



CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS  
DEL INSTITUTO POLITÉCNICO NACIONAL

**Unidad Zacatenco**

**Departamento de Computación**

**Mecanismo seguro de recolección de datos utilizando  
nodos móviles en redes inalámbricas de sensores**

Tesis que presenta

**Iván Cabrera Altamirano**

para obtener el grado de

**Maestro en Ciencias en Computación**

Director de la tesis

**Dr. Francisco Rodríguez Henríquez**

México, D.F.

Diciembre 2010



# Agradecimientos

Gracias a mis padres, Constantino Cabrera y Gloria Altamirano, a mis hermanos, Lina, Omar, Ale y Susy, a las gemelas, Paulina y Carolina y a todos mis familiares cercanos por todo el apoyo brindado a través de los años.

A mi esposa Blanca, por la paciencia, sacrificio y apoyo que ha mostrado en los momentos más difíciles y por la excelente compañía y buenos momentos que hemos vivido juntos.

A mi director de tesis, Dr. Francisco Rodríguez Henríquez, le agradezco por su ayuda y orientación incondicional durante mi estancia en el CINVESTAV.

A mis revisores de tesis, Dr. Luis Gerardo de la Fraga y Dr. José Guadalupe Rodríguez García por sus invaluable comentarios que ayudaron a enriquecer este trabajo de tesis.

A todos los doctores con los que tuve oportunidad de cursar alguna materia (Dr. Adriano de Luca, Dr. Arturo Díaz, Dr. Jorge Buenabad, Dr. Pedro Mejía, Dr. Dominique Decouchant, Dra. Sonia Mendoza, Dr. José Guadalupe Rodríguez, Dr. Francisco Rodríguez, Dr. Luis Gerardo de la Fraga, Dr. Gabriel Ramírez), ya que han aportado mucho a mi formación profesional.

A todo el personal del Departamento de Computación, en especial a Sofía Reza, a Felipa Rosas y Erika Ríos, por su ayuda y gran compromiso al momento de realizar su trabajo.

A mis amigos en el CINVESTAV, que lograron que la estancia en el instituto fuese muy agradable.

Al Centro de Investigación y de Estudios Avanzados del IPN (CINVESTAV-IPN) por abrirme las puertas de una gran institución para la realización de estudios de posgrado.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por la beca otorgada para completar mis estudios de maestría y al proyecto SEP-CONACYT 60240 por otorgarme el apoyo económico para la realización de viajes a congresos internacionales.



## Resumen

Las redes inalámbricas de sensores son una tecnología emergente que puede ser ampliamente utilizada para la creación de aplicaciones novedosas en diversas áreas del conocimiento. Sin embargo aún cuando esta tecnología parece ser una excelente solución para un amplio rango de escenarios, éstas introducen nuevos retos de diseño que los encargados de implementar los nuevos sistemas deben afrontar. Vale recordar que estas redes basan su operación en diminutos dispositivos llamados nodos, los cuales están restringidos en poder de cómputo, memoria y energía.

En este trabajo de tesis se presenta un mecanismo seguro de recolección de datos para redes inalámbricas de sensores, donde a través del análisis realizado hemos identificado características altamente deseables para un mecanismo de recolección de datos. Entre los requisitos más importantes podemos encontrar: escalabilidad, seguridad, diversidad de sensores, entre otros. Otra característica muy importante de este trabajo es que los nodos pueden ser móviles, para cumplir con este requerimiento hemos incorporado funciones de posicionamiento global.

Después de haber realizado el análisis de los requerimientos procedimos a proponer una arquitectura de recolección, tipos de datos y nodos, un procedimiento de reenvío de paquetes, y los componentes del mecanismo de recolección, como son nodos, puertas de enlace y servidores. Finalmente se describen detalles de la implementación de todos los componentes del mecanismo de recolección.

Además en este trabajo de tesis se presentan tres posibles aplicaciones potenciales, se da una descripción de ellas y se comenta como es que el mecanismo de recolección de datos puede aplicarse para dar forma a la aplicación.



## **Abstract**

Wireless sensor network is an emerging technology that can be widely used for the creation of new applications in various fields of knowledge. But even though this technology seems to be an excellent solution for a wide range of scenarios, it also introduces new design challenges that those implementing the new systems must face. It is worth to mention that these networks base their operations on tiny devices called nodes, which are restricted in computing power, memory and energy.

This thesis presents a secure mechanism of data collection for wireless sensor networks, where through a careful analysis we have identified highly desirable characteristics for a data collection mechanism. Among the most important requirements, we have found: scalability, security, diversity of sensors, among others. Another very important feature of this work is that nodes can be mobile; to meet this requirement we have incorporated global positioning capabilities.

After having completed the requirements analysis we proceeded to propose an architecture, that recollects data by means of several classes of nodes, a packet forwarding process, and components of the collection mechanism, such as nodes, gateways and servers. Later, we describe details of the implementation of all components that form part of the main architecture.

Finally, in this thesis we propose three potential application scenarios, a description of each one is given and we also discuss how the mechanism of data collection can be applied.



# Índice general

<b>1</b>	<b>Introducción</b>	<b>1</b>
	Introducción	1
<b>2</b>	<b>Antecedentes</b>	<b>3</b>
2.1	Tercera era de la computación . . . . .	3
2.2	Tecnologías inalámbricas . . . . .	4
2.3	Redes inalámbricas de sensores . . . . .	6
2.3.1	Escenarios típicos y aplicaciones . . . . .	8
2.3.2	Sensores y adquisición de datos . . . . .	9
2.3.3	Movilidad . . . . .	10
2.3.4	Retos de diseño . . . . .	10
2.3.5	Seguridad . . . . .	11
2.4	Trabajos relacionados . . . . .	17
<b>3</b>	<b>Tecnología utilizada</b>	<b>19</b>
3.1	Análisis . . . . .	19
3.1.1	Red Inalámbrica de Sensores . . . . .	20
3.1.2	Software empotrado . . . . .	21
3.1.3	Software de aplicación . . . . .	24
3.1.4	Algoritmos de cifrado . . . . .	25
3.2	Plataforma elegida . . . . .	30
3.2.1	Red Inalámbrica de Sensores . . . . .	30
3.2.2	Software empotrado . . . . .	33
3.2.3	Software de aplicación . . . . .	34
3.2.4	Algoritmos de cifrado . . . . .	34
3.2.5	Otros . . . . .	35

<b>4</b>	<b>Arquitectura</b>	<b>37</b>
4.1	Análisis . . . . .	37
4.1.1	Requerimientos . . . . .	38
4.2	Diseño . . . . .	39
4.2.1	Paquete de datos . . . . .	39
4.2.2	Tipos de nodos . . . . .	41
4.2.3	Tipos de datos . . . . .	42
4.2.4	Reenvío de paquetes . . . . .	42
4.2.5	Nodo de lectura . . . . .	44
4.2.6	Nodo de reenvío . . . . .	45
4.2.7	Nodo de procesamiento . . . . .	45
4.2.8	Puerta de enlace simple . . . . .	45
4.2.9	Puerta de enlace avanzada . . . . .	46
4.2.10	Servidor de almacenamiento . . . . .	47
4.2.11	Servidor de consulta . . . . .	48
<b>5</b>	<b>Desarrollo</b>	<b>49</b>
5.1	Implementación . . . . .	49
5.1.1	Nodo de lectura . . . . .	49
5.1.2	Nodo de reenvío . . . . .	63
5.1.3	Nodo de procesamiento . . . . .	65
5.1.4	Puerta de enlace simple . . . . .	66
5.1.5	Puerta de enlace avanzada . . . . .	68
5.1.6	Servidor de almacenamiento . . . . .	69
5.1.7	Servidor de consulta . . . . .	71
5.2	Pruebas y resultados . . . . .	72
5.2.1	Problemas presentados y su solución . . . . .	75
<b>6</b>	<b>Aplicaciones Potenciales</b>	<b>77</b>
6.1	Sistema de información médica . . . . .	77
6.2	Sistema de seguimiento para atletas . . . . .	79
6.3	Sistema de inventarios . . . . .	81
<b>7</b>	<b>Conclusiones y Trabajo Futuro</b>	<b>85</b>
7.1	Trabajo futuro . . . . .	86

<b>A Anexo A</b>	<b>89</b>
A.1 Compilación y carga del programa en los nodos . . . . .	89
A.2 Instalación del compilador para arquitecturas <i>ARM</i> . . . . .	90
A.3 Cambio del <i>baudrate</i> en los nodos <i>MICAZ</i> . . . . .	91
A.4 Interfaces de comunicación en los nodos <i>MICAZ</i> . . . . .	92
A.4.1 Interfaz <i>I2C</i> . . . . .	92
A.4.2 Interfaz <i>SPI</i> . . . . .	92
A.4.3 Interfaz <i>UART</i> . . . . .	94



# Índice de figuras

2.1	Comparativa de tecnologías inalámbricas. . . . .	6
2.2	Ruteo alterado. . . . .	14
3.1	Comunicación sin cifrado través de un canal inseguro. . . . .	26
3.2	Comunicación con cifrado a través de un canal inseguro. . . . .	27
3.3	Esquema para el cifrado de datos. . . . .	29
3.4	Esquema para el descifrado de datos. . . . .	30
3.5	Nodo <i>MICAZ</i> . . . . .	31
3.6	Puerta de enlace <i>MIB520</i> . . . . .	31
3.7	Puerta de enlace <i>MIB600</i> . . . . .	32
3.8	Puerta de enlace <i>NB100</i> . . . . .	32
3.9	Tarjeta de sensores <i>MDA100</i> . . . . .	32
3.10	Tarjeta de sensores <i>MTS420CC</i> . . . . .	33
3.11	Modulo de lectura <i>RFID</i> . . . . .	33
4.1	Arquitectura del mecanismo de recolección. . . . .	40
4.2	Estructura del paquete de datos. . . . .	41
4.3	Esquema de reenvío de paquetes. . . . .	43
4.4	Nodo de lectura. . . . .	44
4.5	Puerta de enlace simple. . . . .	46
4.6	Petición de almacenamiento. . . . .	46
5.1	Máquina de estados para el nodo de lectura. . . . .	50
5.2	Implementación del cifrado CBC. . . . .	51
5.3	Estructura de la máquina de estados para el nodo de lectura. . . . .	51
5.4	Arquitectura del nodo <i>MICAZ</i> . . . . .	52
5.5	Segmento de código en <i>nesC</i> que inicializa el modulo <i>GPS</i> . . . . .	54
5.6	Inicialización del modulo <i>GPS</i> vista en un analizador lógico. . . . .	55

5.7	Ejemplo de mensajes generados por el <i>GPS</i> en formato <i>NMEA 0183</i> . . . . .	55
5.8	Segmento de código en <i>nesC</i> que interpreta el mensaje <i>\$GPGLL</i> proveniente del modulo <i>GPS</i> . . . . .	56
5.9	Protocolo de comunicación para el sensor de humedad. Imagen original [12].	57
5.10	Implementación de la rutina <i>SecuenciaInicio</i> para el sensor SHT11. . . . .	58
5.11	Implementación de la rutina <i>EnviaDireccion</i> para el sensor SHT11. . . . .	58
5.12	Implementación de la rutina <i>LeeBit</i> para el sensor SHT11. . . . .	59
5.13	Implementación de la rutina <i>LeeSensor</i> para el sensor SHT11. . . . .	59
5.14	Circuito utilizado para adaptar los niveles de voltaje. . . . .	61
5.15	Implementación de la petición de lectura. . . . .	63
5.16	Implementación del evento recibe. . . . .	64
5.17	Implementación del evento completado para el temporizador. . . . .	65
5.18	Implementación del evento recibe para el nodo de procesamiento. . . . .	66
5.19	Petición de almacenamiento para puerta de enlace simple. . . . .	67
5.20	Envío de petición por el puerto serial. . . . .	67
5.21	Formato de los datos recibidos. . . . .	68
5.22	Petición de almacenamiento para puerta de enlace avanzada. . . . .	69
5.23	Envío de datos al servidor remoto a través de un <i>socket</i> . . . . .	69
5.24	Pseudocódigo que recupera los datos de la petición remota y los almacena.	71
5.25	Estructura del programa que muestra los datos almacenados. . . . .	72
5.26	Estructura del programa que muestra los datos almacenados en un mapa. .	73
5.27	Datos recolectados. . . . .	74
5.28	Mapa generado con los datos recolectados. . . . .	74
6.1	Esquema simplificado para la aplicación de información médica. . . . .	79
6.2	Esquema simplificado para la aplicación de seguimiento para atletas. . . . .	81
6.3	Esquema simplificado para la aplicación de levantamiento de inventarios. .	83
A.1	Comando para compilar y cargar los programas en los nodos . . . . .	89
A.2	Estructura del archivo <i>makefile</i> . . . . .	89
A.3	Formato de las señales de comunicación en <i>I2C</i> . . . . .	93
A.4	Formato de las señales de comunicación en <i>SPI</i> . . . . .	94
A.5	Formato de las señales de comunicación en <i>UART</i> . . . . .	95

# Índice de tablas

3.1	Lenguajes de programación disponibles para todas las arquitecturas. . . . .	22
4.1	Tipos de nodos definidos. . . . .	42
4.2	Tipos de datos definidos. . . . .	42
5.1	Señales del interruptor electrónico <i>U7</i> y <i>U9</i> . . . . .	53
5.2	Distribución de terminales en la tarjeta de sensores MDA100. . . . .	61
5.3	Petición para leer un identificador [19]. . . . .	62
5.4	Respuesta del modulo de lectura cuando se ha encontrado un identificador [19]. . . . .	62
5.5	Respuesta del modulo de lectura cuando no se ha encontrado un identificador [19]. . . . .	63
5.6	Estructura de la tabla Registros. . . . .	70
5.7	Problemas presentados en está tesis y su solución. . . . .	75
A.1	Configuración del <i>baudrate</i> para el microcontrolador <i>Atmel ATMEGA128</i> . . . . .	91
A.2	Componentes en <i>nesC</i> para las principales interfaces de comunicación . . . . .	96



# Capítulo 1

## Introducción

Una red inalámbrica de sensores es una infraestructura compuesta por nodos de sensado equipados con procesadores limitados en conjunto con capacidades de comunicación inalámbrica limitadas. A pesar de estas limitaciones, las redes inalámbricas de sensores ofrecen relativamente bajas tasas de transmisión en enlaces de corto alcance, ultra bajo consumo de energía y bajo costo total. Estas características permiten que un administrador de sistemas tenga capacidad de instrumentar y vigilar eventos físicos, así como la oportunidad de reaccionar contra diferentes incidentes y los fenómenos que pueden ocurrir en el ambiente que se encuentra bajo observación.

Aun cuando existen tecnologías bastante útiles, como las redes inalámbricas de sensores, actualmente el uso de sistemas de captura automática de información tiene poca difusión, por lo que este trabajo presenta el análisis, diseño e implementación de un mecanismo seguro de recolección de datos geo-referenciados utilizando nodos móviles en redes inalámbricas de sensores.

El mecanismo seguro de recolección presentado en este trabajo tiene la característica de ser escalable, ya que los nodos pueden entrar y salir en cualquier momento de la red sin necesidad de realizar configuraciones especiales. Además pueden realizar su trabajo fuera de línea y cuando se encuentre una conexión disponible, pueden enviar los datos recolectados previamente. Además el mecanismo debe ser capaz de poder ser adaptado a una gran variedad de sensores sin sufrir cambios en su funcionamiento y/o estructura. Por último, este mecanismo deberá ser capaz de operar con la mínima intervención del ser humano.

Cuando la información haya sido capturada y recolectada, ésta será almacenada en una base de datos central, para que pueda ser utilizada por algún otro componente de *software* y dar forma a una aplicación o sistema de información. En este desarrollo se incluye la utilización de funciones criptográficas para proveer servicios de seguridad a los componentes del sistema.

El resto de este documento está descrito a continuación. El capítulo 2 presenta los antecedentes teóricos sobre los cuales está fundamentado nuestro trabajo. Después, en el capítulo 3, se analiza y presenta la tecnología utilizada en este trabajo. En el capítulo 4 se presenta la arquitectura del mecanismo de recolección, cubriendo dos elementos fundamentales, el análisis y el desarrollo. El capítulo 5 presenta la etapa de implementación y de resultados obtenidos. El capítulo 5 presenta tres aplicaciones potenciales de este trabajo. Después, en el capítulo 6 abordamos las conclusiones y el trabajo a futuro. Finalmente, se presenta el anexo A con detalles de configuración e instalación.

# Capítulo 2

## Antecedentes

El objetivo del presente capítulo es realizar un análisis y presentar las propuestas más relevantes en el área donde el presente trabajo de tesis se encuentra suscrito.

### 2.1 Tercera era de la computación

Las cosas están cambiando continuamente en el mundo de la computación. Todo empezó con la era del *mainframe*: hace unos 30 años, estos grandes dispositivos se utilizaron ampliamente, por ejemplo, en las universidades. Muchos estudiantes y profesores hicieron uso de una computadora central única que compartían entre ellos. Cabe mencionar que estos equipos tenían un alto costo, un poder de cómputo limitado y una gran cantidad de de mantenimiento debido a las dimensiones del sistema.

La tecnología avanzó tal y como estaba previsto por la ley de *Moore* y fue así que entramos en la segunda era de la computación. Esta era, que aún está presente pero que poco a poco va llegando a su final, está caracterizada por la computadora personal, donde ésta es más barata y más pequeña, y cada vez más asequible. Muy a menudo, el usuario promedio tiene acceso y utiliza más de una computadora, estas máquinas están presentes ahora en casi cualquier hogar y lugar de trabajo.

Pero, en este entorno familiar, las cosas empiezan a cambiar y la tercera era de la computación gana más y más terreno cada día. Los avances tecnológicos provocan que las computadoras personales, se vuelvan más pequeñas y baratas. Los equipos de escritorio

tienden a ser sustituidos por computadoras portátiles y otros dispositivos móviles. Lo que antes era un teléfono celular convencional ahora es un dispositivo mucho más sofisticado (incorporando, por ejemplo, una cámara digital o una pantalla de alta resolución) con mayores capacidades de comunicación [53].

El principal factor que está influyendo en la nueva transición es la disponibilidad de tecnologías inalámbricas de comunicación [33]. Los usuarios están utilizando ampliamente dispositivos inalámbricos de comunicación debido a su independencia de equipos fijos. El éxito y la disponibilidad de Internet trajo aún más independencia al usuario: los datos ahora podrán estar disponibles independientemente de la ubicación física del dueño.

Los avances en la tecnología no se detienen aquí: los microprocesadores se han hecho más pequeños y baratos, por lo que ahora se encuentran en casi cualquier dispositivo que nos rodea en el hogar, como un reloj despertador o un aparato electrodoméstico. Los esfuerzos actuales se concentran para que estos dispositivos interactúen entre sí y se organicen en redes *ad-hoc* para cumplir con objetivos específicos de manera más rápida y más confiable.

Esto es, de hecho, la tercera era de la computación prevista hace dos décadas por *Mark Weiser* [51]. El mundo de la computación ubicua tiene una visión contraria sobre el uso de poder de cómputo: en lugar de tener muchos de usuarios reunidos alrededor de una computadora central, ahora cada usuario utiliza los servicios de varias redes inmersas en el medio ambiente.

## 2.2 Tecnologías inalámbricas

El número creciente de sistemas de control y de monitoreo presentes en casi cualquier dispositivo electrónico, y el amplio requerimiento de conectividad en estos sistemas ha causado un cuello de botella en el desarrollo de los mismos [29]. Además, es pertinente recordar que los fabricantes de dispositivos electrónicos utilizan diferentes interfaces de comunicación, ya sea estándares o propietarias, para la comunicación entre componentes en los sistemas anteriormente mencionados.

Los enlaces de comunicación en sistemas electrónicos de control y de monitoreo, son comúnmente cableados, ya que ello permite un intercambio de información confiable entre los diferentes componentes del sistema. Un caso especial, ocurre cuando los periféricos se encuentran fuera del dispositivo de control, pues el cableado requerido trae problemas como: costo de instalación, seguridad, conveniencia, factibilidad, entre otros.

La tecnología inalámbrica es la solución obvia para superar los inconvenientes descritos con anterioridad, aún cuando esta solución implica un nuevo conjunto de retos, por ejemplo, propagación de la señal, interferencia, seguridad, regulaciones y otros. La tecnología para superar estos problemas existe, pero normalmente con complejidad agregada causando un incremento en el costo del sistema.

Ciertamente, muchas aplicaciones pueden pagar el costo de agregar un sistema de comunicación inalámbrica de gama alta, por ejemplo, un teléfono celular, redes de área local, entre otros. Contrariamente, muchas otras aplicaciones pueden resultar ser factibles o verse mejoradas si una solución de comunicación inalámbrica estándar de bajo costo se encuentra disponible.

Una red inalámbrica de área personal con una baja tasa de transmisión, es una red diseñada para las comunicaciones sin cables, de corto alcance, de bajo costo y de muy bajo consumo de potencia. Esta definición está en desacuerdo con la tendencia actual en tecnologías inalámbricas, cuyo enfoque suele darse en el desarrollo de sistemas de comunicaciones con un alto rendimiento y gran volumen de procesamiento de datos, así como una calidad de servicio mejorada [29].

Como se mencionó anteriormente, existen diferentes tecnologías para la comunicación inalámbrica (e.g. *Zigbee*, *Bluetooth*, *IEEE 802.11*, *WiMax*, *3G*, etc), cada una de ellas diseñada para un propósito específico y con características de trabajo muy diferentes, tal y como lo muestra la figura 2.1 [49].

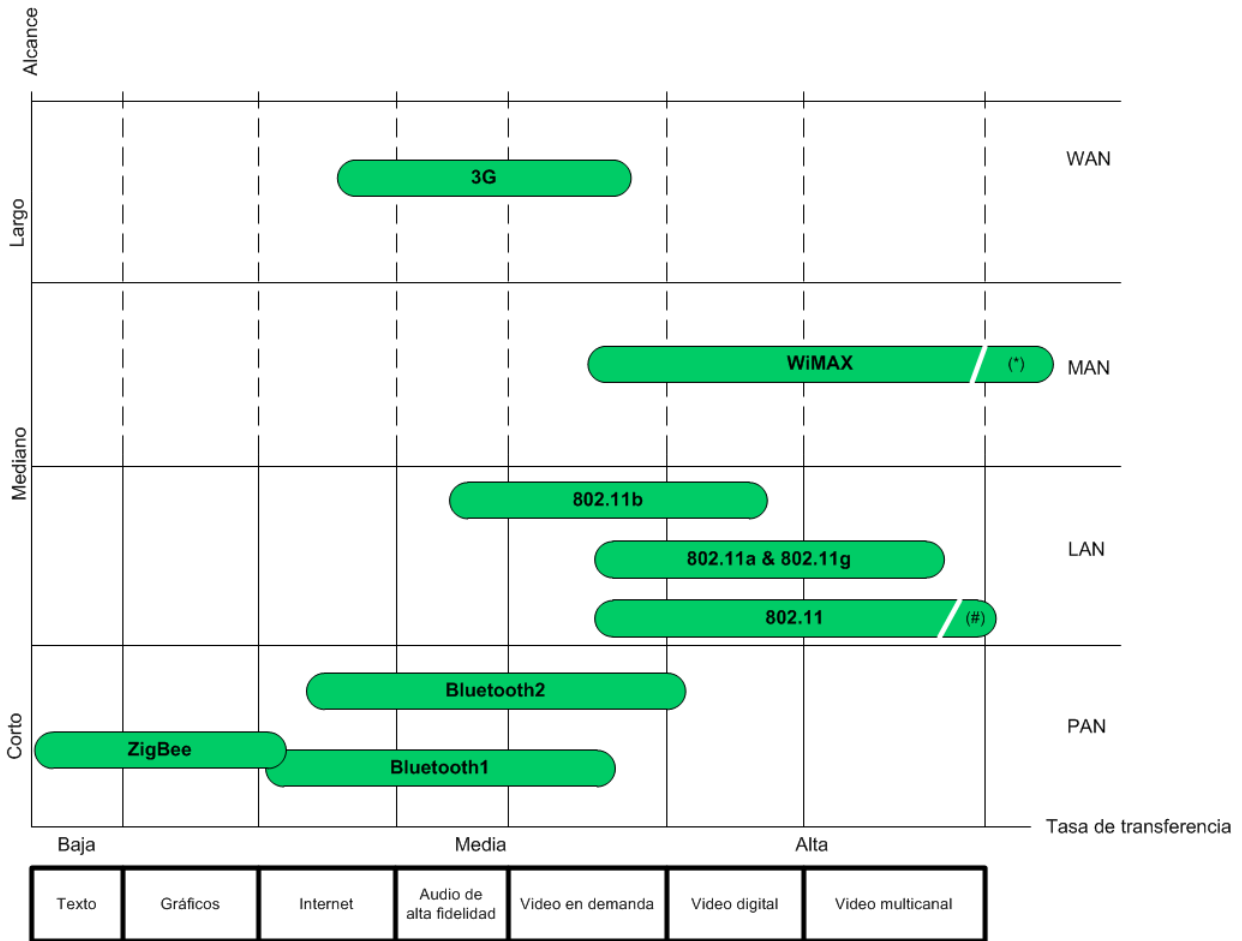


Figura 2.1: Comparativa de tecnologías inalámbricas.

## 2.3 Redes inalámbricas de sensores

Las redes inalámbricas de sensores es el nombre genérico bajo el cual se conoce a una amplia gama de dispositivos de cómputo empotrado. Básicamente, cualquier conjunto de dispositivos equipados con un procesador, con capacidades de sensado y comunicación y con la capacidad de organizarse en una red creada de manera *ad hoc* se consideran dentro de esta categoría [53].

La adición de capacidades de comunicación inalámbrica a los sensores ha aumentado su funcionalidad dramáticamente. Las redes inalámbricas de sensores brindan capacidades de monitoreo lo que cambió para siempre la forma en la que se recogen datos del medio ambiente. Por ejemplo, la grabación de actividad geológica, monitoreo de propiedad

químicas o biológicas, o el monitoreo de fenómenos ambientales como la lluvia.

El enfoque anterior fue: grandes y robustos dispositivos eran necesarios para construir los equipos de sensado, sin olvidar una fuente de gran potencia y capacidades locales de almacenamiento de datos. Un equipo de científicos tenía que viajar hasta el destino del objeto a ser monitoreado, colocar los aparatos caros en las posiciones predefinidas, y calibrar todos los sensores. Luego, era necesario volver después de un cierto tiempo para recoger los datos recolectados. Si por desgracia existía algún fallo en el *hardware*, entonces nada podía hacerse al respecto, ya que la información recolectada sobre el fenómeno se habría perdido.

El nuevo enfoque consiste en la construcción de dispositivos de sensado de bajo costo y de pequeño tamaño, con alta eficiencia energética. Como cientos o incluso miles de estos dispositivos se van a desplegar, las limitaciones de confiabilidad para ellos pierden importancia. No hay almacenamiento local de datos, ya que la información se procesará a nivel local y luego se transmitirá vía radio a uno o más puntos de acceso conectado a una red informática. La calibración individual de cada nodo sensor ya no es necesaria, ya que puede llevarse a cabo mediante algoritmos localizados [52]. La instalación también será más fácil, colocando de forma aleatoria los nodos (por ejemplo, estos se pueden tirar desde un avión) para el seguimiento de una región geográfica.

Teniendo en cuenta este ejemplo, podemos proveer una descripción general de un nodo sensor. El nombre de *nodo sensor* (o *mote*) se utiliza para describir un pequeño dispositivo que tiene comunicación inalámbrica de corto alcance, un pequeño procesador y varios sensores. Puede ser alimentado por baterías y su función principal es recoger datos de un fenómeno, colaborar con sus vecinos, y enviar sus datos (incluso pre-procesando la información o tomando decisiones) hasta el punto final, si así lo requiere. Esto es posible gracias a su procesador, además, contiene el código que permite la comunicación entre nodos y la puesta en marcha, mantenimiento, y la reconfiguración de la red inalámbrica. Cuando se hace referencia a la comunicación inalámbrica, tenemos en mente sobre todo la comunicación por radio (otros medios como la ecografía, luz visible o infrarroja, etc. también pueden ser utilizados [50]).

Las redes inalámbricas de sensores son una de las herramientas más importantes de la tercera era de la computación. Estos son los dispositivos más sencillos, teniendo como objetivo principal la vigilancia del medio y la alerta de los principales acontecimientos

que están sucediendo. Con base en la observación reportada por estos instrumentos, los humanos y las máquinas pueden tomar decisiones y actuar en consecuencia.

### 2.3.1 Escenarios típicos y aplicaciones

En este momento, existe una gran variedad de sensores, éstos se han desarrollado para monitorear casi todos los aspectos del medio ambiente: las condiciones de iluminación, temperatura, humedad, presión, la presencia o ausencia de químicos o biológicos diversos, detección de presencia y movimiento, etc.

Mediante la creación de redes con un gran número de sensores y su despliegue en el interior del fenómeno a estudiar, se obtiene una herramienta de monitoreo más potente que al tener un solo sensor.

Las redes inalámbricas de sensores se pueden clasificar sobre una base de complejidad de las aplicaciones [27]:

- Almacenamiento inteligente: cada elemento a ser controlado dentro de un almacén es etiquetado con un pequeño dispositivo, estas etiquetas son supervisadas por el sensor fijo que se encuentra incrustado en las paredes y estanterías. Con base en los datos leídos, el conocimiento de la posición de los sensores y el tiempo de la comunicación, la red de sensores ofrece información sobre el tráfico de mercancías en el interior del edificio, crea inventarios automáticos e incluso realiza las correlaciones de largo plazo entre los datos leídos.

Desde el punto de vista de la complejidad, este escenario requiere un mínimo de complejidad. Los nodos de sensores son colocados en posiciones fijas, de una manera más o menos al azar. La zona de despliegue es de fácil acceso y algunas infraestructuras (por ejemplo, fuentes de alimentación y equipos) ya existen. En esta categoría, podemos incluir el escenario de un supermercado moderno, donde los productos seleccionados de los clientes se identifican automáticamente a la salida del supermercado.

- Monitoreo ambiental. Esta es el área más amplia de aplicaciones previstas hasta ahora. Una aplicación particular de esta categoría es el monitoreo de desastres. Los nodos de sensores son desplegados en las zonas afectadas y estos puede ayudar a los humanos a estimar los efectos de la catástrofe, construir mapas de las zonas seguras, y dirigir las acciones humanas hacia las regiones afectadas.

Un gran número de aplicaciones en esta categoría se enfocan al control de la fauna. Este escenario tiene una mayor complejidad. La zona de despliegue ya no es accesible de manera fácil y ya no es seguro para los nodos de sensores. Apenas hay infraestructuras actuales, los nodos deben ser diseminados en forma aleatoria y la red puede contener nodos móviles, también será necesario desplegar un mayor número de nodos.

- Redes de sensores de gran escala: el escenario de una gran ciudad donde todos los autos tienen sensores integrados. Estos nodos de sensores se comunican entre sí y recopilan información sobre el tráfico, rutas y condiciones especiales de tráfico. Por un lado, la nueva información está disponible para el conductor de cada vehículo. Por otro lado, una visión global de toda la ciudad también puede estar disponible. Las dos principales limitaciones que caracterizan este escenario son el gran número de nodos y su alta movilidad. Los algoritmos empleados necesitan ser escalables y hacer frente a una red con topología en cambio continuo.

### 2.3.2 Sensores y adquisición de datos

Los sensores son la interfaz con el mundo real, ellos recolectan la información necesaria y éstos son monitoreados por la unidad de procesamiento central. Las plataformas pueden estar construidas en una forma modular de modo que una variedad de sensores puedan ser usados en la misma red.

La tecnología para el sensado y control incluye sensores de campo eléctrico y campo magnético; sensores de radio frecuencia, sensores ópticos, opto-electrónicos e infrarrojos; radares, láseres; sensores de localización y navegación; sensores sísmicos; sensores ambientales (viento, humedad, calor); sensores bioquímicos. Los sensores de hoy pueden ser descritos como *inteligentes*, esto es, dispositivos baratos equipados con varios sensores, que son de bajo costo y bajo consumo.

La mayoría de las plataformas de redes inalámbricas de sensores incluyen la posibilidad de agregar nuevos dispositivos de lectura a la plataforma de cómputo, esto nos ofrece la posibilidad de ampliar las capacidades de sensado.

### 2.3.3 Movilidad

Las redes inalámbricas de sensores se caracterizan por tener topologías muy dinámicas, inducida por las fluctuaciones en la conectividad inalámbrica. Sin embargo, algunas aplicaciones pueden introducir un grado aún mayor de dinamismo, debido a la necesidad de apoyar físicamente a dispositivos móviles [39]. La movilidad puede (o no) manifestarse de diferentes maneras:

En las aplicaciones estáticas, no se mueven los nodos una vez desplegados. Esto es, con mucho, el caso más común en las implementaciones actuales.

Algunas aplicaciones utilizan dispositivos de sensado conectados a entidades móviles (por ejemplo, robots o animales) o capaz de moverse de forma autónoma (por ejemplo, el proyecto nodos XYZ [40]). Un caso típico es el seguimiento de animales en donde los sensores están conectados a los animales, como en el proyecto ZebraNet [32].

Algunas aplicaciones explotan los receptores móviles: los nodos pueden ser fijos o no: el aspecto clave es que la recopilación de datos se realiza de forma oportunista cuando se mueve el receptor en la proximidad de los sensores [48].

### 2.3.4 Retos de diseño

Al diseñar un sensor de una red inalámbrica se enfrenta por un lado la simplicidad del *hardware* subyacente, y, por otra parte, los requisitos que deben cumplirse. Para satisfacerlos, nuevas estrategias y nuevos conjuntos de protocolos se han de desarrollar.

A continuación, vamos a abordar los principales desafíos que están presentes en el campo de redes inalámbricas de sensores. Las líneas de investigación implicadas y las cuestiones pendientes que aún deben ser respondidas se presentarán también. En primer lugar, una

descripción de alto nivel de los objetivos actuales de las redes de sensores pueden sintetizarse como:

- **Larga vida:** El nodo sensor debe ser capaz de vivir el mayor tiempo posible con sus propias baterías. Esta limitación se puede traducir a un consumo energético inferior a  $100\text{ mW}$ . La condición surge de la suposición de que los nodos sensores se desplegarán en un entorno hostil donde el mantenimiento es imposible o tiene un precio prohibitivo. El tiempo de vida de un nodo especifica el periodo de tiempo que éste funciona con dos baterías de tamaño AA, que comúnmente debe ser de un par de años. Este objetivo sólo puede lograrse mediante la aplicación de una estricta política de consumo de potencia, la cual hará uso de los modos de ahorro de energía.
- **Tamaño pequeño:** El tamaño del dispositivo debe ser inferior a  $1\text{ mm}^3$ . Este límite dio el nombre a los nodos sensores de "polvo inteligente", un nombre que da una idea muy intuitiva sobre el diseño final. Recientemente, el procesador y la radio se ha integrado en un chip con un tamaño de aproximadamente  $1\text{ mm}^3$ . Lo que queda es la antena, los propios sensores, y la batería. Los avances se requieren en cada uno de estos tres campos para poder cumplir con esta limitación de diseño.
- **Bajo costo:** La tercera limitación de alto nivel de diseño es el precio de estos dispositivos. Para alentar el despliegue a gran escala, esta tecnología debe ser más barata, lo que significa que los precios de los dispositivos deben estar en el rango de centavos.

### 2.3.5 Seguridad

La seguridad y la privacidad son aspectos importantes para la computación en general, sin embargo para sistemas basados en redes inalámbricas de sensores, estos retos se incrementan dada la dinámica, espontaneidad, heterogeneidad y naturaleza invisible de la aplicación [36].

El desarrollo de sistemas seguros con redes inalámbricas de sensores es un área que ha sido poco explorada, ya que al utilizar un medio de transmisión no guiado, como lo es el aire, se encuentran sujetos a diversos ataques, los cuales pueden variar de acuerdo

a la aplicación, por parte de entidades maliciosas. Además, estos dispositivos no tienen la suficiente capacidad de cómputo para ejecutar algoritmos criptográficos utilizados en aplicaciones cliente - servidor.

Los mecanismos existentes de autenticación, están diseñados para pocas asociaciones de amplia duración entre un usuario y un dispositivo, hecho que no sucede en las aplicaciones basadas en redes de sensores, ya que por la dinámica de la red, una operación de autenticación puede ocurrir repetidas ocasiones en un lapso pequeño de tiempo, lo cual implica que una significativa porción de tiempo el nodo estará ejecutando algún algoritmo criptográfico, situación que puede dar lugar a un comportamiento no deseado.

Otro aspecto importante de seguridad en este tipo de sistemas, es la privacidad donde emergen nuevos desafíos, ya que estas aplicaciones pueden recopilar datos acerca de los usuarios, incluyendo información de localización, actividad, personas que los acompañan, voz, video, datos biológicos, entre otros. Si estos sistemas se encuentran inmersos en el medio ambiente, es común que la gente ignore que se están recopilando datos acerca de su comportamiento.

Es importante mencionar que este tipo de aplicaciones no pueden confiar en un acceso continuo a servidores, ya que como se comentó anteriormente, estas aplicaciones son frecuentemente desconectadas, por lo que es difícil implementar un esquema de seguridad que involucre una conexión remota.

Por último, la naturaleza de estos sistemas, crea la necesidad para un nuevo tipo de seguridad basada en la localización del nodo y en el contexto de operación, y no en el usuario. Por ejemplo, autorizar una transacción sólo si el nodo solicitante se encuentra en un área determinada.

En cuanto al área de seguridad, hemos estudiado varios artículos [23, 34, 38, 44, 47, 53], los cuales nos ofrecen un panorama general del análisis desarrollado previamente y de algunas propuestas efectuadas por otros investigadores. Cuando pensamos en aspectos de seguridad en redes inalámbricas de sensores, una de las preguntas principales que nos vienen a la mente, es ¿cuáles son las diferencias entre la seguridad en redes de sensores y la seguridad en general? [53]

Los siguientes objetivos son considerados como esenciales: autenticidad de entidades y mensajes, integridad de datos, confidencialidad, control de acceso y no repudio de actos de comunicación [47].

La autenticidad de entidades se refiere al procedimiento utilizado para verificar que la identidad de los participantes en un acto de comunicación es verdadera, mientras que para los mensajes significa que existe una forma de comprobar que éste ha sido generado por el remitente y no por un atacante.

Un mecanismo que nos garantiza que los datos no han sido alterados (de forma voluntaria o involuntaria) es la integridad de datos, el cual nos asegura que los datos recibidos coinciden con los datos enviados. Un servicio fundamental cuando se transmiten datos sensibles (e.g. contraseñas, números de tarjetas de crédito, entre otros) es el de la confidencialidad, el cual nos asegura que la información transmitida no puede ser vista por entidades ajenas. El control de acceso consiste en que únicamente los usuarios autorizados puedan hacer uso del sistema. Por último, el no repudio es un servicio de seguridad que impide a un usuario retractarse de un acto de comunicación (e.g. una transacción en línea).

Básicamente, éstos son los mismos objetivos que necesitan ser asegurados en redes inalámbricas de sensores (quizá con excepción del no repudio, que es el de menos interés, debido al nivel en el cual operan las redes de sensores).

Desde un punto de vista muy general, uno podría llegar a la conclusión de que la seguridad en las redes inalámbricas de sensores no agrega mucho a lo que sabemos de la seguridad en general, y que los mismos métodos se pueden aplicar en redes de sensores, como en redes tradicionales y/o inalámbricas. Sin embargo, una consideración más cercana revela varias diferencias en los orígenes de ambos tipos de redes, por lo que las características específicas de las redes inalámbricas de sensores hacen que el uso directo de técnicas de seguridad conocidas no sean necesariamente apropiadas.

La seguridad en redes inalámbricas de sensores es un área de investigación de interés por las siguientes razones [53]:

- Los nodos de la red de sensores se despliegan bajo condiciones particularmente difíciles, ya que una gran cantidad de nodos pueden ser distribuidos sobre un área

geográfica, suponemos que por lo menos algunos nodos podrán ser capturados y comprometidos por un atacante.

- Las restricciones de poder de cómputo, memoria y energía presentes en los nodos, los ponen en desventaja con potenciales atacantes debido a la gran diferencia de recursos disponibles.

En la referencia [34], los autores proveen una descripción de ataques y contramedidas referentes al enrutamiento en redes inalámbricas de sensores.

Uno de los principales ataques en este tipo de redes es la inserción de información de enrutamiento suplantada o alterada, esto es con la finalidad de modificar la topología de la red, permitiendo que el tráfico se pueda atraer o repeler. En la figura 2.2 *inciso a)* podemos observar la topología normal de operación en una red inalámbrica de sensores, en el *inciso b)* de la misma figura la red ha sido particionada por el *equipo A* un atacante causando una desconexión momentánea en la red, por ultimo en el *inciso c)* el atacante ha puesto en operación el *equipo B* el cual se comunica directamente con el *equipo A* proporcionando conectividad entre todos los nodos de acuerdo a la topología original. Sin embargo, toda la información que se transmita podrá ser vista y/o alterada por el atacante.

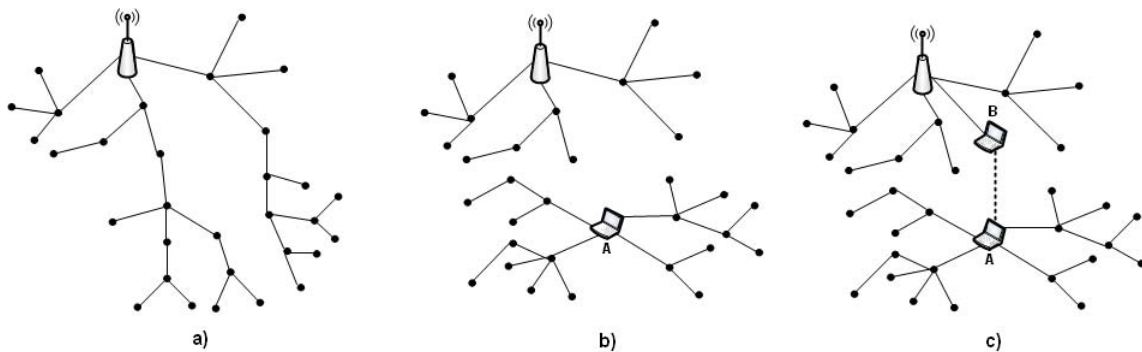


Figura 2.2: Ruteo alterado.

Otro procedimiento utilizado consiste en la creación de mensajes de reconocimiento (*ACK*) para hacer creer a otros que el estado de conexión de un nodo es diferente a su estado real. Un mecanismo más sofisticado consiste en el reenvío selectivo por medio de un atasco deliberado, esto en conjunto con la creación de nodos que absorben el tráfico y lo reenvían a un nodo en específico, permite un control sobre la información que se envía y

la que se descarta. Además se ha mostrado [34] que la simulación de múltiples identidades (*Sybil attacks*), permite reducir la efectividad de esquemas tolerantes a fallos. El último de los ataques descritos se basa en el envío de supuestos mensajes de sincronía (*hello shouting*), en donde el atacante envía o responde mensajes con una energía mayor, para engañar a otro nodo y que éste crea que son vecinos (debido a un mayor nivel de potencia).

De acuerdo con los autores del artículo [34], los problemas de seguridad referentes a la creación de mensajes de reconocimiento e información de ruteo, pueden controlarse de forma efectiva utilizando mecanismos de autenticación del origen de los datos así como la aplicación de esquemas de confidencialidad en los datos a nivel enlace.

En el artículo *Security Protocols for Sensor Networks* [44], se discuten los requerimientos y se propone un conjunto de protocolos para proveer servicios de seguridad de forma eficiente en redes de sensores. Los principales retos en el diseño de estos protocolos vienen dados por las restricciones presentes en estos dispositivos y por el hecho de que estos pueden ser capturados por atacantes. Se considera que algunas alternativas, tales como la criptografía asimétrica conllevan un alto costo computacional, lo cual las hace inadecuadas para este tipo de dispositivos. En este trabajo propone dos protocolos principales de seguridad:

- *Sensor Network Encryption Protocol (SNEP)* para realizar seguridad de extremo-a-extremo de forma eficiente.
- El protocolo *TESLA*, para autenticar las comunicaciones basadas en multidifusión.

Las principales decisiones de diseño en el desarrollo de estos protocolos fueron enfocados en evitar el uso de criptografía asimétrica, mediante la construcción de todas las primitivas de criptografía basadas en un sencillo cifrador por bloques, y con la explotación de tareas en común para reducir el exceso de comunicaciones cuando esto sea posible.

En [38], los autores extienden la idea del protocolo  $\mu$ *TESLA* para solucionar algunos problemas de escalabilidad presentados en el trabajo descrito anteriormente.

El manejo de las llaves es comúnmente una de las partes más difíciles al momento de implementar comunicaciones seguras [53]. El conjunto de protocolos de seguridad no pueden ofrecer protección alguna si las llaves caen en manos de atacantes.

El protocolo *SNEP* [44], incluye la propuesta de un protocolo simple y tradicional para el manejo de llaves, que permite que dos nodos de sensores obtengan una llave secreta compartida con la ayuda de la estación base.

A continuación se tratará el tema del manejo de llaves [47], el cual comprende las siguientes tareas:

- **Generación de llaves:** es la creación de las llaves que se utilizarán para el proceso de cifrado/descifrado. Tales llaves deben crearse ejecutando un proceso con la utilización de números aleatorios, o por lo menos números pseudo-aleatorios. En caso contrario, los atacantes pueden crear sus propias llaves ejecutando el mismo proceso. Si se decide utilizar números pseudo-aleatorios es importante inicializar la generación de la secuencia con un número verdaderamente aleatorio.
- **Distribución de llaves:** es la entrega de las llaves al lugar (o entidad) que hará uso de ellas. En escenarios simples, las llaves pueden ser distribuidas de forma directa. Si los nodos se encuentran físicamente alejados y se utilizan algoritmos de cifrado simétricos, el canal de comunicación necesita ser protegido mediante cifrado. Por lo tanto, una llave es necesaria para la distribución de llaves. Esta necesidad implica la introducción de lo que se conoce como jerarquía de llaves.
- **Almacenamiento de llaves:** cuando se implemente esta funcionalidad se deben tomar medidas para evitar que las llaves sean leídas por usuarios no autorizados.
- **Recuperación de llaves:** es la reconstrucción de llaves cuando éstas han sido perdidas o comprometidas. La manera más simple de instrumentar este mecanismo es mantener una copia de todas las llaves en un lugar seguro. Sin embargo, esto crea un posible problema de seguridad, ya que es necesaria una garantía absoluta de que las llaves no serán robadas.
- **Invalidación de llaves:** es una tarea muy importante del manejo de llaves, especialmente en los métodos de criptografía asimétrica. Si la llave privada ha sido revelada, su correspondiente llave pública debe ser publicada como inválida. En redes de sensores la invalidación de llaves se espera que sea una operación absolutamente eficiente, dado que la posibilidad de que un nodo sea capturado y comprometido es bastante alta.

- Destrucción de llaves no requeridas: esta operación es necesaria con el propósito de asegurar que los mensajes cifrados con estas llaves no puedan ser en un futuro descifrados por personas no autorizadas. Es importante asegurarnos que todas las copias de las llaves sean completamente destruidas, de manera que no puedan volver a ser leídas, aun con la utilización de técnicas sofisticadas de recuperación de datos.

Aunado a las consideraciones anteriores, es necesario especificar algunos requerimientos en particular para el manejo de llaves para esquemas de redes de sensores [23], entre los que destacan:

- Vulnerabilidad de los nodos a ser capturados.
- Desconocimiento previo de la configuración de despliegue de los nodos.
- Dispositivos con restricciones.
- Topología dinámica de la red.

## 2.4 Trabajos relacionados

El primer artículo estudiado [42], consiste en la utilización de redes inalámbricas de sensores para la localización vehicular, su objetivo es el de localizar un vehículo equipado con un transceptor de radio frecuencia compatible con el estándar *IEEE 802.15.4*. En el esquema propuesto por el autor existen tres tipos de nodos: los automóviles a localizar, los cuales poseen un nodo, los autobuses que transitan por una ruta fija, también llamados nodos móviles, que tienen como función recolectar datos de otros nodos utilizando un esquema automóvil-a-automóvil y los nodos fijos, que se encuentran en lugares bien definidos (e.g. las paradas de la ruta de autobuses).

Una característica muy importante en este tipo de redes inalámbricas, es que los nodos se unen y dejan la red inalámbrica en cualquier momento. Cuando los nodos móviles y los nodos fijos entran en contacto debido a su cercanía, se transmite un mensaje al servidor del sistema de localización que contiene el identificador del nodo móvil y el identificador del nodo fijo, por lo tanto la localización del vehículo es actualizada en el sistema.

Cuando la información de localización de un vehículo cambia, se actualiza la posición del autobús en pantalla, y se almacena para ser utilizada por bitácoras y reportes. El

servidor también transmite información basada en la localización que es útil para el pasajero en el nodo móvil colocado en un autobús. La información transmitida está basada en la localización actual del autobús, como pueden ser nombres de complejos turísticos y edificios principales.

El siguiente trabajo analizado [41] consiste en la realización de una estructura para diseminar y reunir información acerca de vehículos que se encuentran en camino, el cual puede ser de utilidad al momento de manejar en situaciones de neblina, lluvia o para encontrar una ruta óptima en un viaje largo.

Un sistema basado en posicionamiento global sólo ofrece una vista estática del mapa, mientras que el trabajo propuesto por el autor ofrece una perspectiva dinámica, la cual se complementa con la información recibida del sistema de posicionamiento global. Este sistema al integrarse con información georeferenciada puede proveer la funcionalidad de un planificador de rutas en tiempo real.

Este trabajo está basado en un computadora de mano *Compaq iPAQ* (expandida con 2 slots *PCMCIA*) con una distribución de *GNU/Linux*, un receptor del sistema de posicionamiento global (*GPS*), una tarjeta de red inalámbrica compatible con *IEEE 802.11b*, una tarjeta *PCMCIA* de 2 puertos seriales y de una interfaz *OBDI-II*.

Una desventaja del trabajo [42] es que para que la localización del vehículo pueda efectuarse en cualquier punto, es necesario contar con una infraestructura de red inalámbrica de cobertura amplia (e.g. a lo largo y ancho de la ciudad). Referente al sistema presentado en [41] podemos destacar que ocupa varios elementos de *hardware*, lo cual inevitablemente incrementa el costo de la aplicación, además de basar su red inalámbrica en el estándar *IEEE 802.11b*, siendo que este fué diseñado para sistemas con alto volumen de transferencia de datos y no para sistemas orientados a la recolección de datos.

# Capítulo 3

## Tecnología utilizada

En este capítulo se describe la tecnología utilizada para la realización de este trabajo. En la primera sección se realiza un análisis de todas las categorías correspondientes a este trabajo (redes inalámbricas de sensores, *software* empotrado, *software* de aplicación y algoritmos de cifrado). Después, en la segunda sección, se define una plataforma de trabajo y se describe a detalle. Finalmente, en la última sección, se presentan algunos otros componentes utilizados en la realización de este trabajo, principalmente útiles para el desarrollo y las pruebas.

### 3.1 Análisis

Esta sección presenta un análisis de las opciones más importantes de desarrollo para las categorías que este trabajo cubre.

Las categorías en las cuales trabajaremos se describen a continuación:

1. Red inalámbrica de sensores: a esta categoría pertenecen los dispositivos electrónicos que conforman la red inalámbrica de sensores, como son los nodos, las puertas de enlace, entre otros.
2. *Software* empotrado: a esta categoría pertenecen los programas que son ejecutados por los dispositivos de la red inalámbrica de sensores. También es comúnmente conocido como *firmware*.

3. *Software* de aplicación: este es el *software* que ejecutan los servidores para acceder y/o modificar las bases de datos.
4. Algoritmos de cifrado: este es el algoritmo que utilizaremos para realizar el cifrado de los datos y proveer los servicios de seguridad para los nodos del sistema.

### 3.1.1 Red Inalámbrica de Sensores

A continuación se presenta un análisis de las diferentes alternativas de *hardware* para redes inalámbricas de sensores. Inicialmente se mencionan las plataformas más representativas del área, para después hacer una descripción en base a la composición de sus elementos.

Existen una gran cantidad de plataformas para redes inalámbricas de sensores tanto en productos comerciales como en prototipos de investigación tales como: *Crossbow* [6], *Body Sensor Node* [2], *BTNodes* [4], *EYES* [5], *SunSPOT* [9], *Meshnetics* [7], *Scatter* [8], *XBee* [10] y *Arduino* [3]. Sin embargo, los componentes utilizados en todas las plataformas anteriores no difieren drásticamente. Muchas de estas plataformas usan el microcontrolador de 16 bits *MSP430* de *Texas Instruments* o un microcontrolador de 8 bits de la familia *Atmel ATMEGA*. Una excepción notable es la plataforma *SunSPOT* la cual utiliza microcontroladores de la familia *ARM9*. Las frecuencias de trabajo para estos procesadores van desde 8 *MHz* hasta arriba de 200 *MHz*, es importante recordar que el consumo de potencia del procesador está relacionado linealmente con la frecuencia de operación [24].

La cantidad de memoria volátil va desde 2 *KB* hasta 512 *KB*. Normalmente esta memoria se utiliza para almacenar las variables utilizadas durante la ejecución del programa. El código binario del programa se almacena en la memoria dedicada, que va desde 32 *KB* hasta 128 *KB*. Además, los nodos son a menudo equipados con dispositivos de almacenamiento externo, por ejemplo, circuitos electrónicos de memoria *flash*, cuyo tamaño varía desde 128 *KB* hasta 4 *MB*.

En cuanto al *hardware* utilizado para enviar y recibir datos de forma digital, la mayoría de las plataformas trabajan en la banda *ISM* de 2,4 *GHz*, y son compatibles con *IEEE 802.15.4*, por ejemplo, el circuito integrado *CC2420* de *Chipcon*. Las soluciones alternativas operan en la banda de 868/916 *MHz*, por ejemplo, utilizando el circuito integrado

transmisor-receptor *CC1000* de *Chipcon* o en otros casos utilizan interfaces *Bluetooth*.

Todas las plataformas están diseñadas para ser operadas a baterías, por lo que en general consideran componentes que ofrecen bajos consumos de potencia para poder trabajar de forma autónoma durante periodos de tiempo extendidos sin la necesidad de reemplazar su fuente de alimentación.

Respecto a los sensores, cada plataforma tiene dispositivos de sensado específicos para la aplicación que fueron diseñados y además con frecuencia permite la adición de nuevos dispositivos de sensado. La diversidad de sensores es una de las características que marcan mayor diferencia entre las plataformas, ya que mientras algunas plataformas cuentan con tarjetas de adquisición de datos con gran variedad de sensores [6] existen otras plataformas donde los sensores no forman parte del dispositivo y estos deben ser añadidos de forma externa [10].

### 3.1.2 Software empotrado

Esta categoría merece atención especial ya que analiza las herramientas disponibles para producir el programa que se ejecutará en los nodos de la red inalámbrica de sensores. Se le conoce como *software* empotrado a todo aquel programa que va integrado dentro de un dispositivo electrónico, i.e. microcontrolador. El principal uso del *software* empotrado no es en el campo de las tecnologías de información, sino más bien está pensando para interactuar con el mundo real. Este *software* se escribe para sistemas que no son las típicas computadoras de escritorio que todo mundo conoce sino que el código binario queda inmerso en los circuitos electrónicos que podemos encontrar en autos, teléfonos, equipo de audio, robots, juguetes, sistemas de seguridad, marcapasos, televisiones aunque también es utilizado en aplicaciones muy sofisticadas como son aviones, misiles, sistemas de control industrial, entre otros [37].

Para poder realizar un análisis de las diversas opciones de *software* empotrado primero debemos definir las arquitecturas de los procesadores utilizados en los nodos. Las arquitecturas de los procesadores para las cuales analizaremos las diversas opciones de herramientas de desarrollo son: *Atmel AVR*, *Texas Instruments MSP430* y *ARM9*. Con esta elección cubrimos una gran cantidad de microcontroladores presentes en redes inalámbr-

cas de sensores, tal y como fue descrito en la sección 3.1.1.

Los lenguajes de programación disponibles para estas arquitecturas son: ensamblador, C y *nesC*. El lenguaje ensamblador es un lenguaje de bajo nivel utilizado para escribir programas y constituye la representación más directa del código máquina específico para cada procesador siendo este legible para un programador. El lenguaje C es un lenguaje estructurado de propósito general, aunque es muy común que este sea utilizado para escribir sistemas operativos y/o otras utilerías debido a un gran potencia, además de permitir manipulaciones a bajo nivel [35]. *nesC* es un lenguaje de programación para sistemas empostrados en red como los *motes*, además implementa un modelo de programación basado en la ejecución de eventos. Este lenguaje soporta concurrencia y simplifica el desarrollo de la aplicación, optimiza el tamaño del código objeto y elimina fuentes de errores potenciales, como lo son la asignación de memoria dinámica y la detección de condiciones de carrera en tiempo de compilación [28].

<b>Lenguaje de Programación</b>	<b>Ventajas</b>	<b>Desventajas</b>
Ensamblador	Control total del dispositivo	Difícil de programar No hay portabilidad
C	Facilidad de programar Gran portabilidad	Eficiencia media-alta
<i>nesC</i>	Portabilidad	Eficiencia media

Tabla 3.1: Lenguajes de programación disponibles para todas las arquitecturas.

La tabla 3.1 muestra una comparativa de lenguajes de programación, es importante

mencionar que los lenguajes listados en la tabla son soportados por todas las arquitecturas analizadas en la presente sección. El lenguaje ensamblador es soportado por todas las arquitecturas, aunque cada arquitectura tiene su propia definición para este lenguaje, mientras que el lenguaje C y *nesC* son los mismos para cualquier arquitectura aunque podrían existir diferencias mínimas dependiendo del compilador utilizado.

Aparte de los lenguajes de programación disponibles para la programación en redes inalámbricas de sensores otro componente fundamental es el sistema operativo para los nodos. Estos sistemas son típicamente menos complejos que los sistemas operativos de propósito general dado los requerimientos especiales de las redes inalámbricas de sensores y por que los recursos disponibles en estos dispositivos son claramente más limitados. En general las aplicaciones que utilizan nodos de sensores son menos interactivas que las aplicaciones para equipos de escritorio. Gracias a esto el sistema operativo no necesita incluir soporte para interfaces de usuario.

*TinyOS* es el primer sistema operativo específicamente diseñado para redes inalámbricas de sensores, este sistema operativo tiene un modelo de programación de aplicaciones orientado a eventos. Varias características importantes que influyeron en el diseño de *nesC* incluyen: una arquitectura basada en componentes, un modelo de concurrencia simple basado en eventos y las operaciones de fase dividida (*split-phase*) [31].

*Contiki* es un sistema operativo ligero con soporte para la carga dinámica de programas y servicios. Este sistema fue diseñado alrededor de un núcleo orientado a eventos pero de forma opcional provee multitarea preferente que puede ser aplicada a procesos individuales [26].

*MANTIS* es un sistema orientado al bajo consumo de recursos, posee un planificador de tareas eficiente en cuestiones de energía, además solo ocupa unos 500 Bytes de memoria RAM y 14KB de memoria *flash*. Provee capacidades multitarea preferente. Se encuentra disponible para las plataformas de *MICA2* y *MICAz* de *Crossbow*, entre otras [21].

*LiteOS* es un sistema operativo con abstracción de tipo UNIX, incluye un sistema de archivos jerárquico y la posibilidad de interactuar con el sistemas a través de una interfaz de comandos (*shell*) de forma inalámbrica, el núcleo soporta la adición de módulos de forma dinámica y proporciona una ejecución multitarea de forma nativa, los programas

se pueden realizar bajo el paradigma orientado a objetos [22].

### 3.1.3 Software de aplicación

En esta categoría analizaremos brevemente el *software* de aplicación utilizado para tres propósitos principales: para la ejecución en el servidor de programas que generan páginas dinámicas, de servidores Web y de servidores de base de datos. Respecto a las opciones de lenguajes utilizadas en la ejecución de programas para páginas dinámicas tenemos a 3 principales: lenguaje *ASP*, lenguaje *PHP* y lenguaje *JSP*.

La primera opción, el lenguaje *ASP*, es producida por *Microsoft*. La programación se realiza bajo el entorno *.NET*. Típicamente se conecta fácilmente con servidores de bases de datos tipo *Microsoft SQL*. Una desventaja es que la mayor compatibilidad de este lenguaje es con productos *Microsoft*.

La segunda opción, el lenguaje *PHP*, es mantenida por la comunidad de *software* libre y es ampliamente usado en Internet. El lenguaje tiene sintaxis específica. Tiene muchas opciones de conexión con otros programas y/o servicios.

La tercera opción, el lenguaje *JSP*, es producida por *Sun Microsystems* (ahora parte de *Oracle*), es libre y la programación se realiza en el lenguaje *Java*. Tiene muchas opciones de conexión con otros programas y/o servicios.

Respecto a los servidores Web tenemos menos opciones, donde el mercado está dividido en 2 principales productos:

Servidor *IIS*. Este servidor es producido por *Microsoft* y es altamente compatible con el lenguaje *ASP*.

Servidor Apache. Este servidor es producido por la fundación Apache, es libre y bastante utilizado en Internet. Soporta al lenguaje *PHP* y al lenguaje *JSP* con sus respectivas adecuaciones.

Por último pasamos a los motores de bases de datos.

*Microsoft SQL*, es un servidor de base de datos producido y soportado por Microsoft. Es un producto bastante estable y muy utilizado por las soluciones implementadas con productos *Microsoft* y tecnología .NET.

*MySQL*, es un servidor de base de datos producido por *Oracle*. Existen versiones libres y de pago. Es ampliamente utilizado por las soluciones construidas con herramientas y tecnología de código abierto.

### 3.1.4 Algoritmos de cifrado

Dentro de la criptografía existen 2 grandes áreas: criptografía simétrica y criptografía asimétrica. A continuación se describen ambos.

Los sistemas de cifrado simétrico también son conocidos como sistemas de clave simétrica, de clave secreta o de clave única. Podemos explicar mejor el concepto de criptografía simétrica presentado un problema fácil de entender: tenemos dos usuarios, *Alicia* y *Beto*, que quieren comunicarse a través de un canal inseguro. El canal de comunicación es sólo un término general para el enlace de comunicación: este canal puede ser el Internet, una porción de aire para el caso de los teléfonos celulares o la comunicación inalámbrica en una red local, entre otros. El problema real comienza con la entidad maliciosa, quién tiene acceso al canal, por ejemplo, interviniendo un ruteador de Internet o escuchando las señales de radio en una comunicación inalámbrica. Obviamente, hay muchas situaciones en las que *Alicia* y *Beto* prefieren comunicarse sin que sean escuchados por una tercera entidad. Por ejemplo, si *Alicia* y *Beto* son dos oficinas de un fabricante de automóviles, y se transmiten los documentos que contienen la estrategia de negocio para la introducción de nuevos modelos de automóviles en los próximos años, estos documentos no deben caer en manos de sus competidores. El escenario descrito anteriormente puede observarse en la figura 3.1.

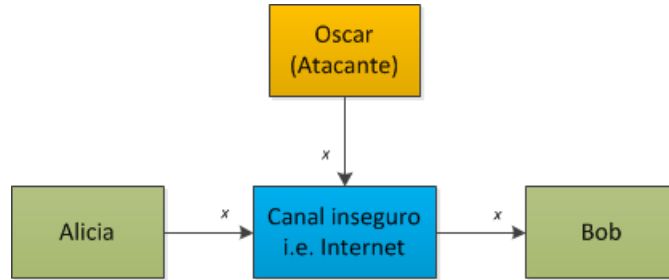


Figura 3.1: Comunicación sin cifrado través de un canal inseguro.

En esta situación, la criptografía simétrica ofrece una potente solución: *Alicia* cifra su mensaje  $x$  utilizando un algoritmo simétrico, resultado el texto cifrado  $y$ . *Beto* recibe el texto cifrado y descifra el mensaje. El descifrado es, por tanto, la inversa del proceso de cifrado. ¿Cuál es la ventaja? Si tenemos un algoritmo fuerte de cifrado, *Oscar* verá el texto cifrado como bits aleatorios y no contendrá información que sea útil.

Las variables  $x$ ,  $y$ ,  $k$  que se aparecen en la figura 3.2 son importantes en criptografía y tienen nombres especiales:

- $x$  es conocido como el texto plano o texto en claro.
- $y$  es conocido como el texto cifrado.
- $k$  es conocido como la llave secreta.
- El conjunto de todas las posibles llaves secretas es llamado el espacio de llaves.

Es importante recordar que el sistema necesita de un canal seguro para la distribución de llaves entre *Alicia* y *Beto*.

La criptografía asimétrica esta basada en la siguiente idea, introducida por Diffie y Hellman en 1976 [25]: no es necesario que la llave utilizada por la entidad que cifra el mensaje (i.e. *Alicia*) sea secreta. La parte crucial es que *Beto*, el receptor, sólo pueda descifrar el mensaje con una llave secreta.

*Beto* publica una llave de cifrado pública que todo el mundo puede conocer. *Beto* también tiene llave secreta correspondiente, que utiliza para el descifrado. Por lo tanto la llave de *Beto* consta de dos partes,  $k_{publica}$  y  $k_{privada}$ .

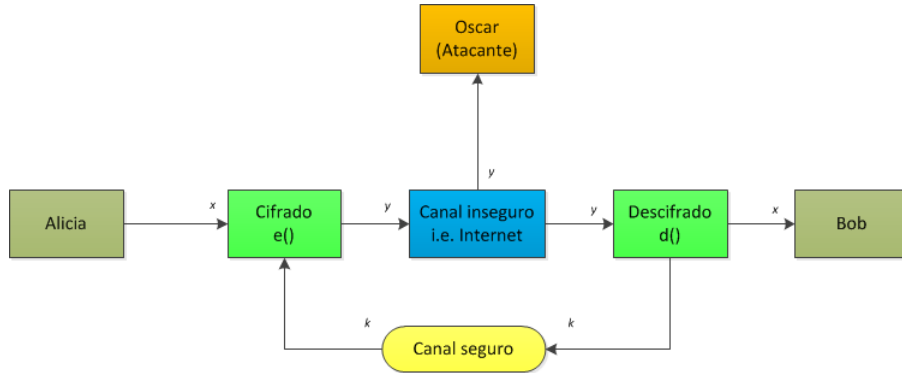


Figura 3.2: Comunicación con cifrado a través de un canal inseguro.

El esquema de comunicación segura bajo un canal hostil utilizando criptografía asimétrica es idéntico al esquema donde se utiliza criptografía simétrica, con la única diferencia es que para la distribución de llaves no se necesita de un canal seguro ya que las llaves utilizadas para el cifrado (i.e. llaves públicas) pueden ser conocidas por cualquier entidad.

Dado que los esquemas de cifrado asimétrico hacen uso de recursos de computo mayores y de que los recursos disponibles en los dispositivos de *hardware* que utilizaremos son limitados, a partir de este momento para propósitos de este trabajo hemos tomado como **única opción** a la **criptografía simétrica** para proveer los servicios de seguridad a los componentes del mecanismo de recolección presentado en este trabajo.

Los principales cifradores simétricos son *DES* y *AES*, los cuales describiremos a continuación.

*DES* ha sido por mucho el cifrador por bloques más popular en los últimos 30 años. Aunque hoy en día no es considerado seguro contra un atacante de fuerza bruta dado que el espacio de llaves en *DES* es pequeño. El proceso conocido como *3DES*, que consiste en cifrar tres veces en fila con el algoritmo *DES*, es considerado como un cifrador muy seguro, el cual es ampliamente utilizado en la actualidad. *DES* utiliza un tamaño de bloque de 64 *bits* para el texto en claro y texto cifrado, mientras que considera una longitud de 56 *bits* para la llave secreta.

*AES* es el algoritmo de cifrado simétrico más empleado en la actualidad, éste fue declarado estándar de *NIST* en Octubre de 2000, siendo el sustituto habitual de algoritmo

*DES*. Está basado en el algoritmo de Rijndael. Hasta la fecha, el mejor ataque conocido contra *AES* es el ataque de fuerza bruta. El tamaño de bloque es de 128 *bits*, mientras que el tamaño de sus posibles llaves son de 128, 192 y 256 *bits*.

Los modos de operación permiten el uso repetido y seguro de un cifrador por bloques con una sola llave. Un sistema de cifrado por bloques por sí mismo sólo permite el cifrado de un bloque de datos. Cuando el mensaje a cifrar es de longitud variable, los datos deben dividirse en bloques de longitud igual al tamaño de los bloques permitidos por el cifrador. Normalmente, el último bloque debe ser ampliado para que coincida con la longitud de cifrado, utilizando un esquema de relleno adecuado. Un modo de operación describe el proceso de cifrar cada uno de estos bloques, y generalmente utiliza un valor de entrada inicial aleatorio, conocido como vector de inicialización (*IV*), para proveer mayor seguridad al cifrado resultante.

Los principales modos de operación para un cifrador por bloques son [43]:

*Electronic Codebook (ECB)*. Este es el modo de operación más simple, el mensaje es dividido en bloques y cada bloque se cifra de forma independiente. Tiene la desventaja de que bloques de texto plano idénticos generan bloques cifrados idénticos, por lo que este modo no oculta los patrones de datos típicos en archivos de imagen, por ejemplo, lo cual no provee confidencialidad de forma efectiva.

*Cipher Feedback (CFB)*. Este modo permite cifrar la información en unidades inferiores al tamaño del bloque y además convierte al cifrador en una unidad de cifrado por flujo de datos, este modo suele utilizarse para cifrar caracteres individuales. Estas características permiten aprovechar totalmente la capacidad de transmisión del canal de comunicaciones, manteniendo además un nivel de seguridad adecuado.

*Output Feedback (OFB)*. Funciona al igual que el modo de operación *CFB*, pero éste utiliza como entradas sus propias salidas, por lo tanto no depende del texto.

*Counter (CTR)*. Al igual que los modos de operación *OFB* y *CFB*, este modo convierte una unidad de cifrado por bloques en una unidad de cifrado por flujo de datos. Genera el siguiente bloque cifrando valores sucesivos de un contador. El contador puede ser cualquier función sencilla que produzca una secuencia de números donde los resultados se

repiten con muy baja frecuencia. Si bien la operación más usada es un contador, el modo *CTR* tiene características similares al *OFB*, pero permite también usar una propiedad de aleatoriedad.

*Galois Counter Mode (GCM)*. Es un modo de operación para cifradores por bloque, el cual utiliza un algoritmo de cifrado que permite proveer los servicios de confidencialidad y autenticación. Para el cifrado utiliza el modo de operación *CTR*.

En el modo de operación *CBC* los bloques cifrados se encuentran encadenados de tal manera que el texto cifrado depende no sólo de bloque de entrada sino también de todos los bloques de texto plano anteriores. En segundo lugar, el cifrado tiene forma aleatoria por la utilización del vector de inicialización (IV) [43]. En el artículo [30] se presenta el diseño de un generador de números aleatorios que pudiese ser utilizado para generar el vector de inicialización.

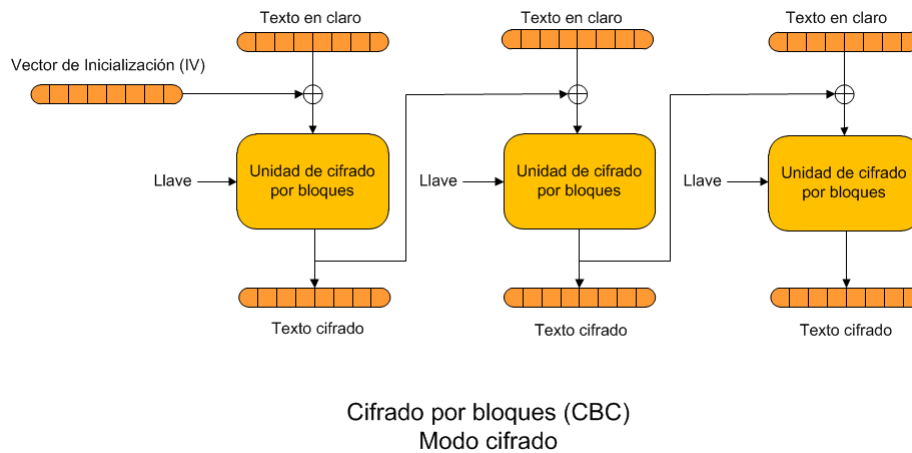


Figura 3.3: Esquema para el cifrado de datos.

El texto cifrado  $y_i$ , que es el resultado del cifrado del texto plano  $x_i$ , se retro-alimenta a la entrada del cifrador y se realiza la operación *XOR* con el bloque de texto plano subsecuente  $x_{i+1}$ . Esta suma *XOR* se cifra, dando el siguiente texto cifrado  $y_{i+1}$ , que se puede utilizar para cifrar  $x_{i+2}$ , y así sucesivamente. Para el caso del primer bloque la operación *XOR* se realiza con el vector de inicialización previamente mencionado, tal y como lo indica la figura 3.3.

El procedimiento detallado en la figura 3.3 es utilizado para proveer el servicio de **confidencialidad**, adicionalmente con el propósito de proporcionar el servicio de **autenticación** haremos uso del esquema conocido como *CBC-MAC*. La autenticación del mensaje es lograda agregando a los datos un bloque extra de información, i.e. conteniendo el número de bloques previos, y cifrando de la misma manera como lo hacemos cuando utilizamos el esquema de cifrado *CBC*.

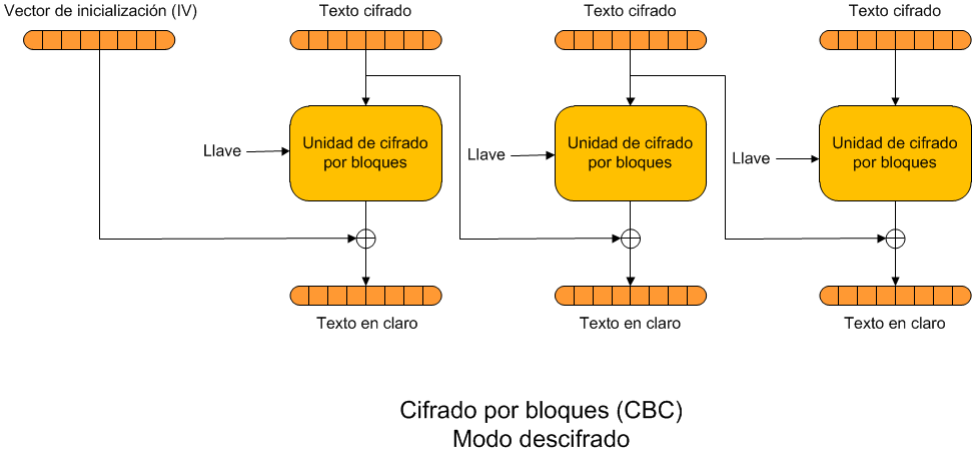


Figura 3.4: Esquema para el descifrado de datos.

El descifrado de los datos en modo *CBC* lo podemos observar en la figura 3.4, básicamente es el mismo principio que para el cifrado de datos, donde el único cambio son las entradas y salidas a las diferentes etapas del esquema de cifrado.

## 3.2 Plataforma elegida

En esta sección se describe la opción elegida para cada categoría definida en la sección 3.1.

### 3.2.1 Red Inalámbrica de Sensores

La red inalámbrica de sensores que utilizaremos en este trabajo fue **adquirida** de la marca *Crossbow*. Elegimos estos componentes por la arquitectura de sus nodos y la gran variedad de sensores disponibles. Otro factor determinante fue la disponibilidad de la puerta

de enlace avanzada, que es una computadora empotrada que permite ejecutar el sistema operativo *GNU/Linux*, lo cual facilita la ejecución de programas avanzados.

Los componentes que utilizaremos en este trabajo se describen a mayor detalle en la presente sección.

Nodos: pertenecientes a la familia *MICAZ* de *Crossbow*, estos nodos están compuestos por un microcontrolador *Atmel ATMEGA128*, un radio digital *Chipcon CC2420*, antena y baterías. Este nodo se muestra en la figura 3.5.



Figura 3.5: Nodo *MICAZ*.

Puertas de enlace: se utilizaron tres modelos diferentes, todos ellos del fabricante *Crossbow*.

*Crossbow MIB520*: permite la programación y conexión de un nodo *MICAZ* con una computadora personal a través del bus serie universal. Esta puerta de enlace se muestra en la figura 3.6.



Figura 3.6: Puerta de enlace *MIB520*.

*Crossbow MIB600*: permite empotrar un nodo *MICAZ*, y permitir que el nodo se comunique a través de un red *Ethernet*. Esta puerta de enlace se muestra en la figura 3.7.



Figura 3.7: Puerta de enlace *MIB600*.

*Crossbow NB100*: es una computadora empotrada que funciona bajo el sistema operativo *GNU/Linux*, cuenta con un procesador *Intel XScale IXP420* a *266 Mhz*, *32MB* de memoria de acceso aleatorio (RAM), memoria *flash* de *8MB*, conectividad *USB 2.0*, red *Ethernet 10/100*, puerto serial. Esta puerta de enlace se muestra en la figura 3.8.



Figura 3.8: Puerta de enlace *NB100*.

Tarjeta de sensores *Crossbow MDA100*: se utilizó esta tarjeta para poder interconectar el modulo de lectura de *RFID* a través del área destinada a prototipos. Esta tarjeta de sensores se puede observar en la figura 3.9.



Figura 3.9: Tarjeta de sensores *MDA100*.

Tarjeta de sensores *Crossbow MTS420CC*: se utilizó esta tarjeta para tener un receptor del sistema de posicionamiento global (*GPS*) y los siguientes sensores: humedad, temperatura, iluminación, aceleración, presión. Está tarjeta de sensores se puede observar en la

figura 3.10



Figura 3.10: Tarjeta de sensores *MTS420CC*.

Sensores adicionales: en este trabajo se presenta el desarrollo de controladores de dispositivo para diferentes sensores, uno de ellos es un modulo de lectura *RFID*, como el que se presenta en la figura 3.11.



Figura 3.11: Modulo de lectura *RFID*.

### 3.2.2 Software empotrado

En este trabajo elegimos como lenguaje de programación para aplicaciones empotradas a *nesC*, dado que permite una programación sencilla y flexible compatible entre una gran variedad de *hardware* para redes inalámbricas de sensores, además de ser un lenguaje muy utilizado para este tipo de tareas. Un punto de gran importancia para ser elegido es la amplia documentación disponible de este lenguaje [46]. Al haber elegido a *nesC* como el lenguaje de programación para esta categoría, elegimos por consecuencia a *TinyOS* como el sistema operativo para los nodos.

Para el caso de la puerta de enlace avanzada, la cual se ejecuta sobre la computadora empotrada *Crossbow NB100* el lenguaje de programación utilizado es el lenguaje C. Dado que la arquitectura de la computadora empotrada es diferente a la arquitectura de la computadora de desarrollo es diferente, debemos de utilizar un compilador cruzado para la arquitectura destino, en este caso *ARM*. En el anexo A, se explica en mayor detalle del proceso de instalación del compilador cruzado bajo el sistema operativo *GNU/Linux*.

### 3.2.3 Software de aplicación

El servidor de almacenamiento y de consulta, se diseñaron sobre una computadora de escritorio con el sistema operativo *GNU/Linux* con el servidor Web *Apache*, base de datos *MySQL* e intérprete de *PHP*. Se realizó esta elección por ser una combinación bastante utilizada hoy en día [45], y que además en su totalidad está basada en *software* libre, con resultados bastante aceptables.

### 3.2.4 Algoritmos de cifrado

Como se mencionó en la sección 3.1.4, la utilización de algoritmos de cifrado asimétricos no será contemplada para dispositivos con restricciones en capacidad de cómputo, por lo tanto la única elección factible es la criptografía simétrica. Dentro de los sistemas de cifrado simétrico, una buena elección es escoger el algoritmo *AES* como sistema de cifrado, además de que el radio digital *CC2420* utilizado en los nodos que anteriormente hemos elegido posee un cifrador *AES* por *hardware*, lo cual permite que el cifrado se realice de forma eficiente. Sin embargo, aun cuando disponible este sistema de cifrado en *hardware*, al momento de la realización de este trabajo no existía la biblioteca o función para hacer uso de este, tal y como se describe en la sección 5.2.1.

El modo de operación *CBC* fue escogido ya que con este podemos cifrar varios bloques de datos de forma encadenada, lo cual permite que un mismo texto en claro resulte en un texto cifrado diferente, además de que podemos obtener una código de autenticación de mensaje al agregar un nuevo bloque al final de los bloques de datos, todo esto ejecutándose sin agregar tiempos significativos de procesamiento.

### **3.2.5 Otros**

Finalmente, como parte de los equipos de pruebas, hicimos uso de un analizador lógico para poder observar las señales digitales generadas por los nodos cuando se comunican con las tarjetas de sensores. Este tipo de herramientas son de gran utilidad ya que nos permite ver el comportamiento de múltiples señales digitales en un intervalo de tiempo. Sin estas herramientas sería imposible ver el comportamiento de dichas señales y por lo tanto el proceso de depuración de los programas sería bastante tedioso.



# Capítulo 4

## Arquitectura

En este capítulo se presenta el análisis y diseño del mecanismo seguro de recolección de datos. En la primera sección, correspondiente al análisis, se describen los requerimientos que el mecanismo de recolección debe cumplir, después en la segunda sección se propone una arquitectura de recolección, se definen el paquete de datos, los tipos de nodos y los tipos de datos, se detalla el procedimiento de reenvío de paquetes y finalmente se describe el diseño de cada uno de los elementos propuestos en la arquitectura de recolección.

### 4.1 Análisis

En esta sección identificamos y describimos los parámetros de funcionamiento deseados para el mecanismo de recolección. Este análisis es de gran importancia ya que nos servirá para poder realizar un diseño adecuado y funcional.

El mecanismo de recolección presentado en este trabajo tiene la finalidad de leer datos provenientes de sensores para su posterior reenvío a otros nodos con el objetivo de alcanzar un servidor de almacenamiento y así permitir que aplicaciones que ocupen procesos de recolección de datos puedan ser fácilmente implementadas por ingenieros y/o científicos que no son expertos en el área de redes inalámbricas de sensores. Trabajos similares a estos se han implementado bajo otras arquitecturas y tecnologías [41] [42], sin embargo no han considerado proporcionar servicios de seguridad para las comunicaciones.

Este trabajo retoma algunas de las ideas presentados en los trabajos similares a este y las extiende para dar forma al mecanismo seguro de recolección de datos con nodos móviles en redes inalámbricas de sensores.

#### 4.1.1 Requerimientos

Los principales requerimientos que deberá cumplir el mecanismo de recolección son los siguientes:

Almacenamiento temporal. La información recolectada por los nodos de lectura deberá ser almacenada de forma temporal en caso de que no haya sido transmitida inmediatamente después de haber sido recolectada. Los nodos deberán tener la capacidad de proveer almacenamiento no volátil, este almacenamiento será de utilidad cuando las baterías se agoten.

Diversidad de sensores. El mecanismo de recolección deberá tener la capacidad de adaptarse a un amplio tipo de escenarios y/o aplicaciones, por lo tanto éste deberá ser capaz de transportar datos recolectados por cualquier tipo de sensor. En caso de que exista algún otro tipo de dato este se deberá poder agregar al mecanismo de recolección.

Escalabilidad. Los nodos podrán entrar o salir en cualquier momento de la red, por lo que unirse o separarse del mecanismo de recolección deberá ser lo más sencillo posible. Por lo tanto, nuestra solución deberá ser capaz de hacer frente a este tipo de características. A fin de permitir la escalabilidad, el mecanismo de recolección deberá incluir un procedimiento de reenvío de paquetes que ayudará a agregar más nodos de sensores a la red.

Movilidad. La mayoría de los nodos deberán tener la facilidad de cambiar su posición en cualquier momento, por lo tanto deberán ser operados a baterías y dependiendo del tipo de nodo, deberán tener un sistema de posicionamiento para registrar la posición donde los datos fueron recabados.

Seguridad. Los datos recolectados y transmitidos deberán viajar de forma segura por la red de nodos inalámbricos, por lo tanto se aplicaran algoritmos criptográficos. Los servicios de seguridad que deberán ser soportados por el mecanismo de recolección son: confidencialidad, autenticidad e integridad.

## 4.2 Diseño

En esta sección se presenta el diseño que se ha realizado en base a los requerimientos, esta es una arquitectura de recolección de datos que está compuesta por: nodos de lectura, nodos de reenvío, nodos de procesamiento, puertas de enlace (simple o avanzada), y servidores de almacenamiento. El esquema que presenta la arquitectura de recolección diseñada se muestra en la figura 4.1.

Adicionalmente a los componentes antes mencionados se contempla un servidor de consulta, es importante mencionar que este servidor de consulta no forma parte del mecanismo de recolección, sin embargo se incluye dentro de la propuesta para ilustrar que el mecanismo de recolección funciona adecuadamente. Este servidor se configura acuerdo a la aplicación final y se pueden agregar los que sean necesarios.

Esta arquitectura de recolección es de tipo jerárquico ya que fue diseñada pensando en que la información de los nodos circule desde los nodos con jerarquía más baja (nodos de lectura) hasta los nodos con jerarquía más alta (nodos de procesamiento) y no en forma contraria.

En las siguientes sub-secciones describimos detalladamente el funcionamiento de cada elemento de la arquitectura.

### 4.2.1 Paquete de datos

Los datos sensados por los nodos de lectura son utilizados para conformar un paquete de datos, como el que se describe en la figura 4.2. La estructura de este paquete de datos, es común para todos los nodos y además es genérica para todo tipo de aplicaciones.

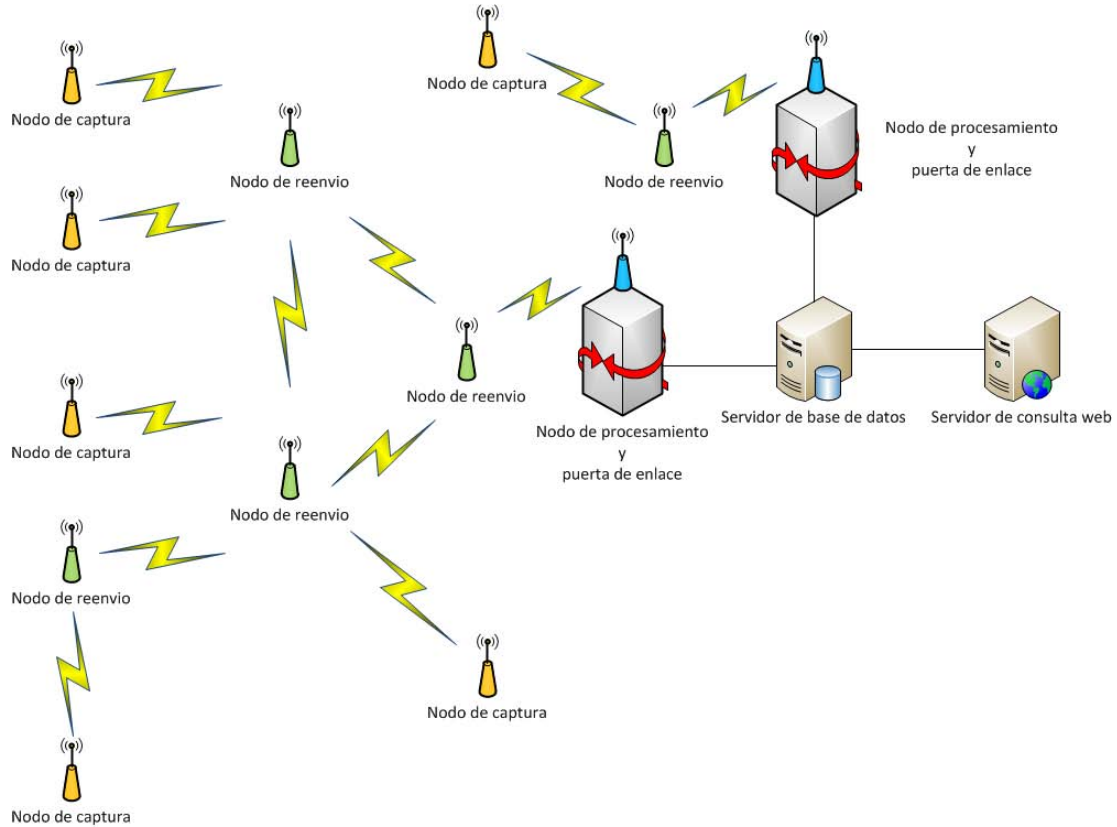


Figura 4.1: Arquitectura del mecanismo de recolección.

La descripción de cada campo se presenta a continuación. El parámetro de *Tiempo\_Vida* especifica el número de saltos restantes o de intentos de reenvío antes de que el paquete sea descartado. En segundo lugar, el parámetro *Tipo\_Nodo* indica el tipo de nodo que origina el paquete de datos. Después, el parámetro *Tipo\_Datos* define el tipo de datos que transporta este paquete de datos. El cuarto lugar, tenemos al parámetro *Identificador\_Nodo\_Fuente* este indica el identificador del nodo que originó el paquete de datos. En quinto lugar está el parámetro *Identificador\_Nodo\_Puerta* el cual especifica el identificador del nodo por donde el paquete se enlaza con otra red. Después, en sexto lugar se define el parámetro conocido como *Identificador\_Mensaje* el cual almacena el identificador único del mensaje que se transporta. En séptimo lugar, tenemos el parámetro *Datos* el cual está destinado a guardar la información que se recolecta. Por último, tenemos al parámetro *Suma\_Verificacion* el cual es un bloque utilizado para verificar la integridad y autenticidad de los datos recolectados. Los campos indicados en **negritas** serán sometidos a algoritmos criptográficos para cifrar su contenido.

```

1. typedef nx_struct message_sensor {
2.     char Tiempo_Vida;
3.     char Tipo_Nodo;
4.     char Tipo_Datos;
5.     char Identificador_Nodo_Fuente;
6.     char Identificador_Nodo_Puerta;
7.     char Identificador_Mensaje[2];
8.     char Datos_A[13];
9.     char Datos_B[13];
10.    char Datos_C[17];
11.    char Suma_Verificacion[16];
10. } message_sensor_t;

```

Figura 4.2: Estructura del paquete de datos.

Los campos que no han sido cifrados viajan de esta forma ya que son necesarios para ejecutar el procedimiento de reenvío de paquetes, si estos campos viajasen cifrados se tendría que descifrar el paquete de datos para poder efectuar el reenvío y actualización del parámetro *Tiempo\_Vida*, lo cual repercutiría en el desempeño del sistema al tener que ejecutar más veces los algoritmos de cifrado y descifrado.

En el paquete de datos se encuentran cifrados un total de 64 *bytes*, formando 4 bloques de 16 *bytes* cada uno, más 2 *bytes* sin cifrar, por lo tanto el tamaño total del paquete de datos es de 66 *bytes*.

## 4.2.2 Tipos de nodos

De acuerdo a la arquitectura de recolección presentada en la figura 4.1 existen diversos tipos de nodos. La tabla 4.1 muestra los tipos de nodos que se han definido, el campo *Tipo* de la tabla 4.1 corresponde con el campo *Tipo\_Nodo* del paquete de datos presentado en la sección 4.2.1.

Tipo	Identificador
NODO_LECTURA	0x01
NODO_REENVIO	0x02
NODO_PROCESAMIENTO	0x03

Tabla 4.1: Tipos de nodos definidos.

### 4.2.3 Tipos de datos

Con el objetivo de cumplir el requerimiento de *diversidad de sensores* presentado en la sección 4.1.1, el mecanismo de recolección tiene la capacidad de manejar diversos tipos de datos. En la tabla 4.2 se muestran los tipos de datos previamente definidos. Los tipos de datos reservados se apartan con el propósito de permitir que el mecanismo pueda ser extendido a más tipos de datos/sensores.

Tipo	Identificador
<i>GPS</i>	0x21
<i>GPS</i> + TEMPERATURA	0x22
<i>GPS</i> + HUMEDAD	0x23
<i>GPS</i> + TEMPERATURA + HUMEDAD	0x24
<i>GPS</i> + <i>RFID</i>	0x25
Reservados	0x26-0x2F

Tabla 4.2: Tipos de datos definidos.

El campo *Tipo* de la tabla 4.2 corresponde con el campo *Tipo\_Datos* del paquete de datos definido en la sección 4.2.1.

### 4.2.4 Reenvío de paquetes

A continuación se describe el procedimiento de reenvío de paquetes, este procedimiento se ha definido con el objetivo de cumplir con el requerimiento de escalabilidad, ya que permite que más nodos puedan agregarse al mecanismo de recolección sin necesidad de ser configurados previamente con una dirección fuente/destino en específico. Esto da la

facilidad de que cualquier tipo de nodo sea energizado y éste comience a capturar y se una al mecanismo de recolección.

Los nodos de jerarquía más baja (nodos de lectura) recaban datos a través de sus sensores, conforman paquetes de datos y los reenvían a otros nodos de jerarquía más alta (nodos de reenvío o nodos de procesamiento). De acuerdo a la arquitectura presentada con anterioridad los nodos de lectura no pueden comunicarse con otros nodos de lectura. Los nodos de reenvío, van almacenando los paquetes recibidos y cuando es posible mandan estos paquetes a otros nodos de reenvío o nodos de procesamiento. Finalmente, los nodos de procesamiento están conectados a una puerta de enlace (simple o avanzada), la cual recibe el paquete de datos y se encarga de procesarlos para su posterior almacenamiento.

Los paquetes de datos circulan desde los nodos de nivel más bajo hasta los nodos de nivel más alto y no de forma contraria. Además todos los nodos poseen mecanismos para desechar paquetes que ya han sido almacenados por mucho tiempo, a través del parámetro de tiempo de vida, el cual especifica el número máximo de saltos o de intentos de reenvío que un paquete puede permitir.

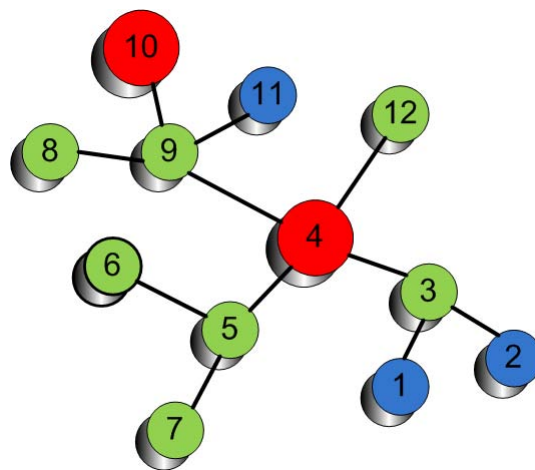


Figura 4.3: Esquema de reenvío de paquetes.

En la figura 4.3 podemos ver los nodos de lectura como los nodos de color azul, a los nodos de reenvío como los de color verde y finalmente a los nodos de procesamiento como los de color rojo.

## 4.2.5 Nodo de lectura

El primer elemento de la arquitectura de recolección presentada en la sección 4.2 es el nodo de lectura. Este dispositivo de lectura se encuentra conformado por un nodo *MICAZ* en conjunto con una tarjeta de sensores *MTS420CC* que se interconectan entre sí a través de los conectores de expansión presentes en ambos componentes, tal y como se ilustra en la figura 4.4. El nodo de lectura y su tarjeta de sensado pueden recabar información de humedad, temperatura, presión, luminosidad, aceleración y posición global.

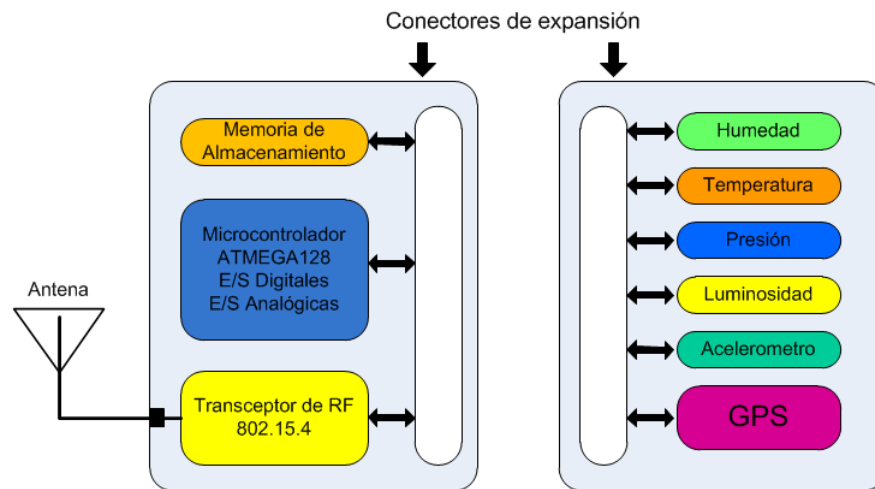


Figura 4.4: Nodo de lectura.

El nodo de lectura se comunica con los sensores a través de interfaces de comunicación seriales (síncronas o asíncronas), tales como *I2C*, *SPI*, *UART*, estas interfaces de comunicación serial se describen a mayor detalle en el Anexo A en la página 92. El primer paso consiste en realizar la inicialización del sensor a través de los comandos destinados para ello, enviar una petición y esperar una respuesta. Cuando se ha recibido la respuesta del sensor se guarda para armar un paquete de datos y aplicar los algoritmos criptográficos.

Este nodo se encarga de realizar lecturas de los sensores cada cierto intervalo de tiempo, conformar con los datos leídos un paquete de datos y finalmente realizar el cifrado de la información. Para realizar la función de cifrado se utiliza el cifrador simétrico *AES* en modo de operación *CBC*. Es importante recordar que la función de cifrado se ejecuta en *hardware*, específicamente en el circuito integrado *CC2420* presente en todos los nodos. Para mayor detalle acerca de los diferentes modos de operación del cifrador *AES* consulte

la sección 3.1.4.

#### **4.2.6 Nodo de reenvío**

El segundo elemento de la arquitectura de recolección presentada en la sección 4.2 es el nodo de reenvío. Este nodo se encarga de recibir paquetes de datos provenientes de nodos de lectura o nodos de reenvío, es decir de nodos de menor jerarquía, ya que ha recibido el paquete de datos lo intenta reenviar a otro nodo de igual o mayor jerarquía, sino tiene nodo alguno a su alcance conserva el paquete de datos para intentar reenviarlo posteriormente, por cada intento de reenvío se descuenta una unidad a la métrica de tiempo de vida. Cuando se ha enviado el paquete de datos, lo elimina de su memoria para liberar el espacio ocupado. Este tipo de nodo no hace uso de las tarjetas de sensores ni de elementos adicionales.

#### **4.2.7 Nodo de procesamiento**

El tercer elemento de la arquitectura de recolección presentada en la sección 4.2 es el nodo de procesamiento. Este nodo se encarga de recibir paquetes de datos vía radio y los reenvía a una puerta de enlace (simple o avanzada) para que los paquetes puedan ser procesados. Debido al poder de cómputo necesario y a la gran cantidad de paquetes que un nodo de procesamiento puede recibir, este nodo no ejecutará ningún tipo de función adicional.

#### **4.2.8 Puerta de enlace simple**

El siguiente elemento de la arquitectura de recolección presentada en la sección 4.2 es la puerta de enlace, ésta puede ser simple o avanzada. En esta sección se presenta el diseño de la puerta de enlace simple.

Esta puerta de enlace se conecta con un nodo de procesamiento para realizar funciones básicas de almacenamiento a los paquetes de datos recibidos. Dado su poder de cómputo limitado, este componente está imposibilitado para realizar funciones avanzadas i.e. funciones criptográficas, por lo que solo se limita a realizar peticiones de almacenamiento

al servidor correspondiente. La figura 4.5 muestra la estructura de una puerta de enlace simple.

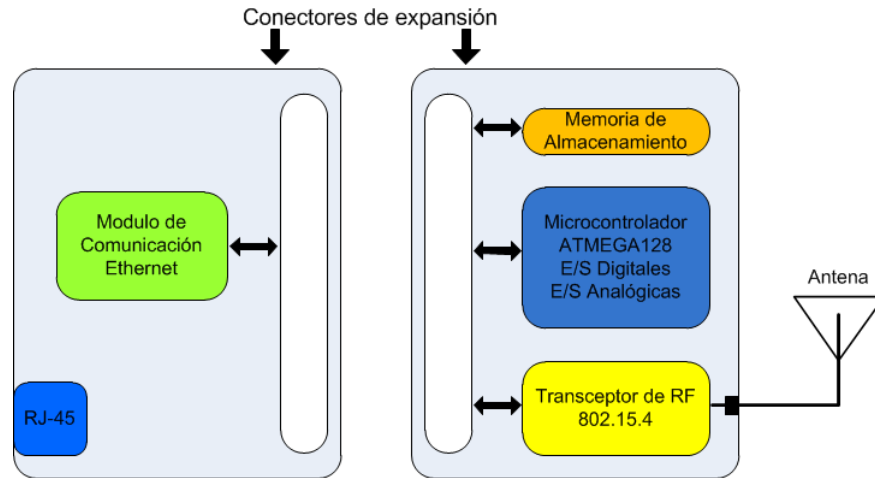


Figura 4.5: Puerta de enlace simple.

Las peticiones de almacenamiento se efectúan por medio del protocolo HTTP, como se muestra en la figura 4.6.

```
http://servidor/almacena.php?bloque1=ww&bloque2=xx&bloque3=yy&bloque4=zz
```

Figura 4.6: Petición de almacenamiento.

La puerta de enlace simple utilizará una puerta de enlace *Crossbow MIB600*, la cual permite empotrar un nodo programado en *nesC* y configurarse para acceder a la red a través de su puerto *Ethernet*.

#### 4.2.9 Puerta de enlace avanzada

En esta sección se presenta el diseño de la puerta de enlace avanzada. Esta puerta de enlace se conecta con un nodo de procesamiento para realizar funciones avanzadas de procesamiento a los paquetes de datos recibidos. Una de sus tarea es la de realizar el

descifrado de datos, tal y como se observa en la figura 3.4, para posteriormente enviar una petición de almacenamiento al servidor correspondiente.

La petición de almacenamiento se efectúa de la misma forma que en la sección anterior. Ver figura 4.6.

Además puede ejecutar otras funciones avanzadas tales como: compresión de datos, almacenamiento y consulta temporal, entre otras.

La puerta de enlace avanzada utilizará una computadora empotrada *Crossbow NB100*, la cual es capaz de ejecutar el sistema operativo *GNU/Linux* y ejecutar programas avanzados escritos en lenguaje C.

La diferencia en *hardware* entre ambas puertas de enlace es significativa, ya que para el caso de la puerta de enlace simple, solo es posible realizar configuraciones para iniciar conexiones a la red de forma automática, mientras que en la puerta de enlace avanzada es posible realizar programas completos en lenguaje C.

#### **4.2.10 Servidor de almacenamiento**

Este servidor tiene el propósito de almacenar los datos enviados por la puerta de enlace, para tales fines contiene un motor de base de datos relacional.

La manera en que recibe los datos desde la puerta de enlace es a través de la ejecución de un programa de página dinámica, el cual recibe los datos a través de la *URL* y del método *GET*.

Al recibir los datos verifica que el mismo mensaje no haya sido insertado anteriormente, esto lo verifica mediante el identificador de mensaje y del código de autenticación de mensaje. Si este mensaje no ha sido guardado, lo inserta en la base de datos o en caso contrario lo descarta.

#### **4.2.11 Servidor de consulta**

Este servidor tiene el propósito de mostrar los datos almacenados posteriormente. Tiene dos formas de consulta, a través de una tabla o a través de un mapa, que nos sirve para indicar donde han sido recolectados los datos.

Como se ha mencionado anteriormente, el servidor de consulta no forma parte del mecanismo de recolección, solamente es utilizado con fines de verificación de resultados.

Este servidor es completamente dependiente de la aplicación final del mecanismo de recolección.

# Capítulo 5

## Desarrollo

Este capítulo trata sobre la implementación del mecanismo de recolección. En la primera sección, correspondiente a implementación, se proporciona una descripción detallada de todos los componentes desarrollados. En la segunda y última sección se presentan las pruebas realizadas al sistema con sus correspondientes resultados. Además se incluye la descripción de los problemas más importantes que se presentaron al realizar este trabajo de tesis. Finalmente, se presenta algunos resultados adicionales como lo son 2 artículos publicados.

### 5.1 Implementación

En esta sección se describen las implementaciones de cada uno de los componentes del mecanismo de recolección. Comenzaremos por los nodos (de lectura, de reenvío, de procesamiento), luego trataremos las puertas de enlace (simple y avanzada) y por último los servidores (de almacenamiento y de consulta).

#### 5.1.1 Nodo de lectura

En esta sección se explica la implementación del primer componente de la arquitectura diseñada en la sección 4.2, componente mejor conocido como nodo de lectura.

Recordemos que este tipo de nodo es una entidad encargada de realizar lecturas por medio de sus sensores integrados, de conformar un paquete de datos y de aplicar algorit-

mos criptográficos para proveer servicios de seguridad. Este nodo es el de jerarquía mas baja dentro del mecanismo de recolección.

El nodo de lectura se ha modelado utilizando una máquina de estados finitos, como la que se muestra en la figura 5.1. El primer estado consiste en realizar la lectura de los sensores, a través de las funciones de lectura diseñadas con tal objetivo, los siguientes cuatro estados realizan el cifrado de los datos en modo *CBC*, tal y como se explico en la sección 4.2.5, finalmente se envía el mensaje a través del radio digital.

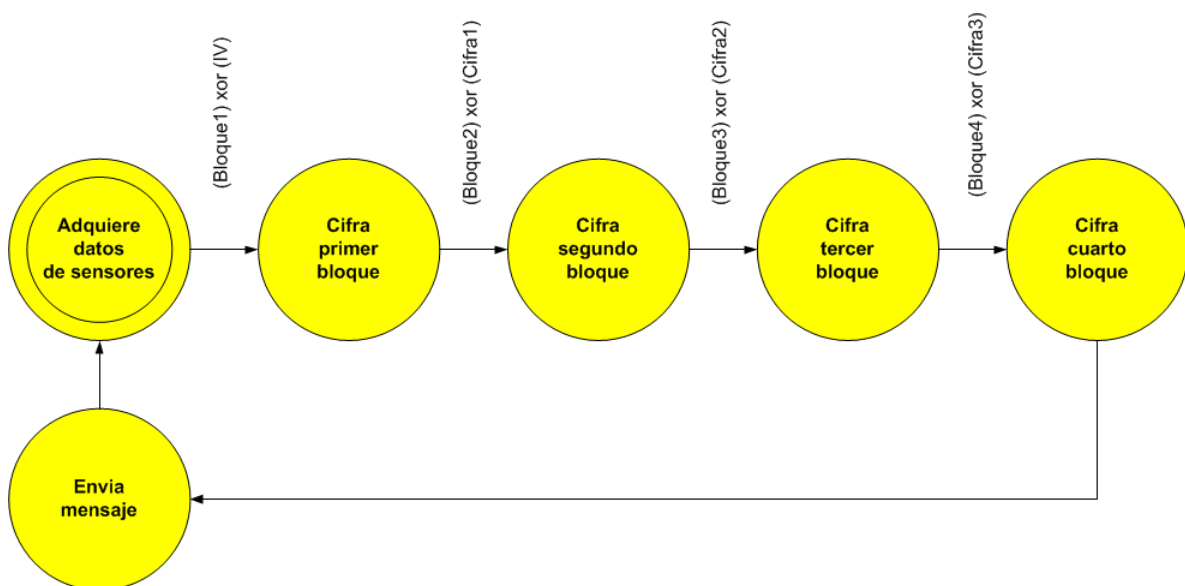


Figura 5.1: Máquina de estados para el nodo de lectura.

En la figura 5.2 observamos la estructura de la función que ejecuta el esquema de cifrado *CBC*. El primer paso es realizar la operación lógica *XOR* a los vectores  $aesTextoPlano_i$  y  $aesTemporal$ , el resultado es utilizado para cifrar el vector  $aesTextoPlano_i$ , después el vector  $aesTemporal$  adquiere el valor del vector  $aesTextoCifrado_i$ , después de esto la secuencia se repite. Para el bloque 1, el vector  $aesTemporal$  se carga con el vector  $aesIV$ .

En la figura 5.3 observamos la estructura de la máquina de estados anteriormente propuesta. El primer estado consiste en la lectura de los sensores, mientras los cuatro estados siguientes son utilizados para generar la secuencia usada en el modo de operación *CBC* para el cifrado *AES*, en el último estado el mensaje es enviado a través del radio digital.

```

1. aesTemporal ← aesIV;
2. ejecuta{
3.   aesResultado ← xor(aesTextoPlanoi, aesTemporal);
4.   resultado ← cifrar(aesResultado, aesTextoPlanoi, aesTextoCifradoi);
5.   aesTemporal ← aesTextoCifradoi;
6. } mientras(resultado es diferente de CIFRADO_SATISFACTORIO);

```

Figura 5.2: Implementación del cifrado CBC.

Al finalizar la acción correspondiente a cada estado, es establecida la variable *estadoSiguiente* con el identificador del siguiente estado a ejecutar.

```

1. si(estado es igual a LEE_SENSORES){
2.   leeSensores(); estadoSiguiente ← CIFRA_BLOQUE1;
3. }sino si(estado es igual a CIFRA_BLOQUE1){
4.   cifraBloque(1); estadoSiguiente ← CIFRA_BLOQUE2;
5. }sino si(estado es igual a CIFRA_BLOQUE2){
6.   cifraBloque(2); estadoSiguiente ← CIFRA_BLOQUE3;
7. }sino si(estado es igual a CIFRA_BLOQUE3){
8.   cifraBloque(3); estadoSiguiente ← CIFRA_BLOQUE4;
9. }sino si(estado es igual a CIFRA_BLOQUE4){
10.  cifraBloque(4); estadoSiguiente ← ENVIA_MENSAJE;
11. }sino si(estado es igual a ENVIA_MENSAJE){
12.  enviaMensaje(); estadoSiguiente ← LEE_SENSORES;
13. }

```

Figura 5.3: Estructura de la máquina de estados para el nodo de lectura.

## Controladores de dispositivo

A continuación se describe el proceso de escritura de nuevos controladores de dispositivo para los nodos *MICAZ*. Primero, se describe la arquitectura de un nodo *MICAZ*, luego la forma en la que se interconectan los sensores al nodo, después se discute de forma general la utilización de cualquier sensor y finalmente se presenta el diseño de controladores de dispositivo para diferentes tres tipos de sensores.

La arquitectura de un nodo *MICAZ* se presenta en la figura 5.4. El componente principal del nodo es un microcontrolador *Atmel ATMEGA128*, este es un procesador de 8 *bits*, posee arquitectura *RISC*, y es capaz de ejecutar una instrucción por ciclo de reloj. Tiene 128KB de memoria *flash* (memoria de programa), tiene una memoria *RAM* de 4KB y una memoria *EEPROM* de 4KB. Este procesador tiene entradas y salidas digitales, un convertidor analógico - digital de 10 *bits* con 8 canales, además tiene interfaces de comunicación seriales como *I2C*, *SPI* y *UART*. El segundo componente más importante de este nodo es el radio digital *CC2420*, el cual es utilizado para las comunicaciones inalámbricas del nodo. Es interconectado con el nodo a través una interfaz de comunicación serial síncrona como *SPI*. Por último tenemos una memoria de almacenamiento no volátil de 512 KB.

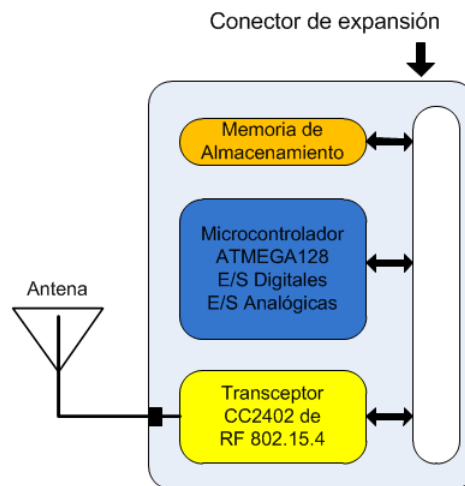


Figura 5.4: Arquitectura del nodo *MICAZ*

Uno de los factores más importantes al interconectar un sensor a un microprocesador es la interfaz mediante la cual se comunicaran, hoy en día existen una gran variedad, ya sea de formato serial o paralelo, síncrono o asíncrono, estándares o propietarias, entre otras. Dado que la interconexión juega un papel fundamental para el desarrollo de controladores de dispositivo en la sección A.4 se describen de manera breve las principales interfaces de comunicación presentes en un nodo *MICAZ* útiles para efectuar un enlace de datos con una unidad de sensado.

### ***Tarjeta de sensado Crossbow MTS420CC.***

La tarjeta *MTS420CC* de *Crossbow*, es una tarjeta de sensado que contiene:

- Receptor *GPS* marca *UBLOX* mod. *LEA-4A* [11].
- Sensor de humedad y temperatura marca *Sensirion* mod. *SHT11* [12].
- Sensor de presión barométrica y temperatura marca *Intersema* mod. *MS5534C* [17].
- Sensor de luz marca *TAOS* mod. *TLS2550* [18].
- Acelerómetro de 2 ejes marca *Analog Devices* mod. *ADXL202JE* [13].

Debido a la gran diversidad de sensores presentes, el consumo de potencia de esta tarjeta puede ser bastante elevado, por lo tanto cuenta con 2 interruptores electrónicos de 8 canales cada uno. Los interruptores electrónicos son de la marca *Analog Devices* y el modelo es *ADG715* [14]. Estos dispositivos se interconectan al nodo mediante la interfaz *I2C*. Los canales de los interruptores electrónico se detallan en la tabla 5.1.

Canal	Señal (U7)	Canal	Señal (U9)
1	<i>LIGHT_POWER</i>	1	<i>GPS_RX</i>
2	-	2	<i>GPS_TX</i>
3	<i>PRESSURE_POWER</i>	3	<i>PRESSURE_SCLK</i>
4	<i>HUMIDITY_POWER</i>	4	<i>PRESSURE_DIN</i>
5	<i>EEPROM_POWER</i>	5	<i>PRESSURE_DOUT</i>
6	<i>ACCEL_POWER</i>	6	-
7	<i>GPS_POWER</i>	7	<i>HUMIDITY_SCK</i>
8	<i>GPS_ENABLE</i>	8	<i>HUMIDITY_DATA</i>

Tabla 5.1: Señales del interruptor electrónico *U7* y *U9*

Los interruptores electrónicos están denotados por *U7* el cual posee la dirección **0x48** y por *U9* el cual posee la dirección **0x49**. Para activar el modulo *GPS* es necesario seguir la siguiente rutina de inicialización, primero hay que activar los canales *GPS\_ENABLE* y *GPS\_POWER* enviando el dato **0x80** a la dirección del circuito integrado *U7*, después hay que activar los canales *GPS\_RX* y *GPS\_TX* enviando el dato **0x03** a la dirección del

circuito integrado *U9* tal y como se muestra en la figura 5.5, la cual contiene el segmento de código en *nesC* que inicializa el módulo *GPS*.

```
1. if (estado == ENERGIA_GPS) {
2.     Energia[0] = 0x80;
3.     if (call I2C.write(START|STOP,0x48,1,Energia)==SUCCESS){
4.         atomic {
5.             estado = DATOS_GPS;
6.         }
7.     }
8. } else if (estado == DATOS_GPS) {
9.     Datos[0] = 0x03;
10.    if (call I2C.write(START|STOP,0x49,1,Datos)==SUCCESS){
11.        atomic {
12.            estado = FIN_GPS;
13.        }
14.    }
15. } else if (estado == FIN_GPS) {
16.    atomic {
17.        estado = FIN_INICIALIZACION;
18.    }
19.    wait();
20.    call MilliTimer.startPeriodic(1);
21. }
```

Figura 5.5: Segmento de código en *nesC* que inicializa el modulo *GPS*

En la figura 5.6 se muestra la captura de las señales *SDA* y *SCL* realizada con un analizador lógico, podemos ver que la ejecución del segmento de código anterior es válida, ya que después de cada petición recibimos la señal de reconocimiento (*ACK*) por parte del periférico.

El siguiente paso para la realización del controlador es interpretar la información recibida el modulo *GPS*, de acuerdo a las especificaciones del fabricante este dispositivo de posicionamiento global genera sus respuestas en base al estándar *NMEA 0183*, entre otros. Este estándar es una combinación de una especificación a nivel eléctrico con una especificación para el intercambio de datos entre dispositivos electrónicos, en especial marítimos. Los datos son transmitidos desde un emisor a múltiples receptores por medio protocolo de comunicación serial basado texto en *ASCII*. En la capa de aplicación el estándar también

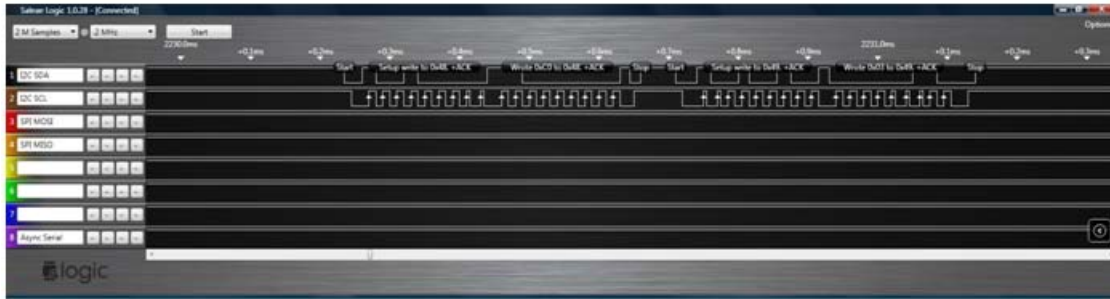


Figura 5.6: Inicialización del modulo *GPS* vista en un analizador lógico.

define el contenido de cada tipo de mensaje para que los receptores puedan interpretarlo correctamente. Un ejemplo de los mensajes generados en formato *NMEA 0183* por este receptor *GPS* se pueden observar en la figura 5.7.

```

$GPGSV,2,1,07,23,02,284,,22,32,126,41,14,26,054,37,03,01,197,*78
$GPGSV,2,2,07,26,05,074,25,16,34,164,47,31,62,006,37*4F
$GPGLL,1932.98922,N,09913.25898,W,033153.00,A,A*71
$GPZDA,033153.00,09,01,2010,00,00*6A
$GPRMC,033154.00,A,1932.98922,N,09913.25901,W,0.009,,090110,,A*6E
$GPVTG,,T,,M,0.009,N,0.016,K,A*2D
$GPGGA,033154.00,1932.98922,N,09913.25901,W,1,04,4.04,2293.0,,,,*68
$GPGSA,A,3,22,14,16,31,,,,,,7.97,4.04,6.87*02

```

Figura 5.7: Ejemplo de mensajes generados por el *GPS* en formato *NMEA 0183*

A continuación, en la figura 5.8 observamos el código fuente en *nesC* que es capaz de reconocer el mensaje de tipo *\$GPGLL*.

El segmento de código mostrado con anterioridad reconoce el mensaje de de tipo *\$GPGLL*, donde este mismo esquema se puede aplicar para todos los mensajes enviados por el modulo de posicionamiento global. En las primeras tres líneas, reconoce que se trata de un mensaje con terminación *GLL*, a partir de este punto comienza a leer los datos recibidos y los almacena dentro de las localidades de memoria destinada para el dato de *Latitud*, esto ocurre hasta que recibe un delimitador de campo, en esta ocasión representado por el símbolo *coma*. Después se adquiere el dato correspondiente a *Longitud* (no mostrado

```

1. if (GPS_Message[GPS_Counter+2] == 'G'){
2.     if (GPS_Message[GPS_Counter+3] == 'L'){
3.         if (GPS_Message[GPS_Counter+4] == 'L'){
4.             GPS_Counter = GPS_Counter + 6;
5.             GPS_tLatitud_counter = 1;
6.             while (GPS_Message[GPS_Counter] != ','){
7.                 GPS_tLatitud[GPS_tLatitud_counter] =
8.                 GPS_Message[GPS_Counter];
9.                 GPS_Counter = GPS_Counter + 1;
10.                GPS_tLatitud_counter =
11.                GPS_tLatitud_counter + 1;
12.            }
13.            GPS_Counter = GPS_Counter + 1;
14.            if (GPS_Message[GPS_Counter] == 'N') {
15.                GPS_tLatitud[0] = '+';
16.            } else if (GPS_Message[GPS_Counter] == 'S'){
17.                GPS_tLatitud[0] = '-';
18.            }
19.            while (GPS_Message[GPS_Counter] != ','){
20.                GPS_Counter = GPS_Counter + 1;
21.            }
22.        }
23.    }
24. }

```

Figura 5.8: Segmento de código en *nesC* que interpreta el mensaje *\$GPGLL* proveniente del modulo *GPS*

en el código anterior). El resto de los campos y mensajes se pueden procesar de la misma manera en que se ha procesado el presente mensaje.

A continuación se describe la implementación realizada para poder utilizar el sensor de humedad *Sensirion* mod. *SHT11*. Este sensor está presente en la tarjeta *Crossbow MTS420CC* y se conecta al nodo *MICAZ* a través de una interfaz serial síncrona, la cual no es ninguna de las descritas en la sección A.4. En este caso tenemos una interfaz propietaria, que sigue un protocolo de comunicación mostrado en la figura 5.9.

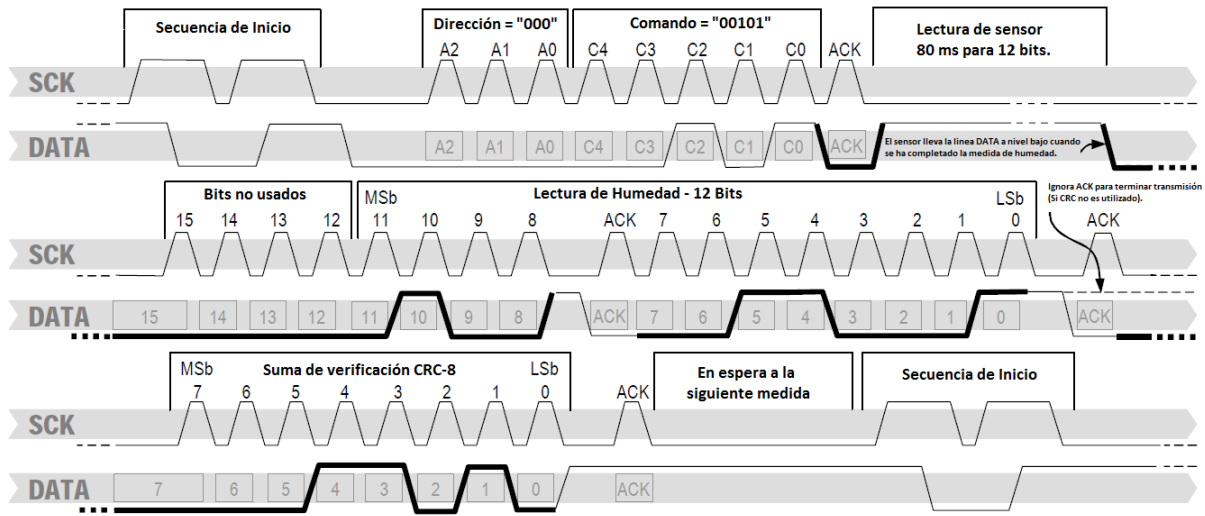


Figura 5.9: Protocolo de comunicación para el sensor de humedad. Imagen original [12].

El primer paso para iniciar comunicación este sensor es generar una secuencia de inicio lo cual se logra, llevando la señal *CLK* a alto, luego asignando la señal *DATA* a bajo, enseguida completando el ciclo de reloj e iniciando uno nuevo, después llevando la señal *DATA* con valor alto y finalmente terminando el segundo ciclo de reloj.

El siguiente paso, es enviar la dirección y el comando al sensor. La dirección consta de 3 bits y en este caso corresponden a "000", por lo cual tendremos que generar tres ciclos de reloj y en cada uno de ellos enviar *bit a bit* la dirección necesario, dado que los 3 bits son igual a "0", la señal *DATA* no sufrirá cambios durante el envío de los tres ciclos de reloj.

Para enviar el comando se hace de la misma forma que con el envío de la dirección, sin embargo como aquí si tenemos bits iguales a "1", en el correspondiente ciclo de reloj debemos llevar la señal *DATA* al valor de "1z regresarlo al valor de "0" al finalizar el ciclo de reloj. Al siguiente ciclo de reloj recibiremos una secuencia de *ACK*, para avisarnos que el sensor ha recibido la dirección y el comando, esta secuencia es lograda por el sensor llevando la señal *DATA* a valor bajo durante un ciclo de reloj.

Después de haber completado la secuencia descrita en el párrafo anterior debemos esperar a que el sensor realice la medida, que para 12 bits el tiempo de espera es de aprox. 80 *ms*. Mientras el sensor se encuentra tomando la medida de humedad, lleva la señal DATA a valor alto, enseguida de que ha terminado de tomar la medida de humedad, el sensor lleva la señal DATA a valor bajo. Para realizar la lectura, se deben enviar 16 señales de reloj, donde las cuatro primeras no son utilizadas y las 12 restantes se ocupan para leer el dato de humedad en cada ciclo de reloj. Por cada byte recibido de datos se debe generar la secuencia de *ACK*. El siguiente paso, el cual es opcional, consiste en leer del sensor la suma de verificación del valor leído anteriormente, con el fin de verificar que existan errores en la transmisión de los datos.

En la figura 5.10 se muestra la rutina *SecuenciaInicio* la cual es utilizada para informar al sensor que debe iniciar una nueva lectura. Después, en la figura 5.11, se muestra la rutina *EnviaDireccion* utilizada para enviar la dirección al sensor, la implementación de la rutina *EnviaComando* no se muestra ya que es muy similar a la rutina *EnviaDireccion*.

```

1. void SecuenciaInicio () {
2.     call HumiditySCK.set();           espera();
3.     call HumidityDATA.clr();         espera();
4.     call HumiditySCK.clr();          espera();
5.     call HumiditySCK.set();           espera();
6.     call HumidityDATA.makeInput();   espera();
7.     call HumiditySCK.clr();          espera();
8.     call HumidityDATA.makeOutput();
9.     call HumidityDATA.clr(); }

```

Figura 5.10: Implementación de la rutina *SecuenciaInicio* para el sensor SHT11.

```

1. void EnviaDireccion() {
2.     call HumiditySCK.set();           espera();
3.     call HumiditySCK.clr();          espera();
4.     call HumiditySCK.set();           espera();
5.     call HumiditySCK.clr();          espera();
6.     call HumiditySCK.set();           espera();
7.     call HumiditySCK.clr();          espera(); }

```

Figura 5.11: Implementación de la rutina *EnviaDireccion* para el sensor SHT11.

En la figura 5.12 se muestra la rutina *leeBit*, la cual es utilizada para leer un bit datos, ésta rutina es principalmente utilizada por la rutina *LeeSensor*, mostrada en la figura 5.13, para leer el dato de humedad generado por el sensor y para la rutina *LeeCRC* (no mostrada), que lee el código *CRC-8* que envía el sensor.

```
1. uint8_t leeBit(){
2.     uint8_t Bit;
3.     call HumiditySCK.clr();           espera();
4.     Bit = call HumidityDATA.get();   espera();
5.     call HumiditySCK.set();          espera();
6.     call HumiditySCK.clr();
7.     return Bit; }
```

Figura 5.12: Implementación de la rutina *leeBit* para el sensor SHT11.

```
1. void LeeSensor(){
2.     leeBit(); leeBit(); leeBit(); leeBit();
3.     bit = leeBit() * 2048; mHumedad += bit;
4.     bit = leeBit() * 1024; mHumedad += bit;
5.     bit = leeBit() * 512; mHumedad += bit;
6.     bit = leeBit() * 256; mHumedad += bit;
7.     wait(); sendACK(); wait();
8.     bit = leeBit() * 128; mHumedad += bit;
9.     bit = leeBit() * 64; mHumedad += bit;
10.    bit = leeBit() * 32; mHumedad += bit;
11.    bit = leeBit() * 16; mHumedad += bit;
12.    bit = leeBit() * 8; mHumedad += bit;
13.    bit = leeBit() * 4; mHumedad += bit;
14.    bit = leeBit() * 2; mHumedad += bit;
15.    bit = leeBit() * 1; mHumedad += bit; }
```

Figura 5.13: Implementación de la rutina *LeeSensor* para el sensor SHT11.

## ***Lector de RFID***

En esta sección se describe la incorporación de un lector de radio frecuencia a un nodo *MICAZ* para generar un nodo de lectura con capacidad de adquirir datos vía *RFID*.

Se utilizó un lector de baja frecuencia de la marca *Texas Instruments* modelo *MRD1*. Este componente lee los identificadores de *RFID* que se encuentran cercanos al área de la antena y envía los datos adquiridos por un enlace serial (configurado como *8N1@9600bps*) al nodo responsable de controlar los dispositivos sensores. Los identificadores (o también conocidos como etiquetas) son reconocidos por el lector de *RFID* al ser aproximados al área de lectura, estos identificadores son de tipo pasivo y tienen un número único de 64 bits.

La conexión se realizó mediante el puerto serie del nodo *MICAZ* y el puerto serie presente en el lector de *RFID*. Un punto importante a destacar, es que el nodo *MICAZ* maneja un voltaje de alimentación de *3V*, mientras que el lector de *RFID* maneja un voltaje de alimentación de *5V*, por lo tanto para que los componentes se puedan interconectar es necesario hacer una conversión de voltajes entre las señales que manejan estos componentes.

El lector de *RFID* debe recibir como señal de entrada, una señal codificada serialmente con nivel de voltaje de *5V*, pero la señal de *TX* del nodo *MICAZ* es de *3V*, sin embargo la conexión se puede efectuar ya que el valor de *3V* es reconocido como un valor lógico de estado alto ("1") por el lector de *RFID*. La señal que hay que adaptar es la transmisión (*TX*) del modulo de lectura *RFID*, o también se puede ver como la señal de recepción (*RX*) del nodo *MICAZ*. El acoplamiento entre dispositivos se realizó mediante un circuito resistivo, mostrado en la figura 5.14, en su configuración *divisor de voltaje*, este circuito recibe a su entrada *5V* y a la salida produce *3V*.

La conexión entre el nodo *MICAZ* y el modulo lector de radio frecuencia se realizó a través de una tarjeta de sensores *MDA100*, como la mostrada en la figura 3.9. Esta tarjeta de sensores tiene la distribución de terminales mostrada en la tabla 5.2

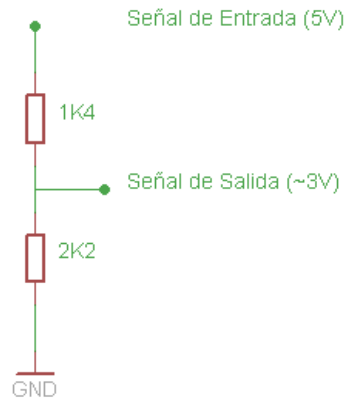


Figura 5.14: Circuito utilizado para adaptar los niveles de voltaje.

	A	B	C	D	E	F
1	GND	GND	GND	VCC	VCC	VCC
2	OPEN	OPEN	USART1_CK	INT3	ADC2	PW0
3	OPEN	OPEN	UART0_RX	INT2*	ADC1*	PW1*
4	OPEN	OPEN	UART0_TX	INT1	ADC0	PW2
5	OPEN	OPEN	SPL_SCK	INT0	THERM_PWR	PW3
6	OPEN	OPEN	USART1_RX	BAT_MON	THRU1	PW4
7	OPEN	OPEN	USART1_TX	LED3	THRU2	PW5
8	OPEN	OPEN	I2C_CLK	LED2	RSTN	PW6
9	OPEN	OPEN	I2C_DATA	LED1	RSTN	ADC7
10	OPEN	OPEN	PWM0	RD	PWM1B	ADC6
11	OPEN	OPEN	PWM1A	WR	OPEN	ADC5
12	OPEN	OPEN	AC+	ALE	OPEN	ADC4
13	OPEN	OPEN	AC-	PW7	OPEN	ADC3
14	GND	GND	GND	VCC	VCC	VCC
15	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
16	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
17	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN

Tabla 5.2: Distribución de terminales en la tarjeta de sensores MDA100.

La comunicación con el lector de radio frecuencia se basa en el principio de petición - respuesta, es decir para obtener datos desde el equipo lector, el nodo debe enviar un comando de petición, esperar un determinado tiempo y leer la respuesta enviada por el modulo de lectura. La tabla 5.3 muestra la estructura del comando de lectura que el nodo debe generar.

Byte	Valor	Comentario	Descripción
0	0x01	Byte de inicio	*
1	0x02	Longitud	Dos bytes siguientes excepto el último.
2	0x08	Comando(1)	Ejecuta un comando y un periodo de carga.
3	0x32	Datos(1)	Duración del periodo de carga I: 50 ms.
4	0x38	Suma de verificación	<i>XOR</i> a los bytes anteriores excepto el primero.

Tabla 5.3: Petición para leer un identificador [19].

A continuación, la tabla 5.4 muestra la estructura del comando devuelto por el modulo lector cuando ha procesado una petición de lectura y ha encontrado un identificador cercano al área de lectura.

Byte	Valor	Comentario	Descripción
0	0x01	Byte de inicio	*
1	0x09	Longitud	9 bytes siguiente excepto el último.
2	0x0C	Estado	Identificador solo lectura valido.
3	0x64	Datos(1)	Datos de identificación ( <i>LSB</i> ).
4	0x58	Datos(2)	Datos de identificación.
5	0x4C	...	...
6	0x00	...	...
7	0x00	...	...
8	0x00	...	...
9	0x00	Datos(7)	Datos de identificación.
10	0x00	Datos(8)	Datos de identificación ( <i>MSB</i> ).
11	0x7B	Suma de verificación	<i>XOR</i> a los bytes anteriores excepto el primero.

Tabla 5.4: Respuesta del modulo de lectura cuando se ha encontrado un identificador [19].

Por último, la tabla 5.5 muestra la estructura del comando devuelto por el modulo lector cuando ha procesado una petición de lectura y no se ha encontrado un identificador cercano al área de lectura.

Byte	Valor	Comentario	Descripción
0	0x01	Byte de inicio	*
1	0x01	Longitud	Un byte después excepto el último.
2	0x03	Estado	Otro, no hay byte de inicio.
3	0x02	Suma de verificación	<i>XOR</i> a los bytes anteriores excepto el primero.

Tabla 5.5: Respuesta del modulo de lectura cuando no se ha encontrado un identificador [19].

La figura 5.15 muestra la implementación del comando mostrado en la tabla 5.3.

```

1. event void Temporizador.fired() {
2.     char peticionLectura[5]={0x01,0x02,0x08,0x32,0x38};
3.     call UartStream.send(peticionLectura,5);
4. }

```

Figura 5.15: Implementación de la petición de lectura.

### 5.1.2 Nodo de reenvío

El segundo componente implementado de la arquitectura diseñada y mostrada anteriormente es el nodo de reenvío. El propósito de este tipo de nodos es de recibir paquetes de datos desde nodos de jerarquía más baja o igual, verificar que estos paquetes aun sean válidos, y reenviarlos a nodos de jerarquía igual o mayor.

La implementación del nodo de reenvío se basa principalmente en el evento de recepción de mensaje, al ocurrir este evento, se recibe el mensaje, se checan los campos de tipo de mensaje y tiempo de vida del mensaje y se reenvía el mensaje si procede, en caso contrario el mensaje se descarta.

El pseudocódigo mostrado en la figura 5.16 implementa la funcionalidad antes mencionada.

```
1. evento mensaje_t* recibe (mensaje_t* Temp, void* Datos, entero Tamano) {
2.
3.     mensaje MensajeRecibido ← Datos;
4.
5.     TipoNodo ← MensajeRecibido.Tipo_Nodo;
6.     TiempoVida ← MensajeRecibido.Tiempo_Vida;
7.
8.     si (TiempoVida - 1 es mayor que 0) {
9.         si(TipoNodo es igual a NODO_LECTOR o
10.        TipoNodo es igual a NODO_REENVIO) {
10.            mensaje MensajeEnviar;
11.            MensajeEnviar ← MensajeRecibido;
12.            Mensaje ← LISTO;
13.        }
14.    }
15.    regresa Temporal;
16. }
```

Figura 5.16: Implementación del evento recibe.

Además se hace uso de un temporizador para enviar el mensaje previamente preparado para reenvío, tal y como se muestra en la figura 5.17. Cuando el mensaje ha sido correctamente procesado por el evento de mensaje recibido se genera una señal para que al completar un temporizador éste sea enviado.

Al entrar a la rutina que procesa el evento completado del temporizador se intenta enviar los mensajes que han sido almacenados, después en caso de tener mensajes listos para ser enviados se procede a ejecutar la rutina que realiza el envío del mensaje, en caso de que un mensaje no se haya podido enviar satisfactoriamente, éste se almacena para su posterior difusión.

```

1. evento temporizador.completado() {
2.     si (Mensaje es igual a LISTO) {
3.         enviaMensajesPendientes();
4.         si (Radio.Envia() es igual a ENVIADO) {
4.             Enviado ← VERDADERO;
5.         } sino {
4.             Enviado ← PENDIENTE;
7.             almacenaMensaje();
8.         }
6.     }
7. }

```

Figura 5.17: Implementación del evento completado para el temporizador.

### 5.1.3 Nodo de procesamiento

El tercer componente implementado de la arquitectura de recolección de datos es el nodo de procesamiento. El propósito de este tipo de nodos es de recibir paquetes de datos provenientes de nodos de reenvío. Al recibir estos paquetes desde los nodos de reenvío, los direcciona inmediatamente a una puerta de enlace (simple o avanzada). Dependiendo del tipo de puerta de enlace es el procesamiento que se realiza.

Al igual que en el nodo de reenvío, presentado en la sección anterior, este componente basa su implementación en el evento de recepción de mensaje. Al activarse este evento, se recibe el mensaje y se envía a la puerta de enlace avanzada en formato de cadena de texto, con los bloques separados por comas.

La estructura del evento **recibe** se muestra en la figura 5.18. Al entrar al evento generado por la recepción de un mensaje, lo primero que se hace es recuperar el mensaje recibido, después éste se divide en 4 bloques de 16 *bytes* de longitud por bloque. Después, con los bloques recibidos se genera una petición que se enviará a través del puerto serial del nodo de procesamiento a una puerta de enlace.

```

1. evento mensaje_t* recibe (mensaje_t* Temp, void* Datos, entero_t Tam) {
2.
3.     mensajeMensajeRecibido ← Datos;
4.
5.     copia_memoria(Bloque1, MensajeRecibido, 16);
6.     copia_memoria(Bloque2, MensajeRecibido, 16);
7.     copia_memoria(Bloque3, MensajeRecibido, 16);
8.     copia_memoria(Bloque4, MensajeRecibido, 16);
9.
10.    generaPeticion(Datos);
11.    UART.envia(Datos, tamano(Datos));
12.    regresa Temp;
13. }

```

Figura 5.18: Implementación del evento recibe para el nodo de procesamiento.

#### 5.1.4 Puerta de enlace simple

El cuarto componente implementado de la arquitectura de recolección de datos es la puerta de enlace simple. El propósito de esta puerta es de procesar paquetes en conjunto con el nodo de procesamiento para su posterior almacenamiento en un servidor remoto a través de una petición basada en el protocolo HTTP.

Dado que las capacidades de cómputo de este tipo de dispositivos son limitadas, la puerta de enlace simple solamente podrá recibir paquetes y formular la petición del almacenamiento al servidor remoto, es decir no tiene la capacidad para hacer un procesamiento del paquete de datos recibido. Además hay que recordar que la puerta de enlace simple junto con el nodo de procesamiento es el punto de acceso para paquetes provenientes de los nodos de jerarquía más baja, por lo que deseamos que esté disponible para procesar paquetes la mayor parte del tiempo.

Para la implementación de este componente se utilizó la puerta de enlace *Ethernet*, de la marca *Crossbow*, modelo *MIB600*, la cual está compuesta por un módulo *Ethernet* modelo *Micro* de la marca *Lantronix* [16]. Este módulo *Micro*, tiene un servidor y dos clientes empotrados, uno para cada puerto serial, el cual se puede configurar como *RS232*, *RS422* u *RS485*. El puerto serial 1 del módulo *Micro* se encuentra directamente conectado a la

interfaz *UART* del nodo *MICAZ*, mientras que el puerto serial 2 se encuentra conectado al programador de nodos, lo cual permite que la reprogramación se haga de forma remota.

Para ésta puerta de enlace utilizamos el puerto serial 1, el que se encuentra directamente conectado a la *UART* del nodo *MICAZ*. El servidor remoto utilizado es `http://www.embedded.com.mx`, al que le corresponde la dirección IP `216.108.239.146` en el puerto 80. Se utilizó el modo *autostart* para iniciar automáticamente una conexión de datos. El envío de la petición remota, ya que se ha iniciado la conexión, se hace a través de un programa en lenguaje *PHP*, utilizando el protocolo HTTP como medio de transporte. El formato de la petición que se envía cuando nos hemos conectado se muestra en la figura 5.19.

```
GET
http://www.embedded.com.mx/almacenaSimple.php?
bloque1=&bloque2=&bloque3=&bloque4=
HTTP/1.0
Accept: */*
Accept: text/html
```

Figura 5.19: Petición de almacenamiento para puerta de enlace simple.

La instrucción que se muestra en la figura 5.20, realiza el envío de la petición en el nodo hacia el servidor remoto. Esto es mediante el envío de la cadena de texto *request* por el puerto serial.

```
1. UART.envia(peticion, tamano(peticion));
```

Figura 5.20: Envío de petición por el puerto serial.

### 5.1.5 Puerta de enlace avanzada

El quinto componente implementado de la arquitectura de recolección de datos es la puerta de enlace avanzada. El propósito de esta puerta de enlace es de procesar paquetes en conjunto con el nodo de procesamiento para su posterior almacenamiento en un servidor remoto a través de una petición basada en el protocolo HTTP.

Dado que la puerta de enlace avanzada es capaz de ejecutar programas más complejos, como se describió en la sección 3.2.1, tenemos una computadora empotrada la cual ejecuta el sistema operativo *GNU/Linux*. Esta computadora posee el procesador *Intel XScale IXP420*, el cual es totalmente compatible con el conjunto de instrucciones *ARM V5T*.

Nuestra aplicación para la puerta de enlace avanzada fue desarrollada en lenguaje *C*. Dado que la arquitectura del equipo de desarrollo, *x86* es nuestro caso, es diferente a la arquitectura de la computadora empotrada donde se ejecutará la puerta de enlace avanzada requerimos hacer uso del proceso de **compilación cruzada**.

Esta puerta de enlace será encargada de recibir paquetes de datos desde nodos de procesamiento, ejecutar la función de descifrado sobre estos, conformar la petición de almacenamiento y enviarla al servidor correspondiente. El formato de los datos que recibe la puerta de enlace avanzada se muestra en la figura 5.21.



```
BLOQUE1 , BLOQUE2 , BLOQUE3 , BLOQUE4
```

Figura 5.21: Formato de los datos recibidos.

La petición de almacenamiento la podemos observar en la figura 5.22.

Finalmente, para enviar la petición de almacenamiento se realiza a través de una conexión directa al servidor remoto por medio de un *socket*. La estructura del programa que cumple con este propósito se muestra en la figura 5.23, el pseudocódigo mostrado en ésta figura realiza el envío de una petición a un servidor remoto, está compuesto por una función llamada *enviaPetición*, el cual toma 2 parámetros, el servidor remoto y la petición a enviar. Después, en las líneas de pseudocódigo 2 a 3, se crea el *socket*, si ocurrió un error la función termina aquí, En la línea de pseudocódigo 4 se obtiene la di-

```

GET
http://www.embedded.com.mx/almacenaAvanzado.php?latitud=&longitud=
&altitud=&data=&nids=&nidg=&ntype=&dtype=&mid=&checksum
HTTP/1.0
Accept: */*
Accept: text/html

```

Figura 5.22: Petición de almacenamiento para puerta de enlace avanzada.

rección del servidor remoto al utilizar la función *obtieneHost*. Después, entre las líneas de pseudocódigo 6 y 7 se completan algunos parámetros de la conexión, i.e. tipo de *socket*, direcciones, puertos. La conexión con el servidor remoto se ejecuta en la línea de pseudocódigo 8. Finalmente, se realiza la escritura de la petición en la línea de pseudocódigo 9.

```

1. enviaPetición(Servidor, Petición) {
2.     sock ← socket(...);
3.     si(sock es menor a 0) regresa error;
4.     servidor ← obtieneHost(Servidor);
5.     si(servidor es igual a NULO) regresa error;
6.     servidord.direccion ← servidor;
7.     servidord.puerto ← 80;
8.     si(conecta(sock, servidord, tamano(servidord)) es menor a 0) regresa error;
9.     si(escribe(sock, petición, tamano(petición)) es menor a 0) regresa error;
10. }

```

Figura 5.23: Envío de datos al servidor remoto a través de un *socket*.

### 5.1.6 Servidor de almacenamiento

El sexto componente implementado de la arquitectura de recolección de datos es el servidor de almacenamiento. El propósito de este servidor es proveer almacenamiento permanente a los datos capturados por el mecanismo de recolección. En este servidor se encuentra un programa en lenguaje *PHP* que se encarga de extraer los datos de la petición originada por la puerta de enlace, convertirlos al formato correcto y almacenarlos en una base de

datos *MySQL*.

La estructura de la tabla **Registros** contenida en la base de datos principal se muestra en la tabla 5.6.

Campo	Tipo	Nulo	Valor por omisión	Comentarios
ID	int(11)	Yes	NULO	Identificador del registro
LATITUD	texto	Yes	NULO	Datos del <i>GPS</i>
LONGITUD	texto	Yes	NULO	Datos del <i>GPS</i>
ALTITUD	texto	Yes	NULO	Datos del <i>GPS</i>
DATOS	texto	Yes	NULO	Datos recolectados
TIEMPO	tiempo	Yes	ACTUAL	Hora de almacenamiento
NIDS	texto	Yes	NULO	Nodo origen
NIDG	texto	Yes	NULO	Nodo de entrada
TIPON	texto	Yes	NULO	Tipo de nodo
TIPOD	texto	Yes	NULO	Tipo de paquete de datos
MID	texto	Yes	NULO	Identificador del mensaje
CHECKSUM	texto	Yes	NULO	Suma de verificación

Tabla 5.6: Estructura de la tabla Registros.

La estructura del programa principal del servidor de almacenamiento se muestra en la figura 5.24, denominado *almacena.php*. Las línea de pseudocódigo 1 y 2 muestran la conexión al servidor remoto, en la línea de pseudocódigo 3 se reciben los datos provenientes del nodo de procesamiento, los datos del *GPS* se convierten de formato en la línea 4, finalmente se realiza la petición de almacenamiento en la línea de pseudocódigo 5.

La ejecución del programa anterior no genera respuesta alguna. Los programas y esquemas mostrados en esta sección corresponden para la puerta de enlace avanzada, respecto a los programas que se ejecutan para la puerta de enlace simple, siguen el mismo principio y estructura.

```

1. conexion ← conecta_bd(servidor, usuario, contraseña);
2. seleccioma_bd(basedatos, conexion);
3. parametros ← PARAMETROS_HTTP;
4. convierte_unidades_GPS();
5. consulta("INSERTA en TABLA (...) valores (parametros)");

```

Figura 5.24: Pseudocódigo que recupera los datos de la petición remota y los almacena.

### 5.1.7 Servidor de consulta

El séptimo componente implementado es el servidor de consulta. El propósito de este servidor es desplegar los datos capturados por el mecanismo de recolección de una forma clara y sencilla con el fin de verificar que los datos generados por el mecanismo de recolección sean correctos. Para poder cumplir con su propósito este servidor ejecuta dos programas en lenguaje *PHP*.

El primer programa, denominado ***consulta.php***, extrae de la base de datos los registros almacenados y los presenta a través de una página HTTP. La estructura de este programa es mostrada en la figura 5.25, las instrucciones mostradas en las líneas de pseudocódigo 1 al 3 se utilizan para realizar la conexión a una base de datos y ejecutar una consulta. Entre la línea de pseudocódigo 4 y 8 se extraen los datos resultantes de la consulta y se anexan a la página *HTML* que posteriormente será mostrada por el navegador.

El segundo programa, denominado ***mapa.php***, el cual hace uso de la *API* de *Google Maps*, extrae los datos almacenados, toma la información geográfica y genera un mapa con la información. La estructura de este programa se muestra en la figura 5.26. En el pseudocódigo mostrado la conexión y la consulta a la base de datos se realizan en las líneas de pseudocódigo 1 al 3, mientras que en las líneas de pseudocódigo 6 al 8 se extraen los registros de la base de datos y se insertan en la página generada.

```

1. conexion ← conecta_bd(servidor, usuario, contraseña);
2. selecciona_bd(basedatos, conexion);
3. resultado ← consulta("SELECCIONA * DESDE TABLA", conexion);
4. si (fila ← obtieneArreglo(resultado)){
5.     imprime nombres de parámetros;
6.     ejecuta{
7.         imprime valores de parámetros;
8.     } mientras (fila ← obtieneArreglo(resultado));
9. } sino {
10.    imprime "La base de datos se encuentra vacía!";
11. }

```

Figura 5.25: Estructura del programa que muestra los datos almacenados.

## 5.2 Pruebas y resultados

La implementación descrita en este capítulo fue evaluada en un ambiente controlado y funcionó como se esperaba. Las pruebas realizadas muestran que la distancia entre nodos fue de 7 a 8 metros pero de acuerdo con otros experimentos realizados la distancia puede llegar a alcanzar más de 15 m. Otros equipos transmitiendo y recibiendo en la misma banda de frecuencia estuvieron operando al mismo tiempo (*WiFi* y *Bluetooth*) con signos menores de interferencia. Los controladores de dispositivo desarrollados (ver sección 4.2.5, página 44) han funcionado de manera adecuada.

Todos los componentes del mecanismo de recolección han sido probados de forma individual y en conjunto. Primero, se han probado de forma individual para verificar que su función la desempeñan adecuadamente y segundo se han probado en conjunto para verificar que el mecanismo de recolección funciona como se ha planeado. Al tratarse de nodos que ejecutan un programa empotrado y que no poseen una salida estándar es complicado presentar el resultado de las pruebas ejecutadas en cada componente. Sin embargo, si es posible mostrar los resultados de las pruebas ejecutadas sobre el mecanismo de recolección, y por consecuencia asumimos que el funcionamiento de los componentes es adecuado. En caso de que algunos componentes fallasen el mecanismo de recolección dejaría de funcionar y el error sería evidente.

La forma en cómo podemos verificar los datos generados por el mecanismo de recolección

```

1. conexion ← conecta_bd(servidor, usuario, contraseña);
2. selecciona_bd(basedatos, conexion);
3. resultado ← consulta("SELECCIONA * DESDE TABLA", conexion);
4. si(fila ← obtieneArreglo(resultado)){
5.   Agrega encabezados necesarios para Google Maps
6.   ejecuta {
7.     Agrega puntos al mapa.
8.   } mientras(fila ← obtieneArreglo(resultado));
9.   } sino
10.   imprime "La base de datos se encuentra vacía!";
11. }

```

Figura 5.26: Estructura del programa que muestra los datos almacenados en un mapa.

es mediante la ejecución de los programas residentes en el servidor de consulta. Así por ejemplo, la salida generada por el programa ***consulta.php*** se muestra en la figura 5.27. El programa mencionado anteriormente se encuentra almacenado en un servidor remoto y es accesible mediante la siguiente dirección: <http://www.embedded.com.mx/consulta.php>.

El otro programa que podemos ejecutar en el servidor de consulta es el llamado ***mapa.php*** el cual nos muestra en un mapa la información generada por el mecanismo de recolección. La salida generada por este programa la podemos observar en la figura 5.28.

Además para verificar que los servicios de seguridad funcionan adecuadamente se procedió a programar un nodo adicional, al cual llamamos ***Nodo Espía*** este nodo ejecuta un programa que se encuentra escuchando en modo continuo y desplegando los datos en pantalla (a través de una puerta de enlace *USB*). Con este nodo pudimos verificar que los datos se cifraban correctamente bajo el modo de operación *CBC*.

El tiempo que le toma al circuito integrado *CC2420* cifrar un bloque de 16 *bits* es de aproximadamente 0.45 *ms* (incluyendo la escritura de la llave, del texto plano y la recuperación del texto cifrado). Para conformar el esquema de cifrado *CBC* es necesario cifrar 4 bloques y realizar operaciones lógicas adicionales. El tiempo que toma ejecutar una función de cifrado en modo *CBC* es de aproximadamente 10.3 *ms*, esto es por que además debemos considerar los retrasos de tiempo concernientes al sistema operativo, como son

Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://www.embedded.com.mx/consulta.php

http://www.embedded.com.mx/consulta.php

ID	LATITUD	LONGITUD	ALTITUD	DATA	NODE_SOURCE	NODE_GATEWAY	NODE_TYPE	DATA_TYPE	MESSAGE_ID	TIME
1	19.510811303542262	-99.12941336631775	0	23	1	3	0x01	0x22	13476	2010-10-07 12:19:22
2	19.510811303543658	-99.12941336635187	0	24	1	3	0x01	0x22	6912	2010-10-07 12:19:41
3	19.510811303547265	-99.12941336639176	0	23	1	3	0x01	0x22	31759	2010-10-07 12:20:02
4	19.510811303547239	-99.129413366387435	0	23	1	3	0x01	0x22	34892	2010-10-07 12:20:23
5	19.510811303542268	-99.12941336631780	0	23	1	3	0x01	0x22	5136	2010-10-07 12:20:42
6	19.510811303542295	-99.12941336631753	0	24	1	3	0x01	0x22	17531	2010-10-07 12:21:03
7	19.510811303542317	-99.12941336631653	0	24	1	3	0x01	0x22	26925	2010-10-07 12:21:24
8	19.510811303543452	-99.12941336632175	0	23	1	3	0x01	0x22	8761	2010-10-07 12:21:43
9	19.510811303542761	-99.12941336634317	0	24	1	3	0x01	0x22	5916	2010-10-07 12:22:05
10	19.510811303545128	-99.12941336638325	0	23	1	3	0x01	0x22	17390	2010-10-07 12:22:25
11	19.510811303542865	-99.12941336635389	0	24	1	3	0x01	0x22	24852	2010-10-07 12:22:47
12	19.510811303545916	-99.12941336633298	0	24	1	3	0x01	0x22	37538	2010-10-07 12:23:06

Done

Figura 5.27: Datos recolectados.

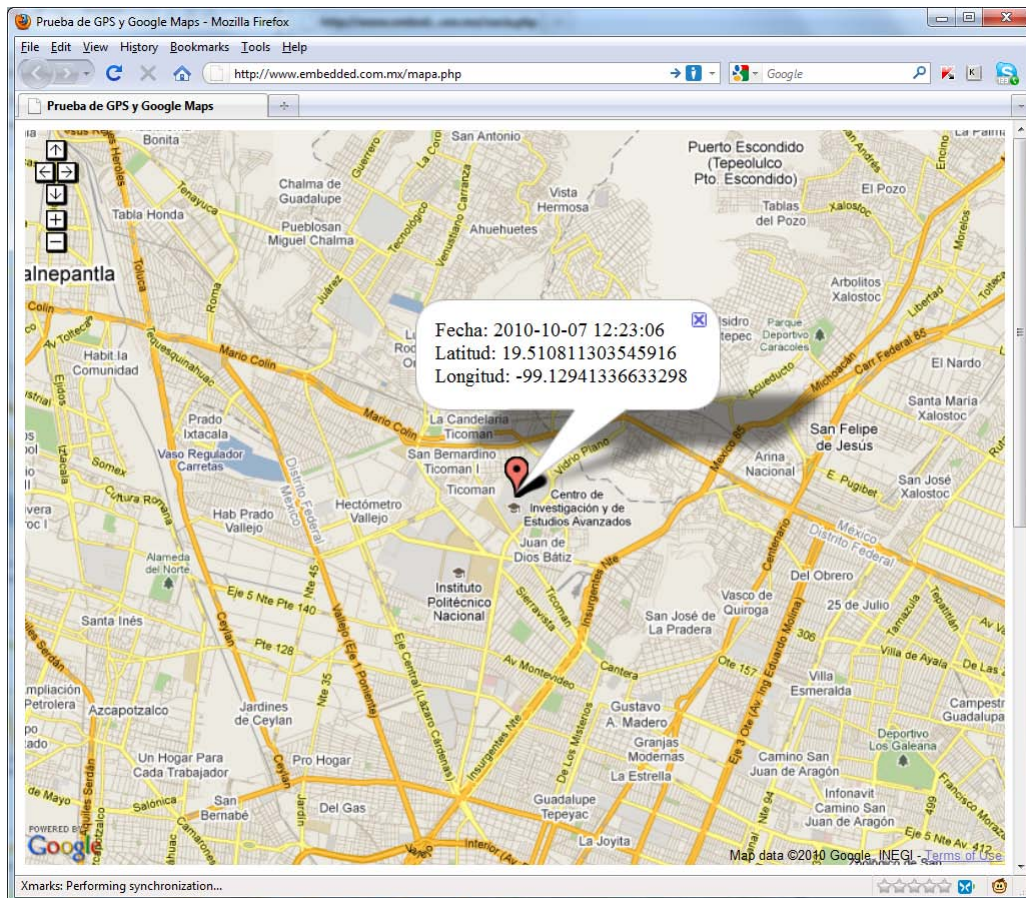


Figura 5.28: Mapa generado con los datos recolectados.

manejo de eventos, interrupciones, cambios de contexto, entre otros. Consideramos que realizando un mejor ejercicio de programación este tiempo puede verse reducido a poco más de 8 *ms*.

### 5.2.1 Problemas presentados y su solución

La tabla 5.7 muestra los problemas más importantes que se presentan al realizar la tesis y su respectiva solución.

Problema	Solución
Los controladores de dispositivo para la tarjeta MTS420CC se encuentran incompletos.	Diseñar un controlador para la tarjeta MTS420CC para el módulo de posicionamiento global <i>GPS</i> . Ver sección 5.1.1.
El programa de recepción de datos desarrollado para una computadora personal no se ha podido compilar para la puerta de enlace avanzada, ya que las librerías utilizadas no se pudieron generar para la arquitectura <i>ARM</i> .	Escribir un programa que reciba datos por el puerto serial y compilarlo para la arquitectura <i>ARM</i> . Ver sección 5.1.5.
El modulo de seguridad <i>AES</i> del circuito integrado <i>CC2420</i> no está siendo utilizado por <i>TinyOS</i> .	Escribir un programa que haga uso de las funciones de seguridad por <i>hardware</i> provistas por el circuito integrado <i>CC2420</i> . Ver sección 5.1.1.

Tabla 5.7: Problemas presentados en esta tesis y su solución.



# Capítulo 6

## Aplicaciones Potenciales

Este capítulo presenta tres aplicaciones potenciales para el mecanismo de recolección de datos desarrollado en esta tesis. A continuación se describen las posibles aplicaciones haciendo énfasis en como el mecanismo de recolección puede aplicarse a estas propuestas.

### 6.1 Sistema de información médica

Este sistema tiene como propósito recolectar información de pacientes que están siendo atendidos en un hospital.

Como parte de los procedimientos médicos, existe información sobre el paciente que debe ser **sensada** y **enviada** cada intervalo de tiempo, por ejemplo, temperatura, presión arterial, frecuencia cardiaca, etc. Adicionalmente, esta información debe ser almacenada y en algunos casos capturada para que se pueda llevar un registro satisfactorio del paciente. Los datos recolectados pueden ser de gran utilidad para el diagnóstico, curación y prevención de enfermedades.

Los pacientes en un hospital comúnmente se encuentran en una habitación, donde permanecen la mayor parte del tiempo, ya sea sentados o acostados, sin embargo durante su estancia pueden requerir visitar otras zonas del hospital, por ejemplo: quirófanos, salas de recuperación, terapia intensiva, laboratorio, etc. Por esta razón los pacientes no podrían hacer uso de dispositivos de recolección (**nodos de lectura**) que requieran de algún cableado, además por seguridad del paciente, cualquier dispositivo eléctrico/electrónico debe estar aislado de la red eléctrica, en nuestro caso al ser una solución operada por ba-

terías, este requerimiento se cumple a la perfección. El nodo de lectura se podría anexar al paciente al momento de su ingreso al hospital, y retirar este equipo cuando el paciente sea dado de alta.

Los **nodos de lectura** utilizarían sensores de temperatura, de presión arterial, de frecuencia cardiaca, entre otros. Para que los datos de estos sensores puedan ser transportados por el mecanismo de recolección sería necesario definir nuevos tipos de datos. Es importante recordar que el mecanismo de recolección permite añadir nuevos tipos de datos, para las lecturas provenientes de diferentes tipos de sensores, y que éstos sean enviados por el mismo mecanismo de recolección sin necesidad de alterar la programación de los demás componentes del sistema.

La **confidencialidad y autenticidad** de los datos del paciente es vital, ya que la información médica debe ser tratada en secreto y solo debe ser conocida por el paciente y los médicos que le atiendan. Es importante mencionar que aun cuando la información sea recolectada por múltiples entidades, esta no puede ser consultada más que por las entidades autorizadas.

Los médicos, enfermeras y demás personal del hospital, pueden actuar como **nodos de reenvío**, ya que éstos se encuentran en movimiento continuo en gran parte de las zonas del hospital, por lo cual se puede cubrir más área con menos nodos. Estos nodos de reenvío tienen la capacidad de almacenar información de bastantes pacientes ya que cuentan con una memoria no volátil de amplia capacidad.

Los **nodos de procesamiento** pueden ser fijos o encontrarse incrustados en mobiliarios del hospital, que pueden ser fijos o móviles. En conjunto con los nodos de procesamiento tendríamos **puertas de enlace avanzadas**, para que reciban los paquetes de datos cifrados y ejecuten los algoritmos criptográficos correspondientes para obtener los datos descifrados. Los **servidores** pueden encontrarse junto a otros equipos de cómputo o en un lugar diferente.

Para completar el sistema sería necesario el desarrollo de una aplicación para administrar la información de los pacientes, la cual se conectaría a la base de datos del servidor de almacenamiento para ejecutar consultas e insertar datos en tablas diseñadas de acuerdo al sistema administrativo. En este sistema se podría manejar toda la información del pa-

ciente, consultar su historial clínico, analizar su comportamiento a través de parámetros médicos, ver estadísticas, o incluso generar alertas cuando se detecten situaciones de riesgo.

La figura 6.1 muestra un esquema simplificado de la aplicación de monitoreo médico.

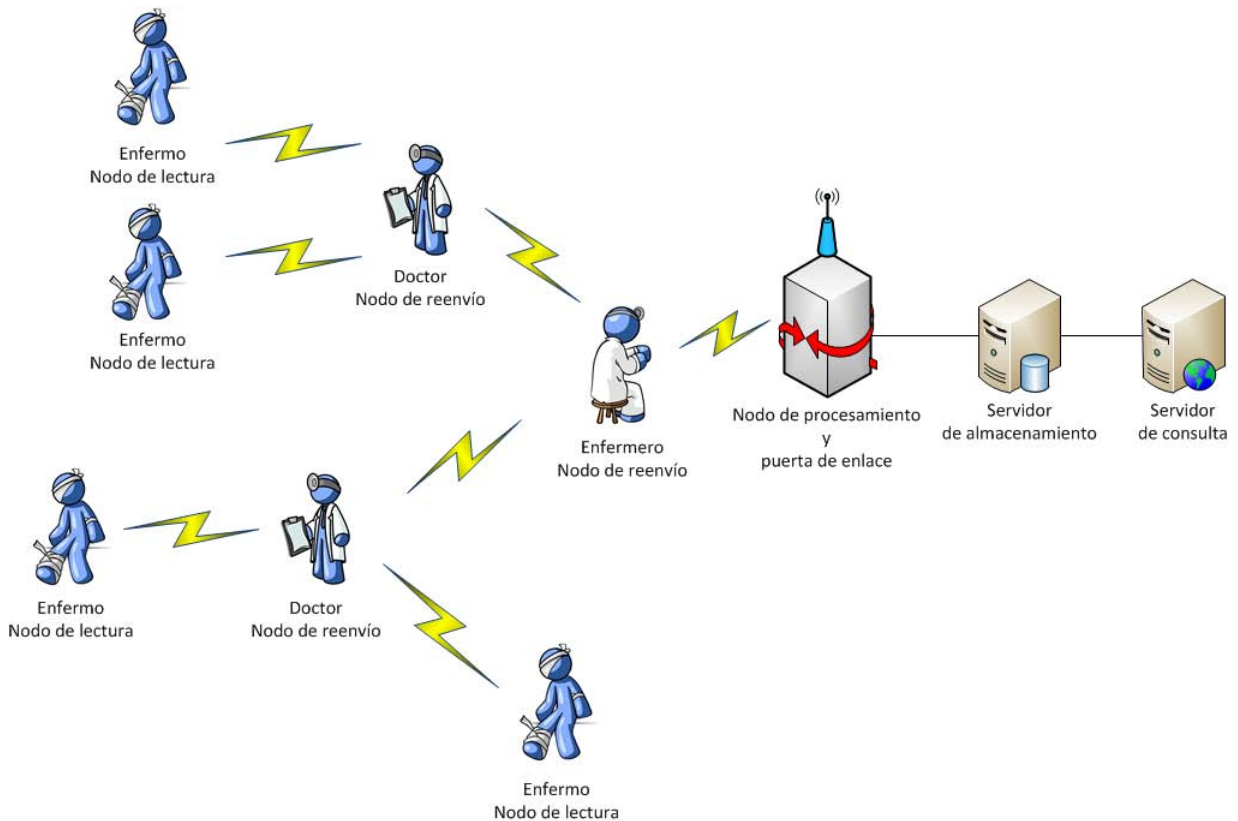


Figura 6.1: Esquema simplificado para la aplicación de información médica.

## 6.2 Sistema de seguimiento para atletas

Este sistema puede ser utilizado para recolectar información de atletas en eventos deportivos.

Los datos que pueden ser recolectados de competidores deportivos pueden incluir: tiempos de carrera, velocidad de desplazamiento, cantidad de pasos ejecutados, además de parámetros del atleta como: temperatura, ritmo cardiaco, presión arterial. Esta informa-

ción recolectada puede ser de gran importancia para los atletas y sus entrenadores con el fin de proporcionar un mejor entrenamiento y en un futuro un mejor rendimiento. Además se pueden detectar problemas potenciales de salud, ya que es posible monitorear el estado del atleta durante el evento.

Aún cuando los datos transmitidos no son totalmente sensibles, es deseable que éstos no puedan ser conocidos por otras entidades, como por ejemplo la competencia, por lo que los servicios de seguridad provistos por el mecanismo de recolección deben ser utilizados.

El mecanismo de recolección puede ser utilizado en eventos deportivos de duración prolongada y en un área geográfica extensa, pero limitada, como por ejemplo en un maratón o medio maratón. Dada la característica de la aplicación es imposible pensar en equipos de lectura cableados.

Los atletas actuarán como los **nodos de lectura**, ya que en este caso son los objetos a monitorear. Los **nodos de lectura** utilizarían sensores de temperatura, de presión arterial, de frecuencia cardiaca, de aceleración, entre otros. Para que los datos de estos sensores puedan ser transportados por el mecanismo de recolección sería necesario definir nuevos tipos de datos. Al igual que en la aplicación anterior, es posible definir nuevos tipos de datos a recolectar.

Los **nodos de reenvío** pueden estar fijos o móviles a través del camino, como en entrenadores o vehículos de auxilio presentes en este tipo de eventos. Los **nodos de procesamiento** pueden ser fijos o encontrarse instalados en los puntos donde se encuentran los equipos de los atletas, en puntos de salida y de llegada. Se utilizarían puertas de enlace avanzadas en conjunto con los nodos de procesamiento para manejar información cifrada y tener el poder de cómputo necesario para procesar los paquetes. Los **servidores** pueden encontrarse junto a otros equipos o en un lugar diferente.

Para completar el sistema sería necesario el desarrollo de una aplicación para administrar la información de los atletas, la cual se conectaría a la base de datos del servidor de almacenamiento para extraer la información de los participantes. Este sistema podría manejar toda la información del atleta, consultar su historial deportivo, analizar su desempeño, ver estadísticas, o incluso generar alertas cuando se detecten anomalías.

La figura 6.2 muestra un esquema simplificado de la aplicación de seguimiento para atletas.

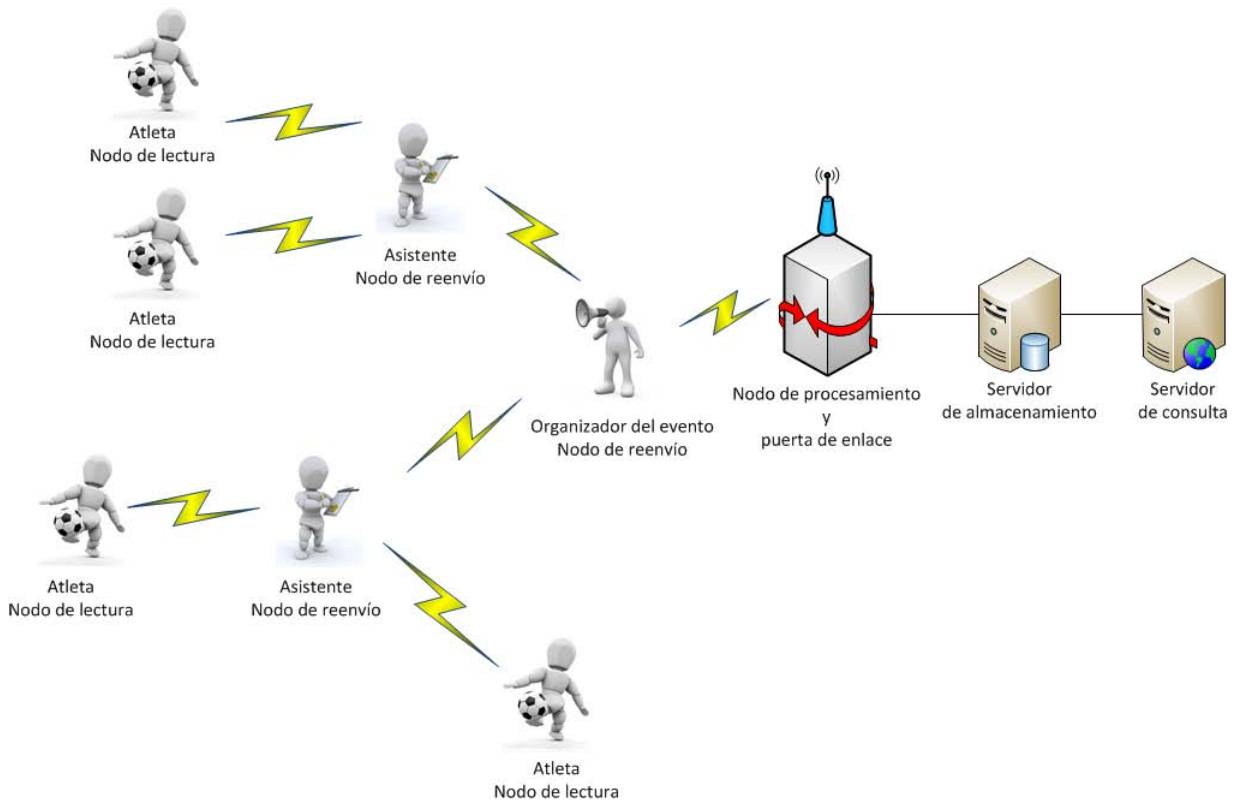


Figura 6.2: Esquema simplificado para la aplicación de seguimiento para atletas.

### 6.3 Sistema de inventarios

Este sistema tiene la finalidad de recolectar información concerniente a inventarios en grandes bodegas.

La tarea de realizar un inventario es una tarea ardua, sí se realiza mediante los procedimientos clásicos, es decir ver el producto, tomar nota de la cantidad existente y proceder con el siguiente producto, para después capturar toda la información y concentrarla en grandes bases de datos. Sin embargo, si hacemos uso de la tecnología, esta tarea puede ser completada en un tiempo visiblemente inferior.

Además, al hacer uso de la tecnología, facilitamos las etapas posteriores a la recolección de datos, por ejemplo en el procesamiento y consolidación de la información. Por último, si la tecnología de recolección está correctamente diseñada y es utilizada de forma adecuada podremos observar que la cantidad de errores asociada con malas lecturas se podrá reducir drásticamente.

La manera en que nuestro mecanismo se puede aplicar a este escenario, es dando a las personas encargadas de levantar el inventario **estaciones de lectura** provistas de **sensores** adecuados para esta tarea, por ejemplo sensores con tecnología *RFID*. Estos operadores tendrán la misión de capturar todos los productos en determinadas áreas asignadas con anterioridad.

Los **nodos de lectura** utilizarían sensores de *RFID*, sensores de posicionamiento global, entre otros.

Los **nodos de reenvío** pueden estar inmersos en estructuras fijas o móviles, como por ejemplo montacargas, ya que estos vehículos se encuentran en movimiento dentro de los almacenes. Los **nodos de procesamiento** pueden estar en localidad fijas, por donde los nodos móviles circulen constantemente.

Al igual que en las otras aplicación, los **servidores** pueden estar junto con otros equipos de cómputo.

En este caso, es deseable que la información que sea capturada sea lo más confiable posible, es decir que no venga alterada ni generada por entidades maliciosas, por lo tanto se puede hacer uso de los servicios de **autenticación**, mientras que el servicio de **confidencialidad** puede quedar como opcional, a consideración del usuario del mecanismo.

Como parte final de este sistema, haría falta una aplicación para administrar y controlar los inventarios, para conocer la existencia o no de ciertos productos, todo esto con la finalidad de mejorar el movimiento de mercancías.

La figura 6.3 muestra un esquema simplificado de la aplicación de levantamiento de inventarios.

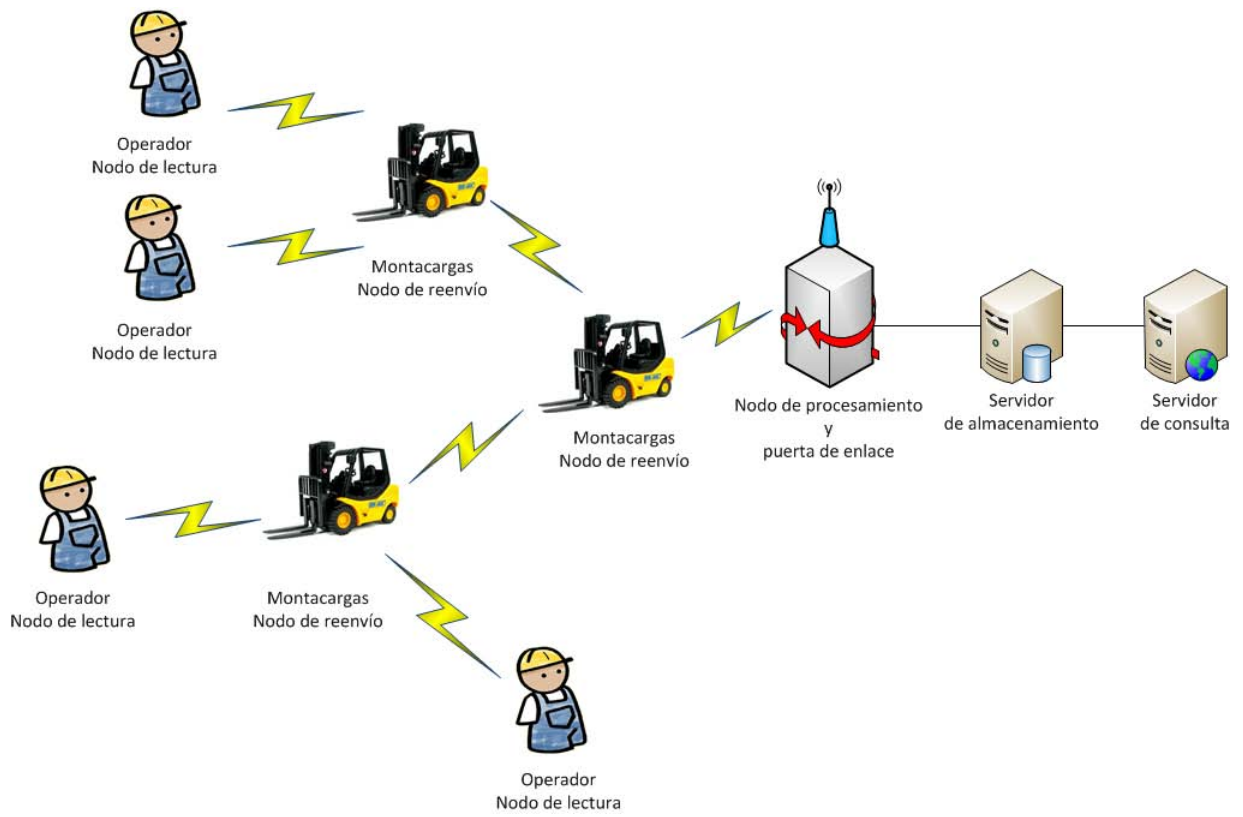


Figura 6.3: Esquema simplificado para la aplicación de levantamiento de inventarios.



# Capítulo 7

## Conclusiones y Trabajo Futuro

En este trabajo se presento el análisis, diseño e implementación de un mecanismo seguro de recolección de datos, el conjunto de pruebas que se han realizado hasta el momento demuestran que el mecanismo de recolección funciona adecuadamente. El resultado de este trabajo permitirá que en un futuro el desarrollo de aplicaciones que ocupen técnicas de recolección de datos sea más sencillo, es así que ingenieros y científicos de otras áreas pueden crear aplicaciones que involucren recolección de datos sin ser expertos en programación de sistemas empotrados, en redes inalámbricas de sensores, en redes de computadoras, en manejo de servidores, entre otras áreas relacionadas con este trabajo.

La arquitectura propuesta incluye diversas categorías de nodos, como son: nodos de lectura, nodos de reenvío y nodos de procesamiento. Además se han definido dos tipos de puerta de enlace (simple y avanzada), esto es para interconectar la red inalámbrica de sensores con otro tipo de red (i.e. una red *Ethernet*). Finalmente se han definido servidores para almacenamiento de datos y de consulta de datos recolectados por el mecanismo diseñado e implementado en este trabajo.

La escalabilidad en este mecanismo juega un papel muy importante ya que permite que las aplicaciones puedan expandirse sin necesidad de reprogramar los nodos. La seguridad juega un papel muy importante y en este mecanismo ha sido contemplada desde sus primeras etapas. Otros requerimientos como la diversidad de sensores está contemplada ya que el mecanismo puede agregar más sensores sin alterar su arquitectura ni la estructura de sus paquetes, por lo cual agregar un sensor más al nodo es un proceso relativamente sencillo.

Otro aspecto de bastante interés en este trabajo, es que a excepción de los servidores (que pueden encontrarse en Internet), todo el mecanismo está programado sobre equipos de cómputo empotrados, los cuales tienen un consumo de potencia muy bajo y en general no requieren ventilación, pudiendo ser operados a baterías en su totalidad, esto nos permite movilidad en nuestras aplicaciones, ya que no dependemos en ningún momento de equipos de cómputo con cableado asociado.

Se han propuesto tres posibles aplicaciones del mecanismo de recolección, éstas son: sistema de información médica, sistema de seguimiento para atletas y sistema de inventarios. En las propuestas realizadas se incluyen las ideas principales de como el mecanismo de recolección diseñado puede ser aplicado para dar forma a estos sistemas.

Los principios básicos de funcionamiento del mecanismo de recolección presentado en este trabajo han sido presentados en el artículo de congreso internacional [20]. Otro trabajo publicado por el autor de ésta tesis, es el artículo [30] donde se describe el diseño e implementación de un generador de números aleatorios, útil para algunas rutinas criptográficas. Además derivado del artículo [30], se realizó una presentación en la sesión de posters en un congreso internacional.

## 7.1 Trabajo futuro

El trabajo a futuro que implica el desarrollo de este mecanismo de recolección se puede resumir en los siguientes puntos:

1. Evaluar el mecanismo a gran escala, en áreas extendidas, con una mayor cantidad de nodos, con una metodología de evaluación previamente definida (i.e. cantidad de paquetes enviados vs. cantidad de paquetes recibidos, velocidad máxima de desplazamiento al momento de efectuar recolección y envío de paquetes de datos).
2. Construir bibliotecas de *software* que permitan que el mecanismo de recolección pueda ser utilizado como componente.
3. Aplicar técnicas de fusión de sensores para obtener mejores datos a partir de una

mayor cantidad de nodos.

4. Construir una interfaz gráfica de usuario que permita seleccionar los parámetros de operación del mecanismo de recolección y que programe automáticamente los nodos con las opciones escogidas.
5. Proponer una solución eficiente para el problema de distribución de llaves para las comunicaciones seguras.
6. Implementar un sistema que haga uso del mecanismo de recolección y evaluar su desempeño.



# Apéndice A

## Anexo A

### A.1 Compilación y carga del programa en los nodos

La compilación y la carga de programas en los nodos es llevado a cabo por el comando mostrado en la figura A.1.

```
make micaz install,id mibXXX, /dev/ttyX
```

Figura A.1: Comando para compilar y cargar los programas en los nodos

La sentencia *make* invoca al comando del mismo nombre, el primer campo especifica la plataforma destino (*MICA*, *MICA2*, *MICAZ*), mientras que el segunda indica la opción a ejecutar (*install* o *reinstall*), el tercero establece el identificador del nodo, el cuarto selecciona el tipo de programador (*MIB510*, *MIB520*, *MIB600*), finalmente el quinto provee la ruta del puerto a utilizar. Este comando debe ser ejecutado en el directorio de cada aplicación, donde se encuentre el archivo *Makefile*. La estructura del archivo *makefile* se muestra en la figura A.2.

```
COMPONENT = nombre_del_componente  
CFLAGS += -I$(TOSDIR)/lib/printf  
MSG_SIZE = tamaño_del_paquete_de_datos  
include $(MAKERULES)
```

Figura A.2: Estructura del archivo *makefile*

## A.2 Instalación del compilador para arquitecturas *ARM*

El proceso de generación del conjunto de herramientas de compilación cruzada es un proceso difícil de lograr, que requiere bastantes conocimientos del sistema, y una gran cantidad de dependencias de paquetes, de versiones específicas de algún compilador. Si bien es cierto que existen *scripts* que nos ahorran mucho no deja ser un proceso difícil y en ocasiones muy tardado.

A continuación se describe la instalación de un compilador cruzado para la arquitectura *ARM* que se encuentra disponible libremente en la red [1].

El compilador quedará instalado en `$HOME/crosstool`

1. Crear directorio del compilador cruzado: `mkdir $HOME/crosstool/`
2. Obtener el conjunto de herramientas de compilación cruzada:  
`wget ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7200-linux/  
cross-toolchains/crosstool-linux-gcc-4.0.1-glibc-2.3.5.tar.bz2`
3. Extraer el contenido del archivo obtenido:  
`tar xvf crosstool-linux-gcc-4.0.1-glibc-2.3.5.tar.bz2  
-C $HOME/crosstool`
4. Agregar la ruta del compilador cruzado a la variable de ambiente `PATH`:  
`PATH=$HOME/crosstool/opt/crosstool/gcc-4.0.1-glibc-2.3.5/  
arm-unknown-linux-gnu/bin/:$PATH`

El llamado al compilador cruzado será de la siguiente manera:

```
arm-linux-unknown-gcc fuentes.c -o objeto.o
```

### A.3 Cambio del *baudrate* en los nodos *MICAZ*

Algunos nodos *MICAZ* deben ser capaces de comunicarse mediante un enlace serial con la puerta de enlace a un mismo *baudrate*, por ello es necesario modificar el archivo y agregar los valores adecuados del registro de configuración para el *baudrate* requerido a la frecuencia de trabajo. Estos valores fueron tomados de las hojas de especificaciones del microcontrolador *Atmel ATMEGA128*. La figura A.1 muestra los diferentes valores que los registros de configuración pueden tomar cuando la frecuencia de trabajo es *7.3728 Mhz* [15].

Baudrate	U2X = 0		U2X = 1	
bps	UBRR	Error	UBRR	Error
2400	191	0.0 %	383	0.0 %
4800	95	0.0 %	191	0.0 %
9600	47	0.0 %	95	0.0 %
14.4k	31	0.0 %	63	0.0 %
28.8k	23	0.0 %	31	0.0 %
38.4k	11	0.0 %	23	0.0 %
57.6k	7	0.0 %	15	0.0 %
76.8k	5	0.0 %	11	0.0 %
115.2k	3	0.0 %	7	0.0 %

Tabla A.1: Configuración del *baudrate* para el microcontrolador *Atmel ATMEGA128*.

## A.4 Interfaces de comunicación en los nodos *MICAZ*

### A.4.1 Interfaz *I2C*

*I2C* es un bus de comunicaciones seriales síncronas, su nombre viene de *Inter-Integrated Circuit*. La velocidad es de *100Kbits* por segundo en el modo estándar, aunque también permite velocidades de *3.4 Mbit/s*. Es un bus muy utilizado, principalmente para comunicar microcontroladores y sus periféricos en sistemas empotrados aunque en general es utilizado para comunicar circuitos integrados entre sí que normalmente residen en un mismo circuito impreso.

La principal característica de *I2C* es que utiliza dos señales para transmitir la información: una señal para los datos y otra señal de reloj. También es necesaria una tercera señal, pero esta sólo es la referencia. Las líneas se llaman: *SDA* (datos) y *SCL* (reloj). Este bus permite la existencia de dispositivos maestros y esclavos, donde pueden existir hasta 128 dispositivos, ya que las direcciones de cada dispositivo están compuestas por 7 *bits*. La señal de datos es bidireccional, por la línea de datos, el maestro realiza los envíos y por la misma línea de datos recibe la respuesta.

En la figura A.3 se muestra el formato de la señales utilizadas en la interfaz de comunicación *I2C*. El primer comando que debe usarse en este bus de comunicaciones es el *Start*, éste se da cuando la línea de datos *SDA* pasa a nivel bajo mientras que la línea de reloj *SCL* se encuentra en nivel alto. Los siguientes 8 *bits* indican la dirección del dispositivo esclavo a utilizar y la operación a realizar (leer o escribir), al finalizar la escritura de estos 8 *bits* el dispositivo esclavo debe enviar una señal de reconocimiento, después de recibir esta señal el dispositivo maestro está listo para enviar la dirección del registro que desea leer o escribir. Después en caso de elegir una operación de escritura se debe enviar el dato a escribir en el registro antes seleccionado. El ultimo comando en este bus de comunicaciones es conocido como *stop*, el cual se da cuando la línea de datos *SDA* pasa a nivel alto mientras que la línea de reloj *SCL* se mantiene en alto.

### A.4.2 Interfaz *SPI*

*SPI* es una interfaz de comunicación serial síncrona, opera en modo *full dúplex*. Los dispositivos que utilizan *SPI* utilizan el modo maestro - esclavo, donde el dispositivo maestro

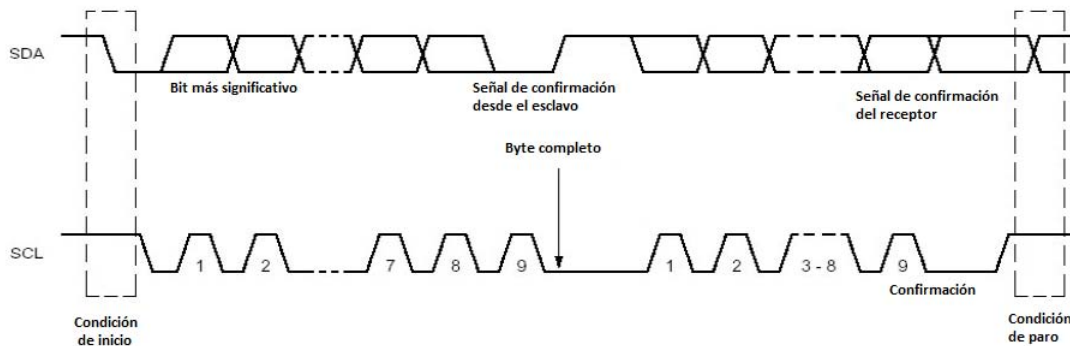


Figura A.3: Formato de las señales de comunicación en *I2C*.

inicia la comunicación al enviar un paquete de datos, este bus permite la conexión de múltiples esclavos. A diferencia del bus *I2C* referido anteriormente, este bus no utiliza un esquema de direccionamiento, por lo que la elección del dispositivo esclavo ocurre a través de la línea de selección provista en cada uno de estos dispositivos. No existe un número máximo de dispositivos esclavos presentes en el bus, ya que solo uno se encuentra activado a la vez.

Esta interfaz está compuesta por dos señales de datos, conocidas como *MOSI* (*Master Output Slave Input*) y *MISO* (*Master Input Slave Output*), *CS* (*Chip Select*), de una señal de reloj (*CLK*) y finalmente de una señal de referencia, o mejor conocida como tierra.

La operación de esta interfaz de comunicación comienza cuando el dispositivo maestro habilita al dispositivo esclavo llevando la línea de selección de nivel alto a nivel bajo. Al escribirse un dato en el bus, a través de la señal *MOSI* automáticamente se genera la señal de reloj correspondiente para sincronizar las comunicaciones. Al completarse el envío de un comando al dispositivo esclavo éste responde con los datos solicitados al escribir los datos en la señal *MISO*, la cual se mantiene en estado de alta impedancia hasta antes de responder con los datos requeridos. La operación antes descrita puede observarse en la figura A.4.

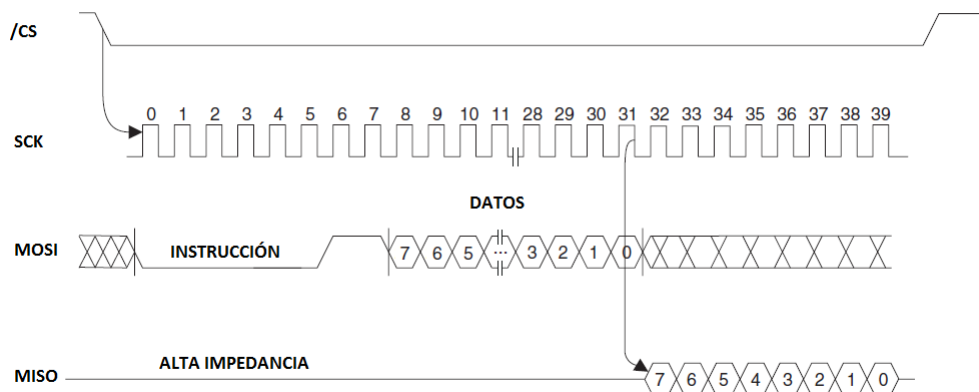


Figura A.4: Formato de las señales de comunicación en *SPI*.

### A.4.3 Interfaz *UART*

Es una interfaz de comunicación serial asíncrona para enlaces de datos. Esta interfaz es muy común en los microcontroladores actuales, y es usado principalmente para comunicarse con otros dispositivos externos. Utiliza una codificación serial para los datos, a una velocidad y formato configurable.

Esta interfaz de comunicación está compuesta por dos señales, la primera conocida como transmisión o *TX*, y la segunda conocida como recepción o *RX*. A diferencia de las interfaces presentadas previamente, esta interfase es asíncrona ya que no tiene una señal de reloj. Por último es necesaria la señal de referencia o tierra.

En esta interfaz de comunicación no existe una señal de reloj para la sincronización de los datos, el ancho de la señal se define a través del parámetro conocido como *baudrate*, donde el ancho del pulso por bit es igual a  $1/\text{baudrate}$ . Los datos se transfieren en secuencias de 8 *bits*, con señalizaciones para indicar el inicio y fin de la transmisión, opcionalmente se puede hacer uso de un chequeo de datos por la paridad de los datos transmitidos.

El lenguaje de programación *nesC* posee los componentes necesarios para efectuar comunicaciones mediante las principales interfaces de comunicación antes descritas, en la tabla A.2 se muestra cada interfaz con su respectivo componente. En caso de que el sensor



Figura A.5: Formato de las señales de comunicación en *UART*.

a comunicar con el nodo utilice una interfaz de comunicación no estándar, ésta se puede implementar a nivel de *software*, si éste es el caso se debe consultar la hoja de datos del componente para poder obtener mayor información respecto a las secuencias de inicialización, control y manipulación del sensor.

Interfaz	Componente
<i>I2C</i>	<i>Atm128I2CMasterC</i>
<i>SPI</i>	<i>Atm128Uart0C</i>
<i>UART</i>	<i>Atm128SpiC</i>

Tabla A.2: Componentes en *nesC* para las principales interfaces de comunicación

# Bibliografia

- [1] Gcc v4.0.1 with glibc v2.3.5 for arm, Nov. 2010, <ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7200-linux/cross-toolchains/crosstool-linux-gcc-4.0.1-glibc-2.3.5.tar.bz2>.
- [2] Body sensor networks nodes, Nov. 2010, <http://vip.doc.ic.ac.uk/bsn/index.php?article=926>.
- [3] Arduino sensor node platform, Nov. 2010, <http://www.arduino.cc>.
- [4] Btnodes - a distributed environment for prototyping ad hoc networks, Nov. 2010, <http://www.btnode.ethz.ch/>.
- [5] Eyes - energy efficient sensor networks, Nov. 2010, <http://www.eyes.eu.org/>.
- [6] Memsic - powerful sensing solutions for a better life, Nov. 2010, <http://www.memsic.com/>.
- [7] Meshnetics - easy wireless for things, Nov. 2010, <http://www.meshnetics.com/>.
- [8] Scatterweb - the self configuring wireless communication platform, Nov. 2010, <http://www.scatterweb.com/index.en.html>.
- [9] Sun spot world, Nov. 2010, <http://www.sunspotworld.com/>.
- [10] Xbee - making wireless m2m easy, Nov. 2010, [www.digi.com/getXBee](http://www.digi.com/getXBee).
- [11] *LEA-4A - ANTARIS 4 GPS Modules Datasheet*. UBLOX, Switzerland, 2007.
- [12] *SHT11 - Humidity and Temperature Sensor Datasheet*. Sensirion, Switzerland, 2010.
- [13] *ADXL202JE - Low-Cost +/-2g Dual-Axis Accelerometer with Duty Cycle Output Datasheet*. Analog Devices, United States of America, 2000.
- [14] *ADG715 - CMOS, Low Voltage Serially Controlled, Octal SPST Switches Datasheet*. Analog Devices, United States of America, 2002.

- [15] *ATMEGA128 - 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash Datasheet*. Atmel, United States of America, 2004.
- [16] *Micro - Ethernet Embedded Device Server*. Lantronix, United States of America, 2006.
- [17] *MS5534C - Barometer Module Datasheet*. Intersema, United States of America, 2007.
- [18] *TLS2550 - Ambient Light Sensor with SMBus Interface Datasheet*. Texas Advanced Optoelectronics Solutions, United States of America, 2007.
- [19] *TIRIS Micro-reader Module Reference Guide - RI-STU-MRD1*. 3rd. Edition, Texas Instruments, USA, 2000.
- [20] I. C. Altamirano and F. R. Henríquez. A scalable intelligent room based on wireless sensor networks and rfid. *CCE2010*, 2010.
- [21] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. pages 563–579, 2005.
- [22] Q. Cao, T. F. Abdelzaher, J. A. Stankovic, and T. He. The liteos operating system: Towards unix-like abstractions for wireless sensor networks. In *International Conference on Information Processing in Sensor Networks*, pages 233–244, 2008.
- [23] H. Chan, A. Perrig, and D. X. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, pages 197–225. IEEE Computer Society, 2003.
- [24] M. de Alba Rosano, I. Cabrera-Altamirano, and C. Ortega-Otero. Estimación del consumo de potencia dinámica en un microprocesador superscalar. *EINVIE05*, pages 251–260, 2005.
- [25] W. Diffie and M. E. Hellman. New directions in cryptography. 1976.
- [26] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN*, pages 455–462, 2004.
- [27] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. pages 263–270, 1999.

- [28] D. Gay, P. Levis, J. R. von Behren, M. Welsh, E. A. Brewer, and D. E. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI*, pages 1–11, 2003.
- [29] J. A. Gutierrez, E. H. Callaway, and J. Raymond L. Barret. *Low-Rate Wireless Personal Area Networks - Enabling Wireless Sensors with IEEE 802.15.4*. 2nd. Edition, IEEE Press, USA, 2007.
- [30] R. E. Henríquez, I. Cabrera, and D. Chakraborty. On the analysis of a practical crypto-system in the limited access model. *CCE2010*, 2010.
- [31] J. L. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *ASPLOS*, pages 93–104, 2000.
- [32] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *ASPLOS*, pages 96–107, 2002.
- [33] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for "smart dust". In *MOBICOM*, pages 271–278, 1999.
- [34] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. *Ad Hoc Networks*, 1(2-3):293–315, 2003.
- [35] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. 2nd. Edition, Prentice Hall, USA, 1988.
- [36] J. Krumm. *Ubiquitous Computing Fundamentals*. 1st. Edition, Chapman & Hall, USA, 2009.
- [37] E. A. Lee. Embedded software. *Advances in Computers*, 56:56–97, 2002.
- [38] D. Liu and P. Ning. Multilevel  $\mu$ tesla: Broadcast authentication for distributed sensor networks. *ACM Trans. Embedded Comput. Syst.*, 3(4):800–836, 2004.
- [39] M. Luca and P. G. Pietro. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Computing Surveys*, 2010.
- [40] D. Lymberopoulos and A. Savvides. Xyz: a motion-enabled, power aware sensor node platform for distributed sensor network applications. In *International Conference on Information Processing in Sensor Networks*, pages 449–454. IEEE, 2005.

- [41] T.Ñadeem, S. D. C. Liao, and L. Iftode. Trafficview: traffic data dissemination using car-to-car communication. *Mobile Computing and Communications Review*, pages 6–19, 2004.
- [42] S. Oh. The vehicle location tracking system using wireless network. In T.-J. Cham, J. Cai, C. Dorai, D. Rajan, T.-S. Chua, and L.-T. Chia, editors, *MMM (2)*, volume 4352 of *Lecture Notes in Computer Science*, pages 651–661. Springer, 2007.
- [43] C. Paar and J. Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer-Verlag New York Inc, 2010.
- [44] A. Perrig, R. Szewczyk, V. Wen, and D. E. Culler. Spins: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, 2002.
- [45] U. Ramana. Some experiments with the performance of lamp architecture. *Computer and Information Technology, International Conference on*, 0:916–921, 2005.
- [46] B. Rubio, M. Díaz, and J. M. Troya. Programming approaches and challenges for wireless sensor networks. In *ICSNC*, page 36, 2007.
- [47] G. Schäfer. *Security in Fixed and Wireless Networks*. 1st. Edition, John Wiley and Sons, USA, 2004.
- [48] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks*, 1(2-3):215–233, 2003.
- [49] K. Sohrawy, D. Minoli, and T. Znati. *Wireless Sensor Networks - Technology, Protocols and Applications*. 1st. Edition, Wiley-Interscience, USA, 2007.
- [50] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992.
- [51] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, January 1991.
- [52] K. Whitehouse, , and D. Culler. Calibration as parameter estimation in sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 59–67, New York, NY, USA, 2002. ACM.
- [53] R. Zurawski. *Networked Embedded Systems*. 2nd. Edition, CRC-PRESS, USA, 2009.

# Glosario

**AES** Advanced Encryption Standard. 24, 30, 37, 43, 62

**API** Application Programming Interface. 58

**ARM** Advanced RISC Machine. 29, 55, 62, 74

**ASCII** American Standard Code for Information Interchange. 46

**ASP** Active Server Pages. 21

**CBC** Cipher-Block Chaining Mode. 25, 30, 37, 42, 60

**CFB** Cipher Feedback Mode. 24, 25

**CS** Chip Select. 77

**CTR** Counter Mode. 25

**DES** Data Encryption Standard. 24

**ECB** Electronic Codebook. 24

**EEPROM** Electrically Erasable Programmable Read-Only Memory. 43

**GCM** Galois Counter Mode. 25

**GHz** Gigahertz. 18

**GNU** GNU is not Unix. 27, 29, 55

**GPS** Global Positioning System. 15, 28, 44–46, 57, 62

**HTML** HyperText Markup Language. 58

**HTTP** HyperText Transfer Protocol. 38, 54, 55, 58

**I2C** Inter-Integrated Circuit. 37, 43, 44, 76, 79

**IIS** Internet Information Services. 21

**ISM** Industrial, scientific and medical. 18

**JSP** Java Server Pages. 21

**KB** Kilobyte. 18, 43

**MB** Megabyte. 18

**MHz** Megahertz. 18

**MISO** Master Input Slave Output. 77

**MOSI** Master Output Slave Input. 77

**ms** milisecond. 60

**mW** miliWatt. 9

**OFB** Output Feedback Mode. 25

**PCMCIA** Personal Computer Memory Card International Association. 15

**PHP** HyperText Preprocessor. 21, 29, 54, 56, 58

**RAM** Random Access Memory. 21, 43

**RFID** Radio Frequency Identification. 28, 35, 47, 48, 68, 69

**RISC** Reduced Instruction Set Computer. 43

**SCL** Serial Clock. 45, 76

**SDA** Serial Data. 45, 76

**SPI** Serial Peripheral Interface. 37, 43, 76, 79

**UART** Universal Asynchronous Receiver Transceiver. 37, 43, 54, 79

**URL** Uniform Resource Locator. 40

**USB** Universal Serial Bus. 60

**XOR** Exclusive OR. 25