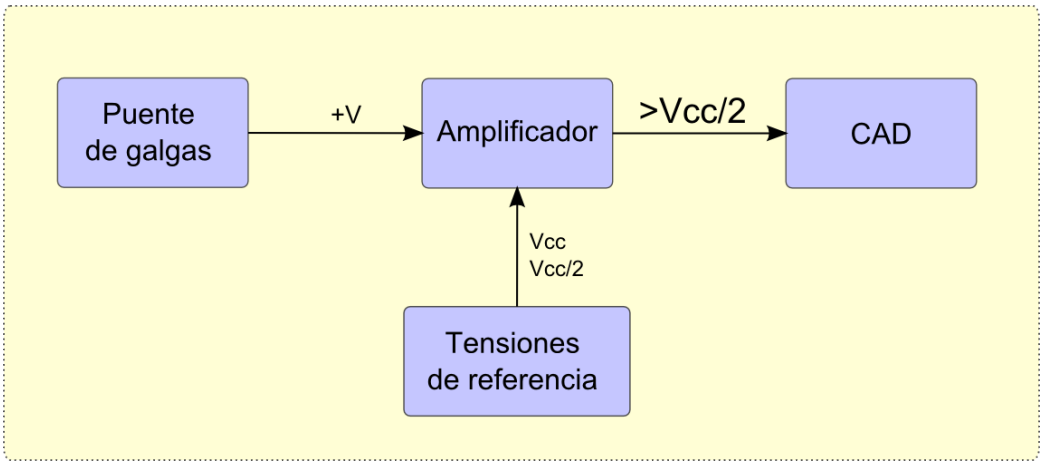


Figura 5.10: Amplificador con tensión de entrada positiva y V_{ref}

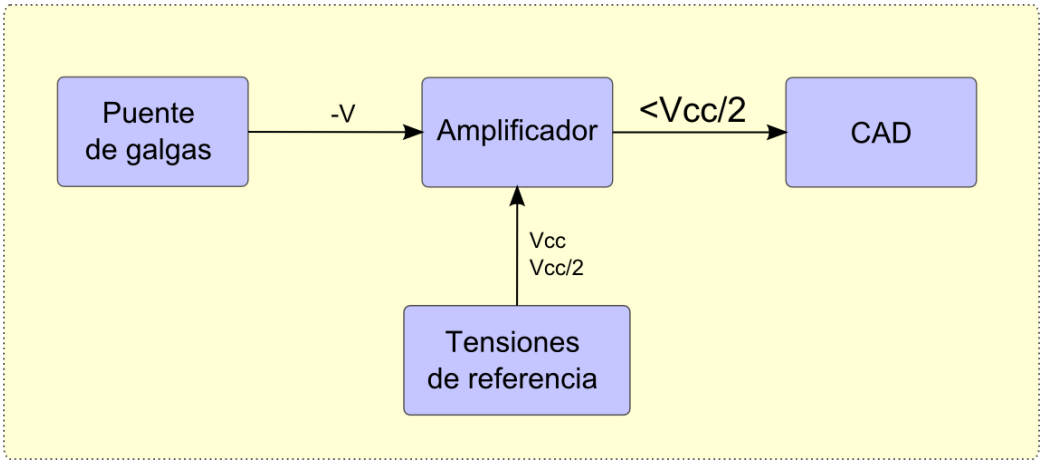


Figura 5.11: Amplificador con tensión de entrada negativa y V_{ref}

el material se deforma en sentido contrario, siendo $V_{cc}/2$ el valor medio en el que no existe ninguna deformación.

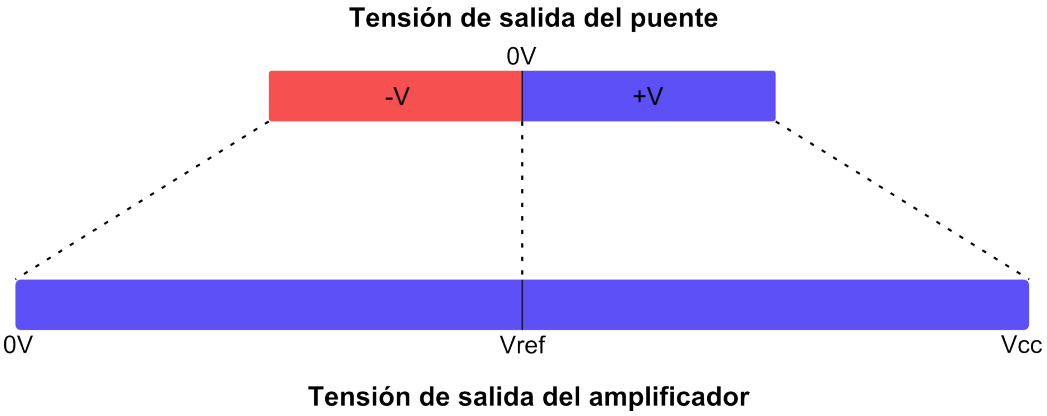


Figura 5.12: Diagrama de correspondencia entre salida del puente y del amplificador

Elección del amplificador

Dado que nuestro amplificador tendrá una entrada del orden de mV, es de gran importancia que toda la electrónica que vayamos a usar sea de gran precisión, en especial el amplificador ya que éste dispositivo además de amplificar la señal en un factor G, también amplificará todo lo que acompañe a ésta, como el ruido. Además, para nuestra aplicación es también muy importante que el amplificador consuma lo menos posible ya que la electrónica que estamos diseñando ésta destinada a ser usada en nodos inalámbricos autónomos, es decir, estará alimentada con baterias y éstas deben tener una gran autonomía. Las características básicas que debe cumplir se pueden resumir en el siguiente listado:

- Gran precisión
- Bajo consumo
- Rango de valores para la tensión de alimentación adecuado
- Factor de amplificación adecuado
- Posibilidad de inyectar una tensión de referencia

Dadas nuestras necesidades en el contexto de precisión y consumo, se ha optado por el amplificador INA333 de Texas Instruments. Éste amplificador

cubre las necesidades requeridas para nuestra aplicación. Además, éste amplificador tiene otra gran ventaja, y es que utiliza la tecnología Rail-To-Rail[4]. Ésta característica le permite entregar tensiones de salidas en el rango de desde casi 0V hasta casi V_{cc}. Ésta característica nos será muy útil ya que no nos veremos limitados por la tensión de saturación típica de los amplificadores que suele estar muy por debajo de la tensión de alimentación y que nos obligaría a alimentar el amplificador con una tensión superior para paliar éste efecto adverso.

Cálculo del factor de amplificación

El factor de amplificación en un amplificador se define como el valor que multiplicará la señal de entrada dando como resultado de la multiplicación el valor de salida del amplificador. Es decir:

$$V_{sal} = V_{ent} * G$$

Como es obvio, el CAD[5] al que se le aplicará la señal de salida del amplificador admitirá un rango determinado de tensiones de entrada. Por ello, hay que ajustar el factor de amplificación en función de la tensión que tendremos a la entrada y la que queremos obtener a la salida.

Para el cálculo del factor de amplificación hay que tener en cuenta siempre el valor máximo que es posible que vayamos a tener en la entrada. Ésto es debido a que, como es obvio, el amplificador nunca podrá entregar en V_{sal} un valor mayor que la tensión V_{cc} a la que esté alimentado el amplificador. Es decir, si ajustamos G para que en V_{sal} obtengamos el valor máximo para un valor de V_{ent} que no es el máximo que puede tomar, cuando V_{ent} tome su valor máximo $V_{ent} * G > V_{cc}$, por lo que el amplificador entrará en saturación[6]. Es por ello, que para el cálculo de G tendremos en cuenta el máximo valor de V_{ent} posible y siempre ajustaremos a la baja el factor G para evitar a toda costa que el amplificador entre en saturación, lo cual daría lugar a medidas erróneas.

Teniendo en cuenta todo ésto calcularemos el valor de G necesario para obtener en la salida una señal adecuada para que el CAD pueda llevar a cabo su función de forma correcta. Para ello tenemos que tener en cuenta que, dado que la entrada pueden ser tanto tensiones positivas como negativas, hemos fijado la tensión en la entrada de referencia del amplificador V_{ref} a $V_{cc}/2$, por lo que debemos amplificar la tensión de entrada para que $V_{ent} * G = V_{cc}/2$.

[4]Topología de interconexión entre transistores

[5]Conversor Analógico/Digital

[6]Se dice que un amplificador ha entrado en saturación cuando su salida está tan cerca de su tensión de alimentación que, por mucho que aumente la tensión de entrada, la salida se mantiene prácticamente estable

Sabiendo que $V_{cc} = 4'096V$, calculamos la ganancia necesaria dada la tensión de salida necesaria:

$$G = \frac{V_{sal}}{Vent} = \frac{V_{cc}/2}{V_{ent}} = \frac{2'048}{13'28571428 * 10^{-3}} = 154'1505377 \approx 154$$

Nota: Se redondea a la baja para evitar la saturación del amplificador

Sabiendo que según la hoja de especificaciones del amplificador $G = 1 + \frac{100K\Omega}{R_g}$, calculamos R_g para que el amplificador obtenga la ganancia deseada:

$$G = 1 + \frac{100K\Omega}{R_g} \rightarrow \frac{100K\Omega}{R_g} = 154 - 1 \rightarrow R_g = \frac{100K\Omega}{153} = 653'59477 \approx 653\Omega$$

Nota: Se redondea a la alza para evitar la saturación del amplificador

Según los cálculos necesitamos una resistencia de aproximadamente 653Ω . Sin embargo, en la práctica no existen resistencias comerciales de ése valor, por lo que tendremos que ceñirnos al valor normalizado que más nos convenga. En ningún caso dicho valor de resistencia puede estar por debajo de 653Ω ya que el amplificador podría llegar a saturar ante un valor de V_{ent} alto (tanto en sentido positivo como negativo). Por ello nos vamos a ajustar al valor normalizado situado inmediatamente por encima de 653. El valor normalizado que usaremos para la resistencia R_g será por tanto de 680Ω. Ésto condicionará el valor de las mediciones realizadas por las galgas, por lo que deberá ser tenido en cuenta.

5.1.6. CAD

Fundamentos básicos de los CAD

Como se ha explicado anteriormente, un CAD es un dispositivo que convierte una señal analógica en una señal digital de n bits. Para realizar éste conversión se emplean distintos mecanismos, sin embargo, en éste documento nos centraremos en la técnica de aproximaciones sucesivas ya que es la técnica empleada en el CAD elegido para nuestro proyecto.

Los CAD disponen de una entrada de datos analógica en donde se le inyecta la señal a convertir, y una salida digital por donde nos entrega la señal digital en binario natural correspondiente a la entrada. El valor en binario natural de la salida será el valor proporcional que representa la entrada respecto de una tensión de referencia que se le aplica al CAD a través de una entrada dedicada a tal efecto. Es decir, una tensión en la entrada $<V_{ref}$ dará como resultado un número en binario natural menor que el máximo respresentable con el número de bits disponibles en el CAD, y para una tensión

$=V_{ref}$ la salida será todos los bits a 1, siendo éste número el máximo representable por el CAD. Por ejemplo, dado un V_{ref} (el cual determinará el valor máximo que tendremos a la entrada del CAD y que se corresponderá con una salida con todos los bits a 1), y una salida de 4 bits (máximo número reprensentable=15), si V_{ent} es 0, a la salida obtendremos 0_2. En el caso de que, por ejemplo, V_{ent}=$V_{ref}/2$ a la salida obtendríamos el número en binario natural correspondiente a la misma proporción en el rango de valores representables en la salida. Es decir, como la entrada es la mitad de V_{ref}, en la salida obtendremos un valor correspondiente a la mitad del rango representable (podemos representar hasta 16 valores), que se corresponde con el séptimo valor, por lo tanto en la salida obtendremos un 7 en binario natural. Si $V_{ent} = V_{ref}$ entonces en la salida obtendremos el número máximo representable, que en nuestro caso es el número 15.

Obviamente el número de bits de los que disponga el CAD condicionará el número de valores distintos que podamos obtener en la salida y, por lo tanto, la exactitud con la que representa la señal analógica que se le está aplicando a la entrada. Dado que las señales analógicas pueden tomar infinitos valores y éstos son discretizados por el CAD, cada valor distinto que puede adoptar la salida binaria representa un rango de valores analógicos, a éste rango se le denomina *intervalo de cuantización*. Dado un rango de valores que puede tomar una entrada analógica, cuantos más bits dispongamos para representar ese rango de valores más pequeño será el intervalor de cuantización, más valores binarios distintos podremos discriminar, tendremos más resolución, y, por lo tanto, el valor digital será mas fiel a la realidad. El intervalo de cuantización o resolución también se podria describir como: la porción más pequeña de señal analógica que produce un cambio en la salida digital, y se calcula como:

$$Res = \frac{1}{2^n}$$

Donde n se corresponde con el número de bits de los que se dispone para representar la señal.

Llegados a éste punto ya disponemos de una señal amplificada de la medición que realiza el puente de galgas. El siguiente paso es convertir ésta señal analógica en una señal digital que sea compatible con las entradas E/S[7] del microcontrolador, sin embargo el CAD que lleva integrado el microcontrolador es un CAD Delta-Sigma. Los CAD tipo Delta-Sigma[8] son muy rápidos pero tienen una relación señal-ruido muy alta en comparación con otros tipos

[7]Entrada/Salida de propósito general del microcontrolador. Para realizar ésta función se podría haber optado por utilizar el propio CAD

[8]Tipo de CAD basado en la técnica de comparar sucesivamente la señal original integrada con 0 e ir restando a la señal original la componente analógica comparada

de CAD, lo cual para nuestras necesidades es inaceptable en términos de precisión. Por consiguiente, nos hemos decantado por la opción de usar un CAD *de aproximaciones sucesivas*[9] externo para realizar ésta tarea. Éste tipo de CAD es notablemente más lento que el Delta-Sigma, pero es mucho más preciso y para nuestra aplicación el tiempo de adquisición de la señal no es relevante. Añadir un CAD externo complica el diseño y la implementación de la tarjeta de adaptación, pero es una complicación asumible teniendo en cuenta la importancia que tiene la precisión en nuestra aplicación.

Para nuestra aplicación hemos optado por el CAD ADS8320 de Texas Instruments por su calidad, precisión, y bajo consumo. Éste además dispone de una salida de 16 bits que cumple con el estandar de comunicaciones SPI.

Serial Peripheral Interface (SPI)

SPI es un estándar de facto de comunicación empleado principalmente para la comunicación entre circuitos integrados en equipos electrónicos en modo maestro y esclavo donde el maestro es el que inicia la comunicación. El estándar de comunicación comprende una señal de reloj, una señal de entrada, una de salida y una de selección de chip. Éstas señales tienen las siguiente denominaciones:

- CLK → Señal de reloj (Clock)
- DI → Entrada de datos (Data In)
- DO → Salida de datos (Data Out)
- CS → Selección de chip (Chip Select)

Una conexión básica maestro-esclavo se representaría de la siguiente forma:

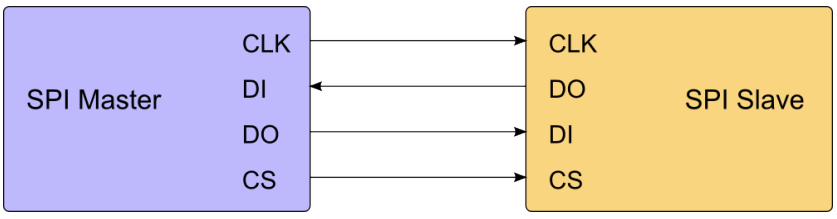


Figura 5.13: Diagrama de conexión básico SPI

El microcontrolador CC1110 que utilizamos dispone de dos controladoras USART las cuales tienen la capacidad de comunicarse mediante el protocolo

[9]Tipo de CAD que basa su funcionamiento en una búsqueda dicotómica comparando la señal a convertir con una señal de referencia que se va aproximando en cada iteración

SPI. Sin embargo, el registro empleado para almacenar los datos leídos tiene una longitud de 8 bits. Ésto plantea muchos problemas ya que nuestro CAD nos entrega señales de 16 bits, por lo que, para leer los datos del CAD mediante una USART nos veríamos obligados a realizar dos lecturas de 8 bits para obtener la palabra de 16 bits que nos proporciona el CAD. En lugar de ésto, por simplicidad se ha optado por no emplear las USART del microcontrolador. En lugar de ello se ha optado por conectar las señales de SPI a los puertos de E/S de propósito general del microcontrolador e implementar el protocolo SPI mediante software. De éste modo, podremos controlar totalmente la comunicación y hacer posible la lectura de 16 bits que requiere la comunicación con el CAD.

Implementación de la comunicación SPI

Para que la comunicación SPI se lleve a cabo con éxito hay que seguir un protocolo determinado entre el dispositivo maestro y el esclavo. En nuestro caso particular el microcontrolador asumirá el papel de maestro y el CAD asumirá el papel de esclavo. El protocolo a emplear se divide en tres fases que llamaremos: muestreo, conversión y apagado. En la fase de muestreo en primer lugar hay que deshabilitar la señal de $\overline{CS}$ (es activa a nivel bajo), a partir de ese instante la señal de reloj debe ser 0 y no puede pasar a nivel alto hasta que pase un tiempo mínimo t_{sucs}. Pasado ese intervalo de tiempo se deben generar entre 4,5 y 5 ciclos de reloj durante los cuales el CAD realiza el muestreo de la señal. Aquí terminaría la etapa de muestreo del CAD y daría paso a la etapa de conversión. En la etapa de conversión se deben generar 16 ciclos de reloj durante los cuales el CAD envía los 16 bits correspondientes a la señal digital codificada. Para leer éstos 16 bits hay que tener en cuenta que el CAD enviará un bit por cada flanco de subida que genere el reloj y que éste lo hará empezando por el bit más significativo. Una vez finalizados los 16 ciclos de reloj, si seguimos generando el reloj sin poner la señal $\overline{CS}$ a nivel alto el CAD volverá a enviar la misma secuencia de bits pero en orden inverso, es decir empezando por el bit menos significativo. Tanto si obtenemos solo la señal MSB como si esperamos a que genere también la señal LSB, al terminar de enviarnos los datos debemos proceder con la etapa de apagado. En la etapa de apagado seguiremos generando 3 ciclos más de reloj al tiempo que ponemos a 1 la señal $\overline{CS}$. Finalizados éstos 3 ciclos, el CAD estará en disposición de volver a convertir otra señal. La figura 5.14 muestra con detalle como quedarían las señales de $\overline{CS}$, reloj y salida digital en ése mismo orden.

Para llevar a cabo todo éste proceso se han usado los puertos de E/S de propósito general del microcontrolador tal y como se describe en la tabla 5.1.

Para implementar el protocolo SPI en el microcontrolador se hizo uso del *timer 1* que proporciona el microcontrolador. Dicho timer se programa para

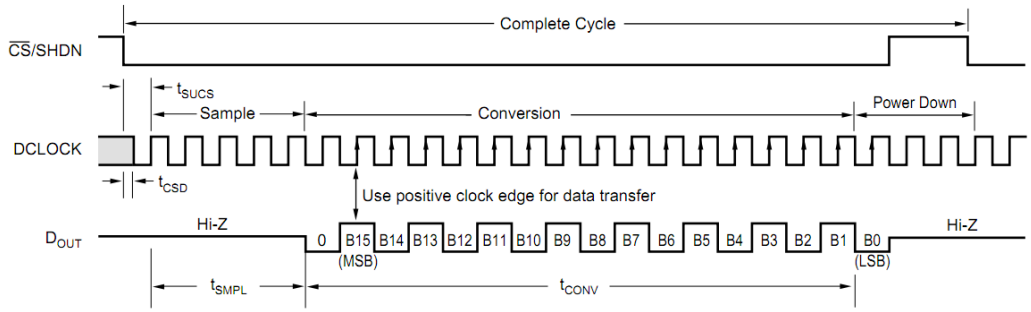


Figura 5.14: Cronograma de las señales del CAD

Puerto E/S	Propósito
P0_2	Señal de reloj
P1_0	$\overline{CS}$
P2_0	Datos digitales

Tabla 5.1: Relación de los puertos de E/S usados para el CAD

que genere aproximadamente una interrupción cada $\frac{1}{25}$ milisegundos (25kHz). Se ha elegido éste periodo porque ésta interrupción será la encargada de generar la señal de reloj para el CAD y éste admite una frecuencia mínima de reloj de 24kHz (se deja 1kHz de margen para no trabajar sobre el mínimo). Para programar el timer 1 para que genere esa frecuencia de llamadas a interrupción se asignan los siguientes valores a los registros correspondientes.

De éste modo, cuando se desee leer el valor del CAD tan sólo habrá que habilitar el timer 1 con éstos valores. La rutina de la interrupción se ha programado de forma que en cada llamada invierta el valor de P0_2 y al mismo tiempo actúe en consecuencia dependiendo del número de ciclo en el que se encuentre. Si está en los ciclos correspondientes a la etapa de muestreo no hará nada mas que generar la señal de reloj, si por el contrario está en algún ciclo correspondiente a la etapa de conversión y el valor de la señal de reloj antes de invertirla era 0 (flanco de subida), acumulará el bit de P2_0 en una variable (desplazando a la izquierda el resto de bits existentes en la variable), y por último, si se encuentra en alguno de los ciclos de apagado no hará nada (excepto activar $\overline{CS}$ en el primero de éstos ciclos). Una vez se cumplen los 24 ciclos de reloj necesarios para el protocolo SPI el timer se desactiva y la variable que acumula los 16 bits transmitidos por el CAD ya está lista para ser leída.

5.1.7. Tensiones de alimentación

En éste tipo de dispositivos en los que se usan señales del orden de milivoltios se hace imperativo la precisión de todos los componentes que vayan a intervenir de alguna forma en dichas señales. Además, en éstas situaciones es extremadamente importante la precisión de las alimentaciones de todos los componentes, ya que cualquier variación en las alimentaciones puede repercutir muy seriamente en el funcionamiento de éstos. En nuestro caso cobra aun más importancia si cabe la alimentación en el puente Wheatstone ya que cualquier variación en su alimentación provocará una variación proporcional en la salida de éste, y dada la amplitud máxima de la señal del puente ($\approx \pm 13,28mV$) no nos podemos permitir variaciones del mismo orden de magnitud o superior. Dado que los nodos están diseñados para ser alimentados con baterías nos alivia un poco la problemática ya que las baterias carecen del rizado típico de las fuentes de alimentación que transforman la señal senoidal de la red de suministro eléctrico en una señal continua. Sin embargo, las baterias no son perfectas y requieren igualmente alguna etapa de estabilización de tensión. Para ello usaremos tensiones de referencia fijas proporcionadas por unos circuitos integrados diseñados ex profeso por Texas Instruments. Éstos circuitos integrados están diseñados para proporcionar una tensión de alimentación fija y extremadamente estable. Cada modelo de integrado proporciona un valor de tensión determinado y fijo, es decir, es un valor prefijado de fábrica, por lo que existen un número limitado de valores de tensión de salida entre los que elegir. Éstos circuitos funcionan de forma muy sencilla, tan sólo hay que alimentarlos con una tensión de alimentación de como mínimo 0.2V superior a la tensión de salida para la que están diseñados y en la salida nos entregarán exactamente la tensión de referencia. Es decir, si el circuito está diseñado para entregar 5V en la salida, como mínimo deberíamos de alimentarlo con 5.2V para que funcione correctamente. En concreto usaremos las tensiones de referencia de Texas Instruments modelos REF50xx.

Para elegir las tensiones de alimentación primero debemos tener en cuenta la restricciones que nos impone nuestro circuito por su diseño. A priori el puente Wheatstone no nos impone restricción en cuanto a tensiones mínimas ya que su salida irá al amplificador y al ser ésta salida una tensión diferencial no importa en exceso el valor de la tensión a la que esté alimentado el puente ya que su salida será muchísimo menor. En todo caso nos interesa que ésta sea lo más alta posible ya que cuanto más alta sea, mas alta será la tensión de salida, lo cual hará ésta débil señal más inmune al ruido y el no deberemos amplificarla tanto en la siguiente etapa, por lo que ganaremos en precisión. Sin embargo, cuanto mas alta sea ésta tensión de alimentación más potencia consumirá el puente, y el consumo es crítico en nodos autónomos como los nuestros. Por esta razón, hemos elegido una tensión de 5V ya que, como

veremos más adelante, es la tensión más alta de entre las que necesitarán el resto de etapas para alimentarse. Podríamos haber elegido una mayor, pero ésto nos habría obligado a incluir en la tarjeta de adaptación otro circuito integrado para proporcionar ésa tensión de referencia exclusivamente para el puente Wheatstone, además de aumentar el consumo como se ha dicho anteriormente.

El amplificador INA333 puede operar con tesiones de alimentación en el rango de 1,8V y 5,5V. Éste es un dato a tener en cuenta, no obstante, el condicionante es que, por el diseño que hemos hecho, éste amplificador necesita que su entrada V_{ref} sea exactamente la mitad de su tensión de alimentación para que funcione correctamente (para tener el mismo rango de salida para tensiones de entrada negativas como positivas). Como ya hemos dicho anteriormente Texas Instrumentas proporciona un número muy limitado de tensiones de referencia distintas (es lógico ya que son tensiones elegidas en fábrica) y el único par de tensiones de referencia que cumplen $V_{ref1} = V_{ref2}/2$ son 2,048V y 4,096V respectivamente, por lo que estamos obligados a usar una tensión de alimentación para el amplificador de 4,096V (por suerte entra dentro del rango de alimentaciones en las que puede operar el amplificador) y 2,048V para la entrada V_{ref} del amplificador. Así cumplimos las restricciones que teníamos impuestas para el amplificador.

Para el CAD tenemos que puede operar en el rango de 2V hasta 5,25V. Además tenemos otra restricción proveniente de la tensión de salida del amplificador. Dado que el amplificador estará alimentado a 4,096V sabemos que como máximo su salida será muy cercana a ése valor, pero nunca se llegará a dar. El CAD necesita dos tensiones estables: la de alimentación, y la de referencia. La tensiones de referencia es la que usará el CAD para definir como valor máximo de entrada, es decir, se corresponde con el valor de tensión de entrada para el que la salida digital será todo 1. En nuestro caso ésta tensión de referencia deberá corresponderse con el valor máximo que nos puede entregar el amplificador. Aunque, como hemos dicho anteriormente, el amplificador teóricamente no llegará nunca a entregar la tensión de alimentación de 4,096V, asumiremos que puede llegar a estar tan cerca que esa será la tensión máxima que le puede llegar a la entrada del CAD, por lo que la entrada V_{ref} del CAD la fijaremos a 4,096V. Dado que la tensión de alimentación del CAD debe estar por encima de la tensión de referencia alimentaremos a éste con 5V ya que es la tensión inmediatamente superior a 4,096V que nos proporciona Texas Instruments. Y con ésto tenemos todas las tensiones estables cubiertas por lo que el diseño de nuestro circuito ya se puede dar por terminado.

5.2. Protocolo de comunicación de la red de sensores

En el Manual del programador del Anexo B.1 se presentan principalmente bibliotecas implementadas para el manejo de las estructuras de datos creadas para el protocolo de comunicación. No en vano el desarrollo de esas estructuras y las bibliotecas han ocupado gran parte del tiempo de desarrollo del proyecto. La otra parcela lógica que se ha llevado gran parte del peso temporal dedicado a la implementación ha sido la comunicación de los nodos, cuya descripción se presenta a lo largo del Tema 3.

En esta sección, por otra parte, se pretende ver de cerca algunos de los algoritmos más representativos que se han desarrollado así como una visión detallada de algunas de las estrategias tomadas en la programación de los nodos.

5.2.1. Implementación de la comunicación en los nodos

Más allá de las estructuras usadas y de las bibliotecas, la clase principal de los nodos que componen la red implementa la comunicación en una serie de bucles que comprueban periódicamente los flags de recepción e ID para situarse en el diagrama de comunicación.

Estos flags conforman una parte importante de la implementación del protocolo de red y por ende, de los nodos. Los bucles iteran con uno de los temporizadores como guarda (además de otras condiciones dependiendo del punto de ejecución en el que se encuentren) del mismo y comprobarán el flag de recepción para ver si se ha recibido algún paquete.

El flag de recepción es un byte que cambia de valor cuando se ejecuta una interrupción de la radio. Durante dicha interrupción también se procesa el paquete almacenándolo en la estructura que facilita el SimpliciTI llamada *ioctlRawReceive_t* y se extrae el valor del ID (los primeros cuatro bits del primer byte recibido) para almacenarlo en otro flag. Finalmente se vacía el buffer de recepción ya que podemos considerar que se ha terminado el procesamiento del paquete.

Cuando se termina la interrupción y se vuelve al programa principal se comprobará de nuevo el flag de recepción, se encontrará el flag de recepción activo y se procederá a desactivarlo y a ejecutar la acción que corresponda según el diagrama de comunicación presentado en el tema 3. El flag ID se utiliza en caso de que se esté en un estado susceptible de recibir distintos tipos de paquetes, utilizándolo por ejemplo como condición de una clausula *case*.

5.2.2. Algoritmos representativos

Formación del grafo a partir de las Tablas de Intensidades

Este proceso se ejecuta siempre que termina la etapa de formación de la red y no se repite hasta que se reinicie la red por algún motivo. Formar el grafo, por tanto, **solo** se produce una vez. El resto de operaciones son de alteración de los atributos, el cálculo de los caminos mínimos sobre el grafo construido o la eliminación de alguno de los vértices si éste ha producido algún error.

Algoritmo 1 Formación del grafo

Entrada: Grafo inicializado (grafo) y las Tablas de Intensidades formadas (iTable)
Salida: Grafo formado con la información de las Tablas de Intesidades

1: **para** $i = 0$ hasta iTable.length (número de tablas almacenadas) **hacer**
2: idO $\leftarrow 0$
3: **si** (idO = indice de del nodo iTable[i]) >grafo.length **entonces**
4: idO = inserta un nuevo vértice con la dirección de iTable[i] y devuelve su ID
5: **fin si**
6: **para** $j = 0$ hasta iTable.iTable[i].length (número de filas en la tabla) **hacer**
7: A = nodo A (tabla «i») respecto al nodo B (fila «j»)
8: $B.RSSI$ = nivel RSSI con que el nodo B ha registrado al nodo A
9: $B.batera$ = nivel de batería del nodo B
10: $pond = A.RSSI * B.RSSI - |A.RSSI - B.RSSI| + A.batera * B.batera - |A.batera - B.batera|$
11: Inserta un nuevo arco de A (idO) a B con valor pond (inserta el vértice B si no existe)
12: **fin para**
13: **fin para**

Para ponderar un arco se debe tener en cuenta tanto el nivel de batería de ambos nodos, como la intensidad de señal con la que ambos nodos se registraron entre sí. Es por este motivo por el que, aunque quizá queda algo ambiguo en el pseudocódigo mostrado en el algoritmo 1, se hace una búsqueda para extraer los datos del nodo «B» antes de ponderar el arco.

En la formación del grafo se utilizan las funciones de la biblioteca de *nodeGprah.h* descrita en la sección B.1.2. Por medio de su uso se intenta facilitar precisamente la formación del grafo según el algoritmo descrito, por esto algunas de las búsquedas incluyen la inserción del nodo si este no se encuentra aún en el grafo. Y de ahí a su vez que antes de insertar un nuevo

vértice se compruebe si ya existe o no en la línea 3 del algoritmo anterior.

Esto se traduce en un varios bucles anidados más en cada búsqueda, cortocircuitados para intentar optimizar el recorrido de búsqueda. De modo que el coste temporal resultante del algoritmo es relativamente elevado. Esto sin embargo no supone ningún problema en la aplicación de la red de sensores. El algoritmo se ejecuta en un punto entre la formación de la red y la etapa estándar en la que la criticidad del tiempo es nula, y más teniendo en cuenta que las limitaciones energéticas no están presentes en el caso del nodo sumidero.

Una vez formado del grafo se calculan los caminos mínimos y se envía toda la información a la estación base a través de la UART.

Programación del *macro-frame*

Este es el algoritmo recursivo que programa el *macro-frame* por ramas según lo explicado en el punto 3.2.6.

El caso base de dicho algoritmo es encontrar un vértice sin enlaces por procesar. La ejecución se produce exactamente como se ha relatado en el punto 3.2.6, bajando de la raíz a las hojas y luego subiendo por la rama del árbol programando marcos temporales.

Algoritmo 2 Programación del *macro-frame*

Entrada: Vértice raíz
Salida: Array de direcciones con la programación y número total de frames

```
    nodesInRoute ← 1
2:  para i = 0 hasta i < longitud de la lista de arcos del nodo raíz hacer
        nodeAct ← nodo «i» de la lista de arcos
4:      si El punto de interés de nodoAct apunta a ráiz entonces
            countSubTree = llamada recursiva a la función con nodoAct como
            raíz
6:          Añadir el enlace en el array de programación
            nodesInRoute = nodesInRoute + countSubTree
8:      fin si
    fin para
10: devolver nodesInRoute
```

Una de las ventajas del algoritmo es que el parámetro de entrada de nodo raíz no tiene necesariamente que coincidir con el nodo raíz del árbol (que sería el sumidero por definición). Así, si se produce algún error y se programa un *macro-frame* de emergencia, como ya se ha indicado con anterioridad, solo se programa para el subárbol que cuelga del nodo que ha producido el error.

De modo que si la llamada a la función desde el código principal se hace con el nodo que produjo el error se obtendrá la programación por ramas del subárbol que cuelga de dicho nodo. Posteriormente tan solo haría falta completar la ruta de puntos de interés desde el nodo que produjo el fallo hasta el nodo sumidero.

5.2.3. Estrategias de programación

Retroceso Exponencial Binario

Como ya hemos indicado en el punto 3.2.2 el paquete de contestación al descubrimiento introduce un retardo mediante un algoritmo similar al de Retroceso Exponencial Binario de Ethernet y precisa de la detección de portadora para hacer efectivo el envío.

El retardo se determina mediante una cantidad aleatoria de medidas temporales, que viene a su vez determinada en módulo por el algoritmo de retroceso exponencial binario mediante la fórmula:

$$R = Random\_Number \text{ mód } (2^k - 1)$$

Donde k es el número de reintentos y $Random\_Number$ es el número aleatorio generado.

En el microcontrolador del CC1110 presenten en los nodos **WSNVAL** existen registros para la generación del número aleatorios que se utilizan en este caso concreto. La entropía necesaria para la generación de números aleatorios se obtiene por defecto de los registros del CAD. Esta opción sin embargo resulta tan solo en número pseudo-aleatorios.

La otra opción para la obtención de la entropía es la de escribir manualmente un registro del microcontrolador dos veces para determinar una semilla. En la generación de números aleatorios participan dos registros: RNDH y RNDL. Cada vez que se escribe en RNDL se copia su valor a RNDH. Así, al escribir dos veces establecemos primero la parte de mayor peso de la semilla y posteriormente la de menor peso.

Para establecer la semilla que se escribirá en los registro del microcontrolador se utiliza una lectura de las galgas extensiométricas como fuente entrópica. Posteriormente para obtener un número aleatorio partiendo de la semilla establecida se hace uso de la función *MRFI_RandomByte()* del SimpliciTI.

Este proceso, junto con la detección de portadora, se repite por completo en cada intento de envío.

5.3. Aplicación puente USB-IGU

La aplicación puente es una aplicación que reside en el terminal al que está conectado el nodo sumidero. Ésta aplicación se encarga de recoger los datos que provienen de la red de sensores a través del sumidero y volcarlos en una base de datos para que la IGU posteriomente pueda consultarlos con el fin de mostrarlos al usuario. Además, la aplicación puente actua de intermediario entre la IGU y la red de sensores, transmitiendo las señales de control de la red con el fin de que el usuario pueda interactuar con la misma. En definitiva la aplicación puente actua como nexo de unión entre la red de sensores, la base de datos y la IGU para que éstos tres elementos funcionen de forma coordinada.

A nivel de implementación cabe destacar que la aplicación puente tan sólo necesita residir en el mismo terminal en el que se encuentra conectado el nodo sumidero. La base de datos puede residir fisicamente en otra maquina siempre y cuando desde el terminal donde se encuentra el puente se pueda acceder a la base de datos a través de cualquier tipo de red, incluyendo el acceso a través de Internet. Dado que la IGU se implementa como una página HTML+PHP que extrae los datos a través de la base de datos y se comunica con la aplicación puente mediante WebSocket, el servidor HTTP que sirve la página correspondiente a la IGU puede de igual forma residir en otra máquina distinta a la de la aplicación puente y a la de la base de datos, con la misma condición respecto de la conectividad citada para la base de datos. A su vez, la IGU puede ser consultada desde cualquier terminal que tenga un navegador con soporte para CSS+JavaScript+WebSocket. En definitiva, gracias al diseño de la arquitectura Puente+BBDD+HTTP, obtenemos un sistema altamente distribuido que nos permite una gran flexibilidad.

5.3.1. Descripción del funcionamiento

En primer lugar la aplicación puente trata de abrir el puerto USB para establecer el canal de comunicación con el sumidero. Obviamente si no lo consigue la ejecución falla y devuelve el control a la consola del SO[10]. En el caso de establecer con éxito el canal de comunicación con el nodo sumidero lanza el hilo de ejecución *conexionIGU* que se encarga de escuchar las peticiones de conexión WebSocket en el puerto 9321. Posteriormente se lanza otro hilo de ejecución llamado *consolaUsuario* que será el encargado de gestionar las ordenes que el usuario introduce desde la consola de comandos. Una vez lanzados éstos dos hilos el hilo principal de ejecución llama a la función *initListadoNodos* que pone a 0 el campo *estado* de todas las tuplas de la tabla *nodos* de la BBDD y queda bloqueado en el semáforo *mutexInicioRed*.

[10]Sistemas Operativos

Éste semáforo sólo se libera cuando desde la consola de usuario o desde alguna IGU se ordena inicializar la red. Cuando ésto sucede el hilo principal vuelve a la ejecución y queda a la espera de datos desde el puerto USB. A partir de éste momento comienza el procesamiento de paquetes provenientes del sumidero.

Cuando llega un trama desde el puerto USB en primer lugar se analiza para determinar el tipo de trama recibida. Si la trama corresponde a la recepción del listado de nodos se llama a la función *getDatosNodos* para que desensamble el paquete y organice los datos un vector de estructuras de tipo *Nodo*. Luego para cada nodo de la estructura comprueba si existe en la BBDD, si no existe lo inserta en la tabla *nodos* con el campo *estado* fijado a 2, de lo contrario lo activa mediante la función *activarNodo*. La función *activarNodo* pone a 1 el campo *estado* del nodo que está procesando en ésa iteración, dado que inicialmente todos los nodos de la tabla *nodos* tenian el campo *estado* a 0, los nodos que ya existian se han activado poniendo a 1 dicho campo y los nodos nuevos se han insertado con el mismo campo a 2 ésta metodología permite saber qué nodos existieron en algún momento en la red y ya no están disponibles (estado=0), los nodos que ya existían y siguen estando disponibles (estado=1) y los nodos nuevos (estado=2).

Si la trama recibida es una trama de listado de nodos pero con longitud 1 (trama vacía, solo contiene el campo identificador) significa que ha terminado el listado de nodos, por lo que se llama a la función *notifyIGU* para notificar a las IGU del suceso.

Si la trama que recibe la aplicación puente es una trama de recepción de medidas se llama a la función *getDatosMedidasNodos*, la cual desensambla el paquete y organiza la información de los nodos en un vector de estructuras de tipo *Nodo* para posteriormente insertar dichas medidas en la BBDD mediante la función *updateMedidas*. Posteriormente se invoca a la función *notifyIGU* que notifica a las IGU la recepción de medidas para que éstas invoquen al script PHP con el fin de obtener los datos de las medidas de la BBDD.

```
typedef struct {
     unsigned char addr[4];
     unsigned char flags;
     unsigned short medida;
     unsigned char addrPadre[4];
     char nivelRSSI;
     unsigned char battery;
} Nodo;
```

Hilo de escucha WebSocket

El hilo de escucha de conexiones WebSocket abre un socket en el puerto 9321 y se queda a la espera de conexiones WebSocket. Dado que pueden coexistir múltiples conexiones WebSocket el hilo las organiza en un *vector<int>* donde almacena los descriptores de fichero de cada conexión abierta. Adicionalmente, se almacena en un *vector<pthread_t>* los identificadores de los hilos correspondientes a cada conexión abierta. De ésta forma se almacenan dichos datos para poder manejarlos con más facilidad. Éstos vectores se inicializan en el hilo principal y son proporcionados mediante parámetro al hilo *conexionIGU* ya que el hilo que controla la consola de usuario los necesitará parar cerrar las conexiones WebSocket de forma ordenada en el momento en el que el usuario ordene el cierre de la aplicación.

En cuanto se recibe una conexión al puerto WebSocket el hilo *conexionIGU* añade el descriptor de fichero al vector de descriptores de sockets, crea otro hilo de tipo *listenClient* para que gestione esa comunicación y agrega el identificador del hilo al vector de identificadores de hilos. A cada hilo que crea para cada conexión le proporciona por parámetro una estructura de tipo *paramThListenClient* que contiene los vectores ántes citados y algunas variables necesarias para la ejecución de dichos hilos. Posteriormente el hilo *conexionIGU* extrae las cabeceras WebSocket e invoca a la función *createHeaderForSend* pasandole como parámetro dichas cabeceras. Ésta función se encarga de analizar las cabeceras y devolver las cabeceras correctas que se deben enviar como respuesta a la petición WebSocket. Una vez obtenidas las cabecera de respuesta el hilo *conexionIGU* las envia y nuevamente queda a la espera de nuevas conexiones. Cabe aclarar que se crea el hilo *listenClient* ántes de devolver las cabeceras de respuesta al cliente WebSocket. Ésto es debido a que el cliente WebSocket enviará un caracter determinado que determinará la validez de la conexión en el momento en el que reciba las cabeceras de respuesta. Éste caracter lo debe recibir el hilo *listenClient* por lo que éste debe estar ya en ejecución y listo para recibir datos en ese momento. Es por ésta razón por la que se anticipa la creación del hilo *listenClient* al envio de las cabeceras de respuesta por parte del hilo *conexionIGU*.

Hilo para la gestión de cliente WebSocket

Éste hilo corresponde con el tipo *listenClient* y es lanzado uno por cada conexión WebSocket que recibe el hilo *conexionIGU*. Éste hilo se encarga de gestionar la comunicación con un cliente WebSocket y recibe como parámetro una estructura de tipo *paramThListenClient* la cual contiene el *vector<int>* que contiene los descriptores de todos los sockets WebSocket activos, el *vector<pthread_t>* que almacena los identificadores de los procesos que escuchan dichas conexiones WebSocket y un puntero al entero que lleva la cuenta del

número de conexiones WebSocket activas.

Nada mas iniciarse el hilo lee tres caracteres del la conexión WebSocket que está gestionando. Éstos tres caracteres deben corresponderse con la secuencia *0x00, 0x76 y 0xFF*. El primer y último caracter se corresponden con los caracteres de inicio y fin de trama del protocolo WebSocket (para más información ver la sección *Comunicación Puente-IGU*), y el segundo debe corresponderse con el caracter UTF-8 0x76 ('L'), el cual identifica una conexión WebSocket procedente de un IGU válido. Si éstos caracteres son correctos el hilo envía el estado de la red almacenado en la variable global *estadoRed*. Ésto sirve para que al conectarse una IGU sepa en qué estado se encuentra la red para que actue en consecuencia. La variable global *estadoRed* es una variable que la aplicación puente va modificando según el punto de ejecución en el que se encuentra en función de las tramas que recibe por parte del sumidero. Si la trama de validación es correcta incrementa el contador de clientes activos a través del puntero a entero que recibió como parámetro.

Posteriormente el hilo queda a la escucha de ordenes por parte de la IGU. Las distintas ordenes que puede recibir por parte de la IGU se distinguen con el segundo byte de la trama recibida (el primero es el caracter 0x00, marca el inicio de trama en WebSocket), y pueden ser las que se observan en la tabla 5.3.1. A su vez, en la tabla 5.3.1 se detallan los valores de las constantes mostratas en la tabla 5.3.1.

Código (2º byte)	**Descripción**
INICIAR_RED	Envía al sumidero la orden 0x02
ENVIAR_CONF	Envía al sumidero la orden 0x03 junto con el periodo de medición, los flags y los tres umbrales
MED_BAJO_DEMANDA	Envía al sumidero la orden 0x04
REINICIAR_RED	Envía al sumidero la orden 0x05

Constante	**Valor**
INICIAR_RED	0x49 ('I')
ENVIAR_CONF	0x43 ('C')
MED_BAJO_DEMANDA	0x4D ('M')
REINICIAR_RED	0x52 ('R')

En el caso de que la trama recibida por parte de la IGU sea la secuencia *0xFF, 0x00* significa que el cliente WebSocket quiere iniciar la desconexión, por lo que se responde al cliente con *TRAMA_FINALIZACION_WEBSOCKET* (0xFF) y se invoca a la función *cerrarConexionWebSocket* para que ésta elimine los datos de ésta conexión del vector de descriptores de socket y del vector de identificadores de hilos.

Hilo de la consola de usuario

El hilo de la consola de usuario es el encargado de recibir y procesar las ordenes que introduce el usuario a través de la consola de comandos de la aplicación puente. Desde ésta consola el usuario puede ordenar el inicio de la red manualmente o bien terminar la ejecución de la aplicación. Éste hilo recibe como parámetros una estructura de tipo *Clientes* que contiene el *vector<int>* que almacena los descriptores de los sockets WebSocket abiertos y el *vector<pthread_t>* que contiene los identificadores de los hilos que escuchan éstos sockets ya que, si el usuario ordena la terminación de la aplicación, éste hilo invoca a la función *cerrarConexionWebSocket* para cerrar de forma ordenada éstas conexiones y que, por lo tanto, requiere dichos vectores.

5.3.2. Comunicación USB-puente

Dado que la aplicación puente siempre reside en el mismo terminal en el que se encuentra conectado el nodo sumidero a través de un puerto USB. La comunicación entre ambos se realiza a través de dicho puerto. Para ello hay que tener en cuenta que el microcontrolador del nodo sumidero emplea una USART para la comunicación. Ésto nos proporciona un flujo de datos en serie pero no es compatible con USB. Para permitir que el nodo pueda ser conectado a un puerto USB el mismo incorpora un puente USART-USB externo al microcontrolador que convierte el flujo de bits de la USART0 del microcontrolador en un flujo compatible con USB. Sin embargo, a bajo nivel el flujo de datos debe ser tratado como si de una UART se tratase. Es por ello que para establecer una comunicación con el nodo sumidero no basta sólo con abrir el descriptor de fichero correspondiente al puerto USB, hay que preparar ciertos parámetros de configuración tanto en el emisor como en el receptor para que el flujo de datos se lea y escriba de forma correcta como se muestra en el siguiente fragmento de código de la aplicación puente en el que se establecen distintos parámetros como el tamaño de caracter (flag *CS8*) y la velocidad de la conexión (115200 baudios).

```
newtermios.c_cflag = CBAUD | CS8 | CLOCAL | CREAD;
newtermios.c_iflag = IGNPAR;
newtermios.c_oflag = 0;
newtermios.c_lflag = 0;
newtermios.c_cc[VMIN] = 1;
newtermios.c_cc[VTIME] = 0;

cfsetospeed(&newtermios,B115200);
cfsetispeed(&newtermios,B115200);
```

En el lado del microcontrolador hay que especificar parámetros similares. Ésto se hace al inicializar la UART con la función *initUART()* de la libreria *UART.c*.

```
U0UCR = 2;          // Stop bit = 1
U0GCR &= ~16;       // Order = LSB
U0GCR |= 13;        // BaudE = 13
U0BAUD = 34;        // BaudM = 34
```

Establecidos éstos parámetros la aplicación puente y el nodo sumidero ya pueden intercambiar flujos de bytes. Sin embargo, no se pueden intercambiar tramas sin más, ya que la recepción de datos tanto en el lado del microcontrolador como en el lado de la aplicación puente presentan ciertas problemáticas.

Recepción de datos en el lado del nodo

En el microcontrolador la lectura de datos se realiza poniendo a 1 el bit *RE* del registro *U0CSR*. Ésto pone en modo recepción a la USART0. A partir de ése instante cada byte que llegue se almacenará en el registro *U0DBUF*. Hay que tener en cuenta que el registro *U0DBUF* tiene un tamaño de un byte, por lo que tan sólo almacena el último byte recibido por la USART0. Ésto puede plantear problemáticas a la hora de recibir mas de un byte ya que obliga a que la lectura sea bloqueante, además de que puede ocasionar problemas si no se lee a tiempo el registro *U0DBUF* ya que podemos acabar perdiendo bytes de la trama. Por lo tanto, éste modo sólo es adecuado si esperamos recibir un sólo byte de datos (ordenes enviadas por la aplicación puente). Éste modo es el que emplea la función *UARTReceive(ENABLE_RX_BLOQ)*.

Otra forma de recibir datos a través de la USART es emplear interrupciones. Si activamos el bit 2 del registro *IEN0* la USART0 generará una interrupción cada vez que se recibe un byte, además de almacenarlo en *U0DBUF*. De ésta forma la interrupción puede guardar los datos en un vector de bytes a medida que los recibe sin necesidad de hacer uso de llamadas a función bloqueantes y sin peligro de perder bytes. Éste es el modo que emplea la función *UARTReceive(ENABLE_RX_INT)*.

Cuando se termina de recibir una trama la función *UARTReceive(ENABLE_RX_INT)* automáticamente desactiva la generación de interrupciones de la USART0 para que haga caso omiso de cualquier dato inesperado. Sin embargo, puede darse el caso de que se active la recepción de datos por interrupción pero nunca llegue ningun dato y, pasado un tiempo de espera prudencial, se necesite dejar de esperar datos. Para ello se implementa *UARTReceive(DISABLE_RX)* cuya única finalidad es la de desactivar las

interrupciones de recepción de datos de la USART0 y así dejar de esperar datos.

Hemos dicho que la función *UARTReceive(ENABLE_RX_INT)* desactiva las recepción de datos cuando recibe una trama completa. Sin embargo, la USART0 sólo contempla flujos de datos, no existe un tamaño de trama, ni delimitador ni nada que permita conocer cuántos datos van a ser leidos. Para conocer de antemano cuántos datos van a ser leidos por la USART0 y de ésta forma saber cuando termina una trama se empla un byte adicional que es insertado por la aplicación puente ántes de transmitir la carga útil de la trama. De ésta forma el primer byte que reciba la rutina de interrupción (la primera llamada a ésta rutina desde que se activó la recepción) se asumirá que contiene el tamaño de los datos que se recibirán a continuación. De éste modo, la rutina irá contando las veces que se ejecuta (es equivalente a los bytes que recibe) y cuando ése contador llegue al valor que tenia el primer byte del flujo de datos dará por finalizada la trama. A continuación se muestra un fragmento de la rutina de interrupción que implementa parte de éste mecanismo.

```
if(packetSize == 0) {
  packetSize = U0DBUF;
} else {
 buffer[iBuffer++] = U0DBUF;
...
```

Recepción de datos en el lado de la aplicación puente

En el cliente la recepción de datos no es tan problemática como en el caso del microcontrolador ya que las librerias de C nos otorgan muchas facilidades y una abstracción suficiente de las problemáticas que existen a bajo nivel. Dado que la aplicación puente está pensada para ejecutarse en SO derivados de UNIX, donde los dispositivos se representan en el arbol de ficheros como un fichero más, se hace uso de la función POSIX *read* para la lectura de ficheros con el fin de leer el puerto USB donde se encuentra conectado el nodo sumidero, en nuestro caso el fichero */dev/ttyUSB**x***, donde ***x*** es un número entero determinado. De éste modo la lectura del puerto USB resulta trivial. El único problema que tenemos en la recepción de datos es el mismo que teníamos en el lado del microcontrolador. La función *read* requiere que se le proporcione mediante un parámetro el número de bytes a leer. Obviamente la solución pasa por la misma solución que en el caso del microcontrolador. A la hora de enviar, la función *UARTSend* usada para enviar datos desde el microcontrolador a la aplicación puente enviará primero dos bytes especificando el tamaño de los datos que enviará posteriormente, y luego enviará la carga útil de la trama. En el lado de la aplicación puente primero se invocará

a la función *read* para que lea dos bytes, una vez ésta función retorna los dos bytes recibidos se invocará nuevamente a la función *read* pero ésta vez se le pasará como tamaño de datos a leer el valor que devolvió la primera llamada a read. De ésta forma podemos recibir tramas de un tamaño arbitrario mediante funciones *read* POSIX. Obviamente, ésto nos limita a un tamaño de trama de 2^{16}, pero dado que la trama mas grande que se enviará corresponderá al listado de nodos y éste está limitado a un nodo por trama por la limitación de memoria del microcontrolador no tendremos problemas en ese aspecto.

Envio de datos en el lado del microcontrolador

El envío de datos desde el microcontrolador es trivial. Para enviar un byte de datos tan sólo tenemos que poner el bit *RE* de *U0CSR* a 0 para asegurarnos de que la USART0 no está en modo de recepción y escribir en el registro *U0DBUF* el byte que queremos enviar. Para enviar otro byte consecutivo primero debemos esperar a que el bit 2 (TX_BYTE) del registro *U0CSR* se ponga a 1 indicando así que el byte ha sido transmitido para posteriormente ponerlo a 0 manualmente y entonces escribir el siguiente byte en *U0DBUF*. A continuación vemos un fragmento de código del envío de datos a través de USART0.

```
U0DBUF = tam >> 8;          // Transmite el byte de mas peso
while(!(U0CSR & 2));        // Espera hasta que el byte se transmita
U0CSR &= ~2;                // Limpia U0CSR.TX_BYTE

U0DBUF = tam;               // Transmite el byte de menos peso
while(!(U0CSR & 2));
U0CSR &= ~2;


for(int i=0; i<tam; i++) {
     U0DBUF = datos[i];

     while(!(U0CSR & 2));
     U0CSR &= ~2;
}
```

Éste mecanismo es el que implementa la función *UARTSend*. Dado un puntero de bytes y un entero de 16 bits envía primero dos bytes que se corresponden con la parte baja y la alta de la variable de 16 bits pasada por parámetro y luego envía los datos apuntados por el puntero a caracteres. Una trama enviada a éste nivel tendrá el formato que se observa en la figura 5.15.

Figura 5.15: Trama enviada por la UART

Envio de datos en el lado de la aplicación puente

Para enviar datos desde la aplicación puente el proceso es similar al seguido en el lado del microcontrolador. Primero se envía un byte que contiene el tamaño de los datos que se van a enviar inmediatamente después, y luego se envian los datos útiles tal y como se muestra en la figura 5.16.

Figura 5.16: Trama enviada por la aplicación puente

Protocolo de comunicación

Cuando la aplicación puente se ejecuta ésta abre el puerto USB donde está conectado el sumidero y se queda a la espera de la orden de inicialización desde la consola o bien desde la IGU. En éste momento tanto el sumdiero como la aplicación puente quedan a la espera. Cuando se recibe la orden de inicialización de la red el sumidero envía una trama al sumidero notificando la inicialización de la red e inmediatamente envía una trama que contiene los parámetros de configuración que deben adoptar todos los nodos de la red. Es necesario enviar éstos parámetros de configuración en el momento de inicializar la red ya que los nodos no conocen ésta configuración que está guardada en la base de datos o bien que ha sido modificada por el usuario en la IGU antes de ordenar la inicialización de la red.

Cuando el nodo sumidero recibe éste par de tramas comienza la etapa de formación de la red. Mientras tanto la aplicación puente queda a la espera de recibir por el puerto USB el listado de nodos que tendrá en posesión el sumidero cuando termine la etapa de formación de la red. Cuando ésta etapa de la red termina el nodo sumidero enviará las tramas que contienen la información sobre los nodos. Éstas tramas forman un listado de los nodos activos que existen en la red. Éste listado es necesario ya que durante la etapa estándar de la red es posible que algunos nodos no emitan ninguna trama por lo que el único momento en el que todos los nodos transmiten sus datos necesariamente es durante la etapa de formación de la red, y la aplicación

puente debe recoger un listado de nodos para que pueda ser mostrado en la IGU ya que el usuario debe saber en todo momento de la existencia de la totalidad de los nodos, no sólo los que transmitan datos en un determinado instante de tiempo.

La aplicación puente debe saber el momento en el que termina la etapa de formación de la red para envíar una notificación a la IGU. No obstante, dado que el sumidero puede enviar el listado de nodos mediante multiples tramas, éste debe notificar a la aplicación puente de alguna forma que el listado de nodos ha terminado y que, por lo tanto, la etapa de formación de la red también lo ha hecho. Para ello el sumidero envía un paquete con el mismo identificador que los paquetes que corresponden al listado de nodos pero con la diferencia de que éste estará vacio. Con ésto el sumidero informa de que la etapa de formación de la red ha finalizado y se procede a iniciar la etapa estándar.

Una vez la aplícación puente detecta que la etapa de formación de la red ha finalizado se queda a la espera de tramas que contengan información sobre las mediciones de deformaciones que están realizando los nodos. Éstas tramas pueden contener un número arbitrario de nodos, sin embargo, por razones de consumo de memoria en el nodo sumidero éste número se ha limitado a un nodo por trama.

Cuando la aplicación puente llega a éste punto se queda indefinidamente esperando tramas que contienen datos de medidas de los nodos y procesandolas para que la IGU las pueda mostrar por pantalla. Sin embargo, en éste punto de la ejecución puede darse el caso de que la IGU solicite realizar acciones sobre la red de nodos. Éstas acciones pueden ser: enviar nuevos parámetros de configuración a la red, solicitar una medición bajo demanda o reiniciar la red. En cualquiera de éstos tres casos la aplicación puente envía un paquete especial que contiene un identificador distinto para cada acción a realizar.

Cabe destacar que la aplicación puente a nivel de implementación no distingue las etapas en las que se encuentra la red, es decir, la aplicación se limita a esuchar el puerto USB y realiza una acción u otra en función del identificador de la trama que recibe, por lo que si recibiera una trama con un identificador que se correpondiera con una trama de listado de nodos, cuando supuestamente la red se encuentra en la etapa estándar, la aplicación procesaria sin problemas dicho paquete y no causaría un mal funcionamiento de la aplicación ni ninguna inconsistencia en la BBDD. Ésta característica permite a la aplicación puente reaccionar ante un reinicio de la red sin tener que realizar ninguna acción especial.

Recepción del listado de nodos

Las tramas correspondientes al listado de nodos las transmite el nodo sumidero al terminar la etapa de formación de la red. La trama se divide en dos partes, la cabecera que contiene únicamente el identificador del paquete y la carga útil que se compone de los datos un número arbitrario de nodos. Como se puede observar el protocolo ofrece la posiblidad de enviar los datos de varios nodos en una misma trama, sin embargo, debido a la limitación de memoria del nodo sumidero se ha restringido a un nodo por trama. La red soporta hasta 2^{32} nodos, lo que supone que para que el nodo sumidero envie los datos de todos los nodos en una trama en el peor de los casos necesitaria primero almacenar en memoria la trama a enviar para posteriormente llevar a cabo el envío. Por cada nodo se necesita almacenar en memoria 4 bytes de la dirección MAC del nodo, 4 bytes de la dirección MAC del nodo padre, 1 byte del nivel de RSSI y otro byte para el nivel de bateria. Ésto implica que para cada nodo que se envíe en la trama se necesitan 10 bytes de memoria, por lo que en el peor de los casos (con 2^{32} nodos en la red) el nodo sumidero debería guardar en memoria $10 * 2^{32}$ bytes, o lo que es lo mismo 40GB de memoria. Sería absudo pues guardar en memoría tal cantidad de datos para luego envíarla por lo que se ha decidido limitar a un nodo por trama. El formato de la trama de éste tipo es el que se muestra en la figura 5.17.

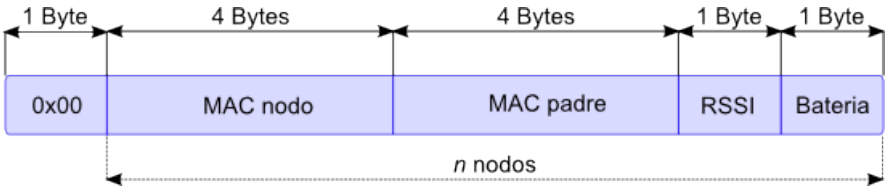


Figura 5.17: Formato de un paquete de listado de nodos

Recepción de medidas

Éste tipo de trama es la enviada por el sumidero cuando éste recoge los datos de las mediciones de los nodos durante la etapa estándar. Cabe notar que la trama es similar a la del listado de nodos. Sin embargo ésta no necesita incluir la MAC del padre ya que se supone que si el padre de un nodo cambia el nodo sumidero enviará una trama de listado de nodos para actualizar dicho dato. No obstante, al contrario que la trama de listado de nodos, en ésta trama sí que es necesario enviar el campo *medición* y el campo *flags*. El campo *flags* ahora se hace necesario ya que es posible que un nodo no envíe alguno de sus datos y la aplicación puente debe saber qué datos contiene la trama y qué datos no. El formato de ésta trama se puede ver en la figura 5.18.

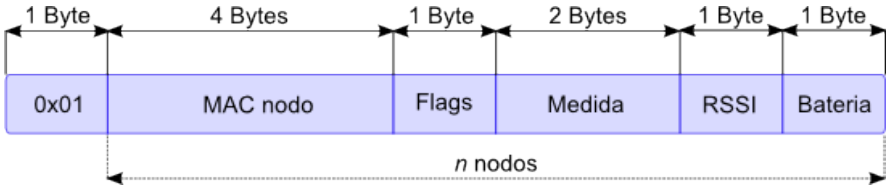


Figura 5.18: Formato de un paquete de recepción de medidas

Envío de la configuración

El envío de los parámetros de configuración se realiza al inicio de la red y en cualquier momento siempre y cuando la red se encuentre en la etapa estándar de la misma. Los parámetros que se deben enviar para que la red pueda funcionar corresponden con el valor del periodo entre mediciones y los valores de los umbrales de medida, nivel de RSSI y nivel de batería. El formato del la trama correspondiente a los parámetros de configuración es el que se muestra en la figura 5.19.

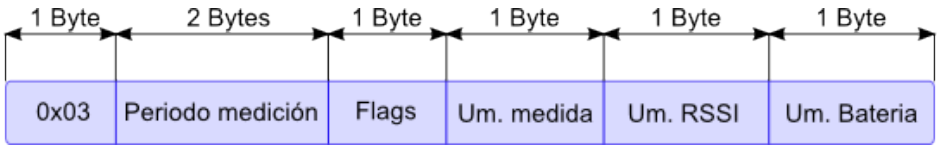


Figura 5.19: Formato de un paquete de envio de parámetros de configuración

Medición bajo demanda

El envío de ésta trama está limitado a realizarse únicamente durante el transcurso de la etapa estándar de la red ya que no tendría sentido en otro momento. Ésta trama, al ser una orden sin necesidad de ningún tipo de parámetro, no necesita más que un identificador para que el sumidero sepa la orden que se le ha ordenado ejecutar. El formato de ésta trama se puede observar en la figura 5.20.

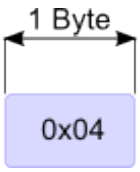


Figura 5.20: Formato de un paquete de envio solicitud de medición bajo demanda

Reinicio de la red

El envío de ésta trama está limitado a realizarse únicamente durante el transcurso de la etapa estándar de la red ya que no tendría sentido en otro momento. Ésta orden está concebida para reiniciar la red en el caso de un mal funcionamiento de ésta o la necesidad de una reorganización de los enlaces, o por la necesidad de añadir nuevos nodos a la red. Ésta trama, al ser una orden sin necesidad de ningún tipo de parámetro, no necesita más que un identificador para que el sumidero sepa la orden que se le ha ordenado ejecutar. El formato de ésta trama se puede observar en la figura **??**.

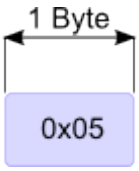


Figura 5.21: Formato de un paquete de envio solicitud de reinicio de la red

5.3.3. Comunicación puente-IGU

Desde el punto de vista de la aplicación puente, la comunicación con el IGU comienza en el hilo *conexionIGU*, cuando éste recibe una conexión de tipo WebSocket al puerto 9321. En ése instante se recogen las cabeceras que envía el cliente y se invoca a la función *createHeaderForSend* pasándole dichas cabeceras como parámetro. Dicha función se encarga de generar las cabeceras de respuesta que deberá envíar la aplicación puente de vuelta al cliente WebSocket. Ésta función genera las campos fijos de las cabeceras:

```
string header("HTTP/1.1 101 Web Socket Protocol Handshake\r\n"
              "Upgrade: WebSocket\r\n"
              "Connection: Upgrade\r\n");
```

y las cabeceras que se pueden generar directamente de los valores de las cabeceras de la petición WebSocket. Para extraer datos de las cabeceras del cliente se utiliza la función *getHeader* a la que se le proporciona las cabeceras originales y el nombre de la cabecera de la cual se quiere extraer su valor.

```
header += string("Sec-WebSocket-Origin: ") +
          string(getHeader(cabeceras, "Origin", NULL, NULL, NULL)) +
          string("\r\n");
```

No obstante para generar la *key* que debe devolver la aplicación puente calculada a partir de las tres *keys* que envía el cliente se delega en la función *solveChallenge*. A dicha función se le proporcionan las tres *keys* enviadas por el cliente y retorna la *key* que debe devolver la aplicación puente al final de sus cabeceras.

Una vez generadas y enviadas todas las cabeceras que debe retornar la aplación puente hacia el cliente WebSocket, un hilo de tipo *listenClient* toma el control de la comunicación. Dicho hilo recoge los datos enviados por el cliente y realiza las acciones oportunas.

Protocolo de comunicación

Una vez se establece la conexión WebSocket la aplicación puente espera que el cliente envie tres bytes. Ésta secuencia de caracteres debe corresponderse con los caracteres *0x00, 0x76, 0xFF*. El primer y último caracter se corresponden con los delimitadores de las tramas WebSocket, y 0x76 es el caracter que desde el código JavaScript de la página se envia nada más iniciarse la conexión. Si ésta secuencia de caracteres es correcta significa que el cliente se corresponde con una IGU válida, de lo contrario se rechaza la conexión. Si los caracteres son correctos se procede a enviar un byte que representa el estado de la red de sensores. Éste byte es de utilidad para la IGU ya que de ésta forma conoce en qué estado se encuentra la red y puede actuar en consecuencia. El byte de estado de la red se puede corresponder con los valores que se muestran en la tabla 5.3.3.

Constante	Valor
RED_PARADA	1
RED_INICIANDO	2
RED_INICIADA	3

Una vez enviado el estado de la red el hilo queda a la escucha de datos por parte de la IGU. Cuando llega una trama tan sólo lee los dos primeros caracteres que envia la IGU ya que el primero siempre es 0x00 que corresponde con el inicio de la trama WebSocket, y el segundo se corresponde con el identificador de la trama, que es el que realmente interesa para discriminar las tramas. Según el valor del segundo byte de la trama se tomará una acción u otra.

Inicio de la red

El hilo *listenClient* lee el identificador de trama 0x49. En ése caso ordena al sumidero que inicie la red y descarta el último byte de la trama (será 0xFF

marcando el final de la trama WebSocket). La figura 5.22 muestra el formato de dicho paquete.

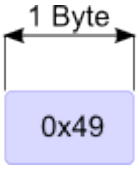


Figura 5.22: Formato de un paquete WebSocket de inicio de la red

Envío de la configuración

El hilo *listenClient* lee el identificador de trama 0x43. En ése caso lee los siguientes 5 bytes que se corresponderán con los flags, periodo de medición, y los umbrales de la medida, nivel de RSSI y nivel de batería en ése orden, tal y como muestra la figura 5.23. Modifica la trama para que sea compatible con la comunicación Puente-USB y la envía hacia el sumidero. Descarta el último byte de la trama (será 0xFF marcando el final de la trama WebSocket)

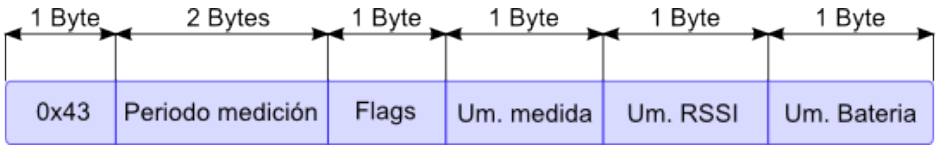


Figura 5.23: Formato de un paquete WebSocket de envío de parámetros de configuración

Medición bajo demanda

El hilo *listenClient* lee el identificador de trama 0x4D. En ése caso ordena al sumidero que realice una medición bajo demanda y descarta el último byte de la trama (será 0xFF marcando el final de la trama WebSocket). La figura 5.24 muestra el formato de dicho paquete.

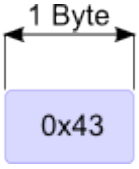


Figura 5.24: Formato de un paquete WebSocket de solicitud de medición bajo demanda

Reinicio de la red

El hilo *listenClient* lee el identificador de trama 0x52. En ése caso ordena al sumidero que reinicie la red y descarta el último byte de la trama (será 0xFF marcando el final de la trama WebSocket). La figura 5.25 muestra el formato de dicho paquete.

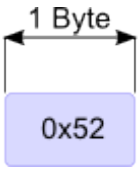


Figura 5.25: Formato de un paquete WebSocket de reinicio de la red

Envío de la notificación del listado de nodos

Desde el hilo principal (main), cuando se recibe desde el sumidero un paquete de listado de nodos vacío significa que la etapa de la formación de la red ha terminado. Para notificar éste suceso a las IGU se envía una trama WebSocket con el caracter 0x50 como contenido como se muestra en la figura 5.26.

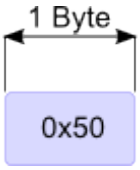


Figura 5.26: Formato de un paquete WebSocket de notificación de listado de nodos

Envío de la notificación de actualización de medidas

Desde el hilo principal (main), cuando se recibe desde el sumidero un paquete de recepción de medidas se notifica a las IGU enviando una trama WebSocket con el caracter 0x51 como contenido como se muestra en la figura 5.27.

5.4. Interfaz Gráfica de Usuario

La interfaz gráfica de usuario se implementa como una página HTML dotada de fragmentos de código JavaScript para otorgar un comportamiento

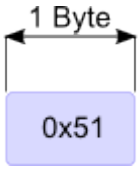


Figura 5.27: Formato de un paquete WebSocket de notificación de actualización de medidas

dinámico al contenido de la página empleando para ello la tecnología AJAX que nos permite realizar peticiones GET (HTTP) sobre un script PHP que nos servirá de intermediario entre la BBDD y la página propiamente dicha con el fin de obtener los datos a presentar en la página. Además, haremos uso de la tecnología WebSocket para establecer una comunicación con la aplicación puente que notificará sobre los eventos acontecidos en la red de sensores y además nos servirá de intermediario entre la red de sensores y la IGU permitiendonos de ésta forma controlar dicha red.

La IGU y el script PHP están estrechamente unidos, es por ello que, dado que PHP permite la inserción de código HTML en el mismo archivo donde reside el código PHP, ambos se implementan sobre el archivo *index.php*. Cuando se accede al archivo *index.php* desde un navegador, en primer lugar se comprueba mediante PHP si el script ha sido llamado mediante un formulario (método *POST*) o si se le proporciona algún parámetro mediante el método GET. De ser falsas ambas condiciones significa que el usuario ha solicitado visualizar la IGU, en caso contrario significa que la propia IGU ha invocado una llamada a sí misma (a *index.php*) mediante AJAX para realizar alguna acción sobre la BBDD (en el caso de que la petición sea de tipo GET) o bien que se está invocando al código PHP para cambiar la imagen de la estructura de nodos (en el caso de que la petición sea de tipo POST).

5.4.1. Conexión WebSocket

Para implementar la conexión WebSocket se hace uso de la API que proporciona JavaScript a tal efecto. Para realizar la conexión con la aplicación puente tan sólo hay que instanciar un objeto de tipo WebSocket proporcionandole como parámetros la URL del servidor y el puerto en el que atiende.

```
ws = new WebSocket("ws://localhost:9321");
```

Automáticamente se envía una petición WebSocket a la aplicación puente y, despues de intercambiarse las cabeceras del protocolo, JavaScript llamará a la función a la que apunta el atributo *onopen* de la instancia del objeto WebSocket.

```
ws.onopen = function() {
     iconoEstado.src = 'img/icono_conectado.png';
     ...
}
```

De igual forma se implementa una función con el atributo *onmessage* para cuando se recibe una trama WebSocket y con el atributo *onclose* para cuando se cierra la conexión.

Una vez abierta la conexión se envía el caracter de validación para identificarse como una IGU válida mediante la función *send* de la instancia de WebSocket.

```
ws.send("L");
```

Una vez validada la IGU la aplicación puente enviará una trama identificando el estado en el que se encuentra la red de sensores. En función del estado en el que se encuentre la red la IGU habilitará o deshabilitará los botones que permiten realizar acciones sobre la red ya que en ciertos estados de la red no es lógico que el usuario pueda realizar algunas acciones, e incluso podría causar un mal funcionamiento en la misma. Los estados posibles en los que se puede encontrar la red y las acciones que se tomarán se detallan en la siguiente tabla 5.2.

Trama	**Descripción**	**Acciones**
1	Red parada	Iniciar red → habilitado Forzar medición → deshabilitado Aplicar configuración → deshabilitado Reiniciar la red → deshabilitado
2	Red iniciando	Iniciar red → deshabilitado Forzar medición → deshabilitado Aplicar configuración → deshabilitado Reiniciar la red → deshabilitado
3	Red iniciada	Iniciar red → deshabilitado Forzar medición → habilitado Aplicar configuración → habilitado Reiniciar la red → habilitado;

Tabla 5.2: Tramas recibidas por la IGU

Una vez se recibe el estado de la red pueden recibirse dos tipos de tramas. Una trama que contenga el valor 0x50 ('P') indica que la red ha finalizado el listado de nodos, en éste caso la IGU no debe realizar ninguna acción. Si, por el contrario, la trama recibia es el valor 0x51 ('Q') significa que se han

procesado medidas nuevas y hay que extraerlas de la BBDD. Para ello se llama a la función *drawNodes*, que obtendrá la información de los nodos y sus medidas y la presentará por pantalla.

5.4.2. Interacción con el script PHP

La interfaz gráfica está compuesta por los botones de control de la red, el panel de parámetros de configuración, la tabla de medidas, y el mapa de nodos. Para todos éstos elementos de la interfaz, excepto para los botones de control de la red, necesita extraer datos distintos de la base de datos. Para la tabla de mediciones necesita extraer un listado de nodos (y sus datos propios como nodos) con sus respectivas últimas medidas tomadas, nivel RSSI y nivel de bateria (cruzando las tablas *nodos* y *mediciones*). Para el panel de configuración necesita extraer los datos de configuración almacenados en la tabla *config* de la BBDD y además poder guardarlos. Y, por último, para el mapa de nodos necesita por una parte los datos de las posiciones de las imagenes de los nodos en el mapa de nodos, por otra poder guardar en la BBDD las posiciones dichos nodos en la imagen, y por último necesita extraer la información de un nodo concreto cuando el usuario posicione el cursor encima de la imagen de un nodo.

De todo ésto extraemos como conclusión que la interfaz gráfica necesita hacer cinco tipos de llamadas distintas al script PHP para obtener la información necesaria para funcionar. Éstas cinco llamadas distintas se traducen en cinco funciones distintas en el código JavaScript que realizarán cinco peticiones mediante AJAX al script PHP. Para que el script PHP distinga éstos cinco tipos de llamadas se incluye en cada petición GET que invoca al script un campo con el nombre *a* (**a**cción). Éste campo que acompaña a la petición GET identifica los cinco tipos de llamadas mediante uno o dos caracteres y se listan en la tabla 5.3.

a	**Descripción**
ln	Devuelve un listado de los nodos con sus últimas medidas
pn	Guarda en la BBDD la posición dada de un nodo concreto
n	Devuelve los datos de un nodo concreto
c	Devuelve los parámetros de configuración existentes en la BBDD
gc	Guarda en la BBDD los parámetros de configuración dados

Tabla 5.3: Tipos de peticiones GET al script PHP

Algunas llamadas aceptan unos parámetros concretos. En la tabla 5.4 se detallan éstos parámetros.

Algunas de éstas llamadas deben retornar una lista de datos. El formato de éstos datos y la forma en la que se procesan se detallarán en los siguientes

a	Parámetros
pn	nodoX → posición en el eje x de la imagen del nodo respecto de la imagen del mapa de nodos nodoY → posición en el eje y de la imagen del nodo respecto de la imagen del mapa de nodos idNodo → Identificador del nodo que se desea posicionar
n	id → Identificador del nodo del que se desea obtener información
gc	t → Periodo de medición uM → Umbral de medición uR → Umbral de RSSI uB → Umbral de batería

Tabla 5.4: Parámetros de las peticiones GET al script PHP

apartados.

5.4.3. Órdenes sobre la red de sensores

En la barra superior de la IGU se encuentran los botones que controlan la red de sensores. Con éstos botones se permite al usuario inicializar la red, reiniciar la red, o forzar que los nodos realicen una medición inmediatamente. Cada uno de éstos botones tiene asociada una función JavaScript de modo que al hacer *click* en ellos dicha función es invocada. La implementación de los mecanismos para que éstos botones cumplan con su cometido es extremadamente simple en el entorno de la IGU ya que ésta tan sólo debe enviar una trama con un determinado identificador a la aplicación puente, siendo ésta última la que lleva a cabo el resto de tareas para que se lleve a cabo la acción que el usuario ha solicitado.

Para enviar la trama se hace uso de la conexión WebSocket que existe entre la IGU y la aplicación puente. Es por ello que basta con invocar al método *send* de la instancia WebSocket proporcionandole como parámetro la cadena de caracteres que será enviada. En la tabla **??** se detallan las tramas enviadas para cada acción ordenada desde la IGU.

Constante	Valor
Iniciar red	0x49 ('I')
Forzar medición	0x4D ('M')
Reiniciar red	0x52 ('R')

Una vez enviada la trama correspondiente se habilitan o deshabilitan ciertos elementos de la interfaz gráfica para adaptarla al estado de la red dada la orden ejecutada.

5.4.4. Gestión de los parámetros de configuración

Al iniciarse la interfaz gráfica ésta realiza una invocación a la función *getConfig*. Ésta función realiza una petición de tipo *c* mediante AJAX al script PHP. Ésta petición obliga al script PHP a realizar una consulta a la tabla *config* de la BBDD. Todas las tuplas encontradas son devueltas por el script mediante una estructura XML que contiene una seria de etiquetas que tienen por nombre la clave del parámetro de configuración (campo *clave* de cada tupla) y envuelven el valor que contiene dicha clave *campo clave de la respectiva tupla*. De ésta forma el código JavaScript desensambla ésa estructura XML y asigna a los elementos *input* (HTML) sus valores correspondientes.

Cabe destacar que el código JavaScript no discrimina por nombre los parámetros de configuración sino que asigna a los elementos *input*los valores recibidos en la estructura XML por orden. Es decir, el mecanismo de asignación de valores a los elementos *input* es sensible al orden de las etiquetas por lo que si el órden de éstas etiquetas XML es alterado provocará un mal funcionamiento en la interfaz. Una estructura XML de éste tipo se muestra en el siguiente ejemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<config>
     <umbralMedida>61</umbralMedida>
     <umbralRSSI>10</umbralRSSI>
     <umbralBateria>21</umbralBateria>
</config>
```

Cuando el usuario presiona el botón *Aplicar configuración* se invoca la función *setConfig*. Ésta función obtiene los valores almacenados en los elementos *input* correspondientes a los valores de los parámetros de configuración. Para cada dato comprueba si es distinto al de una variable global que almacena el valor actual efectivo de ése parámetro. En caso de ser distinto actualiza el valor efectivo de ése parámetro al del elemento *input* y modifica la variable *flagsConfig* que contiene los parámetros que han cambiado con respecto al actual efectivo. Éste procedimiento se repite para cada uno de los parámetros de configuración y finalmente se envía una trama que tiene el valor 0x43 ('C') como identificador seguido del valor de la variable *flags* y los cuatro parámetros de configuración como se muestra en la figura 5.23. Por último se invoca al script PHP con una petición de tipo *gc* para que éste guarde dicha configuración en la BBDD. En dicha petición se le suministrarán los parámetros de configuración a guardar mediante campos en la petición GET. Una petición GET de éste tipo podría ser la que sigue.

```
index.php?a=gc&pM=10&uM=200&uR=10&uB=5
```

Como se puede observar cada parámetro de configuración está identificado en la petición por un par de caracteres. Éstos caracteres se corresponden con los parámetros como se muestra en la tabla 5.4.4.

Constante	Valor
pM	Periodo de medición
uM	Umbral de medida
uR	Umbral de nivel de RSSI
uB	Umbral de nivel de bateria

5.4.5. Listado de nodos

Cuando el usuario presiona el botón *Iniciar red* la red de nodos se prepara para comenzar a realizar mediciones. Una vez éste proceso termina la red de sensores comienza a enviar medidas con un determinado periodo. No obstante éstas medidas se almacenan en la BBDD y, a no ser que la IGU las consulte, obviamente no se actualizarán en pantalla. Por ello, la aplicación puente, cada vez que recibe una trama de medidas provenientes del nodo sumidero, ínmediatamente después de insertar dichas medidas en la base de datos notifica a la IGU mediante una trama formada por el valor 0x51 ('Q'). Una vez la IGU recibe una trama de éste tipo invoca a la función *drawNodes*. Ésta función se encarga de actualizar la tabla de mediciones de la IGU. Con éste fin la función *drawNodes* realiza una petición GET de tipo *ln* al script PHP. Ésta llamada al script devuelve una estructura XML con los datos intrínsecos de cada nodo y sus medidas. Ésta estructura XML está envuelta en una etiqueta *listaNodos* y dentro contiene una serie de etiquetas *nodo* que a su vez envuelven un conjunto de etiquetas que definen los datos de cada nodo tal y como se describe en la tabla 5.4.5.

Un ejemplo de una estructura XML real sería el que sigue.

```
<?xml version="1.0" encoding="utf-8"?>
<listaNodos>
<nodo>
<idNodo>257</idNodo>
<mac>3:3:3:3</mac>
<flags>1</flags>
<idMed>263799</idMed>
<medida>2861</medida>
<RSSI>-86</RSSI>
<nivelBateria>224</nivelBateria>
<fecha>3-12-2010 13:04:45</fecha>
<posX>0</posX>
```

```
<posY>0</posY>
</nodo>
<nodo>
<idNodo>258</idNodo>
<mac>4:4:4:4</mac>
<flags>4</flags>
<idMed>263796</idMed>
<medida>2861</medida>
<RSSI>-78</RSSI>
<nivelBateria>239</nivelBateria>
<fecha>3-12-2010 13:04:24</fecha>
<posX>0</posX>
<posY>0</posY>
</nodo>
<nodo>
<idNodo>259</idNodo>
<mac>2:2:2:2</mac>
<flags>0</flags>
<idMed>263793</idMed>
<medida>2861</medida>
<RSSI>0</RSSI>
<nivelBateria>239</nivelBateria>
<fecha>3-12-2010 12:44:52</fecha>
<posX>0</posX>
<posY>0</posY>
</nodo></listaNodos>
```

Una vez se recibe la estructura XML se eliminan mediante JavaScript todas las filas de la tabla de mediciones excepto la cabecera (desde la 1 hasta n, siendo n la longitud de la tabla en filas). Una vez hecho esto se desensambla la estructura XML para obtener cada dato de cada nodo. Cabe destacar que al desensamblar y recoger los datos de la estructura XML se asume el orden de etiqutas mostrado en la tabla 5.4.5, por lo que alterar dicho orden provocará un mal funcionamiento de la IGU.

Una vez se obtienen los datos del primer nodo de la estructura XML se genera una fila de la tabla de mediciones mediante JavaScript. Posteriormente se van generando cada una de las celdas en orden con el valor que le corresponde extraido de la estructura XML. Cuando se va a insertar alguno de los datos del nodo que son variables como (medida de deformación, RSSI y nivel de batería) se comprueba mediante el dato *flags* si el nodo lo envió en la trama (significa que ha cambiado con respecto a la anterior medida), en caso de ser así se la aplica a la celda un color de fondo de tonalidad verde indicando que dicho valor ha cambiado, de lo contrario se aplica el tono amar-

Constante	Valor
idNodo	Identificador del nodo en la BBDD
mac	Dirección física del nodo
flags	Campo flags enviado por el nodo en la trama inalámbrica al nodo sumidero. Cada uno de los 3 bits de menor peso de éste byte se corresponden con la existencia o no de los campos medida, RSSI y nivel de batería en ése orden
idMed	Identificador de la medición en la BBDD
medida	Última medida válida (se envió el campo flags con el tercer bit a 1) enviada por el nodo
RSSI	Última valor de RSSI válido (se envió el campo flags con el segundo bit a 1) enviado por el nodo
nivelBateria	Última nivel de batería válido (se envió el campo flags con el primer bit a 1) enviado por el nodo
fecha	Fecha y hora en formato dd/mm/aaaa del momento en el que los datos de la trama de medición se insertaron en la BBDD
posX	Posición en el eje X de la imagen del nodo con respecto a la imagen correspondiente al mapa de nodos
posY	Posición en el eje Y de la imagen del nodo con respecto a la imagen correspondiente al mapa de nodos

illo normal. Éste mecanismo se aplica para cada uno de los nodos de la lista XML recibida hasta terminarla.

Para cada nodo procesado además se crea una capa HTML con el atributo CSS *position: absolute* para que pueda ser posicionada sobre otros objetos. Dentro de ésta capa se inserta una imagen que corresponderá a la del nodo. Ésta capa es posicionada usando los atributos de posicion en los ejes X e Y de la imagen del mapa de nodos y sumandole los desplazamientos adiciones en ambos ejes determinados por los valores de las etiquetas *posX* y *posY* extraidos de la estructura XML. De ésta forma se posicionan las imagenes de los nodos dentro de la imagen del mapa de nodos.

5.4.6. Posicionamiento de los nodos en el mapa de nodos

Cuando la IGU está recibiendo medidas en la tabla de medidas se muestra cada nodo disponible en la red. En ése momento el usuario puede usar el botón *Posicionar* para cada nodo con el fin de insertar la imagen del sensor dentro del mapa de nodos de la red. Para lograr éste comportamiento, al pulsar el botón *Posicionar* de un nodo de la tabla de mediciones dicho boton invoca

a la función *botonPos*. Ésta función recibe como parámetro el propio botón y lo usa para retroceder en la estructura DOM[11] de la que cuelga el botón con el fin de acceder a la celda que contiene el identificador de ese nodo (el identificador del nodo que corresponde con la misma fila a la que pertenece el botón).

```
var tr = obj.parentNode.parentNode;
idNodoSelec = tr.cells[0].firstChild.data;
```

Nota: La columna 0 de la tabla se asume que corresponde con la columna que contiene el identificador del nodo. Ésta es otra de las razones por las que alterar el orden de las columnas de la tabla provocaría un mal funcionamiento de la IGU.

Una vez se obtiene el identificador del nodo que el usuario desea posicionar éste es guardado en la variable global *idNodoSelec* y se modifican los atributos de la imagen del mapa de nodos para que cuando el ratón se situe encima de ésta el cursor cambie a una cruz y para que cuando el usuario haga *click* en la imagen ésta invoque a la función *sendCoordinates*. Dicha función se encarga de obtener las coordenadas del cursor con respecto a la imagen (es decir, se toma como coordenada 0,0 la esquina superior izquierda de la imagen) mediante la función *getMouseXY* y realizar una petición GET de tipo *pn* al script PHP mediante AJAX con el fin de que éste guarde las coordenadas del raton con respecto a la imagen para el nodo cuyo identificador sea el que contiene la variable global *idNodoSelec*. Cuando termina éste proceso la función *sendCoordinates* devuelve el cursor por defecto, pone la variable *idNodoSelec* a *NULL* y modifica la imagen para que ésta ya no llame a la función *sendCoordinates* cuando el usuario haga *click* en ella.

Cuando el script PHP termina de procesar la petición el código AJAX invoca a la función *drawNodes* para que se muestre la imagen del nuevo nodo en pantalla.

5.4.7. Cambio de la imagen del mapa de nodos

La imagen del mapa de nodos puede ser cambiada por el usuario presionando el botón *Cambiar imagen*. Cuando ésto sucede se «oscurece» la imagen actual y se muestra sobre la misma un popup con unos elementos HTML que permiten subir al servidor imagenes desde el PC en el que se está visualizando la IGU para que sea ésta imagen la que sustituya a la actual.

[11] El Document Object Model o DOM es esencialmente una interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

Hasta ahora habiamos hecho uso de AJAX para todos los casos en los que necesitamos invocar al script PHP mediante peticiones GET. El mecanismo que permite subir imagenes al servidor también pasa por el script PHP, sin embargo, en éste caso la petición debe ser de tipo POST por la naturaleza de la acción. Además, para poder enviar la imagen al servidor HTTP deberemos de hacer uso de un formulario HTTP ya que así lo requiere ésta funcionalidad. No obstante, en éste caso no vamos a poder hacer uso de AJAX para implementar ésta funcionalidad sin necesidad de recargar la totalidad de la página ya que el formulario que envía la imagen de camino al servidor HTTP debe tener *multipart/form-data* como valor del campo *enctype* del formulario para que la imagen sea enviada al servidor HTTP correctamente, y AJAX no soporta éste tipo de codificación, por lo que éste formulario se implementará a la vieja usanza dentro del código HTML y usando como objetivo del mismo el propio *index.php*, que es el destinatario del mismo, obviamente.

Para obtener el efecto del oscurecimiento de la imagen se ha usado una capa con el atributo CSS *opacity: .9*. Ésta capa inicialmente está oculta mediante el atributo *visibility: hidden*. Cuando el usuario pulsa el botón *Cambiar imagen* se llama a la función *cambiarImgDialog* que adapta el tamaño de la capa al tamaño de la imagen, la posiciona en las mismas coordenadas que posee la imagen y cambia el atributo *visibility* a *visible*.

El popup que aparece en el centro de la imagen es otra capa oculta que es posicionada en el centro de la imagen de la forma que se muestra en el código mostrado abajo, y se hace visible de la misma forma que la capa de oscurecimiento.

```
var input = document.getElementById("divCambiarImgInput");

input.style.left = imgMapaNodos.x+parseInt(
     (imgMapaNodos.width-tabla.clientWidth)/2) + "px";
input.style.top = imgMapaNodos.y+parseInt(
     (imgMapaNodos.height-tabla.clientHeight)/2) + "px";
```

Capítulo 6

Resultados

6.1. Red de sensores

El protocolo de red presentado en el proyecto es el fruto no solo de una larga etapa de diseño inicial, sino además de multitud de pruebas a través de las cuales se ha ido depurando no solo la implementación (que ha sido la que más ajustes ha requerido) sino también en cuanto a diseño.

Es por esto que los resultados obtenidos del funcionamiento del protocolo son en general satisfactorios. A través del Packet Sniffer y de diversas trazas de la memoria de los nodos se ha podido establecer que el protocolo funciona bajo los términos bajo los que se diseñó. Forma la red, lo hace de forma eficiente y envía las mediciones al nodo sumidero periódicamente.

No se han realizado simulaciones del funcionamiento de la red y por tanto se carecen de datos más fiables que los mencionados. El trabajo de campo y las pruebas han sido realizadas con un número limitado de nodos. La mayor prueba realizada en laboratorio fue de cinco nodos sensores y el nodo sumidero funcionando a la vez.

Para ilustrar una de esas pruebas vamos a establecer un caso de prueba con cuatro nodos sensores y un nodo sumidero. Tal y como aparecen en la figura 6.1 cada nodo sensor alcanza tan solo a los nodos que le rodean (situados a derecha o izquierda) y el nodo sumidero alcanza toda la red.

El proceso de formación del árbol si inicia en el nodo sumidero con el descubrimiento. Después de este primer tanteo el nodo sumidero sigue arbitrando la formación de la red mediante el envío de órdenes de descrubrimiento a los nodos que haya registrado en el orden en que han sido registrados.

El proceso se desarrolla como se aprecia en la figura 6.1[1], y continuaría

[1]La gráfica detalla el intercambio de mensajes en el tiempo, desarrollándose este último en el eje vertical en sentido descendente. La inclinación de los mensajes indican el tiempo

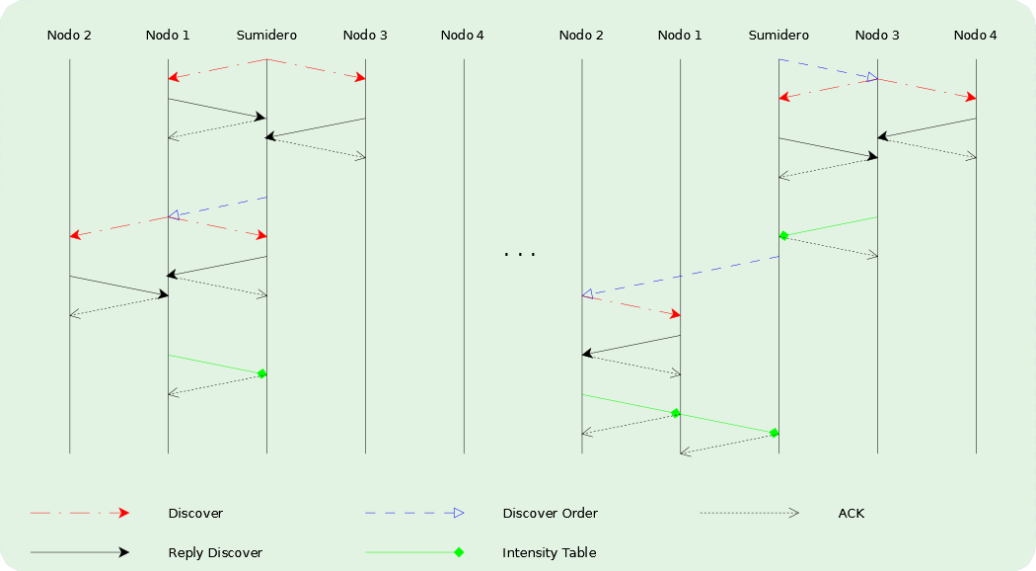


Figura 6.1: Etapa de formación de la red

con una última orden de descubrimiento al Nodo 4. Además, como se puede observar la contestación del Nodo 2 al Sumidero con sus Tablas de Intensidades pasa encaminado a través del Nodo 1, que contesta con un acuse de recibo punto a punto.

Esto resulta en un árbol sencillo en el que los nodos 1 y 3 cuelgan directamente del sumidero y de ellos cuelgan, respectivamente, los nodos 2 y 4.

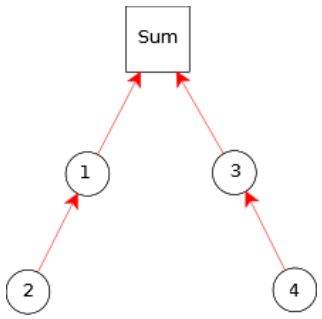


Figura 6.2: Árbol tras la etapa de formación de la red

Sin embargo antes de formar el árbol, con el fin de ilustrar el cambio entre la etapa de formación de la red y la etapa estándar el nodo sumidero enviará un paquete de programación con los umbrales de las medidas que se usarán durante los *macro-frames*. Posteriormente y una vez formado el

de transmisión

árbol y enviados los datos a la estación de monitorización se procederá con la programación del primer *macro-frame* por ramas.

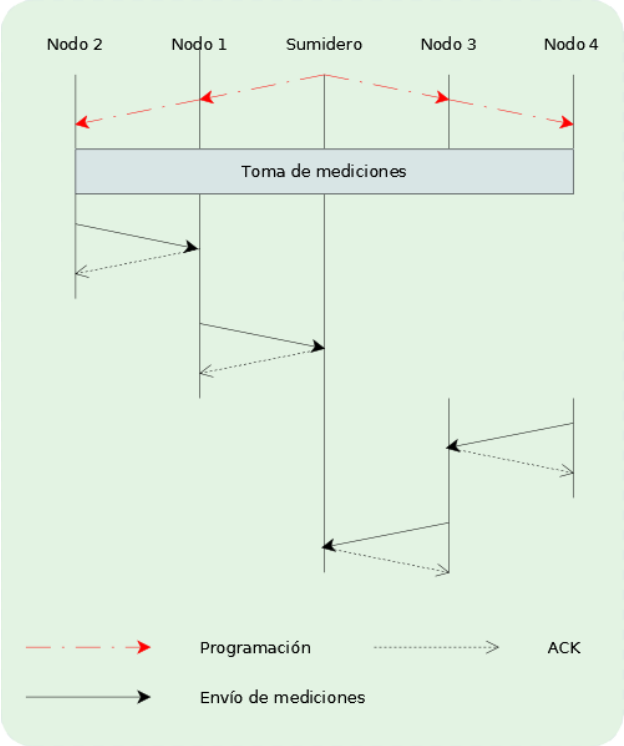


Figura 6.3: Ejemplo de *macro-frame* sin errores

Se puede observar en la figura 6.3 que el eje vertical se usa para representar el tiempo que pasan despiertos los nodos mediante discontinuidades. Según esta característica los nodos hojas en el árbol pasan tan solo un marco temporal despiertos, además del marco temporal que se reserva para la toma de medidas. Los nodos 1 y 3 acumulan las medidas de los nodos que cuelgan de ellos y el sumidero las recibe todas al final del *macro-frame* si no ocurre ningún error.

Se ha implementado también la recuperación ante errores en la red, forzando en el laboratorio dichos fallos para obtener la respuesta del nodo sumidero. En este caso la red responde ante errores resolviendo cada error en un solo *macro-frame* de error en el que se soluciona de dos posibles formas: recibiendo las medidas correctamente de los nodos que fallaron y la etapa estándar continúa sin más, o bien el nodo falla de nuevo, con lo que se elimina de la red y se vuelven a calcular los caminos mínimos sobre el grafo para reencaminar los nodos.

Para el ejemplo ilustraremos dicha situación con un error en el Nodo 1 que hace que no se reciban las mediciones de los nodos 1 y 2. Se recibirían el resto de mediciones y cuando el nodo sumidero compruebe los paquetes

recibidos detectará el error y programará el *macro-frame* de emergencia para los nodos 1 y 2. En la figura 6.4 la red se recupera del error recibiendo el paquete de medidas de los nodos acumuladas en el Nodo 1. En caso de que volviera a fallar eliminaría al Nodo 1 del árbol y el nodo 2 quedaría huérfano, dejando de participar en los *macro-frames* por no tener visibilidad directa con el nodo sumidero ni con ningún otro nodo sensor.

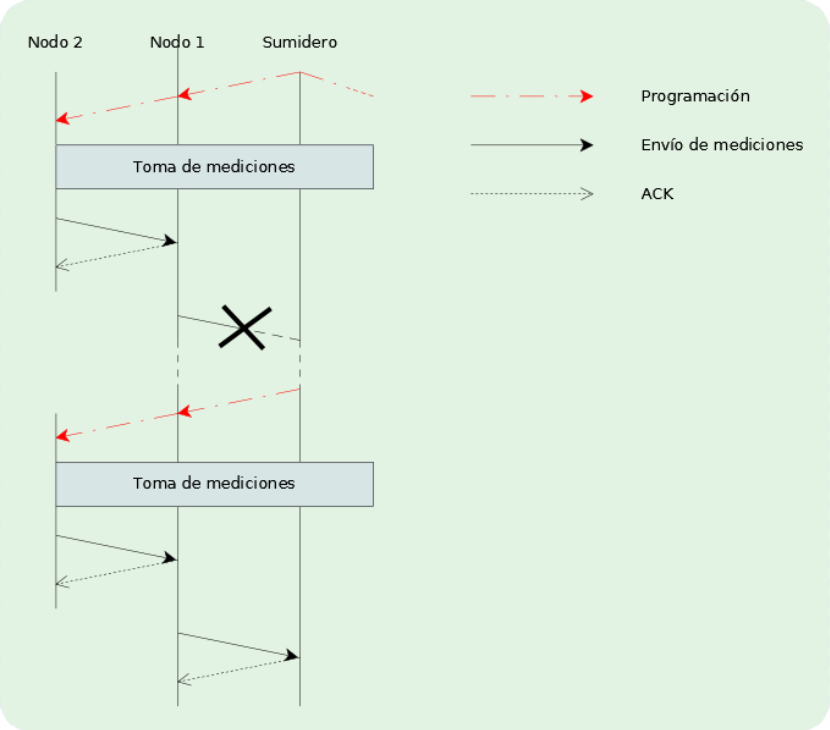


Figura 6.4: Ejemplo de *macro-frame* con un error

Este es solo un ejemplo de lo que sería una ejecución normal en una prueba con cuatro nodos sensores. Un resultado posible, observado durante las pruebas realizadas de forma similar y que cumple con el protocolo de red tal y como se especifica en el Capítulo 3.

6.1.1. Captura del funcionamiento de la Red de Sensores

A continuación presentaremos una serie de capturas parciales del Packet Sniffer en las que se puede comprobar el funcionamiento del protocolo con los formatos de paquete.

P.nbr.	Length	Dest. Address	Source Address	Applicaton payload	User Port	RSSI (dBm)	LQI	FCS
1	12	FF FF FF FF	01 01 01 01	10	0x3F	-31	49	OK
2	12	01 01 01 01	04 04 04 04	20	0x3F	-20	48	OK
3	12	04 04 04 04	01 01 01 01	00	0x3F	-31	50	OK
4	12	01 01 01 01	03 03 03 03	20	0x3F	-33	49	OK
5	12	03 03 03 03	01 01 01 01	00	0x3F	-30	49	OK
6	12	01 01 01 01	02 02 02 02	20	0x3F	-41	47	OK
7	12	02 02 02 02	01 01 01 01	00	0x3F	-34	49	OK
8	16	04 04 04 04	01 01 01 01	30 01 01 01 01	0x3F	-13	48	OK
9	12	FF FF FF FF	04 04 04 04	10	0x3F	-20	48	OK
10	12	04 04 04 04	01 01 01 01	20	0x3F	-23	53	OK
11	12	03 03 03 03	04 04 04 04	00	0x3F	-21	51	OK

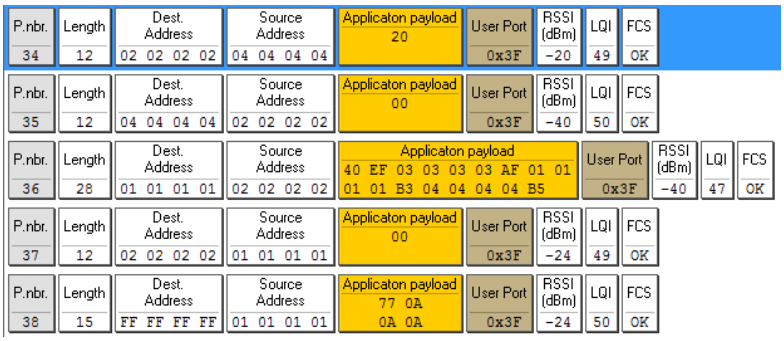
Figura 6.5: Captura del proceso de formación de la red

P.nbr.	Length	Dest. Address	Source Address	Applicaton payload	User Port	RSSI (dBm)	LQI	FCS
12	12	04 04 04 04	01 01 01 01	20	0x3F	-24	48	OK
13	12	01 01 01 01	04 04 04 04	00	0x3F	-20	48	OK
14	12	04 04 04 04	02 02 02 02	20	0x3F	-40	48	OK
15	12	02 02 02 02	04 04 04 04	00	0x3F	-20	49	OK
16	28	01 01 01 01	04 04 04 04	40 EF 03 03 03 03 B1 01 01 01 01 B0 02 02 02 02 AA	0x3F	-20		
17	12	04 04 04 04	01 01 01 01	00	0x3F	-24	48	OK
18	16	03 03 03 03	01 01 01 01	30 01 01 01 01	0x3F	-14	50	OK
19	12	FF FF FF FF	03 03 03 03	10	0x3F	-32	50	OK
20	12	03 03 03 03	04 04 04 04	20	0x3F	-20	49	OK
21	12	04 04 04 04	03 03 03 03	00	0x3F	-31	48	OK
22	12	03 03 03 03	01 01 01 01	20	0x3F	-24	49	OK

Figura 6.6: Captura del proceso de formación de la red

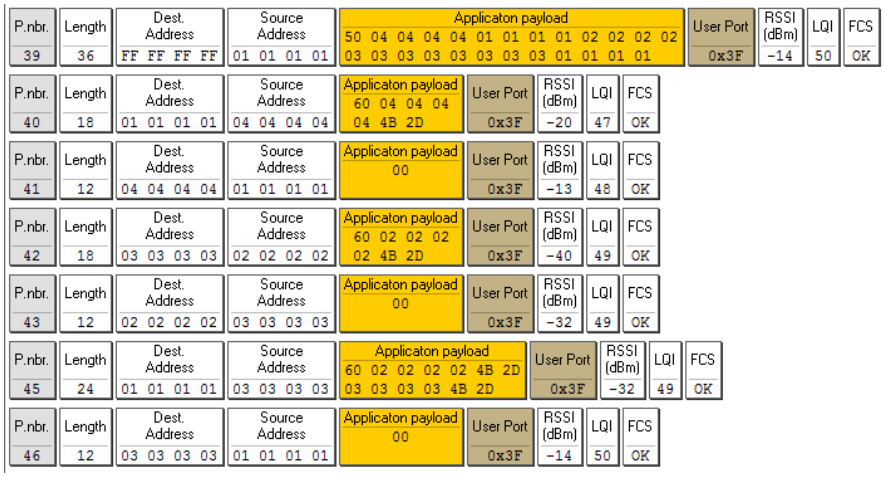
P.nbr.	Length	Dest. Address	Source Address	Applicaton payload	User Port	RSSI (dBm)	LQI	FCS
23	12	01 01 01 01	03 03 03 03	00	0x3F	-31	48	OK
24	12	03 03 03 03	02 02 02 02	20	0x3F	-40	47	OK
25	12	02 02 02 02	03 03 03 03	00	0x3F	-32	46	OK
26	28	01 01 01 01	03 03 03 03	40 EF 04 04 04 04 AE 01 01 01 01 AF 02 02 02 02 AD	0x3F	-32	50	OK
27	12	03 03 03 03	01 01 01 01	00	0x3F	-24	48	OK
28	16	02 02 02 02	01 01 01 01	30 01 01 01 01	0x3F	-14	49	OK
29	12	FF FF FF FF	02 02 02 02	10	0x3F	-40	50	OK
30	12	02 02 02 02	04 04 04 04	20	0x3F	-18	50	OK
31	12	03 03 03 03	02 02 02 02	00	0x3F	-40	50	OK
32	12	02 02 02 02	01 01 01 01	20	0x3F	-24	48	OK
33	12	01 01 01 01	02 02 02 02	00	0x3F	-40	47	OK

Figura 6.7: Captura del proceso de formación de la red



P.nbr.	Length	Dest. Address	Source Address	Applicaton payload	User Port	RSSI (dBm)	LQI	FCS
34	12	02 02 02 02	04 04 04 04	20	0x3F	-20	49	OK
35	12	04 04 04 04	02 02 02 02	00	0x3F	-40	50	OK
36	28	01 01 01 01	02 02 02 02	40 EF 03 03 03 03 AF 01 01 01 B3 04 04 04 04 B5	0x3F	-40	47	OK
37	12	02 02 02 02	01 01 01 01	00	0x3F	-24	49	OK
38	15	FF FF FF FF	01 01 01 01	77 0A 0A 0A	0x3F	-24	50	OK

Figura 6.8: Captura del proceso de formación de la red



P.nbr.	Length	Dest. Address	Source Address	Applicaton payload	User Port	RSSI (dBm)	LQI	FCS
39	36	FF FF FF FF	01 01 01 01	50 04 04 04 04 01 01 01 01 02 02 02 02 03 03 03 03 03 03 03 03 03 01 01 01 01	0x3F	-14	50	OK
40	18	01 01 01 01	04 04 04 04	60 04 04 04 04 4B 2D	0x3F	-20	47	OK
41	12	04 04 04 04	01 01 01 01	00	0x3F	-13	48	OK
42	18	03 03 03 03	02 02 02 02	60 02 02 02 02 4B 2D	0x3F	-40	49	OK
43	12	02 02 02 02	03 03 03 03	00	0x3F	-32	49	OK
45	24	01 01 01 01	03 03 03 03	60 02 02 02 02 4B 2D 03 03 03 03 4B 2D	0x3F	-32	49	OK
46	12	03 03 03 03	01 01 01 01	00	0x3F	-14	50	OK

Figura 6.9: Captura de *macro-frame*

Comentarios de la captura

En las capturas presentadas se pueden observar hasta tres nodos sensores (0x22222222, 0x33333333 y 0x44444444) y un nodo sumidero (0x11111111).

En la formación del árbol el nodo sumidero registra a los tres nodos en la etapa inicial de la formación y los recorre para escrutar sus tablas de intensidades en el orden en que los registró. Podemos ver los paquetes de Tablas de Intensidades en los números de paquete 16, 26 y 36.

El paquete número 38 es el de configuración de los nodos en los que envía los tres umbrales (que en el ejemplo son los tres 0x0A).

El árbol resultante de la captura y que origina el *macro-frame* de la figura 6.9 es el que podemos ver en la figura 6.10.

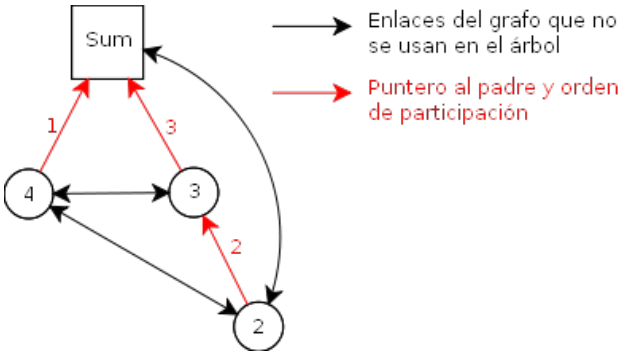


Figura 6.10: Árbol resultante de las capturas mostradas

6.2. Tarjeta de adaptación

Una vez fabricada y montada la tarjeta de adaptación se realizaron pruebas para verificar el correcto funcionamiento de la misma. Con éste fin las galgas extensiométricas fueron montadas sobre un listón de acero similar al tipo de acero para el que ha sido diseñada la electrónica de la tarjeta. Se conectó la tarjeta de adaptación a un nodo que hace uso correctamente de la libreria *leerGalgas* y con la ayuda de un osciloscopio se visualizaron las señales de reloj y de salida del conversor analógico-digital como se ve en las siguientes figuras.

En la figura 6.11 vemos la salida del CAD cuando el metal se encuentra en reposo, es decir, cuando no sufre ningun tipo de deformación. Como se puede observar, la señal se transmite a partir del sexto ciclo de reloj como estipulan las especificaciones del CAD. El valor visualizado en pantalla corresponde con el número 100000011110000 en binario natural, lo que se corresponde con el número 16624 en decimal.

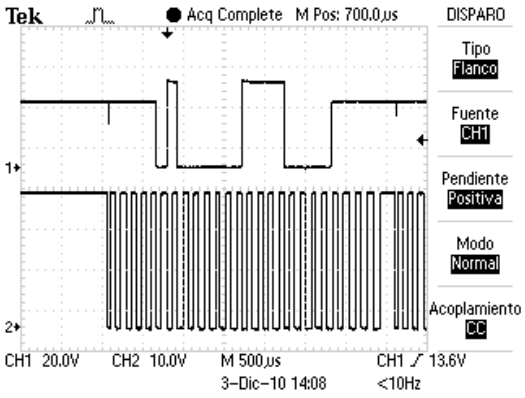


Figura 6.11: Salida del CAD con las galgas extensiométricas en reposo

En la figura 6.12 vemos la señal de reloj y la salida del CAD cuando el metal es deformado en un sentido concreto. Como vimos anteriormente la salida en reposo del CAD debe es 16624 aproximadamente, por lo que la salida del CAD para una deformación en un sentido debe diferir significativamente de 16624. Como vemos el osciloscopio visualiza la señal del CAD correspondiente al número 32761, lo que efectivamente difiere de forma significativa del valor del CAD en reposo.

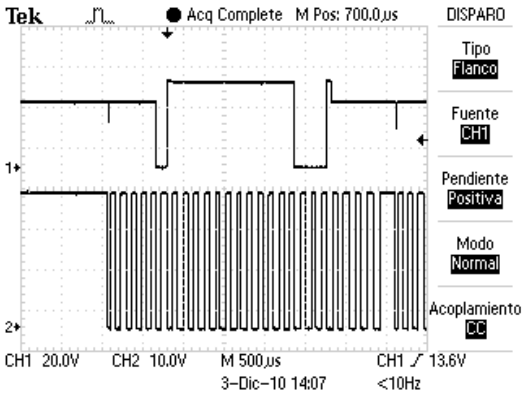


Figura 6.12: Salida del CAD con las galgas extensiométricas deformadas

Finalmente en la figura 6.13 visualizamos la señal de salida del CAD para una deformación pero en sentido contrario a la anterior deformación. Dado que la anterior deformación difería del valor en reposo del CAD en sentido negativo (por debajo del valor de reposo), obviamente cabe esperar que para una deformación en sentido contrario registraremos una medida por debajo de 16624. Como vemos en la imagen de la pantalla del osciloscopio ésta medida corresponde con el valor 7, el cual efectivamente está significativamente por debajo de 16624, por lo que podemos considerar que la tarjeta de adaptación funciona razonablemente bien a falta de un medidor de deformaciones calibrado que contraste éstas medidas.

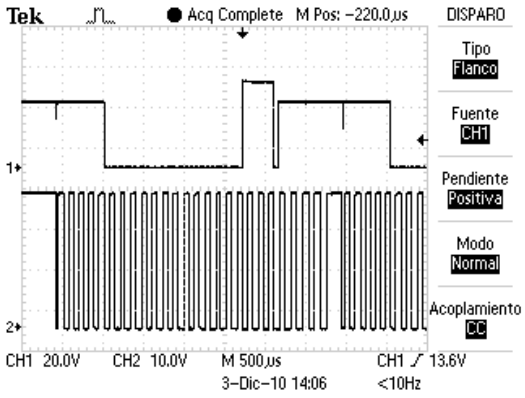


Figura 6.13: Salida del CAD con las galgas extensiométricas deformadas en sentido contrario

6.3. Interfaz gráfica de usuario

Al iniciarse la interfaz gráfica introduciendo la URL de la misma en el navegador ésta se pone en contacto con el servidor WebSocket del cual conoce su URL y puerto ya que está especificado dentro del código JavaScript de la misma. En la figura 6.14 se muestra como la IGU se pone en contacto con la aplicación puente y solicita la conexión mediante el protocolo WebSocket.

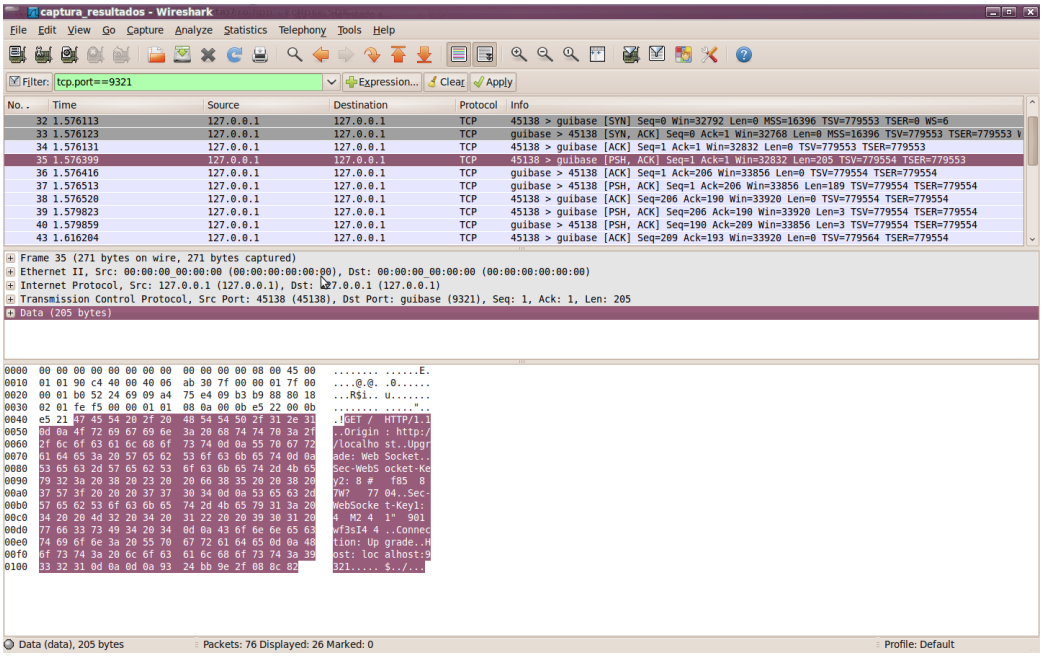


Figura 6.14: IGU contactando con el servidor WebSocket (cabeceras de petición de conexión WebSocket)

La aplicación puente responde con las cabeceras correspondientes según el protocolo WebSocket.

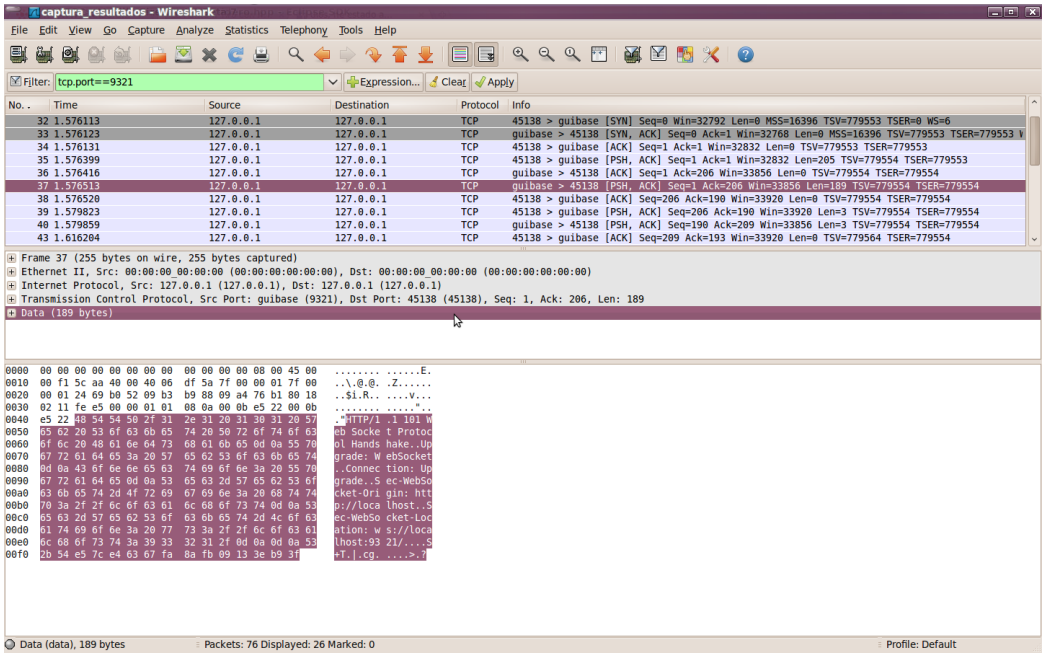


Figura 6.15: La aplicación puente responde con las cabeceras de respuesta WebSocket

Como se explicó en apartados anteriores la IGU debe autenticarse mediante el caracter 'L' (0x4C) (como se observa en la figura 6.16)para confirmar que es una IGU válida de nuestra aplicación de monitorización. Cabe destacar que la trama se compone de los caracteres 0x00, 0x4C y 0xFF ya que el protocolo WebSocket utiliza el caracter 0x00 para marcar el inicio de una trama y 0xFF para marcar el final de la misma. Éste procedimiento se realiza de forma bidireccional, por lo que en la aplicación puente se tiene en cuenta dicho formato. No obstante, las tramas recibidas mediante el código JavaScript en la parte del cliente retornan tan sólo el valor 0x4C, ya que los delimitadores forman parte del protocolo WebSocket y ésto es transparente para el programador.

En la figura 6.17 se muestra cómo la aplicación puente, al validar la IGU, retorna el estado en el que se encuentra la red. En éste caso, obviamente retorna el valor *1* ya que la red está parada.

De forma paralela a éste proceso se invoca al script PHP mediante una petición AJAX en la que solicita que se le proporcionen los parámetros de configuración que se encuentran guardados en la BBDD. Para ello especifica dicha acción mediante los pares clave-valor insertados en la petición GET. Como se puede ver en la figura 6.18 se invoca al script con el valor *c* en la

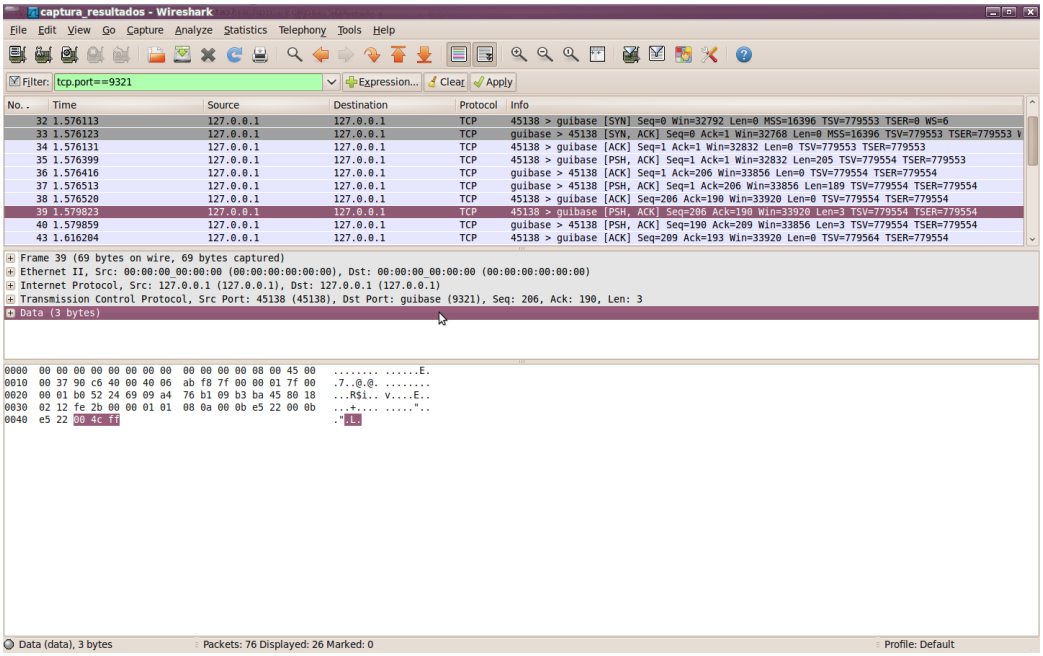


Figura 6.16: La IGU se identifica como una interfaz gráfica válida

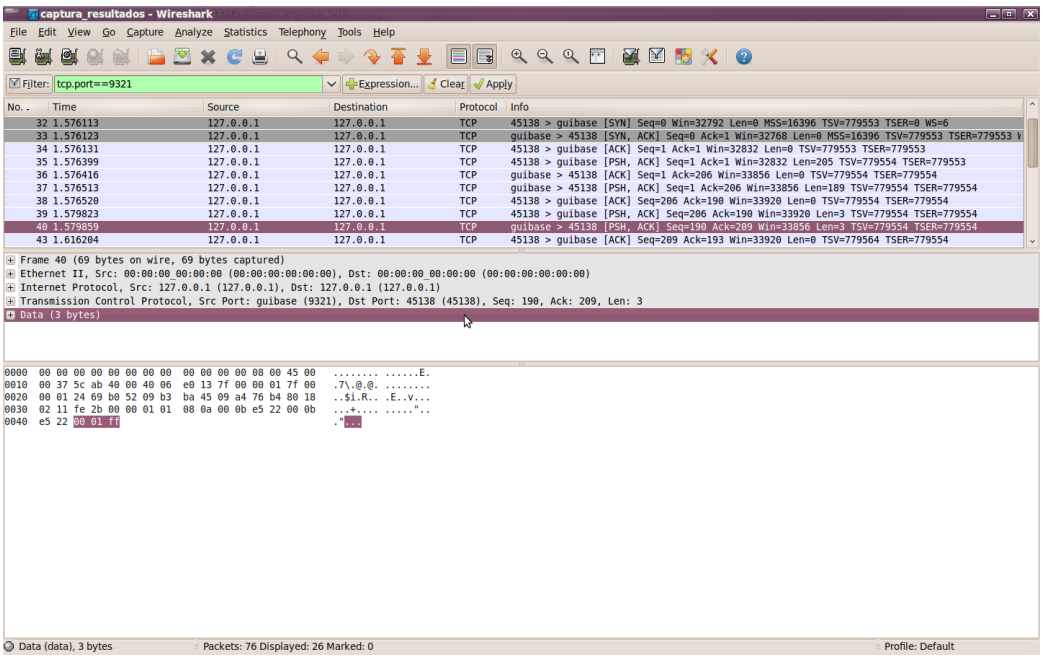


Figura 6.17: La aplicación puente informa del estado de la red

clave *a*, ésto identifica el tipo de consulta que el script PHP debe realizar sobre la BBDD, en éste caso una consulta sobre la tabla *config* para extraer la configuración de la red de sensores.

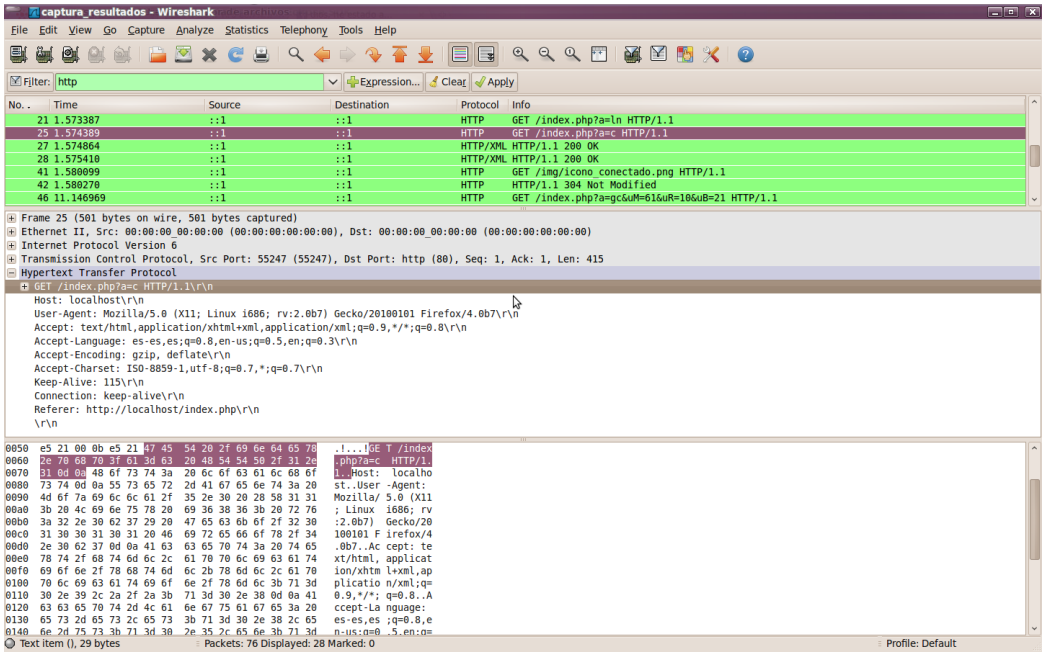


Figura 6.18: La IGU solicita al script PHP el listado de nodos y sus medidas

Cuando el usuario presiona el botón *Iniciar red* la IGU envía la trama correspondiente para que la aplicación puente a su vez transmita la orden al sumidero para que comience con la etapa de formación de red. En la figura 6.19 podemos ver como se envía dicha trama, que se compone únicamente del caracter 'I' (0x49).

Inmediatamente después de ordenar el inicio de la red, la IGU envía los parámetros de configuración que están contenidos en los elementos *input* correspondientes de la interfaz. En la figura 6.20 vemos como envía dichos parámetros por orden en formato hexadecimal.

Pasado un lapso de tiempo la red se inicializa. Dado que la IGU debe reaccionar a éste evento habilitando y deshabilitando ciertos elementos de la página, ésta notificación es recibida por parte de la aplicación puente. En la figura 6.21 podemos ver la trama compuesta por el valor 0x50 ('P') como es enviada por la aplicación puente.

En el momento en el que vence el periodo entre mediciones éste se efectua. La aplicación puente recoge los datos correspondientes a las mediciones de los nodos y los guarda en la BBDD. De forma inmediata notifica a la IGU para que ésta consulte la BBDD en busca de los nuevos datos. Como se muestra

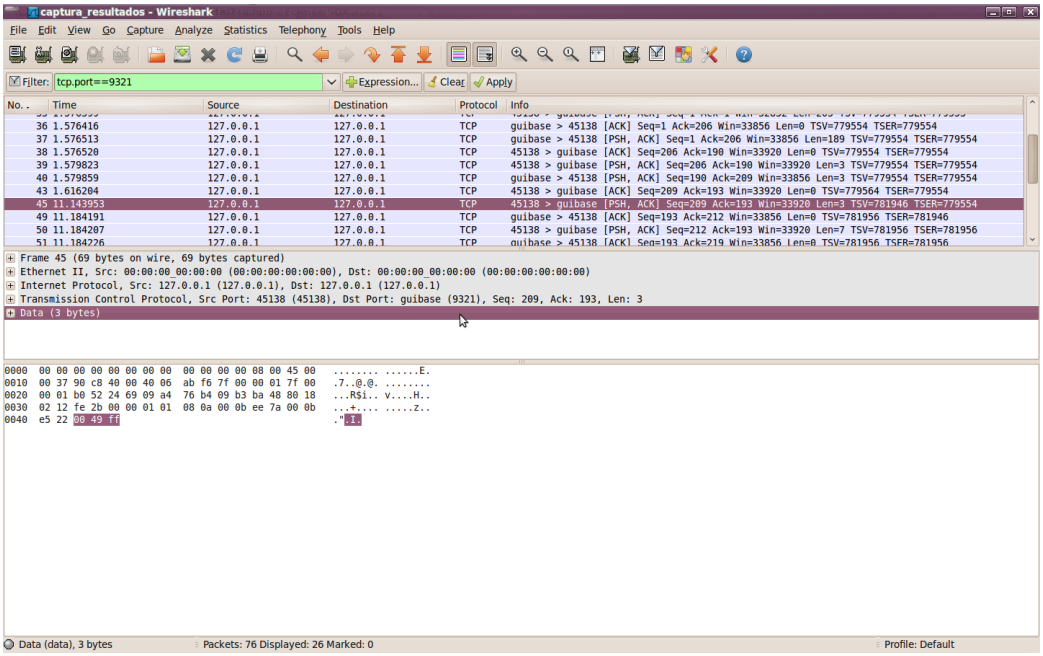


Figura 6.19: La IGU envía la señal para que se inicialice la red

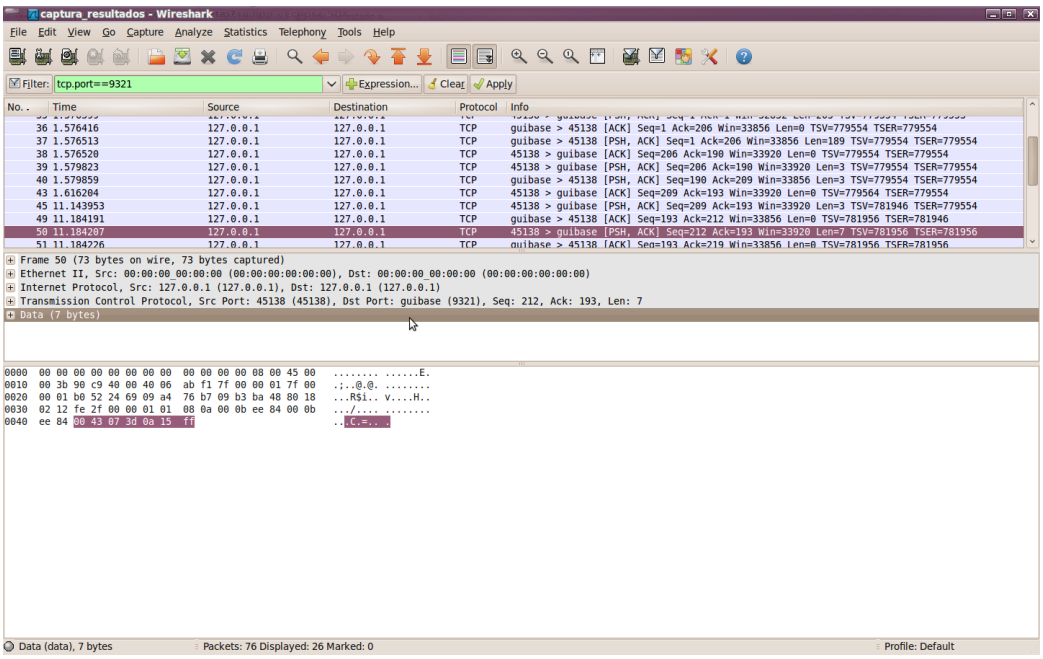


Figura 6.20: La IGU envía los parámetros de la red

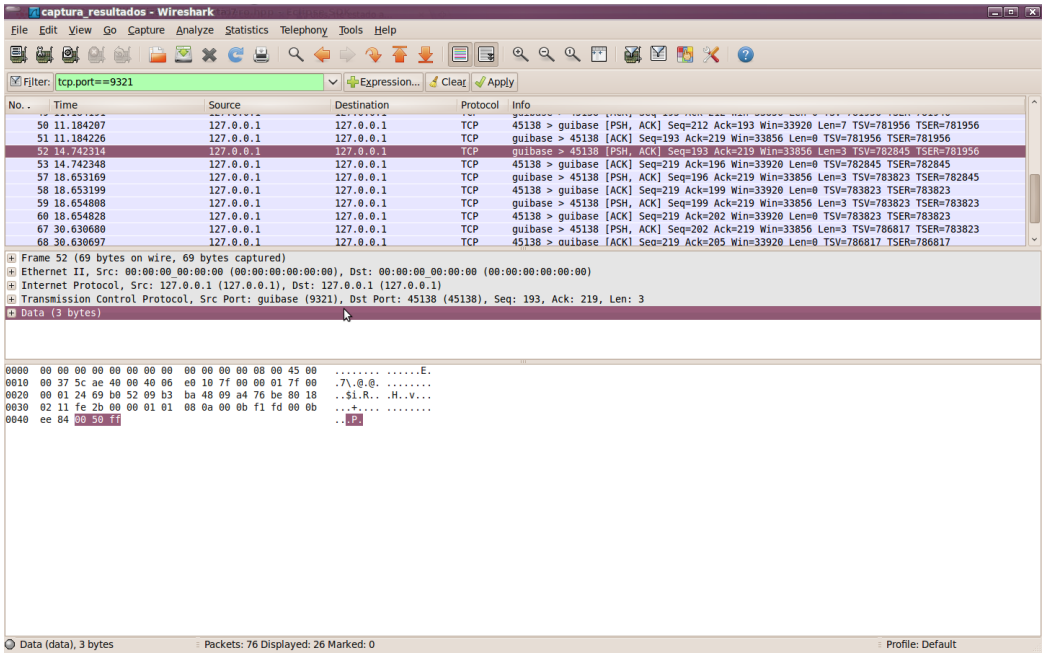


Figura 6.21: La aplicación puente notifica que ha finalizado la etapa de formación de la red

en la figura 6.22 ésta notificación se plasma como una trama que contiene únicamente el caracter 'Q' (0x51).

Cuando la IGU recibe la notificación de nuevas medidas ésta invoca al script PHP con el fin de obtenerlas. Para ello indica al script la acción a realizar mediante los pares clave-valor insertados en la petición GET. Mediante la clave *a* con valor *ln* indica al script PHP que desea que se le proporcione el listado de nodos y sus medidas.

A ésta petición el script responde con una estructura XML dentro de la respuesta del protocolo HTTP. En la figura **??**e puede ver la estructura XML que contiene los datos de los nodos que retornó la BBDD en base a la consulta que realizó el script PHP. En ésta estructura se pueden distinguir los atributos de los nodos como su identificador dentro de la BBDD, la última medición válida que realizó el nodo o el identificador de dicha medición.

Existen mas estructuras XML que son devueltas por el script PHP en función de las peticiones que realiza la IGU. Sin embargo, el propósito de éste documento no es el de ilustrar todas y cada una de ellas sino el proceso de funcionamiento básico de la interacción entre la IGU y la aplicación puente ya que se considera que están suficientemente bien documentadas en los apartados correspondientes a éste protocolo.

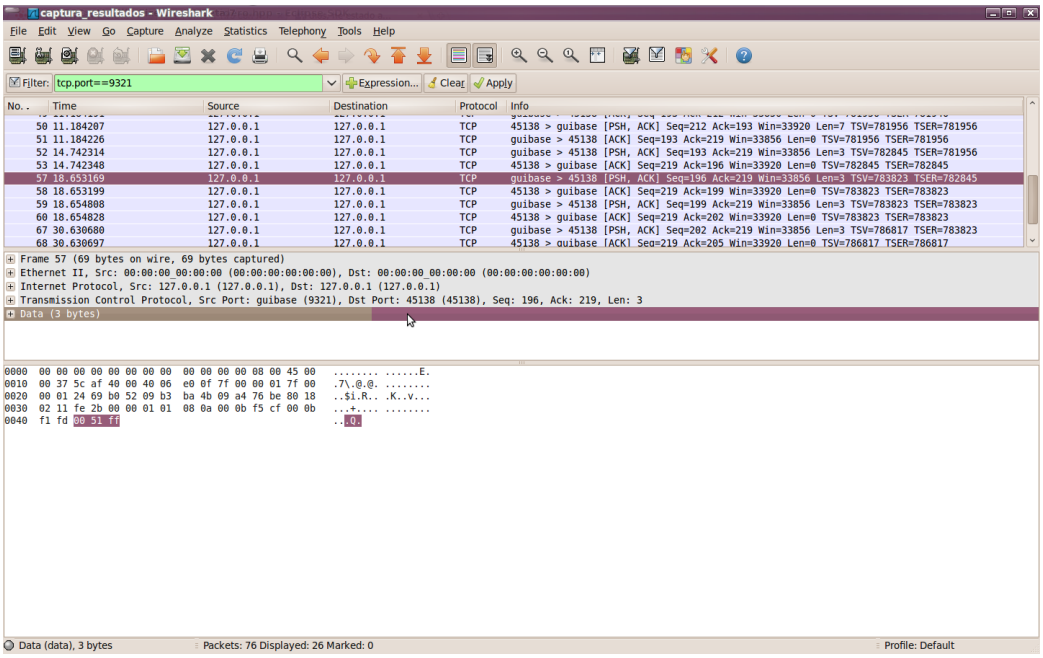


Figura 6.22: La aplicación puente notifica que ha recibido medidas nuevas de los nodos

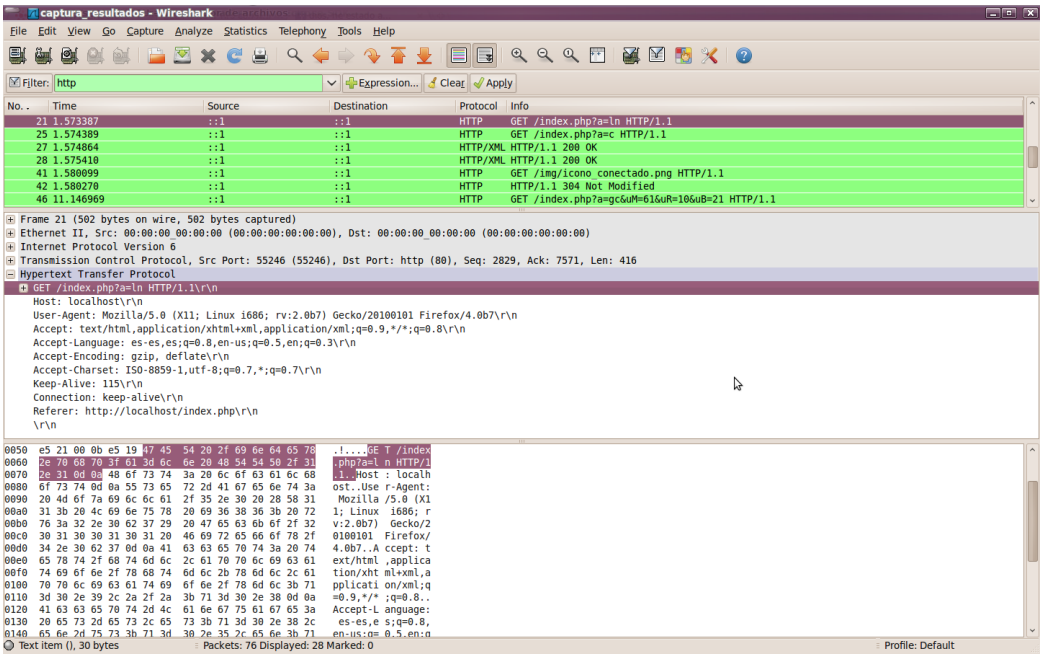


Figura 6.23: La IGU solicita al script PHP la configuración actual

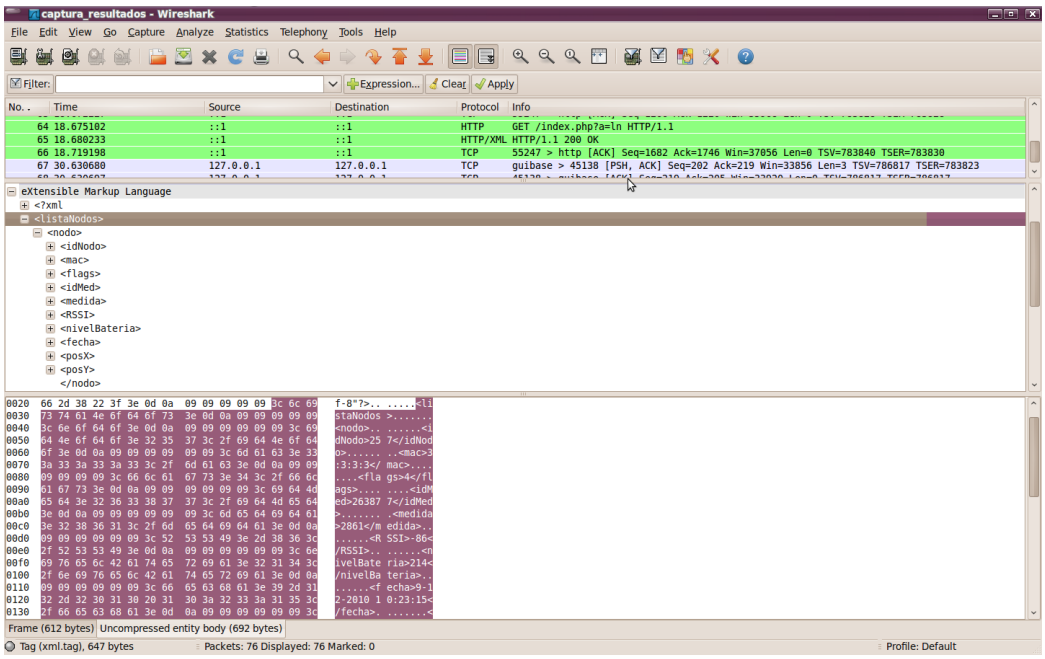


Figura 6.24: La IGU solicita al script PHP la configuración actual

Capítulo 7

Conclusiones y futuras ampliaciones

7.1. Conclusiones

7.1.1. Red de sensores

Los objetivos que se plantearon inicialmente se pueden considerar cumplidos para la red de sensores.

El objetivo principal era conseguir una versión inicial implementada del protocolo y comprobar su viabilidad en una prueba de campo real. No solo se ha cumplido sino que la propia implementación ha suscitado modificaciones y cambios, dando una vuelta de tuerca a la etapa de diseño del protocolo en base a las pruebas de laboratorio.

Es indiscutible que la red requiere poner a prueba su nivel de tolerancia a fallos lo que conllevaría más pruebas, como lo es el hecho de que han quedado cosas en el aire planteadas como ampliación como comprobaremos a continuación. Sin embargo son cuestiones que quedan fuera de los objetivos iniciales.

Entre los objetivos personales estaba el de aprender a programar en microcontrolador y aprender conceptos, tanto básicos como avanzados, de las redes de sensores (un tema muy activo en temas de investigación y con gran proyección de futuro en muchos ámbitos). Ambos objetivos se pueden considerar cumplidos, quizá con algunas lagunas en la programación en microcontrolador, pero con una invaluable experiencia conseguida a lo largo del proyecto.

En general creo que ha sido un trabajo enriquecedor en muchos sentidos, personales y académicos. Lo cual en si mismo puede considerarse como otro

gran objetivo cumplido a lo largo del tiempo dedicado al proyecto.

7.1.2. Tarjeta de adaptación de sensores

La parte de la electrónica del proyecto en principio se planteó como un montaje extremadamente simple con la única finalidad de darle un enfoque mas realista al proyecto de cara a la aplicación teórica que iba a tener. Sin embargo, a medida que avanzó el proyecto ésta parte se tornó relativamente importante en cuanto a complejidad. Ésto causó innumerables problemas en el desarrollo teórico y práctico de la electrónica ya que muchos de los conceptos necesarios para el desarrollo desbordan con creces del contexto de la informática. No obstante, ésto mismo ha sido lo más enriquecedor de la experiencia de ésta parte del proyecto ya que, me he visto obligado a adentrarme de lleno en un campo en el cual soy un profano.

La finalidad de ésta parte del proyecto era conseguir medir deformaciones en el metal, ésto implicaba no sólo adentrarme en el campo de la electrónica de instrumentación, sino además requería adquirir una serie de conocimientos en mecánica de materiales que en un principio ni pensé que fuera a ser necesario. Gracias a ésto he adquirido unos conocimientos (muy básicos) sobre éste campo que, aunque poco o nada tienen que ver con mi especialidad, en cualquier caso son conocimientos, y por lo tanto bienvenidos son.

7.1.3. Aplicación puente y la IGU

En cuanto a la aplicación puente ha sido gratamente satisfactorio ya que éste nexo de unión entre diversas partes del proyecto ha cumplido con creces los objetivos para los que fue pensada. Ha sido una experiencia altamente gratificante el hecho de implementar en C/C++ un servidor WebSocket ya que es un protocolo novedoso, totalmente nuevo, apenas conocido y muy esperado en los entornos del WWW ya que ofrece una solución definitiva a la problemática de la sincronización entre JavaScript y cualquier proceso/servidor remoto. Gracias a éste protocolo, del cual al principio del proyecto no sabía ni de su existencia, creo que la aplicación puente cumple con buena nota sus objetivo y otorga un valor añadido al sistema en general ya que, junto con la tecnología AJAX ofrece una visión de los datos de la red en tiempo real via Web, algo que muy pocas aplicaciones Web pueden garantizar.

Además, gracias al esfuerzo realizado en el desarrollo de la IGU he tenido la oportunidad de aprender a utilizar la tecnología AJAX, la cual hacía mucho tiempo que quería aprender y, por necesidad, también he aprendido a crear y manejar estructuras XML, otra de mis asignaturas pendientes desde hacía mucho.

Creo que AJAX+WebSocket son la piedra angular de ésta parte del proyecto y de los cuales estoy tremendamente satisfecho.

Además, ésta parte de la programación me ha servidor para afianzar sólidamente mis destrezas en lo que a programación en el lenguaje C se refiere y me ha sido muy útil para aprender a usar ciertas facilidades que proporciona C++, sobretodo en el tratamiento de cadenas, un mecanismo altamente usado en la aplicación puente para el tratamiento de los flujos de información con el sumidero y con las IGU.

Otra de las funcionalidades que en principio no iban a ser soportadas pero que finalmente han sido implementadas y de las cuales también estoy muy satisfecho es del soporte multicliente. La aplicación puente es capaz de soportar diversas IGU manteniendo además un mínimo de sincronización entre las mismas de forma que las acciones tomadas en una IGU se verán reflejadas en el resto. No obstante, soy consciente de que éste mecanismo de sincronización es claramente mejorable, pero dado el diseño de la aplicación no supondría añadirle una excesiva complejidad.

7.2. Futuras ampliaciones

7.2.1. Red de sensores

En lo que respecta al protocolo de red diseñado hay varios puntos que se dejaron intencionadamente como ampliación por acotar la complejidad del problema. Otras cosas simplemente se quedaron fuera pero se podría contemplar su implementación futurible como mejora del diseño actual del protocolo.

Entre las cosas que a las que se podrían optar para mejorar las prestaciones del protocolo está la adaptación de la intensidad de señal de los nodos sensores a distintas situaciones, incluso mediante un paquete del nodo sumidero (por ejemplo, cuando tras eliminar un nodo de la red se quede huérfano algún otro nodo). Actualmente los nodos siempre emiten al mínimo de intensidad para favorecer el ahorro de energía. El hecho de hacer que los nodos adaptaran su intensidad de emisión además permitiría estructurar la red con nodos a distancias desiguales sin perder visibilidad entre ellos.

También se contempla como ampliación la implementación de una mejora en el algoritmo de programación de *macro-frames* consistente en la utilización simultánea de enlaces durante un marco temporal siempre que no exista visibilidad entre los nodos que participan. Esto requeriría además replantear la forma en la que se programan los envíos, ya que el algoritmo recursivo no permite este tipo de comprobaciones. En principio los envíos deben seguir siendo desde las hojas a la raíz del árbol, pero con esta mejora varias ramas

(dos o más) podrían enviar a la vez si no son susceptibles de colisionar sus envíos.

Durante algunas pruebas en laboratorio en la fase de desarrollo se detectaron ciertas posibles causas de fallo durante la formación de la red. Se requerirían más pruebas y quizá incluir la posibilidad de que el nodo sumidero eliminara un nodo ya registrado si existen problemas de comunicación, pero quizá hiciera falta un cambio en parte del diseño de la formación de la red para introducir un umbral máximo de reenvíos permitidos.

Por otro lado la señal de «bocinazo» es un requisito del protocolo que no ha pasado de su definición teórica. Se dejó fuera del proyecto desde el principio, como ampliación, el diseño del circuito necesario para emitir las señales de «bocinazo» que permitan despertarse a los nodos sensores del modo de ahorro de energía sin temporizar.

Y por último cabe indicar que no se ha especificado un sistema de encriptación de paquetes, ni tampoco algunos parámetros de red tales como la velocidad de transmisión o el canal o la frecuencia de emisión (en cuyos casos se ha utilizado los mismos que ofrece SimpliciTI por defecto). Estos parámetros formarían lo que sería una ampliación interesante y para ciertos parámetros necesaria. En el caso de la encriptación de la comunicación entre nodos es un paso necesario para su aplicación real, puesto que es necesario cierto grado de seguridad para evitar, no ya que espíen el intercambio de mensajes de la red, sino que suplanten la identidad de alguno de los nodos con fines maliciosos.

7.2.2. Tarjeta de adaptación de sensores

Uno de los principales problemas que tiene el actual diseño de la tarjeta de adaptación de las galgas extensiométricas es que el consumo de la misma es excesivamente alto para unos nodos que están destinados a ser autónomos. Tanto el amplificador como el CAD y las tensiones de referencia tienen un consumo lo suficientemente bajo como para poder ser montados sobre nodos que funcionen con baterías. Sin embargo el puente de galgas consume mucha más energía lo que lo hace ineficiente. Una posible mejor es incorporar a la eletrónica un transistor o un circuito integrado que cumpla la función de «interruptor». Éste interruptor podría estar gobernado por alguna de las salidas del microcontrolador y sería accionado únicamente cuando el nodo va a tomar medidas. Durante el largo transcurso de tiempo durante el cual no hace uso de la electrónica de adaptación de los sensores dicho interruptor quedaría abierto y dejaría de alimentar a dicha electrónica de adaptación. De éste modo se reduciría notablemente el consumo de la tarjeta de adaptación ya que sólo estaría funcionando cuando realmente es necesario.

Tanto las galgas extensiométricas como la electrónica de adaptación no

son perfectas, siempre existe un cierto *offset* en la medida que hay que rectificar. Ésto actualmente no se corrige, pero no sería dificil corregir dicho *offset* mediante software asumiendo que el valor 0 corresponde con una medida inicial sobre las galgas en el momento en el que se activa el nodo. Sin embargo, ésto podría causar también falsas medidas ya que si el metal sobre el que están adheridas las galgas ya se encuentra deformado cuando éstas se adhieren y el nodo es activado, el nodo rectificaría por software la medida de la deformación asumiendo que dicha medida es 0, cuando en realidad es una medida significativa de la deformación.

7.2.3. Aplicación puente USB-IGU

Actualmente la comunicación WebSocket se utiliza con el único fin de notificar a la IGU de que debe actualizar los datos consultandolos a la BBDD a través del script PHP. No obstante, éste mecanismo se podría refinar para que la aplicación puente fuera la que consultara los datos en la BBDD y directamente enviara los mismos a través de la conexión WebSocket a la IGU sin necesidad de utilizar un script PHP creado ex profeso. Actualmente ésto no es posible si nos ceñimos al último borrador de WebSocket, ya que los datos que envíe la aplicación puente deben ser transmitidos en modo binario (se debe poder transmitir valores enteros que no corresponden a caracteres UTF-8), y en el último borrador del protocolo WebSocket se informa de que por ahora no se puede usar el modo de transmisión binario.

7.2.4. Interfaz gráfica de usuario

Una posible mejora de la interfaz gráfica es implementar un mecanismo por el cual se pueda mostrar en la IGU unas gráficas donde se pueda ver el nivel de deformación que han ido midiendo todos los sensores en función del tiempo. Ésta gráfica podría implementarse mediante el objeto *Canvas* que dispone HTML5 o bien con *SVG* e iria actualizandose en tiempo real.

Otra posible mejora de la interfaz gráfica podría ser que el usuario pudiera ver en la imagen del mapa de nodos los enlaces que unen cada par de nodos. Ésto de cara al usuario puede que no sea de mucha utilidad ya que la IGU está pensada para ofrecer un mínimo de abstracción en lo que respecta a la implementación de la red de nodos, pero puede que fuera de utilidad en determinados casos y desde el punto de vista de un PFC orientado a las redes puede que sí fuera interesante.

APÉNDICES

Apéndice A

Manual de Usuario

A.1. Instalación de los nodos

Cada uno de los nodos que conforman la red de sensores está formado por un nodo NK01 (figura A.1) y una tarjeta de adaptación (figuras A.2 y A.3). Ésta tarjeta de adaptación es necesaria para poder conectar al nodo los sensores que irán adheridos a la pieza que será monitorizada.

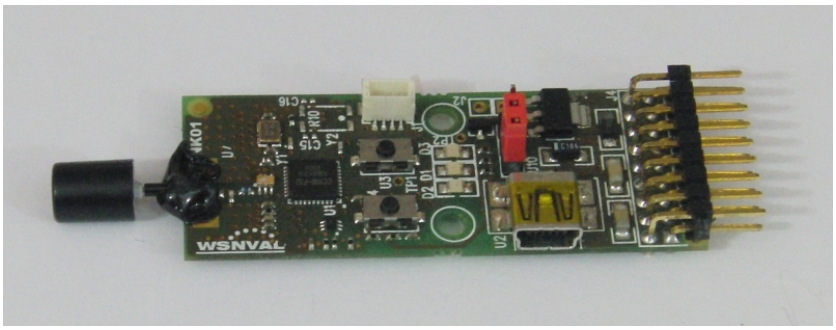


Figura A.1: Nodo NK01 de WSNVAL

El nodo y la tarjeta de adaptación se conectan mediante los conectores IDC-9x2 macho y hembra que incorporan el nodo y la tarjeta de adaptación respectivamente. Para conectar ambos la tarjeta de adaptación debe estar con la cara de la placa que contiene los componentes electrónicos bocabajo mientras el sensor tiene la cara de los componentes bocarriba, tal y como muestra la figura A.4

A.1.1. Adhesión de los sensores a la pieza

Los sensores de los que dispone la tarjeta de adaptación deben ser adheridos con sumo cuidado a la pieza que se desea monitorizar. La instalación de

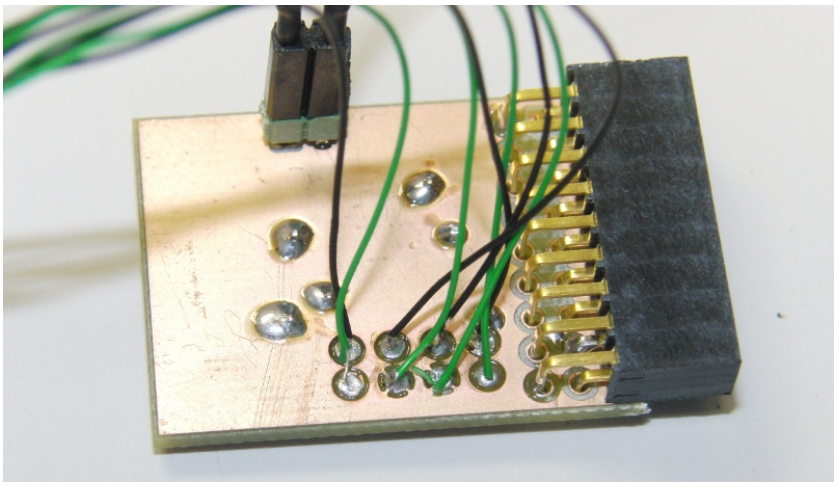


Figura A.2: Tarjeta de adaptación de las galgas extensiométricas (cara superior)

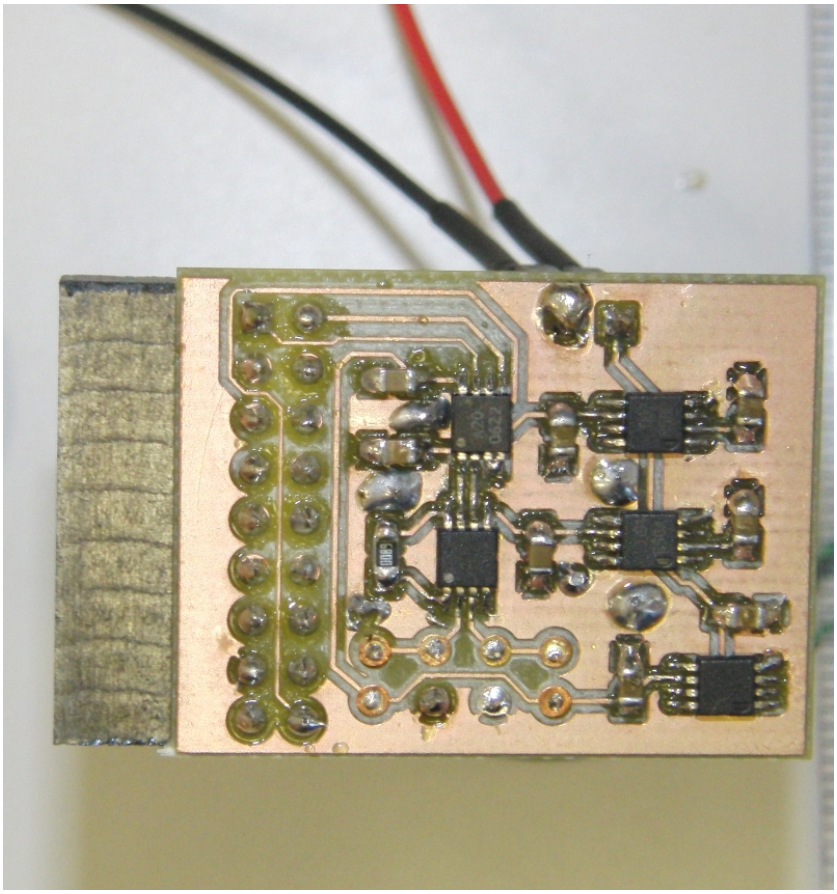


Figura A.3: Tarjeta de adaptación de las galgas extensiométricas (cara inferior)

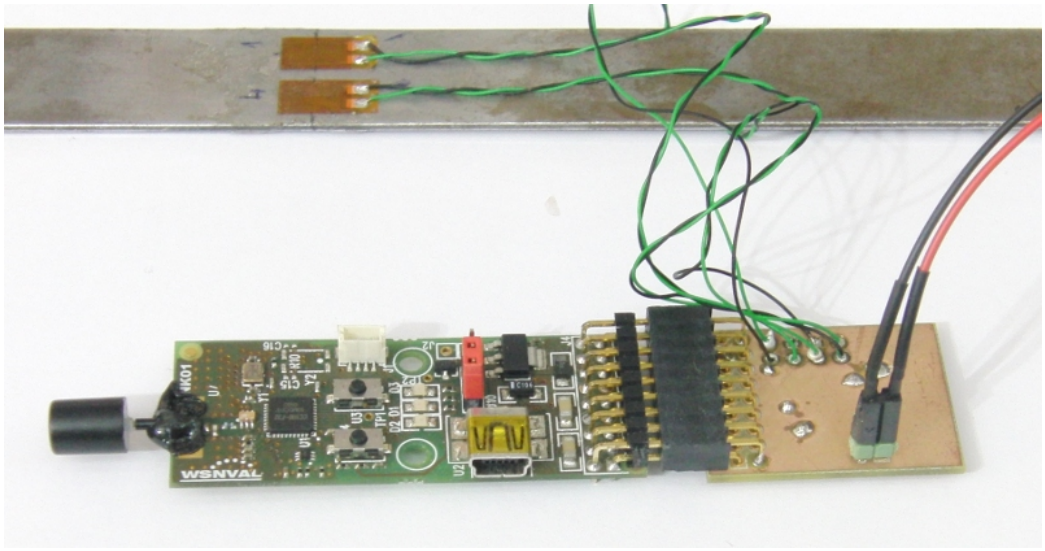


Figura A.4: Nodo conectado a la tarjeta de adaptación de las galgas extensiométricas

éstos sensores es extremadamente importante para obtener medidas fiables. Los sensores son muy sensibles por lo que deben ser manejados con mucho cuidado de deformarlos lo menos posible durante la instalación.

Éstos sensores y la electrónica asociada a ellos están diseñados para medir deformaciones sobre acero, por lo que si se adhieren a otro tipo de material las medidas obtenidas pueden ser erróneas.

Distribución de los sensores en la pieza

Cada tarjeta de adaptación dispone de cuatro sensores con el fin de obtener mas sensibilidad ante una deformación en el material. No obstante hay que tener en cuenta como consecuencia del diseño de la electrónica de la tarjeta de adaptación éstos sensores deben instalarse en una disposición determinada.

Los cuatro sensores se organizan en dos pares, dos de los sensores deben instalarse lo mas cercanamente posible entre sí en un punto de la pieza en el que la deformación sea la misma, es decir, el objetivo es que ambos sensores deformen de igual forma ante una misma deformación. El otro par de sensores debe instalarse también lo mas cercanos posible entre sí de forma que ambos se deformen también de forma solidaria entre sí. Sin embargo ambos pares deben instalarse en puntos de la pieza de forma que ambos pares sufran la deformación contrariar. Es decir, cuando uno de los pares se deforme a tracción el otro par de sensores se deberá deformar a compresión y viceversa.

Limpiar la superficie

Antes de adherir los sensores primero hay que preparar la zona de pieza donde irán adheridos los sensores para que éstos se adhieran correctamente.

Figura A.5: Listón de metal sin preparar

En primer lugar lijaremos la superficie del metal con papel de lija de grano grueso para quitar la suciedad y las imperfecciones más toscas. El metal debe quedar aproximadamente como se muestra en la figura A.6

Figura A.6: Listón después de lijar con papel de grano grueso

En segundo lugar lijaremos con un papel de lija de grano fino para dejar la superficie con un acabado brillante y suave.

Por último limpiaremos con acetona la zona lijada. De ésta forma retiramos los restos que dejó el proceso de lijado y además eliminamos cualquier tipo de grasa o aceite que pudiera tener el metal. Finalmente el metal debe tener un aspecto similar al de las figuras A.8 y A.9

Adhesión de los sensores

El proceso de adhesión de los sensores es especialmente delicado. Los sensores son extremadamente sensibles y una mala adhesión podría dar lugar

Figura A.7: Listón después de lijar con papel de grano fino

Figura A.8: Listón limpiado con acetona

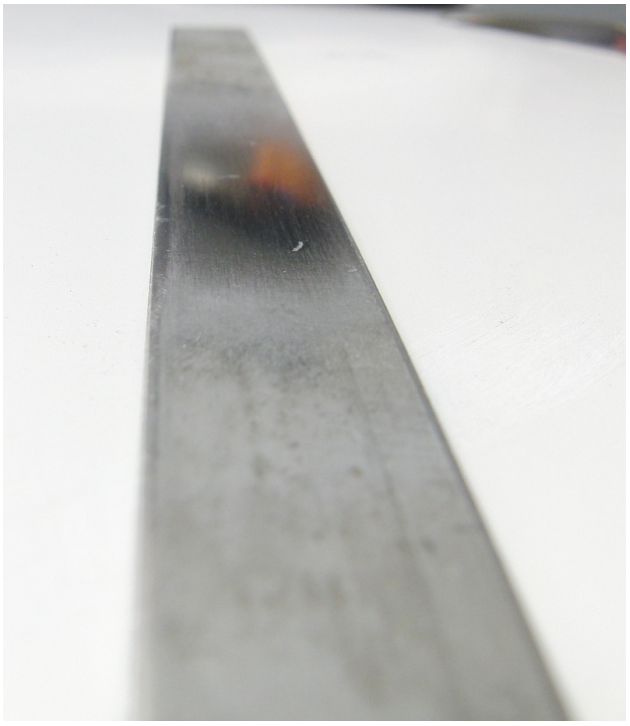


Figura A.9: Listón limpiado con acetona

a medidas erróneas de la deformación del material.

Los sensores llevan unas marcas en forma de flecha o triángulo que sirven como marca de referencia para alinearlos correctamente tanto horizontal como verticalmente. Éstas marcas hay que tenerlas muy en cuenta, pues una mala alineación entre los sensores provocará errores en la lectura de la deformación.

Es muy recomendable dibujar con regla sobre la pieza unos ejes de simetría de forma que se puedan usar en conjunto con las marcas de los sensores para alinearlos correctamente. En el caso de dibujar éstos ejes es recomendable no tocar con los dedos la zona exacta donde irán adheridos los sensores ni dibujar sobre ésta zona ya que la grasa de la piel y las tintas o grafito que se usen para dibujar pueden interferir en la correcta adhesión de las galgas.

Para adherir el sensor en primer lugar cogeremos un trozo de celo y recogemos el sensor usando la parte adhesiva del celo (ver figura A.10. Lo hacemos de ésta forma para evitar tocar en la medida de lo posible el sensor ya que la grasa que cubre nuestra piel puede afectar a la correcta adhesión del mismo.

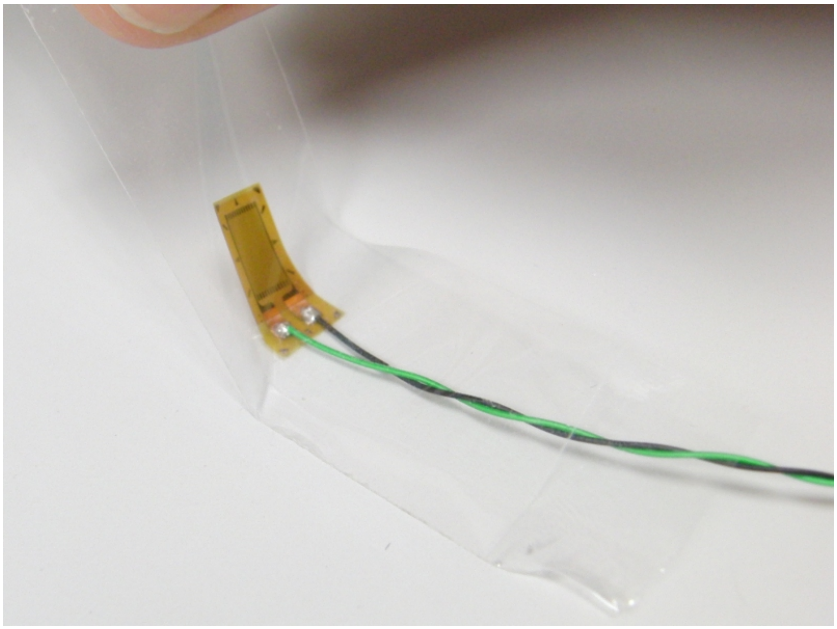


Figura A.10: Recogiendo la galga extensiométrica con celo

Pegamos el celo junto al sensor a la pieza de metal en la posición exacta donde irá adherido finalmente usando para ello los ejes de simetría junto con las marcas de los sensores para alinearlo correctamente (ver figura A.11).

Una vez el sensor está en su posición final despegamos parte del celo de forma que la galga no esté en contacto con el material pero parte del celo si que lo esté y aplicamos una pequeña cantidad de pegamento tal y como se muestra en la figura A.12.

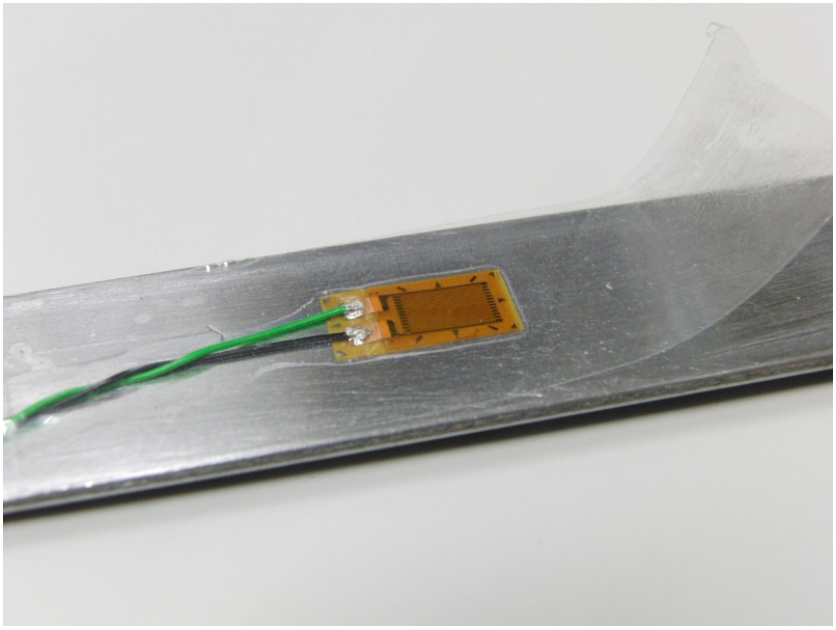


Figura A.11: Pegando la galga con celo provisionalmente

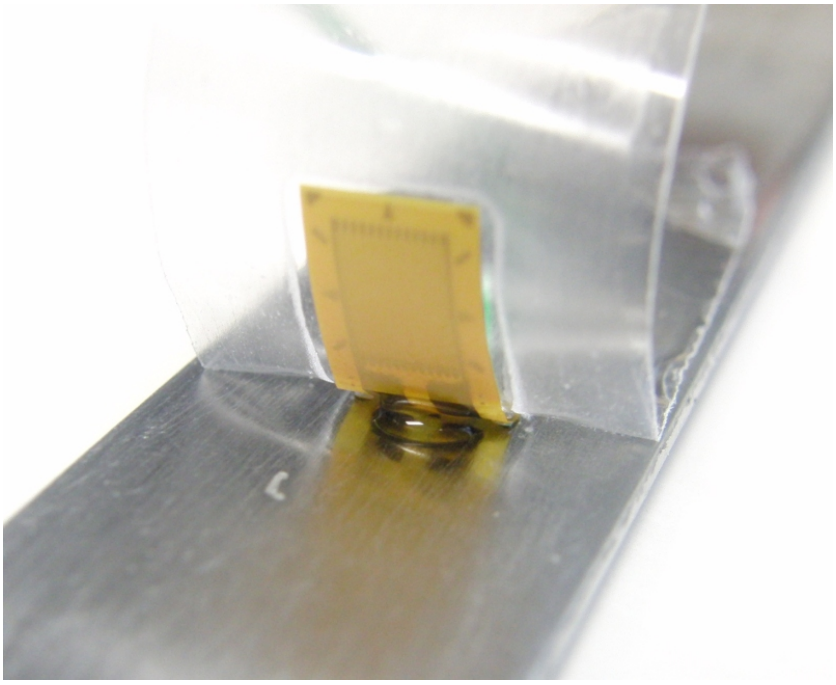


Figura A.12: Aplicando el pegamento con base de cianoacrilato

A continuación, con ayuda del celo volvemos a fijar la galga. Para ello iremos pegando el celo poco a poco y presionando la galga a medida que entre en contacto con el pegamento de forma que entre la galga y el metal tan sólo quede una fina película de pegamento (ver figura A.13. Es recomendable que cuando la galga quede totalmente en contacto con el metal se siga presionando con el dedo durante un minuto aproximadamente.

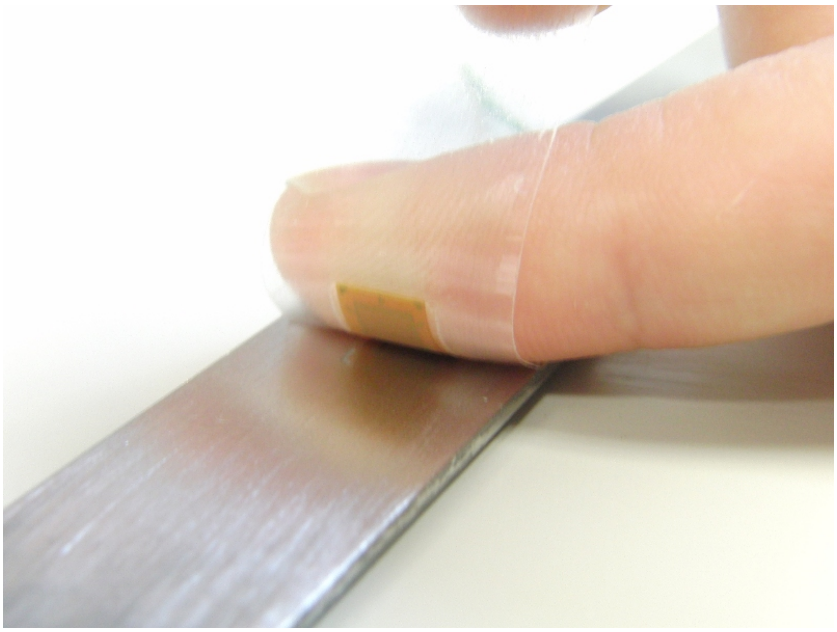


Figura A.13: Adhiriendo la galga definitivamente

Si la instalación de los sensores se va a prolongar durante mas de 6 meses podemos usar pegamentos con base de cionoacrilato (SuperGlue). No obstante, si los sensores van a quedar adheridos durante mas de 6 meses se debe usar pegamentos Epoxy, ya que los pegamentos con base de cianoacrilato a partir de los 6 meses el deterioro de los mismos puede afectar a la precisión de las medidas.

Pasado unos minutos retiramos el celo estirando de él en dirección lo mas paralela a la galga posible y con sumo cuidado. Repetimos el proceso para el resto de galgas del sensor y ya estará listo para medir.

A.1.2. Preparando la BBDD

La aplicación puente necesita una base de datos en la que volcar los datos de la red. Ésta base de datos debe ser MySQL y puede estar ejecutándose en el mismo terminal base donde se ejecuta la aplicación puente o en una maquina distinta, siempre y cuando ésta sea accesible a través del puerto 3306 mediante LAN o Internet desde el terminal base.

Para que los datos puedan ser volcados en la BBDD ésta debe tener disponible una base de datos que contenga una serie de tablas determinadas. Para ello se facilita el archivo *init.sql* que contiene las instrucciones SQL necesarias para crear la base de datos y las tablas necesarias para que funcione todo correctamente. No obstante primero hay que editar el archivo init.sql con el fin de establecer el nombre que tendrá la base de datos que contendrá las tablas. Para ello editamos el archivo *init.sql*. Ejemplo:

```
$ nano init.sql
```

Las dos primeras lineas contienen las siguientes instrucciones:

```
create database rsi;
use rsi;
```

La cadena *rsi* corresponde con el nombre que tendrá la base de datos, por lo que si deseamos cambiarla tendremos que cambiar en ambas lineas la cadena *rsi* por el nombre que deseemos. Por ejemplo, lo cambiamos por *redSensores*:

```
create database redSensores;
use redSensores;
```

Una vez cambiado el nombre de la base de datos, debemos ejecutar las intrucciones SQL que contiene *init.sql*. Para ello ejecutamos:

```
$ mysql -u root -p < init.sql
```

El parámetro *-u* va seguido del nombre del usuario del SGBD que ejecutará las instrucciones contenidas en init.sql por lo que debemos asegurarnos que dicho usuario tiene los permisos necesarios para ejecutar dichas instrucciones. En el ejemplo hemos elegido a root, pero si existiera otro usuario que tuviera los suficientes privilegios en el SGBD para ejecutar dichas instrucciones, podríamos usarlo. Es recomendable crear un usuario ex profeso para todas las acciones relacionadas con la red de sensores ya que la aplicación puente también necesitará un usuario con permisos sobre la base de datos de la red de sensores para ejecutar todas las instrucciones necesarias para hacer uso de las tablas de la misma.

Una vez ejecutamos las instrucciones contenidas en init.sql ya tendriamos la base de datos y sus tablas listas para funcionar.

A.2. Preparación del terminal base

Para que nuestra red de sensores funcione es necesario habilitar un terminal al que se le pueda conectar el nodo sumidero mediante USB. Éste terminal puede ser un PC o cualquier dispositivo que disponga de puerto USB, sea capaz de ejecutar GNU/Linux y tenga acceso a una LAN o a Internet.

En el terminal base deberemos instalar el ejecutable correspondiente a la aplicación puente. Éste ejecutable es el encargado de actuar de intermediario entre el nodo sumidero y las interfaces gráficas de los clientes, así como de volcar en una base de datos los datos de la red de sensores. La base de datos puede estar ejecutándose en el mismo terminal base o en otra máquina a la que tenga acceso el terminal mediante LAN o Internet.

Para instalar ésta aplicación tan sólo hay que copiarla a la estación base. No requiere ningun tipo de configuración extra más que los parámetros proporcionados en linea de comandos en el momento de ejecutarla.

La aplicación siempre debe estar ejecutándose mientras la red está en funcionamiento, de lo contrario las medidas de la red no se volcarán en la base de datos y se perderán. Además, si la aplicación no está ejecutándose obviamente las interfaces gráficas de los clientes tan sólo podrán visualizar las últimas medidas registradas en la base de datos.

Antes de ejecutar la aplicación puente el sumidero deberá estar conectado al puerto USB del terminal, de lo contrario la aplicación fallará.

Una vez tenemos el nodo sumidero conectado al puerto USB del terminal procedemos a ejecutar la aplicación puente. Para ello nos dirigimos al directorio donde resida el ejecutable y la invocamos. Ejemplo:

./puenteUSB-IGU -d /dev/ttyUSB0 –bd-host dataserver –bd-user admin –bd-passwd fRt5AW23Lo –bd-bd rsi

Los parámetros que acepta el ejecutable son los definidos en la tabla A.1. Todos los parámetros son obligatorios.

A partir de éste momento todos los datos procedentes de la red se irán insertando en la base de datos y los clientes ya podrán visualizar en tiempo real las medidas recogidas por la red.

A.3. Instalando la interfaz gráfica

Para instalar la interfaz gráfica descomprimimos el fichero *IGU.zip* en un directorio público del servidor HTTP. El fichero comprimido contiene los ficheros y carpetas que son necesarios para que funcione la interfaz gráfica.

Parámetro	Descripción
-d	Fichero del dispositivo correspondiente al nodo sumidero
–bd-host	Host donde se encuentra la base de datos. El valor para éste parámetro puede ser un nombre de maquina o una IP
–bd-user	El nombre de usuario empleado para autenticarse contra la base de datos
–bd-passwd	El password del usuario empleado para autenticarse contra la base de datos
–bd-bd	El nombre de la base de datos dentro del SGBD

Tabla A.1: Parámetros del ejecutable de la aplicación puente

Dado que la imagen del mapa de nodos puede ser cambiada mediante la propia interfaz gráfica, la carpeta *img* debe ser escribible por el usuario que use el servidor HTTP para ejecutarse. En la mayoría de sistemas GNU/Linux el servidor HTTP se ejecuta con el usuario *www-data* y grupo *www-data*, por lo que la carpeta *img* debe tener permisos de escritura para éste usuario o bien ser el propietario de la misma. Hay diversos métodos de permitir que el usuario *www-data* escriba en una carpeta. Una posible opción es cambiar el grupo de la carpeta a *www-data* y dar permisos de escritura al grupo. Para ello ejecutamos:

```
$ chown www-data img
$ chmod g+w img
```

Dado que el script PHP necesita acceder a la BBDD deberemos proporcionarle los datos correspondientes de ésta para que el script pueda realizar acciones sobre la misma. Para ello editamos el fichero *index.php*:

```
$ nano index.php
```

En las lineas 3,4,5 y 6 se encuentran las variables que corresponden con los datos de la BBDD. En la tabla A.2 se detallan las variables y los valores que deben contener.

En éste punto ya está todo listo para poder conectar el nodo sumidero, ejecutar la aplicación puente y acceder a la interfaz gráfica.

Variable	Descripción
$BDServer	Nombre de la maquina o IP en la que se ejecuta el SGBD
$BDUser	Usuario usado para autenticarse contra la BBDD
$BDPass	Password usado para autenticarse contra la BBDD
$BDBD	Nombre de la base de datos

Tabla A.2: Variables correspondientes a los datos de la BBDD en el script PHP

A.4. Accediendo a la interfaz gráfica

Una vez tenemos la BBDD lista, el nodo sumidero conectado al puerto USB y la aplicación puente en ejecución ya podemos acceder a la interfaz gráfica. Para ello abrimos un navegador y accedemos a la URL correspondiente de la máquina que ejecute el servidor HTTP. Si todo ha ido bien nos aparecerá en la ventana del navegador una página como la de la figura **??**.

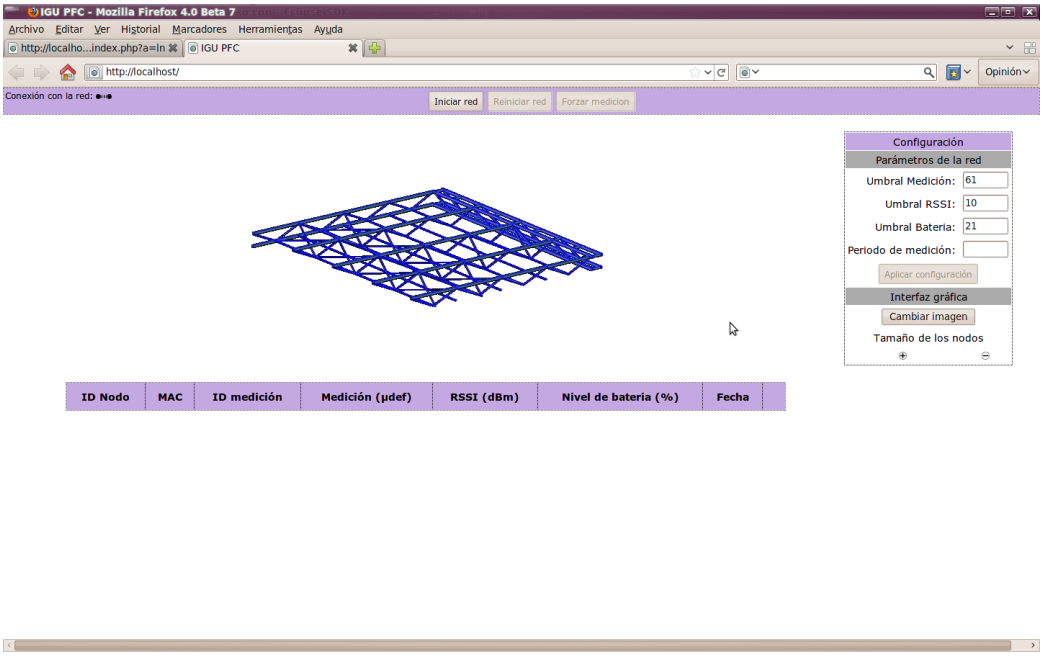


Figura A.14: IGU con la red sin inicializar

En éste momento ya tenemos todo listo para empezar a recoger medidas, sin embargo si no realizamos ninguna acción la red se encuentra en reposo. Es decir, la infraestructura está lista para funcionar pero la red todavía no

está funcionando. Es por ésta razón por la que no se muestra ningún tipo de información sobre la red de sensores.

A.4.1. Elementos de la IGU

Al acceder a la interfaz gráfica nos aparece la interfaz de la red estando ésta en reposo. En ella podemos distinguir varios elementos etiquetados en la figura A.14.

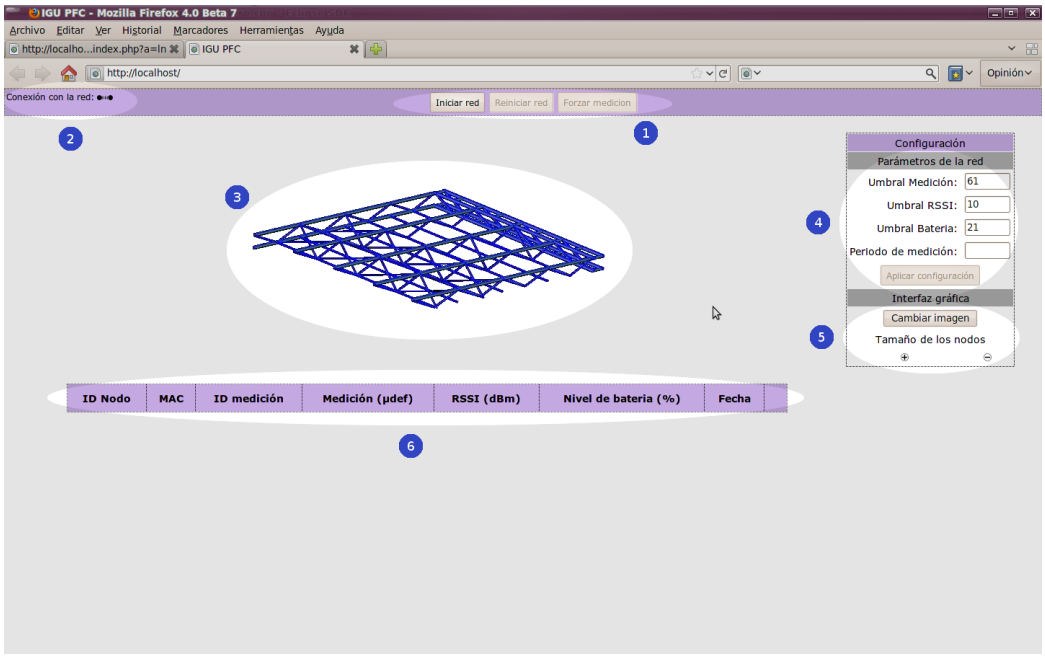


Figura A.15: Elementos del IGU

A.4.2. Parámetros de la red

El diseño de la red de sensores permite ciertas variables que definen ciertos comportamientos de la misma. Éstos parámetros pueden ser configurados desde la interfaz gráfica. Los umbrales de medición, RSSI y bateria son variables que establecen en cuántas unidades debe aumentar cada variable para que un nodo envíe los datos de esa variable a través de la red. Por ejemplo, un valor de umbral de RSSI = 20, establece que cada nodo sólo enviará el dato correspondiente a su RSSI si dicho valor aumenta en 20 unidades o más. Ésto afecta de forma importante a la autonomía de los nodos ya que la acción que más batería consume en un nodo es el hecho de enviar datos por RF. Si un nodo está constantemente enviando todos los valores de su

Num.	Descripción
1	Botones para el control de la red.
2	Indicador de conexión con la red.
3	Imagen del mapa de nodos. En ella se muestra una imagen de la estructura o pieza que estemos monitorizando en la que se pueden superponer los nodos para saber donde se encuentra físicamente cada nodo en la estructura o pieza. Ésta imagen puede ser cambiada por el usuario.
4	Configuración de los parámetros de la red. Se pueden definir distintos parámetros que condicionaran el comportamiento de la red.
5	Configuración de la interfaz gráfica. Se pueden modificar distintos parámetros relativos a la visualización de la interfaz gráfica.
6	Tabla de medidas. Aquí se mostrarán las medidas y otros datos de cada nodo en tiempo real.

Tabla A.3: Elementos de la interfaz gráfica

medición, RSSI y nivel de batería afectará sensiblemente a su autonomía. Sin embargo, si establecemos valores de umbral demasiado altos, las variables tendrán que variar mucho para que sean enviadas a través de la red y posteriormente actualizadas en la BBDD. Ésto tendrá como consecuencia que en la interfaz gráfica visualizaremos los datos actualizados con una cadencia excesiva. Por lo tanto siempre hay que buscar una solución de compromiso que dependerá de las preferencias del usuario. Si el consumo de los nodos no es crítico estableceremos valores de umbrales pequeños, si el consumo sí es crítico estableceremos valores de umbral relativamente alto.

Los parámetros de la red son enviados automáticamente al iniciar la red (ver siguiente apartado) por lo que ésta acción aparece deshabilitada mientras la red no esté incializada. Una vez la red está funcionando los parámetros de la red pueden ser modificados en cualquier momento.

A.4.3. Iniciar la red

Cuando se accede por primera vez a la interfaz gráfica la red de nodos se encuentra en reposo. Para empezar a recibir medidas hay que presionar el botón *Iniciar red* de la barra superior de la página. Una vez se presiona la red empezará a incializarse, éste proceso puede durar varios segundos, el tiempo de inicialización es directamente proporcional al número de nodos de los que disponga la red. Pasado éste tiempo de inicialización se comenzarán a recibir las medidas de los nodos.

A.4.4. Reiniciar la red

La red tolera caídas de nodos de la red, sin embargo si se desea añadir nuevos nodos a la red es necesario reiniciarla. Para reiniciar la red tan sólo hay que presionar el botón *Reiniciar red* de la barra superior de la página. Automáticamente la red se reiniciará, enviando de nuevo los datos de configuración de los parámetros de la red. Al igual que cuando se inicia la red por primera vez ésta quedará bloqueada durante unos segundos mientras se incializa de nuevo la misma.

A.4.5. Forzar medición bajo demanda

La red está diseñada para que se realicen mediciones con un periodo determinado por el parámetro de configuración *Periodo de medición*. Sin embargo, existe la posibilidad de realizar una medición bajo de manda de forma inmediata. Para ello presionamos el botón *Forzar medición* de la barra superior de la página. Inmediatamente se ordenará a todos los nodos que realicen una medición en ese momento. Ésto no afecta al periodo de medición normal, por lo que se volverán a recibir medidas cuando el periodo programado venza.

Cabe destacar que una medición bajo demanda está sujeta a los mismos parámetros de umbrales que las mediciones periódicas, por lo que si alguna variable de un nodo no ha variado lo suficiente como para ser actualizada, éste nodo no la enviará.

A.4.6. Mapa de nodos

La interfaz gráfica dispone de una imagen personalizable que representa la estructura o pieza que se está monitorizando con la red de sensores. El usuario puede cambiar ésta imagen con el botón *Cambiar imagen* que se encuentra en las opciones de configuración de la interfaz gráfica. Una vez la red se inicia y se muestran los nodos disponibles en la tabla de mediciones aparecerá para cada nodo el botón *Posicionar* como se muestra en la figura A.16.

ID Nodo	MAC	ID medición	Medición (µdef)	RSSI (dBm)	Nivel de bateria (%)	Fecha	
254	3:3:3:3	262611	-0.00116	-88	83.92	29-11-2010 13:27:26	Posicionar
255	2:2:2:2	262612	-0.00116	-94	93.73	29-11-2010 13:27:26	Posicionar

Figura A.16: Botón de posicionado del nodo

Se debe posicionar manualmente cada nodo para que éste aparezca en la imagen. Para ello presionamos el boton *Posicionar* del nodo que deseemos

que aparezca en la imagen. Desplazamos el ratón sobre la imagen y automáticamente el cursor del ratón se mostrará como una cruz.

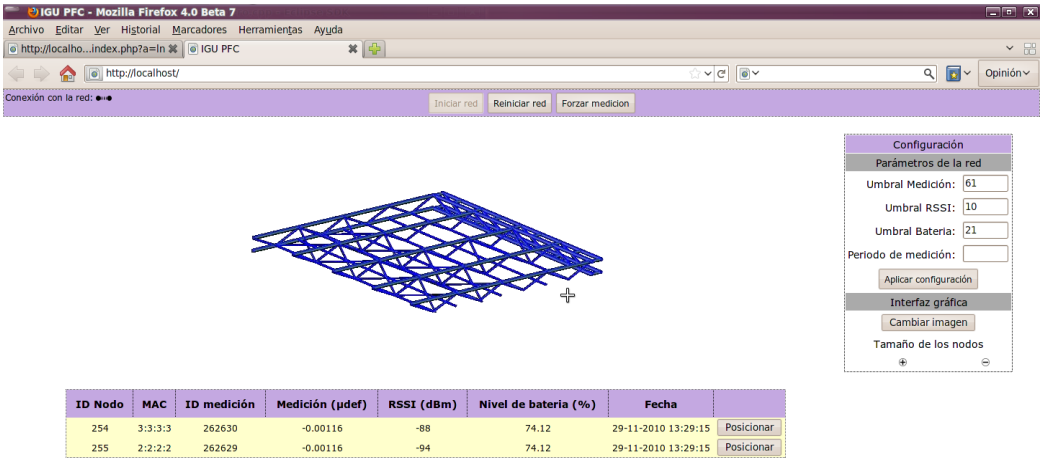


Figura A.17: Posicionando un nodo en la imagen

Hacemos *click* en la posición de la imagen donde queremos posicionar el nodo e inmediatamente aparecerá la imagen del nodo superpuesto a la imagen en la posición donde hemos hecho *click*.

La misma operación la podemos repetir para todos los nodos que aparezcan en el listado de mediciones. Si queremos cambiar de posición un nodo ya posicionado tan sólo debemos repetir el proceso con la nueva posición.

Una vez posicionado un nodo en la imagen, si pasamos el ratón por encima del nodo aparecerá un popup mostrando la información en tiempo real para ése nodo para facilitar al usuario la tarea de vigilar las mediciones de un nodo en particular. De igual forma éste popup desaparecerá cuando el ratón salga de la imagen del nodo.

La posición de los nodos en la imagen es guardada en la base de datos, por lo que aunque se cierre la interfaz gráfica, al volver a acceder los nodos seguirán en la posición donde quedaron guardados. Así mismo, si se accede de forma simultanea desde dos interfaces gráficas distintas, la posición de los nodos será la misma. Es lógico pensar que para cada imagen el mapa de nodos será distinto. Es por ello que al cambiar la imagen de la estructura o pieza a monitorizar, las posiciones de los nodos se borrarán y deberemos posicionar los nodos de nuevo.

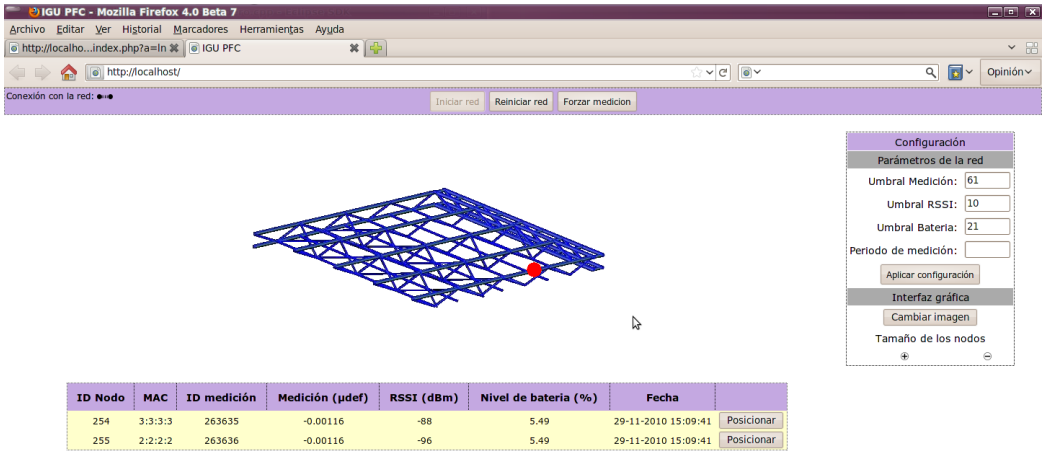


Figura A.18: Nodo posicionado en la imagen

A.4.7. Cambiar la imagen del mapa de nodos

Para cambiar la imagen de la estructura o pieza que estamos monitorizando presionaremos el botón *Cambiar imagen* situado en las opciones de configuración de la interfaz gráfica. Superpuesto a la imagen aparecerá un popup como el que se muestra en la figura A.19, al hacer *click* sobre el botón *Examinar* del popup se abrirá una ventana del navegador de archivos de nuestro sistema operativo. A continuación elegimos la imagen deseada de nuestro disco duro y presionamos el botón *Enviar archivo* del popup. La imagen de nuestro disco se transferirá al servidor y automáticamente se mostrará en la interfaz gráfica.

Cabe destacar que los nodos que hayamos posicionado en la imagen anterior se borrarán de la imagen, por lo que tendremos que volver a posicionarlos para la nueva imagen.

A.4.8. Recibiendo medidas de la red

Cuando se inicializa la red las medidas empezarán a ser mostradas en la tabla de mediciones de la interfaz gráfica como se muestra en la figura A.20. Junto a las medidas se visualizan también datos como el identificador del nodo en la base de datos, la MAC del nodo (identificador de red), el identificador de la medición, nivel de RSSI del nodo (calidad de la señal de radio)

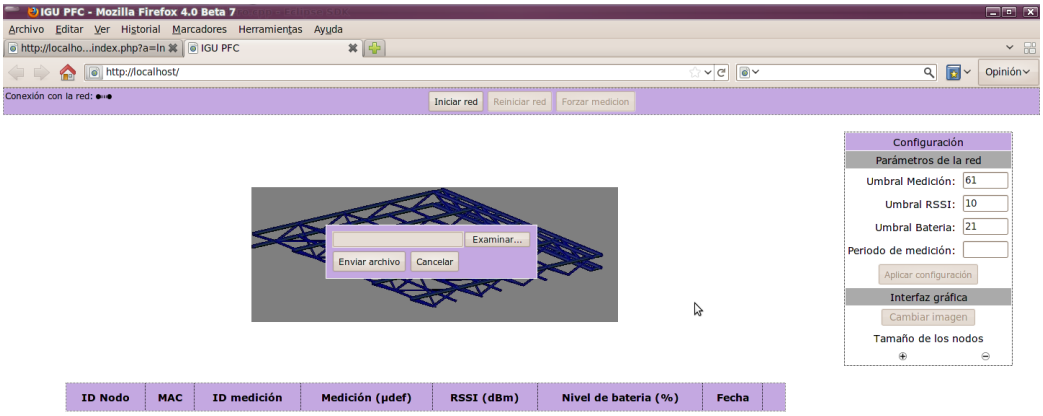


Figura A.19: Popup para cambiar la imagen

y el nivel de batería del nodo.

Los nodos no siempre envían los datos de la medición, RSSI, y nivel de batería. Éstos datos sólo los envían si la variación respecto de la última vez que lo enviaron supera los umbrales definidos en la configuración de los parámetros de red. Por ello, cuando los datos de medición, RSSI o nivel de bateria de algún nodo cambian en la tabla de mediciones de la interfaz gráfica, la celda que contiene el valor que ha variado se marca en verde, para destacar que ése dato ha cambiado con respecto al valor anterior. Un ejemplo se puede ver en la figura A.21, donde el valor de la batería cambia con respecto al valor anterior.

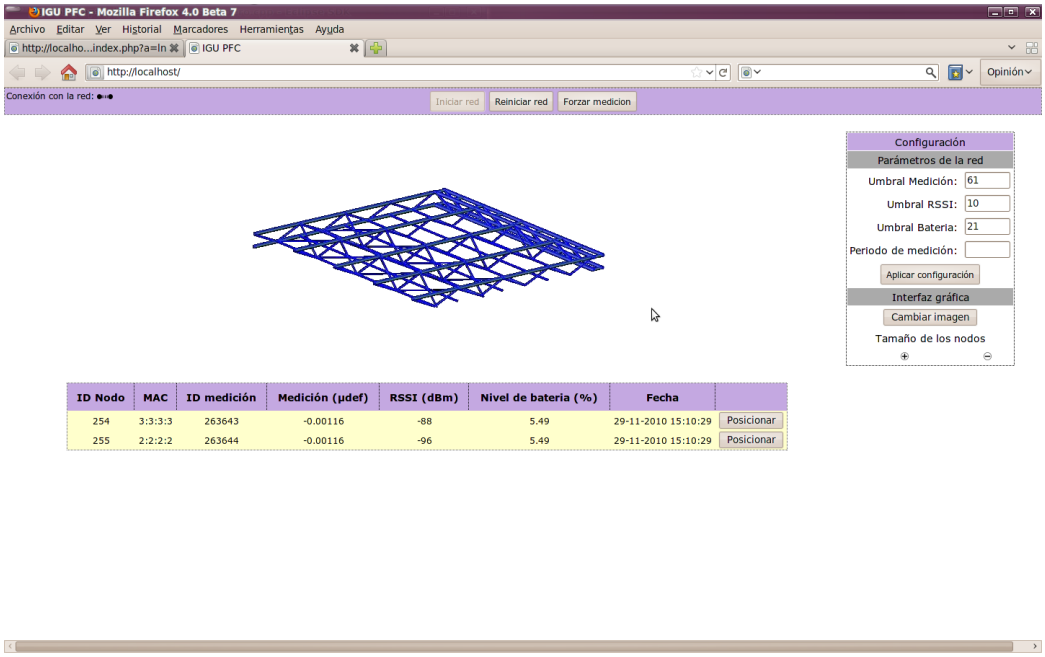


Figura A.20: IGU recibiendo medidas

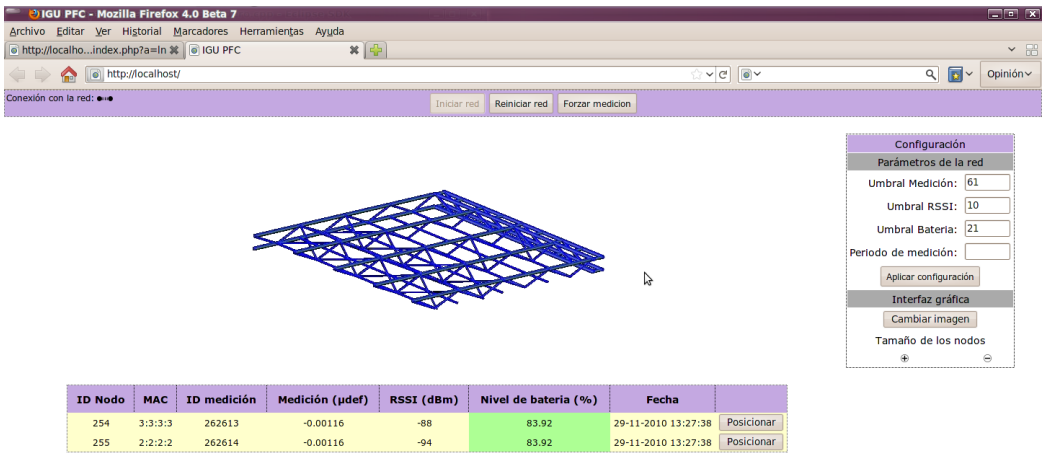


Figura A.21: IGU recibiendo medidas nuevas

Apéndice B

Manual del programador

B.1. Manual de la implementación del protocolo de red

B.1.1. Introducción

La implementación del protocolo de comunicación propuesto se puede entender desde dos puntos de vista distintos, como el propio protocolo: la implementación del nodo sumidero y la implementación de los nodos sensores.

Aunque en la formación de la red tienen comportamientos similares, las estructuras utilizadas y los algoritmos implementados en el sumidero son sustancialmente más complejos.

Por tanto para exponer el desarrollo del protocolo de comunicación se seguirá dicho criterio diferenciador entre el nodo sumidero y el resto de nodos.

B.1.2. Estructuras de datos

Las estructuras usadas se pueden clasificar también según su pertenencia al nodo sumidero, a un nodo sensor o a ambos.

Pero antes de relatar las estructuras usadas cabe comentar una estrategia común utilizada para la práctica totalidad de estructuras que requieren asignación de memoria de un conjunto de elementos (arrays). En lugar de asignar la memoria dinámicamente mediante el uso de librerías de C (como malloc), en la programación del microcontrolador se requiere declarar arrays específicamente como tales.

Esto produce que muchas estructuras sean simplemente conjuntos de elementos con un atributo añadido para indicar la longitud del array que esta

actualmente ocupado. Estas **estructuras array** serán referidas como tales en la descripción de las estructuras a continuación.

Estructuras del nodo sumidero

Lista de nodos

Esta estructura es la que contendrá los nodos registrados durante la etapa de formación de la red y funcionará como una lista enlazada de nodos con un puntero al nodo que lo descubrió (para establecer las rutas en dicha fase). El uso de las listas enlazadas se fue dejando paulatinamente de lado como opción en posteriores estructuras, así pues ésta es una de las primeras estructuras que se utilizaron en esta parte del proyecto y debido a su sencillez no ha sufrido prácticamente ninguna modificación.

```
typedef struct {
    addr_t address;
    struct nodo_t *sig;
    struct nodo_t *desc;
} nodo_t;

typedef struct {
    nodo_t *first;
    nodo_t *last;
    uint8_t length;
} NodeList_t;
```

Nombre	Descripción
address	Dirección física del nodo registrado
sig	Puntero al siguiente nodo de la lista
desc	Puntero al nodo a través del cual se descubrió al actual

Tabla B.1: Descripción de las variables de un nodo_t

La inicialización de esta estructura supone siempre la inserción del propio nodo sumidero con su dirección física y el puntero de *nodo descubridor* inicializado como puntero a nulo, al que apuntarán los primeros nodos registrados.

El control de la lista se realiza a través de la biblioteca *nodeList.h* que incluye también la declaración de las estructuras.

- **initList():** Función que inicializa las variables de la estructura nodeList_t.

Parámetro	Descripción
Parámetros de entrada	
nodeList	Puntero a la estructura declarada que se desea inicializar

- **Prototipo**
  *void initList(NodeList_t *nodeList);*

- **fInsert():** Inserta el primer nodo de la lista (usualmente el nodo sumidero) que no asigna el puntero al descrubridor.
  insert(): Inserta un nodo detrás del último de la lista y asigna el puntero al nodo que lo descubrió.

  - **Prototipos**
    *void fInsert(NodeList_t *nodeList, addr_t *address);*
    *void insert(NodeList_t *nodeList, addr_t *address, struct nodo_t *desc);*

Parámetro	Descripción
Parámetros de entrada	
nodeList	Lista de nodos en la que se desea insertar el nuevo nodo
address	Dirección del nodo que se desea insertar
desc	Puntero al nodo (ya en la lista) que descubrió al que se esta insertando

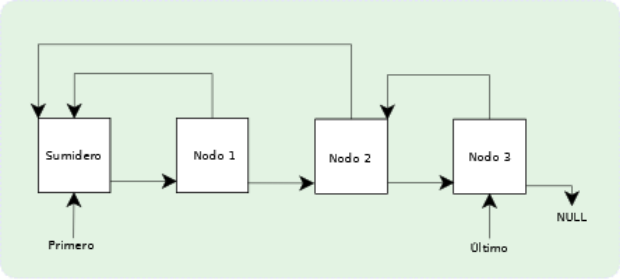


Figura B.1: Diagrama de una Lista de Nodos

Grafo y Variables Dijkstra

Esta estructura es la que contendrá los nodos y los enlaces que formarán el grafo tal y como se ha descrito en el capítulo 3.

```
typedef struct{
    int id;
    addr_t addr;
    uint8_t batt;
    struct ArcList_t *list;
    uint8_t nivel;
    /** Variables Dijkstra **/
    uint8_t cost;
    struct nodoVertice_t *pi;
} nodoVertice_t;

typedef struct{
    struct nodoVertice_t *nodo;
    uint8_t val;
    int8_t nRSSI;
} nodoArco_t;

typedef struct{
    struct nodoVertice_t *first;
    uint8_t length;
} NodeGraph_t;

typedef struct{
    int id;
    nodoArco_t arcos[MAX_ARC_NUM];
    uint8_t length;
} ArcList_t;
```

Los vértices del grafo poseen un identificador para distinguirlos unívocamente del resto de nodos y posicionarlos en el array que los contiene. Se identifican también por la dirección física del nodo que representan. Contienen además el nivel de batería actualizado del nodo, la lista de arcos que salen del vértice y su nivel en el árbol (que tomará valor una vez éste este formado).

El resto de variables del vértice son variables que se usan durante el cálculo de caminos mínimos: las variables Dijkstra. El coste supone el coste acumulado de un nodo para llegar al sumidero pasando por los enlaces que hagan falta. El punto de interés es un puntero al nodo que hará de padre del actual una vez establecido el árbol de caminos mínimos.

En los que respecta a los enlaces del grafo éstos contienen un puntero al nodo al que apunta el arco, el valor de la ponderación del mismo y la intensidad de la señal con la que el nodo que posee el arco percibe al nodo al que éste apunta.

En el cálculo de los caminos mínimos se usa el valor de la ponderación de los arcos para establecer el coste acumulado. Dicha ponderación se basa en los atributos de batería y nivel de intensidad de señal que poseen los elementos relacionados con el cálculo como se ha descrito en el capitulo 3.

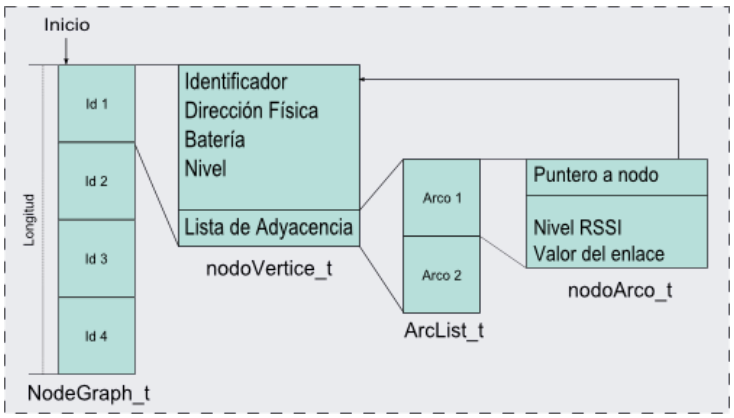


Figura B.2: Diagrama del estructura del grafo

En este caso el manejo de la estructura de grafo se realiza a través de la biblioteca *nodeGraph.h*.

- **initGraph():** Inicializa el grafo insertando al nodo raíz del mismo.
  - **Prototipo**
    *void initGraph(NodeGraph_t *nodeGraph, addr_t address);*

Parámetro	**Descripción**
Parámetros de entrada	
nodeGraph	Puntero al grafo que se desea inicializar
address	Dirección del nodo que se desea insertar como raíz (normalmente la del sumidero)

- **insertVertice():** Inserta un vértice en el grafo inicializándo sus atributos.
  - **Prototipo**
    *int insertVertice(NodeGraph_t *nodeGraph, addr_t address, uint8_t battery);*
- **insertArco():** Inserta un nuevo arco ponderado en el grafo uniendo a dos vértices identificados por su ID.
  - **Prototipo**
    *void insertArco(NodeGraph_t *nodeGraph, int idO, int idD, uint8_t val, uint8_t nRSSI);*

Parámetro	Descripción
Parámetros de entrada	
nodeGraph	Puntero al grafo al que se desea insertar el vértice
address	Dirección del nodo que se desea insertar
battery	Nivel de batería del nodo a insertar

Parámetro	Descripción
Parámetros de entrada	
nodeGraph	Puntero al grafo al que se desea insertar el arco
idO	Identificador del nodo origen del arco
idD	Identificador del nodo destino del arco
val	El peso con el que se ponderará al nuevo arco (el mismo valor para los dos arcos que conforman el enlace)
nRSSI	Nivel de RSSI con que el nodo origen detecta al nodo destino

- **buscarIdVertice():** Busca un vértice en el grafo respecto a su ID.
  - **Prototipo**
    *nodoVertice_t *buscarIdVertice(NodeGraph_t *nodeGraph, int id);*

Parámetro	Descripción
Parámetros de entrada	
nodeGraph	Puntero al grafo en el que se buscará el vértice
id	Identificador del nodo que se desea buscar
Parámetros de salida	
retorno	Puntero al vértice si se ha encontrado o a NULL en caso contrario

- **buscarDirVertice():** Busca un vértice en el grado comparándolo con su dirección física y lo inserta en caso de no encontrarlo.
  - **Prototipo**
    *uint8_t buscarDirVertice(NodeGraph_t *nodeGraph, addr_t address, uint8_t battery);*
- **actualizarArco():** Actualiza el peso de un arco del grafo.
  - **Prototipo**
    *void actualizarArco(NodeGraph_t *nodeGraph, int idO, int idD, uint8_t val);*

Parámetro	Descripción
Parámetros de entrada	
nodeGraph	Puntero al grafo en el que se buscará el vértice
address	Dirección Física del nodo que se desea buscar
battery	Nivel de batería del nodo en caso de que se inserte
Parámetros de salida	
retorno	Identificador del nodo

Parámetro	Descripción
Parámetros de entrada	
nodeGraph	Puntero al grafo a actualizar
idO	Identificador del nodo origen del arco
idD	Identificador del nodo destino del arco
val	Nuevo valor de ponderación de un arco

- **actualizaPondBateria():** Actualiza el nivel de batería de un arco y su peso.

  - **Prototipo**
    *void actualizaPondBateria(NodeGraph_t *nodeGraph, addr_t node, uint8_t batt);*

Parámetro	Descripción
Parámetros de entrada	
nodeGraph	Puntero al grafo a actualizar
node	Dirección del vértice que se desea actualizar
batt	Nuevo nivel de batería del vértice

- **actualizaPondRssi():** Actualiza el nivel de RSSI de un arco y su peso.

  - **Prototipo**
    *void actualizaPondRssi(NodeGraph_t *nodeGraph, addr_t dest, int8_t rssi);*

Parámetro	Descripción
Parámetros de entrada	
nodeGraph	Puntero al grafo a actualizar
dest	Dirección del vértice que se desea actualizar (el nodo destino del arco es el punto de interés de dicho vértice)
rssi	Nuevo valor de RSSI del arco

- **deleteVertice():** Elimina un vértice y con él todos los enlaces (arcos salientes y entrantes) asociados.

  - **Prototipo**
    *int deleteVertice(NodeGraph_t *nodeGraph, nodoVertice_t *vertToDel);*

Parámetro	**Descripción**
Parámetros de entrada	
nodeGraph	Puntero al grafo en el que se eliminará el vértice
vertToDel	Puntero al vértice que se desea eliminar
Parámetros de salida	
retorno	Identificador del nodo eliminado

Cola de prioridad

El sumidero posee una estructura de cola de prioridad implementada con un montículo binario minimal[1], implementado a su vez como un array de elementos de la cola.

```
typedef struct{
    uint8_t indice;
    uint8_t val;
} pQueueItem;

typedef struct{
    pQueueItem items[maxQueue];
    int count;
} pQueue;
```

La cola de prioridad es de uso aconsejable según el desarrollo del algoritmo de caminos mínimos de Dijkstra y a tal fin se ha desarrollado.

Al insertar un elemento en la cola éste se inserta al final de la misma y se reflota hasta su posición según las propiedades de orden del montículo. Por contra, al desencolar el mínimo valor se devuelve la raíz del montículo y se sustituye por el último elemento de la cola para posteriormente hundir la raíz hasta su posición según las propiedades de orden del montículo.

Con este procedimiento se garantiza la propiedad de orden y la obtención del mínimo de entre todos los datos en tiempo constante.

[1]Estructura que almacena un conjunto ordenado de elementos, en el que cada nodo padre tiene un valor menor que el de todos sus hijos

La biblioteca encargada del control de la cola de prioridad es *priorityQueue.h*.

- **initQueue():** Inicializa la cola de prioridad, reiniciando sus atributos.
  - **Prototipo**
    *void initQueue(pQueue *C);*

Parámetro	Descripción
C	Puntero a la cola de prioridad que se inicializará

- **insertQueue():** Inserta un elemento al final de la cola, reflotándolo hasta su posición.
  - **Prototipo**
    *void insertQueue(pQueue *C, uint8_t indice, uint8_t value);*

Parámetro	Descripción
Parámetros de entrada	
C	Puntero a la cola de prioridad sobre la que se insertará el elemento
indice	Atributo índice del elemento a insertar
value	Atributo valor del elemento a insertar

- **getMinQueue():** Devuelve la raíz del montículo (elemento con valor mínimo de la cola) y lo elimina, restaurando el orden.
  - **Prototipo**
    *pQueueItem getMinQueue(pQueue *C);*

Parámetro	Descripción
Parámetros de entrada	
C	Puntero a la cola de prioridad de la que se extrae la raíz
Parámetros de salida	
retorno	Valor literal de la raíz antes de restaurar el orden

Control de Errores

Para los errores en la recepción de medidas el nodo sumidero consulta un flag contenido en una estructura array con dicha variable de control. La estructura indica si se ha producido error, en cuántos nodos y cuales son las direcciones físicas de dichos nodos.

```
typedef struct {
    addr_t *nodes;
    uint8_t len, errFlag;
} errorNodes_t;
```

Cada vez que se encuentra el flag activo se programa un *macro-frame* de emergencia para recibir las medidas del subárbol que produjo el error y se decrementa el contador de nodos de la estructura. Si llega a cero se desactiva el flag.

Control de recepción de medidas

El nodo sumidero espera la recepción de medidas durante todo el *macro-frame* directamente de sus nodos hijos que habrán acumulado las del resto del subárbol que cuelga de cada uno de ellos.

Esta estructura sirve para contener las direcciones físicas de los nodos hijos y el número de hijos de los que debe esperar medidas y como contador para el número de nodos recibidos actualmente. La estructura se actualiza cada vez que se recalcula la programación del *macro-frame* y permite reducir el coste de tener que consultar el grafo para comprobar cuántos y cuáles son los nodos hijos del sumidero al de consultar ésta estructura array.

```
typedef struct{
    addr_t recNodes[6];
    uint8_t length;
    uint8_t act;
} recFramesSum_t;
```

Estructuras de los nodos sensores

Control de recepción y envío de medidas

Similar a la estructura de control de recepción de medidas del sumidero, pero además debe almacenar los tiempos de espera que se producen bien antes de la primera recepción (tiempo que pasará en un estado de bajo consumo), o bien entre la recepción de un paquete de medidas y el siguiente (tiempo que pasará en espera activa). También almacena la dirección del nodo padre al que debe enviar su paquete de medidas y cuántos marcos temporales deberá consumir enviándolos.

```
typedef struct{
    addr_t recNodes[6], sendNode;
```

```
    int waitTimes[6];
    uint8_t length, sendFrames, act;
} procFrames_t;
```

Esta estructura se renueva en cada *macro-frame* al procesar la programación recibida del sumidero.

Estructuras comunes

Tablas de Intensidades

La Tabla de Intensidades se refiere a la estructura que los nodos sensores envían al nodo sumidero conteniendo su nivel de batería y la dirección física de los nodos que ha registrado junto con la intensidades con la que los percibe durante la formación de la red. Ésta última información relativa a los nodos registrados es lo que compone cada una de las filas en la tabla de intensidades.

```
typedef struct {
    addr_t  address;
    uint8_t nivelRSSI;
} intensityRow_t;

typedef struct {
    addr_t address;
    intensityRow_t table[MAX_TABLE_ROWS];
    uint8_t battery;
    int length;
} intensityTable_t;

typedef struct {
    intensityTable_t iTable[MAX_TABLE_ARRAY];
    int length;
} iTableArray_t;
```

El nodo sumidero utiliza además una estructura array de tablas de intensidades para almacenar las tablas de todos los nodos que registre en la red. La primera de las tablas almacenada en dicho array será siempre la del propio sumidero.

Contenedores de medidas

Esta estructura esta diseñada con el fin de facilitar la toma, envío y recepción de medidas por parte de los nodos de la red. Se acumulan y almacenan

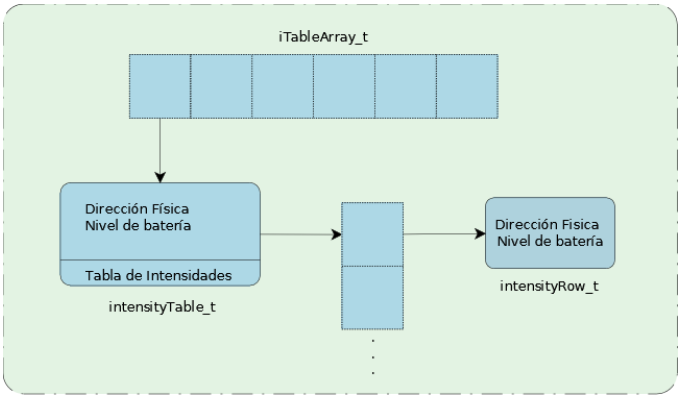


Figura B.3: Diagrama de las Tablas de Intensidades

haciendo uso de una estructura array que contiene las medidas de los distintos nodos que se han acumulado hasta la recepción del paquete.

```
typedef struct {
    addr_t node;
    uint8_t flags, rssi, battery;
    uint16_t measure;
} mNode_t;

typedef struct {
    mNode_t *measures_array;
    uint8_t length;
    uint8_t index;
} measures_array_t;
```

De cada nodo se almacena su dirección física, los flags que indican qué medidas de la estructura se han recibido y las correspondientes medidas.

La estructura se debe inicializar y práctica todo el manejo de la misma se produce en la recepción del paquete de medidas. Sin embargo se facilita su uso desde la biblioteca *controlList.h*. A continuación mostraremos una breve descripción de las funciones más importantes y sus prototipos (figura B.4):

- **initIndex():** Inicializa la estructura que contendrá las mediciones de los nodos
- **insertNode():** Inserta un nuevo elemento con los datos que se le proporcionen
- **insertFlags() - insertMeasure() - insertRssi() - insertBattery():** Inserta cada dato por separado sobre el último elemento

- **updateFlagRssi():** Inicializa el flag de medida de RSSI (que es el único que se actualiza al recibir una medición).

```
28 void initIndex(measures_array_t *meas);
29 void insertNode(measures_array_t *meas, addr_t nodeAddr, uint8_t nodeFlags)
30 void insertNodeNoFlags(measures_array_t *meas, addr_t nodeAddr);
31 void insertFlags(measures_array_t *meas, uint8_t nodeFlags);
32 void updateFlagRssi(measures_array_t *meas);
33 void insertMeasure(measures_array_t *meas, uint16_t m);
34 void insertRssi(measures_array_t *meas, uint8_t rssi);
35 void insertBattery(measures_array_t *meas, uint8_t battery);
```

Figura B.4: Prototipos de las funciones de *controlList.h*

B.1.3. Bibliotecas Software

En esta sección se detallarán las bibliotecas desarrolladas específicamente para la implemententación del protocolo. Esta vez no se hará la distinción entre nodos ya que las bibliotecas desarrolladas son en su mayoría de uso general por todos los nodos de la red.

No todas las bibliotecas, por otro lado, están directamente relacionadas con el desarrollo del protocolo en sí mismo sino más bien con el manejo de ciertas funciones del microcontrolador o del propio nodo WSNVAL. La finalidad del desarrollo de las últimas es por comodidad en el manejo de varias características que sí son habituales en el protocolo y a las que no dan soporte otras bibliotecas (SimpliciTI).

Por último cabe mencionar que en esta sección no se hará referencia a las bibliotecas software para el manejo de las estructuras vistas en el punto anterior, ya que ya aparecen explícitamente mencionadas junto con las estructuras que gestionan. La mayor parte de las estructuras desarrolladas poseen una biblioteca que permite hacer uso de ellas sin gestionarlas directamente desde la clase principal. Las estructuras son contenedores de datos que se inicializan y en los que se insertan o se extraen datos a través de funciones de sus respectivas bibliotecas.

Manejo de temporizadores - *timerLayer.h*

El manejo de los cuatro temporizadores del microcontrolador se hace a través del uso de esta biblioteca desarrollada específicamente a tal fin. En la biblioteca se encuentras funciones para el manejo de los temporizadores 2, 3 y 4, algunas macros[2] asociadas al temporizador de ahorro de energía o *sleep-timer* e incluso el manejo de los divisores del reloj y la selección del oscilador utilizado o su velocidad.

[2]Abreviaturas que el compilador sustituye en cada aparición por un fragmento de programa asociado.

El temporizador 2 no se usa en el protocolo, pero se usó en sus etapas iniciales de desarrollo y se han mantenido las funciones relacionadas con su manejo. Los temporizadores 3 y 4 son gemelos y se usan ambos dos, cambiando solo el registro sobre el que funcionan pero haciéndolo de forma pareja. El temporizador 1 se usa de forma específica en la biblioteca desarrollada para el manejo de la UART pero de forma trivial, así que se optó por manejar sus registros asociados directamente sin pasar por la biblioteca.

Los temporizadores 3 y 4 que son los que se utilizan específicamente en el protocolo de comunicación se utilizan para gestionar los reenvíos de mensajes en caso del vencimiento del temporizador. Se necesitan al menos dos temporizadores ya que en algunos momentos un nodo puede estar esperando varios acuses de recibo o contestaciones de dos paquetes distintos. También se utilizan para las esperas activas y la sincronización de los marcos temporales.

La biblioteca permite establecer los indicadores (*flags*) de interrupción de los temporizadores soportados, establecer los contadores, el modo de funcionamiento, etcétera (detallado en el capítulo 4).

El uso del temporizador 3 es compartido con ciertas funciones de las bibliotecas del SimpliciTI, así que se fuerza a restaurar su configuración entre usos del mismo.

```
61 void global_IntEnable(void);
62 void global_IntDisable(void);
63
64 void sleepTimer_IntEnable(void);
65
66 void timer2_IntEnable(void);
67 void timer2_IntDisable(void);
68 void timer2_SetTickPeriod(uint8_t period);
69 void timer2_SetPrescaler(uint8_t prescaler);
70 void timer2_SetCounter(uint8_t counter);
71 void timer2_SetMode(uint8_t mode);
72 void timer2_INIT(void);
73
74 void timer3_IntEnable(void);
75 void timer3_IntDisable(void);
76 void timer3_Start(void);
77 void timer3_Stop(void);
78 void timer3_ClearCounter(void);
79 void timer3_SetPrescalerDivider(uint8_t divider);
80 void timer3_SetMode(uint8_t mode);
```

```
82 void timer4_IntEnable(void);
83 void timer4_IntDisable(void);
84 void timer4_Start(void);
85 void timer4_Stop(void);
86 void timer4_ClearCounter(void);
87 void timer4_SetPrescalerDivider(uint8_t divider);
88 void timer4_SetMode(uint8_t mode);
89
90 void setClockControlSpeed(uint8_t tickSpd);
91 void selectOscillator(uint8_t option);
```

Figura B.5: Prototipos de las funciones de *timerLayer.h*

Macros

Constante	Valor	Descripción
T2CTL_TIP_64	0x00	$T = T2CTL * 64$
T2CTL_TIP_128	0x01	$T = T2CTL * 128$
T2CTL_TIP_256	0x02	$T = T2CTL * 256$
T2CTL_TIP_1024	0x03	$T = T2CTL * 1024$

Tabla B.2: Multiplicadores disponibles para el periodo del temporizador 2

Constante	Valor	Descripción
TIMER2_USE_REG	0x00	El generador de ticks funciona cuando el contador es distinto de 0
TIMER2_FREE	0x01	Funciona repetidamente generando interrupciones en 0

Tabla B.3: Modos disponibles para el temporizador 2

Constante	Valor	Descripción
T3_T4CTL_DIV_1	0x00	Frecuencia de tick / 1
T3_T4CTL_DIV_2	0x20	Frecuencia de tick / 2
T3_T4CTL_DIV_4	0x40	Frecuencia de tick / 4
T3_T4CTL_DIV_8	0x60	Frecuencia de tick / 8
T3_T4CTL_DIV_16	0x80	Frecuencia de tick / 16
T3_T4CTL_DIV_32	0xA0	Frecuencia de tick / 32
T3_T4CTL_DIV_64	0xC0	Frecuencia de tick / 64
T3_T4CTL_DIV_128	0xE0	Frecuencia de tick / 128

Tabla B.4: Divisores disponibles para los temporizadores 3 y 4

Constante	Valor	Descripción
T3_T4CTL_MODE_FREERUN	0x00	Free Run → Cuenta repetidamente desde 0x00 a 0x00
T3_T4CTL_MODE_DOWN	0x01	Down → Cuenta desde el valor del contador hasta 0x00
T3_T4CTL_MODE_MODULO	0x02	Modulo → Cuenta desde 0x00 hasta el valor del contador
T3_T4CTL_MODE_UP_DOWN	0x03	Up / Down → Cuenta repetidamente desde 0x00 hasta el valor del contador y de nuevo desde el valor del contador a 0x00

Tabla B.5: Modos disponibles para los temporizadores 3 y 4

Empaquetado y desempaquetado de mensajes - *packetFormat.h*

Esta biblioteca es la encargada de empaquetar (y enviar) y desempaquetar los mensajes del protocolo de comunicación descrito en el Capítulo 3. Cuando una interrupción de radio es recibida se lanza, entre otras, una función para establecer el identificador del paquete para poder distinguir qué hacer con él.

Si el paquete es esperado según los diagramas de comunicación establecidos éste se procesará con alguna de las funciones de la biblioteca. En general para desempaquetar un mensaje se lee el mensaje contenido en la estructura *ioctlRawReceive_t* del SimpliciTI byte a byte y se insertan los datos con su

Constante	Valor	Descripción
CLKCON_TICKSPD_DIV_1	0x00	Frecuencia de reloj
CLKCON_TICKSPD_DIV_2	0x08	Frecuencia de reloj / 2
CLKCON_TICKSPD_DIV_4	0x10	Frecuencia de reloj / 4
CLKCON_TICKSPD_DIV_8	0x18	Frecuencia de reloj / 8
CLKCON_TICKSPD_DIV_16	0x20	Frecuencia de reloj / 16
CLKCON_TICKSPD_DIV_32	0x28	Frecuencia de reloj / 32
CLKCON_TICKSPD_DIV_64	0x30	Frecuencia de reloj / 64
CLKCON_TICKSPD_DIV_128	0x38	Frecuencia de reloj / 128

Tabla B.6: Divisores disponibles para reloj del sistema

debido formato en las estructuras que debiera en su caso (facilitadas a la función por referencia) para que el nodo las maneje.

Las funciones de recepción devuelven además un booleano indicando el correcto procesamiento del mensaje.

El empaquetado incluye el envío del mensaje ya que son funciones parejas. Empaquetar un mensaje supone recibir por valor en la función los datos que se deseen enviar, dependiendo del tipo de paquete, y que la función los procese y los asigne a una cadena de bytes en la estructura *ioctlRawSend_t* del SimpliciTI para enviarlos a la dirección deseada a través del puerto 0x3F (como se indicó en el Capítulo 4).

```
21 /** Envío de mensajes del protocolo de comunicación **/
22 void sendACK(addr_t *address, ioctlRawSend_t *msg);
23 void sendDiscover(ioctlRawSend_t *msg);
24 void sendReplyDiscover(addr_t *address, ioctlRawSend_t *msg);
25 void sendDiscoverOrder(addr_t *address, route_array_t *route, ioctlRawSend_t *msg);
26 void sendIntensityTable(addr_t *address, route_array_t *route, uint8_t battery, intensityTable_t intensityTable, ioctlRawSend_t *msg);
27 void sendFrameProg(frame_array_t *frame, ioctlRawSend_t *msg);
28 void sendMeasures(addr_t *address, measures_array_t meas, ioctlRawSend_t *msg, int i);
29 void sendReConfig(uint8_t flags, uint8_t measure, uint8_t rssi, uint8_t battery, ioctlRawSend_t *msg);
30
31 /** Recepción de mensajes del protocolo de comunicación **/
32 uint8_t receivePacket(ioctlRawReceive_t msg);
33 int receiveACK(ioctlRawReceive_t msg);
34 int receiveDiscover(ioctlRawReceive_t msg);
35 int receiveReplyDiscover(ioctlRawReceive_t msg);
36 int receiveDiscoverOrder(ioctlRawReceive_t msg, route_array_t *res);
37 int receiveIntensityTable(ioctlRawReceive_t msg, addr_t address, intensityTable_t *resTable, route_array_t *resRoute);
38 int receiveFrameProg(ioctlRawReceive_t msg, frame_array_t *frame);
39 int receiveMeasures(ioctlRawReceive_t msg, measures_array_t *meas);
40 int receiveReConfig(ioctlRawReceive_t msg, uint8_t *measure, uint8_t *rssi, uint8_t *battery);
41
42 /** Funciones auxiliares **/
43 void addrToUint8(addr_t addr, uint8_t *res);
44 void uint8ToAddr(uint8_t *pointer, addr_t *res);
45 void routeToUint8(route_array_t *route, uint8_t *res);
46 void intensityRowToUint8(intensityTable_t table, uint8_t *res);
47 void frameArrayToUint8(frame_array_t *frame, uint8_t *res);
48 void getRoute(ioctlRawReceive_t msg, uint8_t len, addr_t **res);
49 int getTable(ioctlRawReceive_t msg, uint8_t lenRoute, intensityRow_t res[]);
50 uint8_t getFrames(ioctlRawReceive_t msg, addr_t **res);
```

Figura B.6: Prototipos de las funciones de *packetFormat.h*

Manejo de LEDs - *LEDlib.h*

Los nodos sensores de WSNVAL poseen tres LEDs[3]: uno verde, uno amarillo y uno rojo. Ésta biblioteca permite controlar dichos LEDs (encendiéndolos y apagándolos) previa inicialización.

El control de los LEDs se realiza a través del Puerto 0 del microcontrolador, que es un puerto de propósito general al que están conectados directamente los diodos luminosos en configuración de ánodo común con tres de los pines.

Color del LED	Pin CC1110	Uso en la aplicación
Amarillo	P0_1	Marca los plazos de inactividad del nodo (espera activa o *sleep time*)
Rojo	P0_0	Permanece encendido durante el modo de formación de la red
Verde	P0_6	Permanece encendido durante el modo de funcionamiento estándar

Tabla B.7: Pines del Puerto 0 a los que están conectados los LEDs

Cálculo de rutas mínimas sobre el grafo - *graphPaths.h*

Esta librería se encarga de recorrer el grafo y formar el árbol de expansión que determinará la topología en árbol del protocolo propuesto. Para conseguirlo se hace uso del algoritmo de caminos mínimos de Dijkstra, implementado con cola de prioridad (existe una versión del algoritmo que no requiere dicha cola de prioridad).

Desde el exterior de la biblioteca solo se hace uso de la función que calcula los caminos mínimos (recibiendo la estructura del grado como argumento), y ésta se encarga de formar el árbol de expansión con las variables reservadas en la estructura grafo y de establecer los niveles de los nodos en el árbol resultante.

La primera llamada se debe realizar al formar el árbol en la transición entre la fase de formación de la red y la fase de funcionamiento estándar. Durante la fase de funcionamiento estándar hay dos situaciones que motivan que se forme de nuevo el árbol de expansión: la eliminación de uno o varios de los nodos del grafo o la actualización de uno o varios de los enlaces del grafo.

[3]Pequeño diodo semiconductor que emite luz de espectro reducido cuando se polariza de forma directa y se alimenta.

Bibliotecas de nodos

Biblioteca de los nodos sensores

- **getFramePart():** función encargada de procesar la programación recibida del macro-frame (almacenada en una estructura array de direcciones) y establecer los marcos en los que participará y las direcciones de los nodos con los que interactuará en cada uno de los marcos temporales.

  - **Prototipo**
    *void getFramePart(frame_array_t prog, addr_t ownAddr, procFrames_t *framesPart);*

Parámetro	Descripción
prog	Estructura array que contiene las direcciones físicas que componen el *macro-frame*
ownAddr	Dirección física del nodo que procesa la programación recibida
framesPart	Estructura que contiene los marcos, los tiempos y las direcciones físicas con los que interactuará durante el *macro-frame*

Biblioteca del nodo sumidero

- **extractRows():** recorre las filas de la tabla de intensidades de un nodo para encontrar nuevos nodos no registrados en la lista de nodos.

  - **Prototipo**
    *void extractRows(intensityTable_t iTable, NodeList_t *nodeList, nodo_t *act);*

Parámetro	**Descripción**
iTable	Tabla de intensidades que se debe recorrer en busca de nuevos nodos
nodeList	Lista de nodos en la que se insertará los nodos que no estén registrados
act	Nodo propietario de la tabla de intensidades cuya dirección física se establecerá como descubridora de los nuevos nodos en la lista

- **buscarEnLista():** busca una dirección física en la tabla de intensidades y extrae los niveles de RSSI y batería para ponderar un arco.

- **Prototipo**
  *void buscarEnLista(intensityTable_t iTable[], addr_t sourceAddr, addr_t destAddr, uint8_t iTableIndex, uint8_t *nRSSI, uint8_t *batt);*

Parámetro	Descripción
iTable	Array que contiene las tablas de intensidades recibidas de los nodos
sourceAddr	Nodo fuente para identificar su batería y el arco que lo une con el destino
destAddr	Nodo destino del arco cuyo nivel de intesidad se pretende extraer
nRSSI	Puntero a byte en el que se devolverá el nivel de RSSI
batt	Puntero a byte en el que se devolverá el nivel de batería del nodo fuente

- **selectRoute():** selecciona una ruta del nodo actual al sumidero usando los punteros de descendencia de la lista de nodos.

  - **Prototipo**
    *route_array_t selectRoute(nodo_t *act, nodo_t *sink);*

Parámetro	Descripción
act	Puntero al nodo actual
sink	Puntero al nodo sumidero
retorno	Estructura que contendrá la ruta calculada

- **calcFrameProg():** calcula la programación del *macro-frame* por ramas.

  - **Prototipo**
    *uint8_t calcFrameProg(nodoVertice_t *source, frame_array_t *prog);*

Parámetro	Descripción
source	Puntero del vértice que se procesa actualmente (la llamada se hace sobre la raíz)
prog	Estructura en la que se almacena la programación que se calcule
retorno	Número de marcos temporales programados

Biblioteca común a todos los nodos de la red

- **BSP_SleepFor():** configura el *sleep-timer* y cambia el modo de funcionamiento a un nivel de ahorro de energía indicado.

  - **Prototipo**
    void BSP_SleepFor(uint8_t PM, uint8_t resolution, uint8_t count);

Parámetro	Descripción
PM (Power Mode)	Indica el nivel al que pasará cuando pase a modo de ahorro de energía
resolution	Cambia la resolución del periodo del *sleep-timer*
count	Cambia el contador del *sleep-timer*, estará en el PM indicado durante $count * resolution$ tiempo

  - **PMs disponibles**:
    - SLEEP_MODE_PM0
    - SLEEP_MODE_PM1
    - SLEEP_MODE_PM2
    - SLEEP_MODE_PM3
  - **Resoluciones disponibles del *sleep-timer***:
    - WORCTL_31_25_US_RES → Resolución de 31,25 μs
    - WORCTL_1_MS_RES → Resolución de 1 ms
    - WORCTL_32_MS_RES → Resolución de 32 ms
    - WORCTL_1_S_RES → Resolución de 1 s

- **carrierSense():** comprueba si el bit PKTSTATUS.CS esta activo, lo que significa que se detecta señal portadora en el medio inalámbrico

  - **Prototipo**
    uint8_t carrierSense(void);

Parámetro	Descripción
retorno	Byte utilizado como booleano que indica si el medio está o no ocupado

- **compareAddr():** recorre los bytes de dos direcciones de tipo addr_t e indica si son iguales o distintas.

  - **Prototipo**
    uint8_t compareAddr(addr_t addr1, addr_t addr2);

Parámetro	Descripción
addr1	Primera dirección física de las dos a comparar
addr2	Segunda dirección física de las dos a comparar
retorno	Byte utilizado como booleano que indica si son o no la misma dirección

B.2. Manual de la implementación del puente USB-IGU

B.2.1. Instruducción

En ésta sección se explicarán detalles de implementación de la aplicación puente que actua de intermediaria entre la IGU y el nodo sumidero además de la librería *UART.c* empleada para hacer uso de la UART con el fin de establecer la comunicación entre el nodo sumidero y la aplicación puente.

B.2.2. Flujo de ejecución

En la figura B.7 se muestra el diagrama de flujo de datos que sigue la apliación puente. En él se distinguen por colores diferentes hilos de ejecución que se ejecutan de forma concurrente. En azul se muestra el flujo de ejecución del hilo principal, el cual a su vez lanza el hilo que escucha las peticiones WebSocket y el hilo que escucha de forma permanente las ordenes de la consola de usuario. En el diagrama no se muestra por simplificación y por ceñirse a la estructura de un DFD pero el proceso *Escuchar puerto WebSocket* lanza un hilo distinto para cada conexión entrante que correspondería a los procesos que cuelgan del mismo.

B.2.3. Estructuras de datos

Constantes predefinidas

Miscelánea

En la tabla B.2.3 se detallan las constantes definidas necesarias para distintos mecanismos empleados en la aplicación puente.

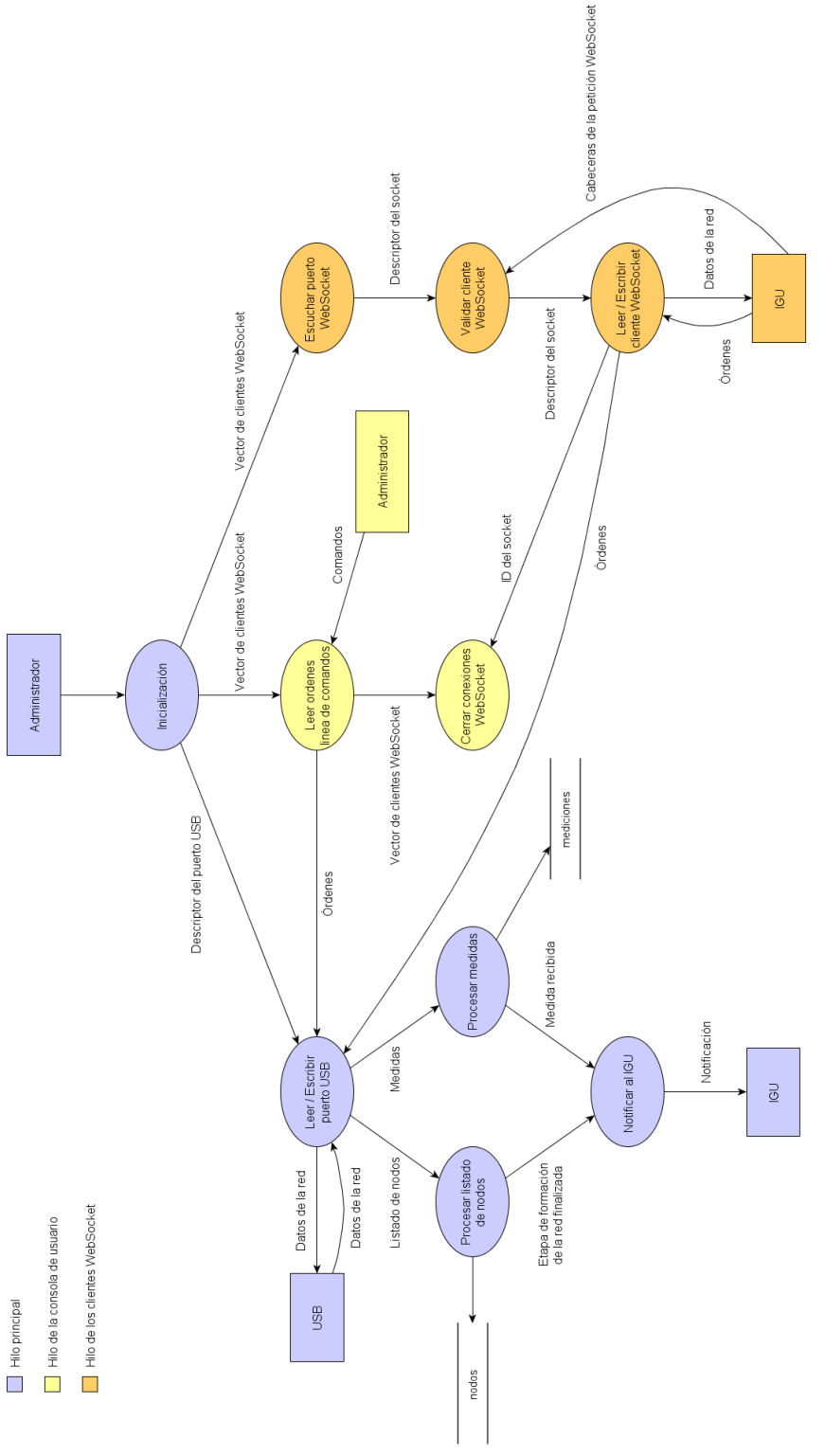


Figura B.7: Diagrama de flujo de datos de la aplicación puente USB-IGU

Constante	Valor	Descripción
NUM_MAX_NODOS	255	Número máximo de nodos que puede contener la red
MAX_PACKET_LENGTH	32	Tamaño máximo que tendrán los paquetes que se intercambien a través de la UART
NUM_MAX_CLAVES_TABLA	5	Número máximo de campos que contienen los paquetes que se intercambian a través de la UART para cada nodo
NUM_MAX_CONEXIONES_IGU	16	Número máximo de conexiones WebSocket simultaneas
TRAMA_FINALIZACION_WEBSOCKET	0xFF	Trama que se envía a un cliente WebSocket para finalizar la conexión

Notificaciones USB-IGU

En la tabla B.2.3 se listan los identificadores de las tramas que envía el nodo sumidero hacia la aplicación puente.

Constante	Valor	Descripción
FIN_LISTADO	0x50	Código enviado a la interfaz gráfica a través de WebSocket para notificar la llegada de un listado de nodos
MEDIDA_RECIBIDA	0x51	Código enviado a la interfaz gráfica a través de WebSocket para notificar la llegada de una medición

Notificaciones IGU-USB

En la tabla B.2.3 se listan los identificadores de las tramas que envía la aplicación puente hacia el nodo sumidero.

Estructuras

```
typedef struct {
    unsigned char addr[4];
    unsigned char flags;
    unsigned short medida;
    unsigned char addrPadre[4];
    char nivelRSSI;
```

Constante	Valor	Descripción
INICIAR_RED	0x49	Código enviado desde la interfaz gráfica a la aplicación puente para ordenar la inicialización de la red
ENVIARCONF	0x43	Código enviado desde la interfaz gráfica a la aplicación puente para ordenar el envío de parámetros de la red
MED_BAJO_DEMANDA	0x4D	Código enviado desde la interfaz gráfica a la aplicación puente para ordenar una medición bajo demanda
REINICIAR_RED	0x52	Código enviado desde la interfaz gráfica a la aplicación puente para ordenar el reinicio de la red

```
    unsigned char battery;
} Nodo;
```

Variable	Descripción
addr[4]	Dirección MAC del nodo
flags	Flags con los que el nodo envió los datos a la red. Determina la validez de los campos medida, RSSI y batería
medida	Medida tomada por el nodo
addrPadre[4]	Dirección MAC del padre del nodo
nivelRSSI	Nivel de RSSI del nodo
battery	Nivel de batería del nodo

```
typedef struct {
    vector<int> fd;
    vector<sockaddr> dir;
    vector<pthread_t> thread;
} Clientes;
```

Variable	Descripción
fd	Vector de los descriptores de socket de las conexiones WebSocket
dir	Vector con las direcciones de los clientes WebSocket (actualmente no se usa)
thread	Vector que contiene los hilos que están escuchando conexiones WebSocket

```
typedef struct {
```

```
    int fd;
    vector<int> *fdList;
    vector<pthread_t> thread;
    int *numClientesActivos;
} paramThListenClient;
```

Variable	Descripción
fd	Descriptor del socket que el hilo que recibe ésta estructura debe escuchar
*fdList	Vector de los descriptores de socket de las conexiones WebSocket
thread	Vector que contiene los hilos que están escuchando conexiones WebSocket
*numClientesActivos	Contador del número de clientes WebSocket activos

B.2.4. Referencia de funciones

- `int activarNodo(MYSQL *conn, string MAC);`

  Activa el nodo en la base de datos. Para ello pone a 1 el campo *estado* de la tupla correspondiente al nodo en la tabla *nodos*. Cuando se inicia la aplicación puente pone a 0 el campo *estado* para todos los nodos de la tabla *nodos*. Cuando se recibe el listado de nodos disponibles en la red, si el nodo ya existe, se activa con ésta función. Si el nodo fuera nuevo se inserta en la tabla con el campo *estado*=2. De ésta forma se distingue qué nodos ya existían la última vez que se inició la red, qué nodos son nuevos, y cuales están inactivos (tendrán el campo *estado*=0).

Variable	Descripción
Variables de entrada	
conn	Conexión con la BBDD
MAC	MAC del nodo que se desea activar
Variables de salida	
return	1 si la operación se lleva a cabo con éxito, 0 si falla

- `unsigned char* calculateMD5(const void *content, ssize_t len);`

  Calcula el hash MD5 de los datos apuntados por *content*. Ésta función es necesaria para uno de los pasos del procesamiento de las cabeceras WebSocket.

- `int cerrarConexionWebSocket(int fd, vector<int> *fdVector, vector<pthread_t> *thVector, int *numClientesActivos,`

Variable	Descripción
Variables de entrada	
*content	Datos de los cuales se desea calcular el hash MD5
len	Longitud de los datos
Variables de salida	
return	Hash MD5 calculado en base a *content*

`unsigned short decrementar)`

Cierra una conexión WebSocket. Cada conexión WebSocket se corresponde con un hilo de escucha para ésta conexión y un descriptor de socket. A ésta función se le proporcionan ambos vectores para que sea la misma función la encargada de eliminar los datos correspondientes a ésta conexión y decrementar el número de clientes activos en cuanto se haga efectiva la desconexión. Ésta función puede ser llamada para ordenar la terminación de una conexión WebSocket por un error durante la negociación de la conexión. En ése caso, por coherencia en la semántica del código todavía no se habrá incrementado el número de clientes activos (realmente todavía no sería una conexión activa aun) por lo que es posible que ésta función no deba decrementar el número de clientes activos ya que el contador no se habría incrementado por ése cliente. Es por ello que se le proporciona una variable decrementar que índica si se debe decrementar o no el contador de clientes activos al cerrar dicha conexión.

Variable	Descripción
Variables de entrada	
fd	Descriptor del socket que se debe cerrar
*fdVector	Vector de descriptores de conexiones abiertas
*thVector	Vector de hilos de los sockets abiertos
*numClientesActivos	Puntero al contador de clientes activos
decrementar	1 si se debe decrementar el número de clientes activos, 0 en caso contrario
Variables de salida	
return	Siempre devuelve 1

- `MYSQL* conectarConBD(char *servidor, char *usuario, char *password, char *bd);`

  Conecta con la base de datos.

- `void conexionIGU(void *paramTh);`

  Hilo de ejecución encargado de escuchar en el puerto de WebSocket y gestionar las conexiones. Éste hilo lanza un hilo *listenClient* por cada conexión que llegue al puerto de WebSocket.

Variable	Descripción
Variables de entrada	
*servidor	Servidor de la base de datos
*usuario	Usuario con el que autenticarse contra la base de datos
*password	Password con el que autenticarse contra la base de datos
Variables de salida	
return	1 si la operación se lleva a cabo con éxito, 0 si falla

Variable	Descripción
Variables de entrada	
*paramTh	Estructura del tipo *Clientes*. Ésta estructura contiene diversas variables necesarias para gestionar las conexiones WebSocket

- `void *consolaUsuario(void *paramTh);`

  Hilo de ejecución que se encarga de gestionar todo lo referente a la consola de comandos del administrador. Se le proporciona como parámetro una estructura del tipo *Clientes* ya que, en el caso de que el administrador ordene la finalización de la aplicación, ésta debe cerrar de forma ordenada las conexiones WebSocket, para lo cual necesita llamar a la función *cerrarConexionWebSocket* para conexión activa y ésta función requiere que se le pase por parámetro un puntero a la estructura *Clientes*.

Variable	Descripción
Variables de entrada	
*paramTh	Estructura del tipo *Clientes*

- `unsigned char* createHeaderForSend(char *cabeceras, size_t *length);`

  Dada una cadena de caracteres que contiene todas las cabeceras de una petición de conexión de un cliente WebSocket proporciona las cabeceras que se debe devolver como respuesta.

Variable	Descripción
Variables de entrada	
*cabeceras	Cabeceras recibidas del cliente
*length	Longitud de las cabeceras
Variables de salida	
return	Cabeceras a devolver como respuesta

- `int existeNodoEnBD(MYSQL *conn, string MAC);`

  Comprueba si un nodo ya existe en la base de datos.

Variable	Descripción
Variables de entrada	
*conn	Conexión con la base de datos
MAC	MAC del nodo del cual se desea comprobar si existe en la base de datos
Variables de salida	
return	1 si el nodo existe en la base de datos, 0 si no existe

- `string fromCharToString(char c);`

  Convierte una cadena de caracteres en un *string*.

Variable	Descripción
Variables de entrada	
c	Cadena de caracteres a convertir
Variables de salida	
return	Cadena de caracteres convertida a *string*

- `void fromCharToUnsignedChar(char c[], unsigned char uc[]);`

Variable	Descripción
Variables de entrada	
Variables de salida	
return	1 la operación se lleva a cabo con éxito, 0 si falla

- `string fromTime_tToString(time_t time);`

  Convierte una estructura *time_t* en un *string*.

Variable	Descripción
Variables de entrada	
time	Estructura a convertir
Variables de salida	
return	*time* convertido a *string*

- `string fromUCharToString(unsigned char c);`

  Convierte un caracter en un *string*.

Variable	Descripción
Variables de entrada	
c	Caracter a convertir
Variables de salida	
return	Caracter convertido a *string*

Variable	Descripción
Variables de entrada	
c	Caracter a convertir
Variables de salida	
return	Caracter convertido a *string*

- `string fromUShortToString(unsigned short c);`

  Convierte un caracter sin signo a *string*.

- `void getDatosMedidasNodos(unsigned char datos[], int tamPaquete, Nodo nodos[], unsigned char *numNodos);`

  Procesa un paquete de mediciones enviado por el sumidero a través de la UART. Actualmente la comunicación está implementada de forma que cada paquete recibido solo contiene un nodo. Sin embargo, ésta función puede procesar un número arbitrario de nodos en un mismo paquete.

Variable	Descripción
Variables de entrada	
datos	Datos recibidos por la UART
tamPaquete	Tamaño de los datos recibidos por la UART
Variables de salida	
nodos	Vector de estructuras de tipo *Nodo* extraída de *datos*
numNodos	Número de nodos que contiene el vector *nodos* (longitud del vector)

- `void getDatosNodos(unsigned char datos[], int tamPaquete, Nodo nodos[], unsigned char *numNodos);`

  Procesa un paquete de listado de nodos enviado por el sumidero a través de la UART. Actualmente la comunicación está implementada de forma que cada paquete recibido solo contiene un nodo. Sin embargo, ésta función puede procesar un número arbitrario de nodos en un mismo paquete.

- `string getUltimaMedidaValida(MYSQL *conn, string MAC);`

  Los nodos no siempre envían todos sus datos. Sin embargo, la interfaz gráfica siempre debe mostrar algún valor. El valor que es lógico mostrar

Variable	Descripción
Variables de entrada	
datos	Datos recibidos por la UART
tamPaquete	Tamaño de los datos recibidos por la UART
Variables de salida	
nodos	Vector de estructuras de tipo *Nodo* extraida de *datos*
numNodos	Número de nodos que contiene el vector *nodos* (longitud del vector)

en la IGU es el último valor válido que envío cada nodo. Por ello, en caso de que un nodo no envíe su valor de medida se realiza una búsqueda en la base de datos para encontrar el último valor de medida válido.

Variable	Descripción
Variables de entrada	
conn	Conexión con la base de datos
MAC	MAC del nodo del cual se desea saber la última medida válida que envió
Variables de salida	
return	Última medida válida que envió el nodo

- `string getUltimoBattValido(MYSQL *conn, string MAC);`

  Los nodos no siempre envían todos sus datos. Sin embargo, la interfaz gráfica siempre debe mostrar algún valor. El valor que es lógico mostrar en la IGU es el último valor válido que envío cada nodo. Por ello, en caso de que un nodo no envíe su valor de nivel de batería se realiza una búsqueda en la base de datos para encontrar el último valor de nivel de batería válido.

Variable	Descripción
Variables de entrada	
conn	Conexión con la base de datos
MAC	MAC del nodo del cual se desea saber el último nivel de batería válido que envió
Variables de salida	
return	Último valor de nivel de batería válido que envió el nodo

- `string getUltimoRSSIValido(MYSQL *conn, string MAC);`

  Los nodos no siempre envían todos sus datos. Sin embargo, la interfaz gráfica siempre debe mostrar algún valor. El valor que es lógico mostrar en la IGU es el último valor válido que envío cada nodo. Por ello, en caso de que un nodo no envíe su valor de nivel de RSSI se realiza una

búsqueda en la base de datos para encontrar el último valor de nivel de RSSI válido.

Variable	**Descripción**
Variables de entrada	
conn	Conexión con la base de datos
MAC	MAC del nodo del cual se desea saber el último nivel de RSSI válido que envió
Variables de salida	
return	Último valor de nivel de RSSI válido que envió el nodo

- `int initListadoNodos(MYSQL *conn);`

  Pone a 0 el campo *estado* de todos los nodos. Esta función es llamada al iniciarse la aplicación puente. Cuando se recibe el listado de nodos disponibles en la red, si el nodo ya existe en la base de datos, se activa con el nodo con la función *activarNodo*. Si el nodo fuera nuevo se inserta en la tabla con el campo *estado*=2. De ésta forma se distingue qué nodos ya existían la última vez que se inició la red, qué nodos son nuevos, y cuales están inactivos (tendrán el campo *estado*=0).

Variable	**Descripción**
Variables de entrada	
conn	Conexión con la base de datos
Variables de salida	
return	1 si la operación se lleva a cabo con éxito, distinto de 0 si falla

- `int insertarNodo(MYSQL *conn, string values[]);`

  Inserta un nodo en la base de datos.

Variable	**Descripción**
Variables de entrada	
conn	Conexión con la base de datos
values	Vector con los valores del nodo a insertar: 0 → MAC del nodo 1 → Nivel de bateria del nodo 2 → MAC del padre del nodo 3 → Nivel de RSSI del nodo 4 → Fecha actual
Variables de salida	
return	1 la operación se lleva a cabo con éxito, 0 si falla.

- `void *listenClient(void *paramTh);`

  Hilo de ejecución encargado de escuchar una conexión WebSocket. Cada conexión WebSocket tiene asignado un hilo de éste tipo que escucha constantemente los datos enviados por el cliente. Según qué datos envíe el cliente el hilo puede tomar acciones como ordenar la inicialización de la red, el reinicio de la red, forzar mediciones bajo demanda del cliente, enviar configuraciones de la red o terminar la conexión.

  Dado que éste hilo es el encargado de verificar la validez de la conexión WebSocket a nivel del protocolo de comunicación IGU-USB se le suministra una estructura de tipo *paramThListenClient* que contiene el vector de hilos de tipo *listenClient*, el vector de descriptores de sockets WebSocket y el número de clientes activos ya que ningún hilo ni su respectivo descriptor de fichero debe ser agregado a éstos vectores hasta que se verifique su validez, de igual forma no se debe incrementar el número de clientes activos hasta que dicha condición se cumpla. Una vez se verifica su validez el hilo procederá a insertar el descriptor de socket y el hilo en los respectivos vectores así como incrementará el contador de clientes activos.

Variable	**Descripción**
Variables de entrada	
*paramTh	Estructura del tipo paramThListenClient

- `void notifyIGU(Clientes clientes, char codigo);`

  Cada vez que la aplicación puente recibe un paquete de listado de nodos o de mediciones ésta notifica a la interfaz gráfica para que actualice los datos presentados en pantalla. Ésta función envía un código a todos los clientes (interfaces gráficas). El código identifica el evento sucedido (recepción de listado de nodos o recepción de mediciones).

Variable	**Descripción**
Variables de entrada	
clientes	Estructura de tipo Clientes que contiene los clientes activos
codigo	Código a enviar a los clientes. Los caracteres posibles son: 0x50 → FIN_LISTADO 0x51 → MEDIDA_RECIBIDA

- `unsigned char* solveChallenge(const char *number1, const char *number2, const char *key3);`

  Dadas las tres claves proporcionadas en las cabeceras de la petición de conexión WebSocket calcula la clave a enviar en la respuesta a la petición.

Variable	Descripción
Variables de entrada	
*number1	Valor del campo Sec-WebSocket-Key1 de la petición de conexión WebSocket
*number2	Valor del campo Sec-WebSocket-Key2 de la petición de conexión WebSocket
*key3	Últimos 8 bytes de la petición de conexión WebSocket correspondientes a los 8 bytes aleatorios generados por el cliente
Variables de salida	
return	Clave a enviar en la respuesta a la petición de conexión

- `int updateMedidas(MYSQL *conn, string values[], unsigned char flags);`

  Inserta una nueva tupla en la tabla *mediciones* correspondiente a una medición de un nodo.

Variable	Descripción
Variables de entrada	
*conn	Conexión con la base de datos
values	Vector con los datos de la medición: 0 → MAC del nodo 1 → Medición del nodo 2 → Nivel de RSSI del nodo 3 → Nivel de bateria del nodo 4 → Fecha actual
Variables de salida	
return	1 la operación se lleva a cabo con éxito, 0 si falla.

B.2.5. Libreria para el manejo de la UART

- `void initUART();`

  Ésta función inicializa diversos parámetros de configuración de la UART.

- `void UARTSend(uint8_t *datos, uint16_t tam);`

  Ésta función se usa para enviar datos a través de la UART.

- `uint8_t UARTReceive(uint8_t mode);`

  Ésta función se usa para recibir datos de la UART. Soporta dos modos de invocación: bloqueante y no bloqueante. En el modo de recepción bloqueante sólo se puede recibir un byte por llamada y éste se devuelve

Variable	Descripción
Variables de entrada	
*datos	Datos a enviar
tam	Tamaño de los datos a enviar

en la variable de retorno. En el modo no bloqueante la recepción se realiza atendiendo a las interrupciones generadas por la UART al recibir un dato. En éste modo es posible recibir una cantidad arbitraria de datos y éstos pueden ser leidos a través de la función *UART_ReadBuffer* cuando los datos hayan sido recibidos por completo. Se puede asumir que los datos son recibidos por completo cuando la variable global *RXflag* es distinta de 0.

Variable	Descripción
Variables de entrada	
mode	Determina el modo de la recepción ENABLE_RX_BLOQ → Recepción no bloqueante ENABLE_RX_INT → Activa la recepción de datos en modo no bloqueante DISABLE_RX → Desactiva la recepción de datos en el modo no bloqueante
Variables de salida	
return	Byte leido en el modo no bloqueante

Constante	Valor
ENABLE_RX_BLOQ	0
ENABLE_RX_INT	1
DISABLE_RX	2

- `UARTbuffer_t UART_ReadBuffer();`

  Cuando la recepción de datos no bloqueante ha terminado de leer los datos (*RXflag != 0*) se debe llamar a ésta función para obtener dichos datos. La función devuelve los datos leidos dentro de una estructura de tipo *UARTbuffer_t* que contiene además la longitud de dichos datos.

```
typedef struct {
     uint8_t *buff, len;
} UARTbuffer_t;
```

Variable	Descripción
Variables de salida	
return	Datos recibidos

- `__interrupt void UART_ReceiveInterrupt(void);`

  Ésta es la rutina de la interrupción que genera la UART al recibir un byte de datos. Ésta rutina se activa cuando se solicita una recepción de datos en modo no bloqueante. La rutina guarda cada byte de datos recibido en un buffer que posteriormente será leido por la función *UART_ReadBuffer*. De forma adicional, cuando se termina de leer una flujo de bytes, ésta rutina identifica el tipo de paquete recibido y consecuentemente cambia la variable global *RXflag* a un valor distinto de 0 que identifica el tipo de paquete recibido.

```
extern uint8_t RXflag;
```

RXflag	Descripción
0	Datos no recibidos
1	Datos recibidos. Paquete de configuración
2	Datos recibidos. Paquete de petición de medición bajo demanda
3	Datos recibidos. Paquete de solicitud de reinicio de la red

B.3. Manual de la implementación de la interfaz gráfica / Script PHP

Aunque la interfaz gráfica y el script PHP conceptualmente se pueden considerar como dos entidades distintas a nivel de implementación, sin embargo ambos conviven en el archivo *index.php* ya que PHP permite fusionar código HTML/JavaScript en un mismo archivo y a nivel de implementación facilita la tarea del programador, sobretodo a la hora de implementar el envío de la imagen de la estructura de la IGU. Dado que la interfaz gráfica y el IGU están estrechamente unidos y el script PHP de por sí no tiene suficiente entidad como para dedicarle un apartado exclusivo para detallar su implementación se incluye dicha información en éste apartado junto con la información concerniente a la implementación de la IGU.

La interfaz gráfica hace uso de dos vías de comunicación con el fin de poder mostrar la información relativa a la red en tiempo real. Por un lado mantiene una conexión WebSocket con la aplicación puente para saber en qué

momento consultar las medidas y para enviar ordenes de control y parámetros de configuración a la red, y por otro se comunica con el script PHP mediante peticiones GET a través de AJAX para interactuar con la base de datos cuando lo necesite. Todas las interacciones que la IGU realiza con el script PHP hacen uso de AJAX para evitar la recarga total de la página cada vez que se necesiten datos de la BBDD. Sin embargo AJAX no soporta el envío de datos de tipo *multipart/form-data*, que es el tipo de datos que se emplea para enviar mediante POST datos como imágenes al script PHP. Por ésta razón el IGU no es capaz de usar AJAX a la hora de cambiar la imagen de la estructura y por lo tanto el IGU se ve en la necesidad de recargar la página por completo cuando el usuario decide cambiar dicha imagen.

B.3.1. Interacción con el script PHP

Ante ciertos eventos es posible que el IGU requiera interactuar con los datos que contiene la BBDD, ya sea para extraer o bien para guardar datos en la misma. Para interactuar con la base de datos el IGU hace uso de la tecnología AJAX para ejecutar peticiones GET (HTTP) que invoquen al script PHP para que posteriormente éste ejecute las sentencias SQL oportunas, ya sea para extraer o para guardar los datos que el IGU le proporcione a través de la propia petición GET.

Según que acción concreta solicite realizar el IGU contra la BBDD el script PHP ejecuta unas sentencias SQL u otras. La acción concreta a realizar la especifica el IGU en el campo *a* de la petición GET. Según que código albergue éste campo, el script PHP realiza una acción u otra. Éstas diferentes acciones predefinidas se describen en la tabla B.3.1.

B.3.2. Estructuras de datos

En la interfaz gráfica se hace uso de Ajax para intercambiar información con el script PHP de forma dinámica sin vernos en la necesidad de hacer una recarga total de la página cada vez que la interfaz gráfica tiene la necesidad de interactuar con dicho script. La comunicación con el script PHP se lleva a cabo intercambiando listas de datos fácilmente estructurables, es decir. Por ésta razón tanto el script PHP como la IGU estructuran dichos datos en bloques de código XML. Dichas estructuras XML son las siguientes:

Solicitud del listado de nodos

Cuando el IGU solicita un listado de nodos al script PHP éste le devuelve una estructura XML como la siguiente. Toda la estructura está envuelta con una etiqueta *listaNodos* y la etiqueta *nodo* y sus etiquetas anidadas se repiten

Acción	Descripción
ln	El script PHP devuelve en una estructura XML el listado de nodos activos de la BBDD
pn	Ésta acción se ejecuta cuando el usuario posiciona un nodo en el mapa de nodos. El IGU proporciona en la petición GET los campos *nodoX* y *nodoY* que contienen la posición del nodo dentro del mapa de nodos con respecto a la imagen (no la posición con respecto a la página). El script PHP guardará dichos datos en los campos *posX* y *posY* de la tupla de la tabla *nodos* correspondiente al nodo cuyo identificador (campo *id* de la tabla) sea el mismo que el identificador proporcionado en el campo *idNodo* por el IGU en la petición GET.
n	El script PHP devuelve en una estructura XML los datos del nodo cuyo identificador en la tabla *nodos* sea el mismo que el proporcionado en el campo *id* por el IGU en la petición GET.
c	El script PHP devuelve en una estructura XML los datos de configuración guardados en la tabla *config* de la BBDD.
gc	El script PHP guarda los datos de configuración enviados por el IGU en la petición GET. El IGU debe enviar tres campos: uM → Contiene el valor del umbral de medida uR → Contiene el valor del umbral del nivel RSSI uB → Contiene el valor del umbral del nivel de batería

para cada nodo. Las etiquetas anidadas dentro de *nodo* envuelven a su vez cada uno de los datos del nodo correspondiente.

Las etiquetas *posX* y *posY* representan la posicion del nodo en pixeles dentro de la imagen del mapa de nodos. Si los campos homologos de éstas etiquetas en la BBDD son NULL, las etiquetas contendrán *0*.

```
<listaNodos>
     <nodo>
          <idNodo>idNodo</idNodo>
          <mac>mac</mac>
          <flags>flags</flags>
          <idMed>idMed</idMed>
          <medida>medida</medida>
          <RSSI>RSSI</RSSI>
          <nivelBateria>nivelBateria</nivelBateria>
          <fecha>fecha</fecha>
          <posX>posX</posX>
```

```
            <posY>posY</posY>
      </nodo>
</listaNodos>
```

Solicitud de datos de un nodo

El IGU puede solicitar datos de un nodo en particular. Éste caso se da cuando el usuario posiciona el ratón sobre la imagen de un nodo dentro del mapa de nodos ya que el IGU debe mostrar un popup con la información concreta de ése nodo en particular. Cuando ésto sucede el nodo solicita dichos datos al script PHP y éste devuelve una estructura como la que sigue.

```
<listaNodos>
      <nodo>
            <id>id</id>
            <mac>mac</mac>
            <flags>flags</flags>
            <medida>medida</medida>
            <macPadre>macPadre</macPadre>
            <RSSI>RSSI</RSSI>
            <nivelBateria>nivelBateria</nivelBateria>
            <fecha>fecha</fecha>
      </nodo>
</listaNodos>
```

Solicitud de datos de configuración

El IGU puede solicitar los datos de configuración guardados en la BBDD. Cuando ésto sucede el script PHP devuelve una estructura envuelta en la etiqueta *config*. Cada una de las etiquetas anidadas dentro de *config* tendrá como nombre de la etiqueta el nombre del parámetro de configuración, y el valor de la etiqueta será el valor correspondiente al parámetro de configuración.

```
<config>
      <clave>valor</clave>
</config>
```

B.3.3. Referencia de funciones

- `function init();`

Ésta función se ejecuta al cargar la página en el navegador. Inicializa variables globales, y abre la conexión WebSocket. Además, implementa el protocolo de comunicación con la aplicación puente.

Una interfaz gráfica puede conectarse a la aplicación puente en cualquier momento. Ésto implica que en el momento en el que una IGU se conecta a la aplicación puente la red puede encontrarse en cualquier estado ya que otra interfaz gráfica (o la misma que se desconectó anteriormente y ha vuelto a conectar). Dada ésta tesitura se hace necesario que la aplicación puente suministre información sobre el estado actual de la red a cada IGU que se conecte a ésta para que la IGU actue en consecuencia habilitando o deshabilitando distintos elementos HTML para permitir o impedir ciertas acciones desde la IGU en función del estado de la red. La información sobre el estado de la red se suministra en forma de códigos de estado a través de la conexión WebSocket inmediatamente después de establecer la conexión.

Las acciones que realiza en función del código que recibe a través de la conexión WebSocket son las siguientes:

Código	**Descripción y acciones**
'P'	Informa de que la red está inicializada. Habilita y deshabilita varios botones de la interfaz gráfica
'Q'	Informa de que se han recibido medidas de los nodos. Llama a la función *drawNodes* para que actualice la tabla de medidas de la IGU
Cualquier valor distinto de los anteriores	Código de estado de la red: 0 → Red sin inicializar 1 → Red inicializandose 2 → Red inicializada, recibiendo medidas

- `function getMouseXY();`

  Devuelve la posición del cursor respecto de la imagen del mapa de nodos

- `function sendCoordinates();`

  Ésta función se invoca al hacer *click* sobre el mapa de nodos para posicionar un nodo. Invoca al script PHP (acción *pn*, ver tabla B.3.1) para que guarde en la BBDD la posición del nodo a posicionar.

- `function drawNodes();`

  Solicita al script PHP el listado de nodos (acción *ln*, ver tabla B.3.1) para posteriormente mostrar los datos en la tabla de medidas de la

IGU. Para ello primero borra las filas desde la 1 hasta la última (la fila 0 corresponde con la cabecera de la tabla), y luego las vuelve a insertar con los datos actualizados. Es importante no modificar la organización de la tabla y de las columnas ya que las celdas con sus datos se insertan siempre en el mismo orden y el resto de funciones que hacen uso de los datos de las celdas también hacen uso de éstos siguiendo el mismo orden, si se altera el orden de las columnas o se elimina alguno podría causar un mal funcionamiento de la interfaz gráfica e incluso propagar el fallo a la BBDD.

Además, si procede, posiciona los nodos en el mapa de nodos en función de los datos de la posición de cada nodo almacenados en la BBDD. El mecanismo de posicionamiento de los nodos se implementa creando para cada nodo una capa (etiqueta *layer* HTML) con el atributo CSS *position: absolute* e insertando dentro de cada capa la imagen del nodo. De ésta forma la capa se puede posicionar encima de la imagen mediante los atributos CSS *left* y *top* que definen la posición en el eje X e Y respectivamente. Adicionalmente, en la etiqueta de la imagen del nodo se le asigna la función *selecNodo* al evento *onMouseOver*, la función *deSelecNodo* al evento *onMouseOut* y el identificador del nodo al atributo HTML *id* de la imagen para relacionar ésa imagen con el nodo que represente (es necesario para luego extraer la información del nodo para mostrarla en el popup del nodo). De ésta forma, cuando el usuario posiciona el ratón sobre la imagen del nodo se mostrará un popup con la información del nodo y cuando el ratón si situe fuera de la imagen del nodo, el popup desaparecerá.

- `function botonPos(obj);`

  Ésta función es invocada al presionar el botón *Posicionar* para un nodo. Prepara el IGU para que el usuario pueda presionar sobre la imagen del mapa de nodos para posicionar un nodo sobre la imagen. Para ello guarda en la variable *idNodoSelec* el identificador del nodo que se va a posicionar. Éste identificador lo extrae de la primera columna de la fila de la tabla de mediciones a la que pertenece el botón que ha sido presionado. Además modifica el estilo de la imagen del mapa de nodos para que cuando el ratón se sitúe sobre dicha imagen el cursor cambie al tipo *crosshair*.

Variable	**Descripción**
obj	Botón *Posicionar*

- `function selecNodo(obj);`

  Ésta función es invocada cuando el cursor del ratón se posiciona sobre la imagen de un nodo. Solicita al script PHP la información de dicho

nodo y la muestra como un popup. Para ello envía mediante AJAX una petición GET al script PHP (acción *n*, ver tabla B.3.1) con el identificador del nodo extraído del campo HTML *id* de la etiqueta de la imagen. Cuando se recibe la información del nodo se crea una capa (con el atributo CSS *position: absolute* para poder posicionarla sobre la imagen del mapa de nodos) que conformará el popup con dicha información y se inserta dentro de la capa *contenedorPopupsNodos* que actúa como contenedor de capas de popups.

Variable	Descripción
obj	Imagen del nodo

- `function deSelecNodo();`

  Vacía la capa *contenedorPopupsNodos* que contiene las capas que conforman los popups de información de los nodos cuando se sitúa el ratón sobre la imagen del algún nodo;

- `function iniciarRed();`

  Envía la orden de inicio de red y la configuración a la aplicación puente mediante la conexión WebSocket.

- `function enviarConfiguracion(overridePreviousValues);`

  Ésta función es invocada cada vez que se ordena el inicio de la red o se presiona el botón *Actualizar*. Envía a través de la conexión WebSocket la configuración hacia la red y al mismo tiempo llama a la función *setConfig* para guardar dicha configuración en la base de datos.

  Dado que la red de nodos necesita una configuración de la que partir de inicializarse, ésta configuración debe enviarse siempre desde la interfaz gráfica cuando el usuario ordene la inicialización de la red. Sin embargo, cuando el usuario ordena el envío de configuración la red ya dispone de una configuración activa por lo que la interfaz gráfica sólo debería enviar la configuración de los parámetros que han cambiado respecto de los parámetros anteriores. Para lograr éste comportamiento se usa el parámetro *overridePreviousValues*. Si éste parámetro es *True* enviará los parámetros de configuración a pesar de que no hayan cambiado respecto a los anteriores, si por el contrario *overridePreviousValues* es *False* sólo se enviarán los parámetros que hayan cambiado. De ésta forma se evita enviar a la red de nodos datos innecesarios.

- `function getConfig();`

  Ésta función es llamada nada mas iniciar la interfaz gráfica para mostrar los parámetros actuales de configuración en los objetos *input* (HTML) corrrespondientes. Realiza una petición GET (HTML) mediante AJAX

Variable	Descripción
overridePreviousValues	Si es *True* envía todos los parámetros de configuración incondicionalmente, de lo contrario sólo se enviarán los parámetros que sean distintos de los que habían la última vez que se enviaron

al script PHP (acción *c*, ver tabla B.3.1) para obtener una estructura XML con los parámetros de configuración guardados en la tabla *config* de la BBDD.

- `function setConfig();`

  Ésta función es llamada cuando el usuario presiona el botón *Aplicar configuración* y alguno de los parámetros de configuración ha cambiado respecto de la última vez que fue guardado en la BBDD o bien cuando el usuario presiona el botón *Iniciar red*. Realiza una petición GET (HTML) mediante AJAX al script PHP (acción *gc*, ver tabla B.3.1) para que guarde en la BBDD los parámetros de configuración enviados como campos de la petición GET.

- `function medicionBajoDemanda();`

  Ésta función es llamada cuando el usuario presiona el botón *Forzar medición*. Envía una orden a la aplicación puente mediante la conexión WebSocket para que ésta ordene a la red que realice una medición bajo demanda.

- `function reiniciarRed();`

  Ésta función es llamada cuando el usuario presiona el botón *Reiniciar red*. Envía una orden a la aplicación puente mediante la conexión WebSocket para que ésta ordene a la red se reinicialice.

- `function cambiarImgDialog();`

  Ésta función es llamada cuando el usuario solicita cambiar la imagen de la estructura. Muestra el popup que permite al usuario elegir la imagen que desea insertar en lugar de la imagen actual. Para ello posiciona la capa que oscurece la imagen actual sobre la imagen actual y la capa que contiene los campos *input* (HTML) que permiten seleccionar otra imagen sobre en el centro de la imagen actual. Posteriormente examina los atributos CSS *visibility* de ambas capas (puede tomar el valor *visible* o *hidden*) y lo cambia al valor contrario al que tiene actualmente.

- `function cambiarTamNodo(incrementar);`

  Ésta función es llamada cuando se presiona sobre las imágenes que permiten cambiar el tamaño de las imágenes de los nodos. Aumenta

o disminuye en un pixel el tamaño de las imágenes de los nodos en función del parámetro de entrada.

Variable	Descripción
incrementar	Si es True aumenta el tamaño de las imágenes, de lo contrario disminuye dicho tamaño

B.4. Manual de implementación de la libreria para la lectura de los sensores

B.4.1. Descripción

La finalidad de ésta libreria es controlar la electrónica de la tarjeta de adaptación de los sensores con el fin de proporcionar el valor de salida en binario natural que entrega el CAD que a su vez se corresponde con la medida de deformación realizada por las galgas extensiométricas. Para ello ésta libreria genera las señales de reloj y CS (Chip Select) que necesita el CAD para funcionar de forma correcta, recoge la señal digital que entrega el CAD y la almacena en la variable global *galgasValue* para que pueda ser leida en cuanto esté lista para ello.

B.4.2. Referencia de funciones

- `uint16_t leerGalgas();`

  Ésta función bloqueante proporciona el valor de salida del CAD en binario natural de 16 bits.

Variable	Descripción
Variables de salida	
return	Valor de salida del CAD

- `void initTimer1();`

  Inicializa los valores del *timer1* del microcontrolador que será usado para generar las señales necesarias para hacer funcionar el CAD.

- `__interrupt void timer1_interrupt(void);`

  Ésta rutina de interrupción es llamada en cada *tick* que genera el *timer1* del microcontrolador. Se encarga de general la señal de reloj para el

CAD, la señal de CS (Chip Select) y recoger los datos proporcionados por el CAD que se corresponden a la medida de las galgas extensiométricas. El valor de salida del CAD es almacenado secuencialmente (bit a bit) en la variable *galgasValue*. Cuando la secuencia de señales necesarias para la lectura del valor termina se detiene el *timer1*. La detención del *timer1* se aprovecha para notificar a la función *leerGalgas* de que el valor de las galgas ya puede ser leido.

Bibliografía

[1] Wireless Sensor Networks, Architectures and Protocols - Callaway

[2] Wireless Sensor Network Designs - Anna Hac

[3] SimpliciTI: Simple Modular RF Network Specification - Larry Friedman

[4] SmartRF^{TM} Packet Sniffer User Manual

[5] JavaScript: Cuso de Programación - Edgar D'Andrea

[6] Ajax - Francisco Charte Ojeda

[7] JavaScript - Steve Suehring

[8] JavaScript: La Guía definitiva - David Flanagan

[9] JavaScript y DHTML - Danny Goodman

[10] C/C++ Curso de Programación. 3ª Edición Ed. Ra-Ma - Francisco Javier Caballos Sierra

[11] Wikipedia: http://en.wikipedia.org

[12] CPlusPlus: http://www.cplusplus.com

[13] MySQL Doc: http://dev.mysql.com/doc

[14] C Programming FAQs: Frequently Asked Questions: http://c-faq.com/

[15] National Institute of Standards and Technology - Dictionary of Algorithms and Data Structures: http://xw2k.nist.gov/dads/