View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by UPCommons. Portal del coneixement obert de la UPC

***Título: Prometeo***

***Volumen: Primero***

***Alumno: David Antolino Rivas***

***Director/Ponente: Manel Guerrero Zapata***

***Departamento: Arquitectura de Computadores***

***Fecha: 15/05/2008***

**DATOS DEL PROYECTO**

*Título del Proyecto: Prometeo*

*Nombre del estudiante: David Antolino Rivas*
*Titulación: Ingeniería Informática*
*Créditos: 37,5*
*Director/Ponente: Manel Guerrero Zapata*
*Departamento: Arquitectura de Computadores*

---

**MIEMBROS DEL TRIBUNAL** *(nombre y firma)*

*Presidente: Rubén Tous Liesa*

*Vocal: Francesc Tiñena Salvañà*

*Secretario: Manel Guerrero Zapata*

---

**CALIFICACIÓN**

*Calificación numérica:*
*Calificación descriptiva:*

*Fecha:*

---

# Índice de contenido

2.Introducción....7
3.Entorno de Programación y Herramientas....9
3.1 TinyOS y nesC....9
3.2 TinyOS-1.x vs. TinyOS-2.x....13
3.3 Entornos de Programación....14
3.3.1 TinyOS en Linux Fedora....14
3.3.2 XubunTOS....15
3.3.2.1 Herramientas....16
3.3.2.1.1 TOSSIM....16
3.3.3 TinyOS y Cygwin....18
3.3.3.1 Herramientas....19
3.3.3.1.1 Programmers Notepad 2....19
3.3.3.1.2 MoteConfig....19
3.3.3.1.3 XSniffer....20
4. Descripción del Hardware....22
4.1 MPR2400 (MICAz)....22
4.1.1 IEEE 802.15.4....25
4.1.1.1 Modelo de Red....26
4.1.1.2 Arquitectura de Transporte....27
4.1.2 ZigBee....28
4.2 Placa de Sensores MTS400....28
4.3 Placa MIB520....30
4.3.1 Grabación....31
4.3.1.1 Over The Air Programming....32
4.3.2 Comunicación....32
5. El Incendio Forestal....37
5.1 El Triángulo del Fuego....39
5.2 Propagación del Calor....41
5.2.1 Radiación....42
5.2.2 Conducción....42
5.2.3 Convección....43
5.2.4 Pavesas....43
5.3 Tipos de Incendios....44
5.3.1 Incendios de Superficie....44
5.3.2 Incendios de Copas....45
5.3.3 Incendios de Subsuelo....45
5.4 Comportamiento del Incendio....46
5.4.1 Velocidad de Propagación....46
5.4.2 Forma del Incendio....47
5.5 Primeras Conclusiones....49
5.6 Combustión de Combustibles....50
5.7 La Humedad en los Combustibles....52
5.7.1 Efectos en la Combustión....52
5.7.2 Transmisión de Calor....55
5.7.3 Disponibilidad del Combustible....55
5.7.4 Variación de la Humedad....56

5.7.4.1 Combustibles Muertos....56
5.7.4.1.1 Humedad de equilibrio y tiempo de respuesta....59
5.7.4.1.2 Otros Factores de Variación de la Humedad....61
5.8 Modelo de Predicción de la Humedad....63
6. Modelo Vigente de Detección de Incendios ....69
6.1 Contenido de la Información....70
6.2 Tipos de Humos....70
6.3 Vigilancia Fija....71
6.4 Vigilancia Móvil....73
6.5 Otros Sistemas de Vigilancia....74
6.6 Comparación con la Red de Sensores....75
7. Prometeo....77
7.1 Detección de Incendios....77
7.1.1 Disposición Octogonal....77
7.1.2 Disposición Hexagonal....80
7.1.3 Formación Cuadrada....82
7.1.4 Zona de Fresnel....84
7.2 Protocolo Utilizado....86
7.3 Protocolo con Seguridad....90
7.4 Comparativa: Seguridad vs Raw....92
8. Aplicación: Prometeo – Fire Detection System....95
8.1 Diagrama de Casos de Uso....96
8.2.1 Calcular DC....97
8.2.2 Selección por Humedad....97
8.2.3 Selección por Temperatura....97
8.2.4 Selección por Drought Code....98
8.2.5 Nodos Inactivos....98
8.3 Base de Datos....99
8.4 MsgParser....100
8.5 Interfaz....103
8.5.1 StoryBoard....104
9. Conclusiones Finales....108
10. Bibliografia....109
Apéndice....111
A. Reuniones....111
A.1 Reunión con el Cuerpo de Bomberos de Mollet del Vallés....111
A.2 Reunión con el Ingeniero de Montes Ramón Costa....112
A.3 Reunión con los Responsables del GRAF....113
B. Pruebas y Resultados....114
B.1 Test1....116
B.2 Test2....121
B.3 Test3....125
B.3 Test4....128
C Código....129
C.1 Código TinyOS-1.X....129
C.1.1.1. FireDetection.h....129
C.1.1.2. FireDetectionApp.nc....132
C.1.1.3 FireDetectionC.nc....133

C.2 Código JAVA....................................................................................................160
C.2.1 MsgParser.java..........................................................................................160
C.2.2 FireMenu.java............................................................................................170
C.3 Código TinyOS-2.X.............................................................................................184
C.3.1.1. FireDetectionApp.nc.............................................................................184
C.3.2.1.FireDetectionC.nc..................................................................................186

# *2.* Introducción

Según el avance informativo del Ministerio de Medio Ambiente en el año 2007 se registraron exactamente 10.915 incendios en nuestro país arrasando con un total de 82.048,76 hectáreas, entre pastos, dehesas, matorral, monte abierto y superficie arbolada. Además, debemos tener presente que solamente en el pasado año se destinaron 1.618 millones de euros para la restauración medioambiental y forestal de los territorios incendiados. Con estos datos en la mano resulta natural pensar que los incendios forestales son una desgracia natural y al mismo tiempo conllevan un gasto importante del dinero del contribuyente. Así pues, creemos que nuestro interés en intentar aportar soluciones y mejorar la situación queda más que justificado.

A lo largo de este proyecto desarrollaremos una aplicación destinada a monitorizar el estado del bosque con dos objetivos bien definidos. El primero y principal es realizar un mapa de riesgo de incendios por zonas en las que se distribuyan los sensores. El segundo consiste en detectar incendios ajustándonos a la tecnología que tenemos a nuestra disposición.

Para cumplir nuestros objetivos trabajaremos con sensores MICAz de Crossbow y para ello deberemos familiarizarnos con el hardware que compone los sensores, con el lenguaje de programación propio de los nodos y con el sistema operativo que utilizan. Además, diseñaremos un protocolo de establecimiento de redes ad-hoc y de recolección de datos y discutiremos sobre la importancia de la seguridad en nuestra solución.

Contaremos con la ayuda y el consejo del Grup de Recolzament a les Actuacions Forestals (GRAF) y de un compañero Ingeniero de Montes. Con sus consejos y consultando una extensa biografía argumentaremos sobre el método más preciso para establecer el riesgo en la zona basándonos en los datos que recogen los sensores y en modelos de predicción de la humedad de combustibles muertos.

También hablaremos en detalle de cómo se producen los incendios, qué condiciones les son más propicias. Estudiaremos estadísticas e informes de años anteriores y adaptaremos lo mejor posible nuestro sistema a una situación real.

Finalmente, presentaremos las conclusiones a las que nos ha llevado el proyecto.

# *3.* Entorno de Programación y Herramientas

## *3.1 TinyOS y nesC*

TinyOS es un sistema operativo de código abierto diseñado para redes de sensores impulsado por la Universidad de California, Berkeley, e Intel que a lo largo de los años ha terminado consolidándose en lo hoy que se conoce como TinyOS Alliance. Se asienta sobre una arquitectura basada en componentes que permite una rápida innovación e implementación a la vez que minimiza el tamaño del código de acuerdo con las restricciones de tamaño inherentes a las redes de sensores. La librería de componentes incluye protocolos de red, servicios distribuidos, drivers para acceder a los sensores y herramientas para la obtención de datos.

Así pues, la idea consiste en combinar componentes y articular sus funcionalidades hasta obtener los resultados deseados.

NesC es el lenguaje en el que se escriben y se articulan los módulos. Una forma de describirlo es comparándolo con el propio C. Podríamos decir que nesC es un subconjunto de C, un subconjunto de sus librerías y con una orientación a la arquitectura de componentes. La idea es que cada componente o modulo se componga de dos ficheros, uno donde se declaren las funciones que utilizará el modulo (interfaz) y otro que contendrá la implementación propiamente dicha. Veamos un ejemplo.

En el proyecto nos hemos visto obligados a implementar un componente que sea capaz de guardar las listas de muestras que recogen los sensores. Lo primero que hacemos es crear un fichero SamplesList.nc donde se declara una interfaz y las operaciones que forman el componente.

```
interface SamplesList<samples_t>
{
	command uint8_t getNextSample();
	command uint8_t maxSize();
	command samples_t getLastSample();
	command samples_t getSample(uint8_t i);
	command int8_t addSample(samples_t s);
	command void reset();
}
```

Fig 1. Código de SamplesList.nc

Es probable que lo que más nos llame la atención sea la palabra clave *command*. TinyOS además de trabajar con una arquitectura basada en componentes funciona según un modelo de gestión basado en eventos. Así pues, una interfaz puede declarar un *command*, que es una operación implementada en la propia interfaz y que puede ser utilizada por todos aquellos componentes que enlacen con el componente SamplesList. Por otro lado también se pueden declarar *events*, que aunque también sean declarados en la interfaz, su implementación es responsabilidad del componente que enlace con SamplesList. En este ejemplo, un posible evento podría ser listaLlena, así cuando la lista estuviera llena se haría un signal de listaLlena y entonces en el componente que ha enlazado con SamplesList se empezaría a ejecutar el código de correspondiente al evento listaLlena.

Continuando con el ejemplo, lo segundo que quizá pueda parecernos extraño sea el *<samples_t>* en la primera línea. Simplemente se utiliza para definir el tipo de la lista de muestras y forzar que cuando se llame a la operación addSample se haga la comprobación de tipo y así asegurar que todas las muestras que contiene la lista serán del mismo tipo.

En un segundo plano tendremos el archivo con la implementación de las operaciones,

que por convención se llamará nombreComponenteC.nc y deberá tener este aspecto:

```
generic module SamplesListC(typedef samples_t, uint8_t LIST_SIZE)
{
        provides interface SamplesList<samples_t>;
}

implementation
{
        samples_t samples[LIST_SIZE];
        uint8_t nextSample=0;

        command uint8_t SamplesList.getNextSample()
        {
                return nextSample;
        }

        command uint8_t SamplesList.maxSize()
        {
                return LIST_SIZE;
        }
        .........
}
```

Fig 2. Fragmento de código de SamplesListC.nc

De este archivo cabe resaltar la declaración del módulo o componente donde además se especifica en qué interfaces están declaradas las operaciones que podrán utilizar todos aquellos otros módulos que enlacen con SamplesList.

Llegados a este punto ya tenemos nuestro componente listo para ser utilizado. Para

ello sólo nos queda decir en la implementación de nuestra aplicación que utilizaremos la interfaz SamplesList y en el fichero de configuración especificar qué componente proveerá dicha interfaz. Es perfectamente posible que varios componentes provean una interfaz de igual nombre y distinta implementación.

```
module FireDetectionC
{
        uses interface SamplesList<nx_struct Sample> as List0;
        uses interface SamplesList<nx_struct Sample> as List1;

        ...
}
```

Fig 3. Fragmento de código de FireDetectionC.nc donde anunciamos que utilizaremos SamplesList.

En lenguaje natural este código vendría a decir que FireDetectionC utiliza una vez la interfaz SamplesList como List0 y de nuevo como List1.

```
configuration FireDetectionAppC
{
}
implementation
{
        components new SamplesListC(nx_struct Sample,10) as List0;
        components new SamplesListC(nx_struct Sample,10) as List1;
        FireDetectionC.List0->List0;
        FireDetectionC.List1->List1;
}
```

Fig 4. Fragmento de código de FireDetectionApp.nc donde se especifica que las interfaces List0 y List1 de FireDetectionC son componentes SamplesListC (proceso de enlace).

De nuevo este código se traduciría por: creamos dos SamplesListC que se llamarán List0 y List1, y entonces enlazamos los componentes creados con las interfaces que los necesitan, tal que la interfaz List0 que usa FireDetection será proporcionada por la List0 que acabamos de crear y que la interfaz List1 que usa FireDetection será proporcionada por la List1 que acabamos de crear.

## *3.2 TinyOS-1.x vs. TinyOS-2.x*

TinyOS-2.0 es la evolución de TinyOS-1.1, avanzando hacia mayor robustez, eficiencia, flexibilidad y mayor portabilidad hacia nuevas plataformas. Sin embargo, la primera versión del sistema está tan implantada que aunque TinyOS Alliance no dé soporte activamente, la propia comunidad sí lo hace.

En nuestro caso nos hemos encontrado con que el fabricante (Crossbow) proporciona un entorno de programación y unas librerías orientadas completamente a TinyOS-1.1 resistiéndose así a la adopción del nuevo sistema. Viéndonos en esta tesitura decidimos que lo mejor era hacer dos versiones del código, una para cada sistema. De este modo tendremos una versión para TinyOS-1.1 que será la que se instale en los nodos mientras que la TinyOS-2.0 se reservará para el simulador. El objetivo es desarrollar la parte específica del código que trabaja con el establecimiento de la red y la recolección de datos y probarla en el simulador. Una vez funcione y cumpla con el protocolo que hemos diseñado la portaremos a TinyOS-1.1 y añadiremos el código encargado de recoger los datos de los sensores.

## *3.3 Entornos de Programación*

En la realización del proyecto hemos estudiado y probado múltiples entornos de programación. En esta sección hablaremos de todos ellos, de sus ventajas e inconvenientes y argumentaremos porque hemos escogido uno u otro para trabajar. Sin embargo, hay un elemento común a todos ellos y éste es que los sistemas fueron probados en máquinas virtuales, concretamente con la aplicación Vmware Server. La razón radica en la inestabilidad propia del software, ya que el entorno proporcionado por Crossbow no funcionaba en modo nativo en nuestra estación de trabajo (Windows XP Professional 64 bits) y por ello instalamos múltiples sistemas operativos y probamos múltiples configuraciones hasta que dimos con la acertada.

### *3.3.1 TinyOS en Linux Fedora*

En primer lugar resaltar que para tener TinyOS embebido en Fedora deberemos instalar una quincena de paquetes y configurar diversas variables de entorno. Este conjunto de paquetes está formado por compiladores nativos del microcontrolador instalado en nuestros nodos, compiladores para nesC y el código fuente de TinyOS.

No es un proceso arduo ni difícil, pero sí largo y por tanto es bastante fácil olvidarse instalar algún paquete o configurar alguna variable de entorno con lo que encontrar la explicación a un fallo determinado puede ser complicado.

Como algo positivo cabe destacar que si ya trabajamos con una distribución de Linux la podremos adaptar a TinyOS y mantener nuestro entorno de desarrollo habitual.

## *3.3.2 XubunTOS*

XubunTOS simplifica enormemente la instalación de TinyOS a través de un live CD de Linux. Nace de la unión de Xubuntu con TinyOS 2.x y a su vez Xubuntu es una versión más compacta de Ubuntu con el gestor de ventanas XFCE, que es más reducido que Kde o Gnome y deja así más espacio en el CD para el software propio de TinyOS.

De todos los métodos de instalación, distribuciones de Linux y versiones de Windows probadas sin lugar a dudas nos quedamos con XubunTOS. En primer lugar por la facilidad de instalación y en segundo porque una vez arranca la máquina virtual del CD el entorno ya está preparado para empezar a trabajar con los sensores. Además, el hecho de ser una distribución Live nos permite modificar los ejemplos una y otra vez porque cuando volvamos a arrancar lo encontraremos todo como el primer día. Naturalmente, también incluye la opción de instalación permanente en disco si así lo preferimos.

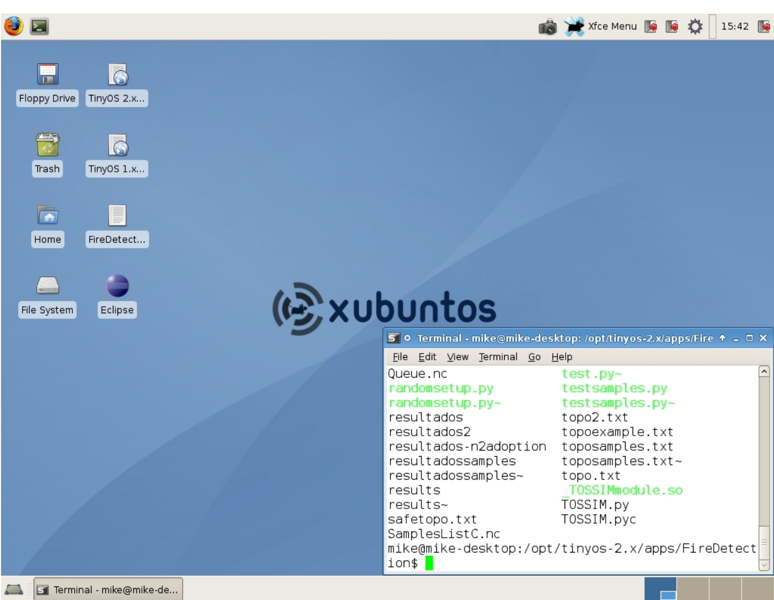


Fig 5. Escritorio de XubunTOS

### 3.3.2.1 Herramientas

En lo que se refiere a herramientas en XubunTOS hemos trabajado con el propio terminal del sistema para grabar los sensores y para programar hemos utilizado el editor Kate. Para comprobar el correcto funcionamiento de la aplicación hemos recurrido al simulador TOSSIM.

Además, el entorno cuenta con un conjunto de herramientas Java que nos ayudan a familiarizarnos con el uso del puerto serie, y que nos han permitido construir la aplicación de usuario de nuestro sistema.

#### 3.3.2.1.1 TOSSIM

TOSSIM simula aplicaciones enteras de TinyOS. Funciona reemplazando los componentes físicos por implementaciones simuladas y el nivel al que los componentes son reemplazados es muy flexible. Por ejemplo, hay una implementación para simulación de un el reloj HilTimerMilliC, que es general y puede ser utilizada por cualquier plataforma, y al mismo tiempo existe una implementación específica para plataformas Atmega que solamente reemplaza los componentes hardware estrictamente necesarios. Del mismo modo que sustituye relojes, TOSSIM también puede simular la comunicación entre nodos mediante la radio.

A nivel conceptual, TOSSIM no es más que una librería, ya que cada vez que queramos crear una simulación tendremos que crear un programa que la configure y la ejecute. Estos programas pueden estar escritos en C++ o en Python. Nosotros nos hemos decantado por esta última opción por su sencillez y porque con Python podemos escoger entre interactuar con la simulación aunque ya esté en marcha, como si de un debugger se tratara, y automatizarlo todo con scripts o programas como haríamos en C++.

En capítulos más avanzados trataremos en detalle los códigos que hemos utilizado dejando de lado la interacción entre la simulación y el programador. Por eso ahora nos centraremos solamente en el simulador.

```
Terminal - mike@mike-desktop: /opt/tinyos-2.x/apps/FireDetection
File Edit View Terminal Go Help
mike@mike-desktop:/opt/tinyos-2.x/apps/FireDetection$ python
Python 2.5.1 (r251:54863, May  2 2007, 16:56:35)
[GCC 4.1.2 (Ubuntu 4.1.2-0ubuntu4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from TOSSIM import *
>>> t=Tossim([])
>>> t.runNextEvent()
0
>>> m=t.getNode(4);
>>> m.bootAtTime(500);
>>> import sys
>>> t.addChannel("Boot",sys.stdout);
>>> t.runNextEvent()
DEBUG (4): app booted
1
>>>
```

Fig 6. Terminal de XubunTOS ejecutando TOSSIM.

En esta imagen se puede ver como lo primero que hacemos es arrancar el intérprete de Python y a continuación importar el contenido de la librería TOSSIM. Seguidamente creamos un objeto Tossim y ejecutamos el siguiente evento, lo cual nos devuelve 0 porque no hay ningún evento por ejecutar. En las líneas siguientes obtenemos un nodo, le decimos cuando debe arrancar, relacionamos el canal de salida del nodo con el terminal y finalmente arrancamos el nodo. El mensaje de debug así lo confirma.

TOSSIM ha resultado ser una de las herramientas más útiles ya que nos ha permitido desarrollar la aplicación aún cuando no teníamos los sensores y porque debugar en un entorno de simulación aporta muchísima más información acerca de la lógica del programa y es más sencillo que hacerlo instalando la aplicación en nodos y observando su comportamiento.

### *3.3.3 TinyOS y Cygwin*

Finalmente sólo queda por describir el entorno que más dificultades, errores e incompatibilidades ha causado. Crossbow nos facilita un CD con herramientas para grabar y programar los sensores o nodos. Algunas de estas herramientas se instalan directamente en el sistema operativo que estemos corriendo en la máquina. Sin embargo, hay otras que necesitan funcionalidades Linux aunque nos encontremos en un sistema Windows. Y para ello el fabricante recurre a Cygwin.

Básicamente, Cygwin es un entorno Linux para Windows compuesto de:

- Una DLL que actúa como una capa de emulación de la API de Linux y provee gran parte de sus funcionalidades.
- Una colección de herramientas para mostrar el look and feel de Linux.

En este punto nos parece interesante remarcar que la instalación del paquete Cygwin suministrado por Crossbow sólo funcionó en una instalación limpia de Windows XP Professional Edition 32 bits a pesar de que el fabricante asegura la compatibilidad con otras versiones del sistema operativo.

Lo único que nos habría empujado a trabajar con este entorno es el hecho de que el fabricante incorpora unas placas de sensores para las que diseña componentes (módulos e interfaces) que permiten acceder a los datos que muestrea, y ya vienen instaladas en el entorno de Crossbow. Sin embargo, no cuenta con el simulador TOSSIM, ni con herramientas Java orientadas a la comunicación entre el ordenador y la estación base utilizando el puerto serie.

Afortunadamente, TinyOS es un proyecto de código abierto y los desarrolladores de XubunTOS han tenido a bien incluir en el sistema esas mismas librerías, con lo que hemos podido dejar el entorno de Crossbow aparcado.

### 3.3.3.1 Herramientas

#### 3.3.3.1.1 Programmers Notepad 2

Pn2 es un editor de texto especialmente diseñado para nesC. Cuenta con un terminal Cygwin embebido que nos permite compilar e instalar nuestra aplicación en un mote sin salir del programa.

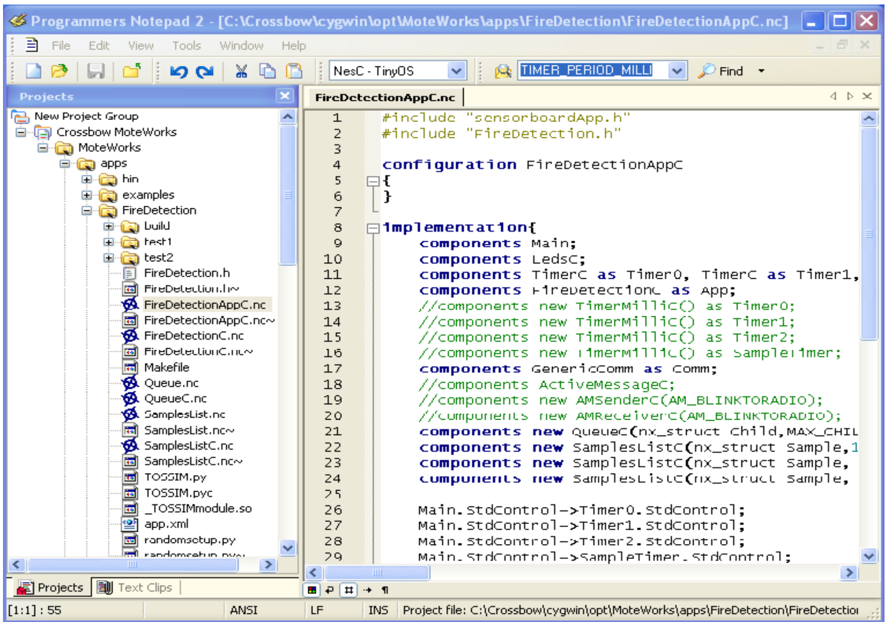



Fig 7. Programmers Notepad 2

#### 3.3.3.1.2 MoteConfig

MoteConfig es la herramienta que se utiliza para grabar los sensores. El propio programa permite validar el ejecutable que queremos instalar y a la vez extraer parámetros fijados en tiempo de compilación, como el identificador del nodo o la potencia de la radio, y modificarlos antes de programar la placa.

De esta herramienta cabe destacar que es la única que permite el modo de programación OTAP (Over The Air Programming). Programas como Pn2 o bien el propio terminal Cygwin tienen comandos específicos para grabar sensores, pero sólo pueden grabar el nodo que esté físicamente conectado a la placa programadora. Sin embargo, MoteConfig explota una particularidad de los nodos MICAz que permite que el código que queremos instalar viaje por la red como un paquete más, llegue al nodo de destino, se instale y cuando ordenemos el reinicio del nodo éste empiece a ejecutar el nuevo código.

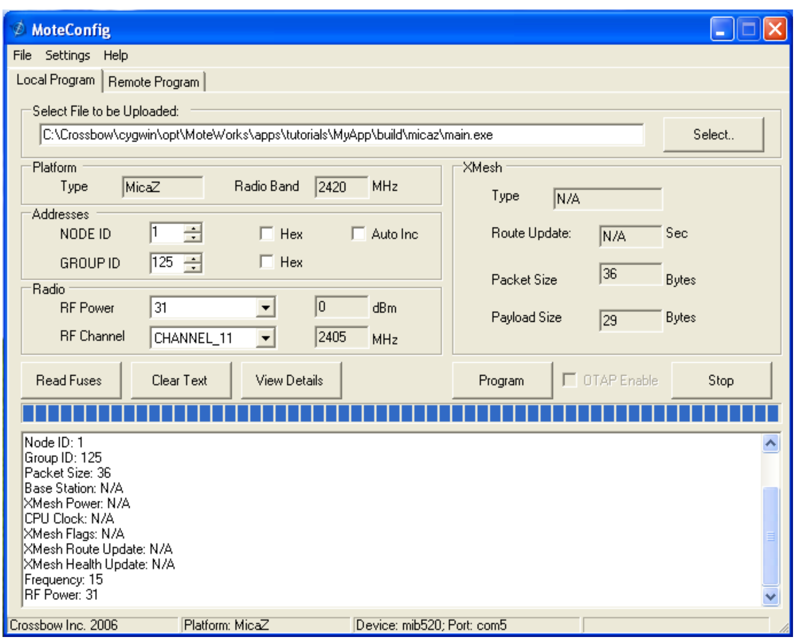


Fig 8. Programmers Notepad 2

### *3.3.3.1.3 XSniffer*

Tal y como su propio nombre indica, XSniffer es una herramienta cuya tarea se reduce a escuchar el tráfico de la red. Si utilizáramos el formato de paquete de Crossbow además podría parsearlos y mostrarnos el contenido.

Este programa nos habría sido especialmente útil a la hora de debugar el código de

TinyOS-1.X ya que el entorno de Crossbow no cuenta con un simulador.

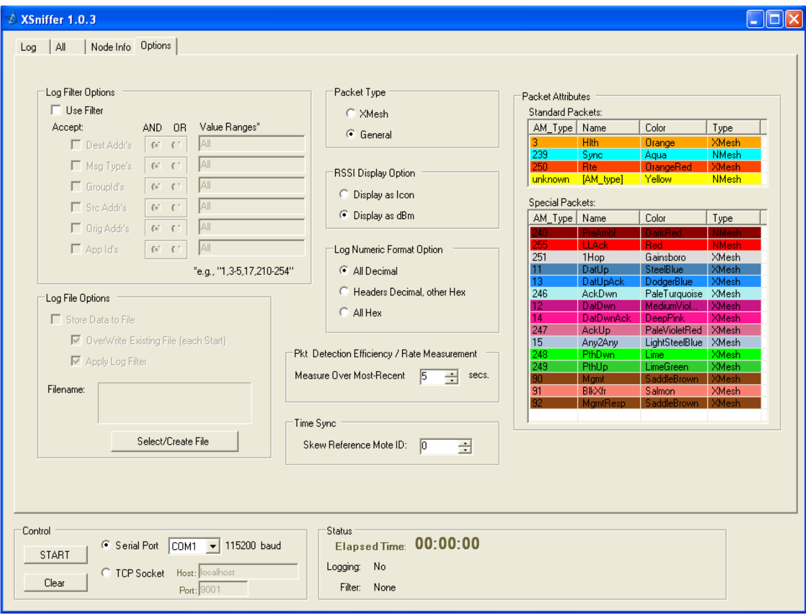



Fig 9. XSniffer

# 4. Descripción del Hardware

En este proyecto trabajaremos con sensores MICAz en los que montaremos unas placas de sensores MTS400 y que serán programados por una placa MIB520.

En un primer momento es normal que lo dicho en la frase anterior no suene más que a jerga incomprensible. Por eso vamos a hacer un repaso en detalle de cada uno de los componentes involucrados en el proyecto.

## *4.1 MPR2400 (MICAz)*

MICAz es la última generación de sensores de Crossbow. Utiliza una radio Chipcon CC2420 (emite en bandas desde 2400MHz hasta 2483.5 Mhz), cumple con el estándar IEEE 802.15.4 y está preparado para el uso del modelo de comunicaciones ZigBee.

Siempre que trabajemos con redes de sensores hay tres características muy importantes que deberemos tener en cuenta: tamaño de memoria, alcance de la radio y el consumo de las baterías. De momento nos centraremos en esta última.

Para que un nodo MICAz opere con normalidad debemos suministrarle entre 2.7 y 3.6 V. En la figura 10 podemos ver en detalle cual es el gasto de corriente del nodo en función de la operación que realiza.

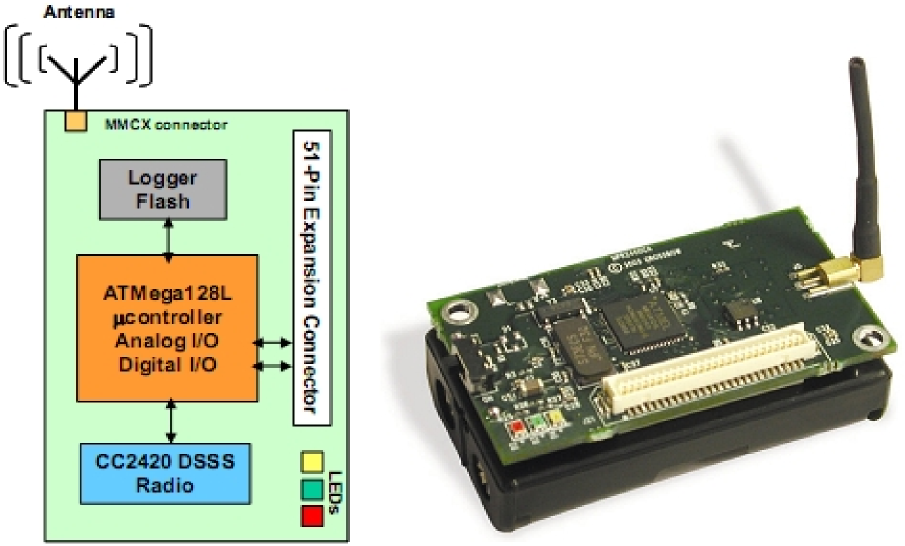



Fig 10. Diagrama de bloques de un mote MICAz y su equivalente real

| Corriente de Operación (mA) | MICAz |
|---|---|
| Procesador, modo trabajo | 12 (7.37 MHz) |
| Procesador, reposo | 0.010 |
| Radio - recepción | 19.7 |
| Radio – transmisión (1mW de potencia) | 17 |
| Radio - reposo | 0.001 |
| Escritura en memoria flash | 15 |
| Lectura de memoria flash | 4 |
| Memoria flash - reposo | 0.002 |

Fig 11. Tabla de consumo

Basándonos en esta tabla de consumo podemos hacer un cálculo aproximado de la vida de las baterías. Recordemos que los sensores funcionan con dos pilas AA que aportan unos 2000 mA-hr. Nuestra aplicación debe monitorizar un bosque y aunque entraremos en detalles en capítulos venideros podemos adelantar que debemos tomar muestras de los sensores una vez cada diez segundos. A groso modo, significa

que cada 10 segundos enviamos un paquete y recibimos otro, es decir, cada diez segundos tenemos un consumo de 36.7 mA, 220.2 mA por minuto, 13.2 A por hora. Despreciando el consumo del procesador llegamos a la conclusión de que las baterías se agotarían a los 0.21 meses, lo cual es inaceptable para una aplicación que pretende ser autónoma. Así pues, las baterías convencionales no son una solución que podamos contemplar. Sin embargo, sí vemos viable realizar el proyecto con placas solares de tamaño reducido que recarguen las baterías en las condiciones de mayor energía lumínica que coinciden siempre con las situaciones de mayor peligro de incendio.

Siguiendo el repaso de características importantes que condicionan el funcionamiento de la red es el turno del tamaño de memoria. En nuestro caso la memoria no será una limitación ya que los nodos cuentan con 128Kbytes de memoria de programa mientras que nuestra aplicación necesita de unos 15kbytes aproximadamente.

Finalmente, en lo que a la radio se refiere, sabemos que es el componente del nodo que más consume y que emite en bandas de frecuencia comprendidas entre los 2400 y los 2483.5 MHz. Además, es útil saber que tiene entre 75 y 100 metros de alcance en exteriores y que puede recibir tráfico en condiciones de hasta -90 dBm de ruido. Sin embargo, en nuestro proyecto los sensores no tendrán que poner a prueba estos límites porque estarán relativamente cerca, con lo que podremos rebajar la potencia de emisión y por tanto reducir el consumo.

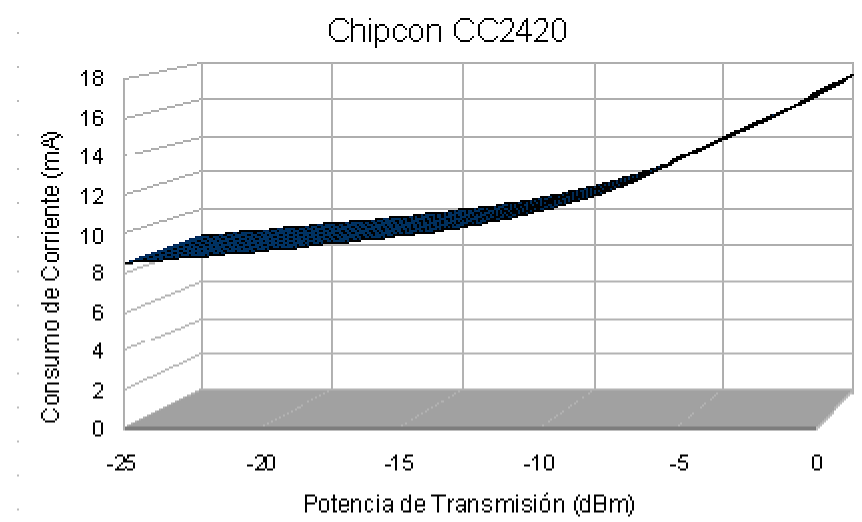



Fig 12. Gráfico de consumo en función de la potencia

Según lo que podemos observar en la gráfica el coste de emitir a -25 dBm es de 8.5 mA. Si retomamos los cálculos que hicimos unas líneas atrás obtenemos que las baterías nos durarían 0.45 meses, lo cual es sensiblemente mejor a lo que teníamos antes pero no lo suficiente como para descartar las placas solares de nuestro sistema.

### *4.1.1 IEEE 802.15.4*

El estándar IEEE 802.15.4 especifica la capa física (PHY) y la de control de acceso al medio (MAC) para WPANs (Wireless Personal Area Network), que se centra en el bajo coste y en mantener comunicaciones de baja velocidad entre dispositivos, en contraste con otros enfoques orientados al usuario como puede ser Wi-Fi. En este caso el énfasis se encuentra en las comunicaciones entre dispositivos utilizando poca o ninguna infraestructura específica, intentando explotarlo en favor de un menor consumo.

El marco básico de trabajo trata comunicaciones a diez metros de distancia con una tasa de transferencia de 250 kbit/s. Aunque, es común encontrarse con tasas inferiores en la búsqueda de consumos aún menores.

Otras características a tener en cuenta son el soporte integrado para comunicaciones seguras, la evitación de colisiones mediante CSMA/CA (Carrier Sense Multiple Access – Collision Avoidance) y la reserva de slots de tiempo garantizados.

#### *4.1.1.1 Modelo de Red*

El estándar define dos tipos de nodos de red:

- Full-function Device (FFD): puede actuar de coordinador en una WPAN o bien como lo haría cualquier otro nodo. Implementa un modelo general de comunicación que le permite hablar con otros dispositivos así como hacer de repetidor de mensajes, en cuyo caso estaríamos hablando de un coordinador parcial de la red.
- Reduced-Function Device (RFD): estos dispositivos son extremadamente simples y de recursos muy limitados por lo que solamente pueden comunicarse con FFDs y nunca pueden llegar a actuar de coordinadores.

Las redes pueden ser construidas con una topología de estrella o bien de punto a punto. Sin embargo, cada red necesita como mínimo un FFD que ejerza de coordinador. Además, cada dispositivo tiene un identificador único cuya longitud oscilará entre los 8 y los 64 bits dependiendo de las necesidades de la red que estemos diseñando. En nuestro caso hemos considerado que con 8 bits y por tanto 256 nodos tenemos suficiente para realizar un prototipo del proyecto.

Las redes punto a punto pueden formar patrones arbitrarios de conexión, limitados

por la distancia entre nodos. Éstas conexiones están llamadas a servir de base para las redes ad – hoc capaces de organizarse y mantenerse a si mismas.

Tal y como decíamos antes, las redes estrella también están soportadas, y tienen en el nodo central a su coordinador. Estas redes se originan cuando un dispositivo FFD decide crear su propia PAN y declararse a si mismo coordinador de ella. A continuación otros dispositivos pueden pasar a formar parte de la red.

#### *4.1.1.2 Arquitectura de Transporte*

Los frames o paquetes son la unidad básica de transporte en la red, y podemos distinguir cuatro tipos:

- datos
- ack
- beacon
- comandos MAC

Además, puede darse el caso de que el coordinador de la red decida usar una estructura de superframe, definida por él mismo, y en ese caso dos beacons marcarán los límites de la estructura proveyendo sincronización e información de configuración a otros nodos.

Un superframe es una serie de 16 slots de igual medida, que a su vez pueden ser subdivididos en parte activa e inactiva durante la cual el coordinador puede entrar en modo de reposo sin necesidad de controlar su red. En el uso de superframes la contención se resuelve por CSMA/CA. Además, cada transmisión debe finalizar antes de la emisión del segundo beacon que marca el final del superframe.

La transmisión de datos hacia el coordinador de la red requiere una fase de sincronización de beacons seguida de una transmisión CSMA/CA. La transmisión en

sentido inverso suele responder a peticiones de los dispositivos, así que en primer lugar el coordinador hace un ack de la petición y luego envía los datos al dispositivo que responde con una serie de acks.

En redes punto a punto es posible la utilización de CSMA/CA u otros mecanismos de sincronización.

### *4.1.2 ZigBee*

ZigBee es el nombre que recibe una serie de protocolos de comunicación de alto nivel que usan radios basadas en el estándar IEEE 802.15.4 para WPANs. El objetivo es conseguir una tecnología más simple y barata que la utilizada en otras WPANs, como por ejemplo el Bluetooth. ZigBee está pensado para aplicaciones que requieran maximizar la vida de la batería, que puedan trabajar con una tasa de transferencia baja y necesiten securizar las comunicaciones.

## *4.2 Placa de Sensores MTS400*

Desarrollada entre UC Berkeley e Intel Research Labs, la MTS400 ofrece cinco sensores de ambiente básicos. Incorporando la tecnología más reciente en circuiteria integrada de sensores resulta ideal en aplicaciones donde queramos extender al máximo la vida de la batería y reducir al mínimo el mantenimiento de los nodos. Lo que la hace especialmente apropiada para nuestra aplicación de monitorización de bosques.

A continuación presentamos la especificación de los sensores ya que muchas de las decisiones que tomemos en la distribución de los nodos y por tanto en la formación de la red dependerá de ella.

Fig 13. MTS400CB

- Placa:
    - Opera en un rango de temperatura de -10°C hasta +60°C.
    - Opera en un rango de humedad de 0% HR (Humedad Relativa) hasta un 90%.
- Acelerómetro de doble eje:
    - resolución de +-2 g.

Sería posible utilizarlo para saber si alguien esta manipulando el nodo.

- Sensor de humedad relativa y de temperatura
    - Humedad:
        - resolución de 0 a 100% HR.
        - margen de error +-3.5% RH.
    - Temperatura:
        - margen de error +- 0.5°C con 25°C.

Los utilizaremos para establecer factores de riesgo y para la detección de incendios mediante cambios bruscos de temperatura.

## 4.3 Placa MIB520

La placa MIB520 proporciona conectividad por USB con los sensores de la familia MICA ya sea para comunicarse con ellos o para programarlos.

La placa cuenta con un ISP (In-System Processor) para programar los nodos. Básicamente, el código se descarga en el ISP a través del puerto USB y a continuación el ISP programa el código en el mote propiamente dicho.

A nivel de sistema operativo, cuando conectamos la placa al ordenador el driver se encarga de transformar ese puerto USB en dos puertos Serie virtuales: com<n> y com<n+1> . Com<n> es el puerto que se utiliza para programar el mote mientras que com<n+1> es el destinado a la comunicación.

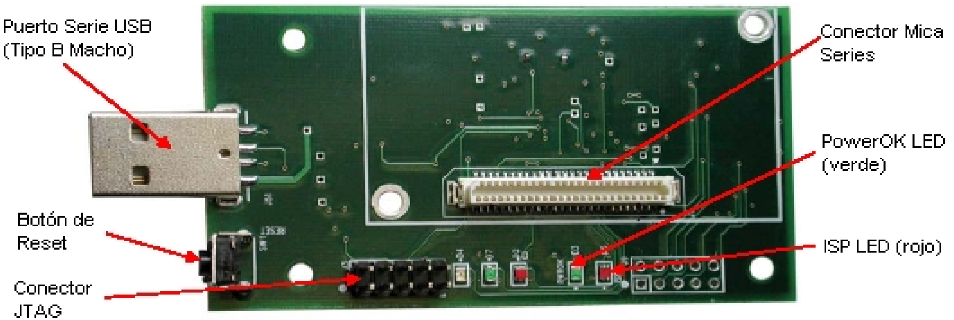



Fig 14. MIB520

Cuando hablamos de comunicación nos referimos al hecho de que la placa puede actuar de enlace entre el ordenador y la red de sensores. Podemos conectar un MICAz a la placa programadora, renunciando así al slot de expansión para la placa de sensores, instalar en ese nodo un programa *BaseStation* y mediante otras herramientas el PC puede acceder al tráfico de la red desde su puerto Serie virtual.

Ahora bien, es muy importante remarcar que el nodo que conectemos a la MIB520

esté desconectado de su fuente de alimentación habitual porque la placa programadora ya cuenta con su propia alimentación a través de la interfaz USB. De lo contrario correremos un riesgo elevado de quemar el chip.

### *4.3.1 Grabación*

Grabar un mote en XubunTOS es relativamente sencillo. Basta con conectar la placa a un puerto USB del ordenador y modificar los permisos para permitir lectura y escritura en el puerto.

*chmod 666 /dev/ttyUSB0*

A continuación nos situaremos en el directorio y compilaremos e instalaremos el programa con una sola instrucción.

*make micaz install mib510,/dev/ttyUSB0*

Nótese que en lugar de mib520 hemos incluido mib510 en la instrucción. No es un error, lo que ocurre es que XubunTOS no incluye soporte específico para la MIB520 porque es igual que la MIB510. La única diferencia es que una tiene un puerto Serie real mientras que la otra lleva un Serie simulado que en realidad es un puerto USB.

En cambio, si nos encontráramos en el entorno de Crossbow la forma descrita arriba también funcionaría en un terminal Cygwin con la diferencia de que aquí si que tenemos soporte para la placa MIB520 y por ello deberemos modificar esa parte de la instrucción.

También podemos programar los nodos desde Programmers Notepad 2 así como utilizando MoteConfig. Utilizando este último descubriremos el modo de programación OTAP y ahí es donde estriba una de las pocas ventajas que tiene Crossbow sobre XubunTOS.

#### *4.3.1.1 Over The Air Programming*

OTAP es una idea sencilla pero ingeniosa con la que se pretende programar sensores remotamente, es decir, sin necesidad de conectarlos a la placa grabadora. Básicamente consiste en enviar el código que queremos programar en el nodo como si fuera un paquete más, a través de la radio, y el mote destinatario lo recibirá y lo grabará en la Logger Flash (figura 9). Finalmente cuando sea reiniciado empezará a ejecutar el código nuevo.

Viéndolo a más bajo nivel, cada nodo tiene 512kB de memoria flash dividida en 4 slots, con 128kB por slot. El Slot 0 está reservado para la imagen OTAP, algo parecido a un gestor de arranque, mientras que los slots 1, 2 y 3 están disponibles para albergar programas.

El servidor o nodo base, ordena al nodo que reinicie desde el slot 0 para que cargue la imagen OTAP. A continuación, la imagen del firmware especificado por el usuario es dividida en fragmentos y transmitida al mote, que la recibirá y la guardará en los slots disponibles. Finalmente, el servidor puede enviar un mensaje para transferir el firmware del slot a la memoria de programa y reiniciar el nodo.

Es interesante remarcar que si queremos hacer que nuestra aplicación sea programable remotamente deberemos incluir el componente XOTAPLiteM.

### *4.3.2 Comunicación*

En XubunTOS, TinyOS incluye una serie de herramientas que trabajan con el puerto Serie y un conjunto de programas para instalar en el nodo que hará de estación base. El objetivo es que la estación base recoja el tráfico que nosotros deseemos y que lo envíe al puerto Serie de nuestra máquina.

Hasta ahora hemos visto que TinyOS trabaja con Python, C++ y nesC. Sin embargo, será código Java el que se ocupe de procesar el tráfico de la red.

Supongamos que tenemos instalado en el nodo base la aplicación *BaseStation* (nesC). Sin necesidad de entrar en más detalle, nos basta con saber que esta aplicación envía por radio todos los paquetes que le llegan por el puerto Serie y por el puerto Serie todos aquellos paquetes que recibe por radio. Esto soluciona la primera parte del problema, pero seguimos necesitando leer los datos que recibimos del puerto Serie. Aquí es donde entra un programa como *Listen* (Java) que recoge los datos y los muestra por pantalla.

```
public class Listen
{
        public static void main(String args[]) throws IOException
        {
                String source = null;
                PacketSource reader;
                ...
                reader = BuildSource.makePacketSource(source);
                try
                {
                        reader.open(PrintStreamMessenger.err);
                        for (;;) {
                        byte[] packet = reader.readPacket();
                        Dump.printPacket(System.out, packet);
                        System.out.println();
                        System.out.flush();
                 }
                ...
        }
```

```
}
```

Fig 13. Fragmento de Listen.java, donde source deberá ser /dev/ttyUSB1.

Tal y como se puede apreciar, el código es sencillo y compacto. Un rápido vistazo sirve para ver que contamos una clase que lee los paquetes directamente del puerto, y Listen.java se limita a llamarla y a volcar el contenido en la salida estándar. El problema está en el formato de presentación de los datos.

*00 FF FF 00 00 04 22 06 00 02 00 01*

Fig 15. Resultado de Listen.java

Mayormente, Listen sólo se utiliza para comprobar que estamos recibiendo algo por el puerto Serie. En el caso que quisiéramos trabajar con los mensajes recomendamos encarecidamente utilizar una clase como MsgReader junto con MIG.

MIG (Message Interface Generator) es una herramienta para parsear los paquetes de forma automática. Dada una secuencia de bytes el código generado por MIG parsea cada uno de los campos en el paquete y además provee una serie de métodos estándar que acceden y modifican los campos con el fin de mostrar los paquetes recibidos e incluso generar paquetes nuevos. En nuestro caso sería suficiente modificar el Makefile de la aplicación tal que así:

```
COMPONENT=FireDetectionAppC
BUILD_EXTRA_DEPS += FireDetectionSample.class
FireDetectionSample.class: FireDetectionSample.java
        javac FireDetectionSample.java
FireDetectionSample.java:
        mig java -target=null -java-classname=FireDetectionSample
FireDetection.h FireDetectionSample -o $@
```

```
include $[MAKERULES]
```

Fig 16. Makefile modificado para usar MIG

Ahora que ya hemos generado la clase FireDetectionSample.java podemos utilizarla para que un lector como MsgReader parsee los paquetes.

```
String className = args[i];
try
{
        Class c = Class.forName(className);
        Object packet = c.newInstance();
        Message msg = (Message)packet;
        v.addElement(msg);
}
```

Fig 17. Extracto de MsgReader.java

El código es muy parecido al de la clase Listen, con la diferencia que vemos en la figura 16: toma la clase del paquete de uno de los argumentos del programa y crea una instancia para trabajar con ella.

Por motivos de diseño del sistema operativo dos programas no pueden escuchar del puerto Serie a la vez, por eso TinyOS recomienda utilizar la herramienta SerialForwarder.

SerialForwarder es un programa en Java que tal y como su nombre indica propaga los paquetes que recibe a los clientes que tiene. La idea es conectar SerialForwarder al puerto Serie, y a su vez conectar todos aquellos programas que necesiten leer de ese puerto al SerialForwarder. Éste mantiene una lista de clientes conectados y cada vez que recibe un paquete lo propaga.

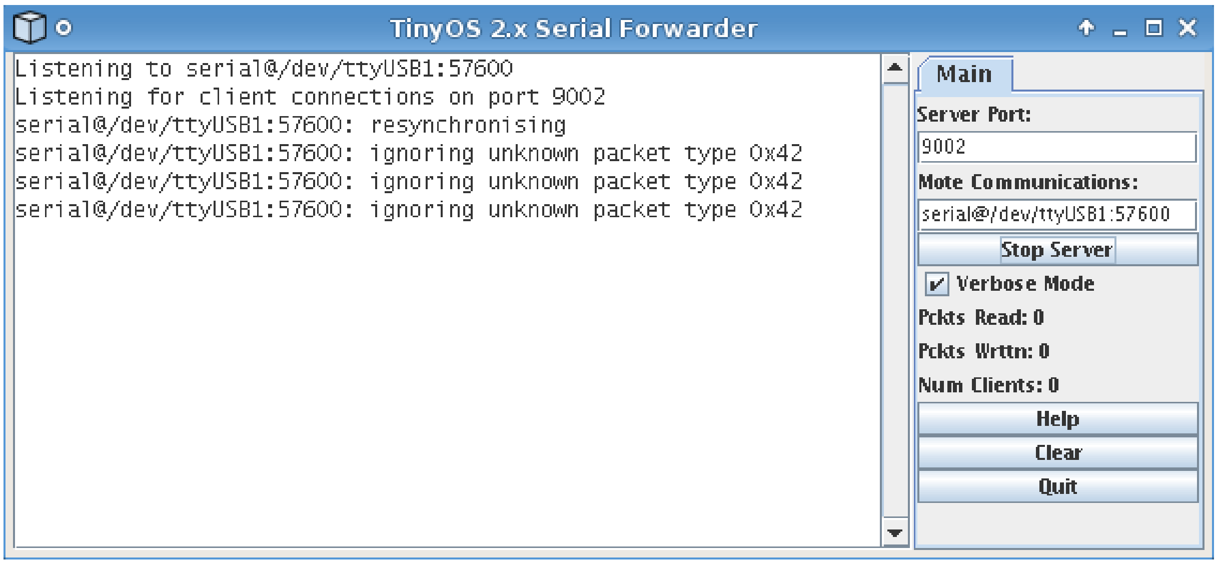



Fig 18. SerialForwarder

En el caso de que ahora quisiéramos conectar el MsgReader haríamos lo siguiente:

*java net.tinyos.tools.MsgReader -comm sf@localhost:9002 FireDetectionMsg*

E incluso es posible conectar un SerialForwarder tras otro, aunque la utilidad de hacerlo sea discutible.

Respecto a nuestras opciones y posibilidades de comunicación en el entorno de Crossbow en comparación con lo que hemos visto hasta ahora en XubunTOS son muy limitadas. Por defecto, aplicaciones como MsgReader y SerialForwarder no están instaladas. Ahora bien, en su lugar contamos con la herramienta XSniffer que a más alto nivel nos permite monitorizar el tráfico de la red. Naturalmente, al subir de nivel perdemos la posibilidad de introducir parseos personalizados de los paquetes que nos muestra XSniffer.

# 5. El Incendio Forestal

El fuego es un elemento más que pertenece a la naturaleza, que lo ha utilizado como una herramienta para modelar su cara, marcando y condicionando la distribución de especies y su extensión en el territorio.

Tradicionalmente, el hombre ha utilizado el fuego para reconstruir su entorno, empleándolo en la agricultura, ganadería o en actividades forestales como por ejemplo en las quemas de restos, rastrojos, etc.

Ahora bien, cuando el fuego escapa al control se produce el incendio, y cuando éste pasa al monte se produce el incendio forestal.

La Ley de Montes vigente actualmente dice textualmente en su artículo 6),k):

"Incendio Forestal: el fuego que se extiende sin control sobre combustibles forestales situados en el monte."

Y su artículo 5:

"Concepto de monte: a los efectos de esta Ley, se entiende por monte todo terreno en el que vegetan especies forestales arbóreas, arbustivas, de matorral o herbáceas, sea espontáneamente o procedan de siembra o plantación que cumplan o puedan cumplir funciones ambientales, protectoras, productoras, culturales, paisajísticas, o recreativas.

Tienen también la consideración de monte:

a) Los terrenos yermos, roquedos y arenales.

b) Las construcciones e infraestructuras destinadas al servicio del monte en el que se ubican.

c) Los terrenos agrícolas abandonados que cumplan las condiciones y plazos que determine la comunidad autónoma, y siempre que hayan adquirido signos

inequívocos de su estado forestal.

d) Todo terreno que, sin reunir las características descritas anteriormente, se adscriba a la finalidad de ser repoblado o transformado al uso forestal, de conformidad con la normativa aplicable."

De lo que se desprende que el incendio forestal es el fuego que se propaga sin control sobre el terreno forestal, quemando vegetación que no estaba destinada a arder (Vélez, 2000).

En nuestro país, el único fuego que tiene su origen en la naturaleza es el que provocan los rayos. Tanto éstos como los provocados por el hombre, por descuido o intencionadamente, cuando las condiciones meteorológicas son favorables para ello, se extienden con virulencia y rapidez causando grandes daños.

En el momento en que un foco de fuego se convierte en un incendio, se produce una secuencia de sucesos, que es en definitiva una suma de tiempos, desde que empieza hasta que acaba.

1º Empieza => t1

2º Se ve => t2

3º Se transmite => t3

4º Alarma => t4

5º Se moviliza => t5

6º Se llega => t6

7º Se combate => t7

8º Se acaba => t8

El objetivo que pretendemos lograr mediante la implantación de una red de sensores en el monte es reducir al mínimo el tiempo de detección y prácticamente eliminar el intervalo de tiempo que se produce entre que se detecta el fuego y se da la alarma.

## *5.1 El Triángulo del Fuego*

Para que la combustión ocurra hay tres elementos que deben estar presentes, y además deben hacerlo en la proporción adecuada. En concreto hablamos de: calor, oxígeno y combustible.

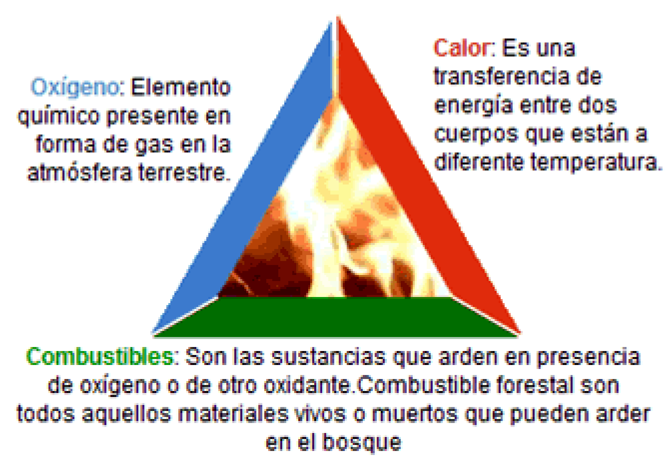



Fig 19. Triángulo del fuego

El oxígeno es uno de los elementos que forman parte de nuestra atmósfera, en una proporción de un 21%, y en contra de la creencia popular, no es necesario eliminarlo por completo para romper el balance del triángulo del fuego, sino que basta con reducir su presencia al 14%. A medida que vaya disminuyendo la proporción de oxígeno la combustión se va retardando hasta llegar a desaparecer por completo.

El calor es una forma de energía y la energía es átomos en movimiento. Sabemos que todos los elementos y materiales están compuestos de partículas en constante movimiento. Ahora bien, cuando aplicamos calor a un combustible las partículas que lo componen empiezan a moverse cada vez más y más rápido. Cuando el combustible alcanza una determinada temperatura (punto de vaporización) empieza a liberar

gases: vapor de agua primero y a continuación otros gases altamente inflamables. En ese momento, si acercáramos una cerilla serían los gases los que arderían, en lugar del propio combustible.

La temperatura de ignición de un combustible es la temperatura a la cual estos gases arderán en llamas y provocarán que continúe la combustión incluso después de que el foco del calor exterior, como una cerilla o un encendedor, sea retirado. La temperatura de ignición de muchos de los combustibles puede variar entre los 260 y los 400 grados centígrados.

El calor desprendido por un combustible ardiendo se mide en calorías y una caloría es el aumento de calor requerido para que un gramo de agua aumente un grado centígrado su temperatura. Así se puede entender que al hablar de un fuego podamos decir que está produciendo más calor en una zona que en otra, a pesar de que la temperatura de combustión de las dos zonas sean aproximadamente las mismas. Por ejemplo, supongamos que un fósforo representa una caloría, entonces una caja de 50 cerillas representa 50 calorías, pero en ambos casos, la temperatura de combustión será la misma. Sin embargo, la caja arderá más intensamente que una simple cerilla, porque produce más calorías.

Finalmente, un fuego no existe si no hay material para quemar, es decir, combustible. E incluso aunque haya combustible, dependiendo de sus características éste podrá arder o no. Nos referimos más concretamente a la forma, el contenido de humedad, el volumen y la superficie ocupada, así como otras características que analizaremos más adelante, pero por ahora es importante recordar que el combustible no arde sin la presencia de calor y oxígeno en la proporción adecuada.

Cuando un combustible humea (sin llama) se debe normalmente a la falta de oxígeno suficiente, o a que el contenido en humedad del mismo es tal que todo el calor se emplea en evaporar el agua y no se liberan gases inflamables, retardando de esta forma el proceso de oxidación. Una vez empezado, este proceso de combustión

continuará mientras haya combustible, calor y oxígeno en proporciones adecuadas.

Si los elementos del triángulo se encuentran en proporciones adecuadas el proceso de combustión será completo y el incendio no dejará restos. Ahora bien, si se rompe el balance la combustión será incompleta. Un humo marrón oscuro o negro es signo de combustión incompleta, indicador habitual de combustibles pesados. En bosques densos y espesos la combustión puede ser incompleta debido a factores variados como un contenido de humedad elevado del combustible, o un aporte de oxígeno limitado. Es en estos casos en los que estamos desprotegidos, porque los combustibles víctimas de combustiones parciales tienen lo que se denomina potencial de retorno. Así que, lo que un momento puede parecer un foco contenido o extinguido al siguiente podemos encontrarnos con un foco vivo simplemente porque las condiciones del triangulo han cambiado y ahora se encuentran en la proporción adecuada. Decimos que nos encontramos desprotegidos, porque una vez se hayan quemado los sensores no podremos detectar un rebrote del fuego.

## *5.2 Propagación del Calor*

Algunas veces el oxígeno se consume tan rápido en la base del fuego que disminuye su proporción, y ésta pronto está fuera del balance, aunque quedan todavía suficiente calor y vapor. A medida que los gases calientes ascienden en el aire vuelven a saturarse de oxígeno y pueden aparecer llamas en la columna de humo muy por encima del fuego. Está claro que el aporte de oxígeno para la combustión depende de la capacidad del aire para circular libremente entre los combustibles, aunque también el calor debe poder llegar al combustible para elevar su temperatura a la de ignición.

Debemos distinguir tres formas distintas de transmisión de calor:

- radiación

- conducción
- convección

Pero además, en el caso de los incendios forestales debemos considerar también:

- pavesas (que saltan o ruedan)

### *5.2.1 Radiación*

La radiación es la transferencia de energía calorífica a través del espacio, por medio de ondas, como las de energía calorífica que recibe la tierra de los rayos del Sol.

Los combustibles que están muy cerca del fuego se calientan por radiación, pero según nos vamos alejando la cantidad de energía irradiada disminuye y disminuye más deprisa de lo que nos separamos.

De hecho, ésta es la principal forma de transmisión de calor que hemos escogido para detectar incendios. Los sensores de temperatura de los nodos detectarán cambios en la temperatura del aire que los rodea, por eso a la hora de distribuir los nodos en el monte deberemos tener en cuenta la máxima distancia hasta el fuego donde aún podemos percibir el incremento de temperatura debido a la radiación.

### *5.2.2 Conducción*

La conducción es la transferencia de energía calorífica a través de una sustancia por acción atómica directa, es decir, pasa de un átomo a los que tiene al lado, y así sucesivamente. Por ejemplo, si colocamos el extremo de una rama junto a una llama y

cogemos la rama por el otro extremo no notaremos el calor, pero si cambiamos la rama por un trozo de metal no tardaremos en notarlo. Así pues, debido a que la madera es un mal conductor este método de transferencia es el de menor importancia en cuanto a lo que incendios forestales se refiere.

### *5.2.3 Convección*

La convección es un método de transmisión de calor a través de masas de fluidos, como por ejemplo el ascenso de gases calientes. El aire caliente sube porque es más ligero que el aire frío, y el aire frío va a llenar el vacío dejado por el aire caliente que está en ascensión.

En el caso de que aún quedará alguna duda respecto al poder de transmisión de calor por convección veamos el siguiente ejemplo. Si situamos un papel encima de una llama sin tocarla, éste se calienta hasta llegar a su punto de ignición y empieza a arder. Sin embargo, si colocamos el papel a un lado de la llama, en un lateral, el papel arde con mayor dificultad o simplemente no arde. Esto se debe a que en el último caso el papel sólo recibe calor por radiación, y si no arde es porque no recibe suficiente energía como para llegar a la temperatura de ignición.

### *5.2.4 Pavesas*

Podemos considerar como pavesa el fenómeno que describe el transporte de puntos de ignición. Básicamente, podemos distinguir dos formas de transporte:

- Pavesas volantes: puntos de fuego provocados por pavesas que se mueven por la

convección, por delante del frente de avance del incendio. Por ejemplo, ciertas especies pirófilas, como el eucalipto, se desprenden de las hojas ante el fuego para que éstas sean arrastradas por corrientes de convección y puedan propagar el incendio.

- Pavesas rodantes: se dan cuando tenemos un incendio combinado con un cierto desnivel. Este podría ser el caso de piñas que explotan a consecuencia del calor para precipitarse a continuación colina abajo.

## *5.3 Tipos de Incendios*

### *5.3.1 Incendios de Superficie*

Son aquellos incendios en los que se queman los combustibles que hay en el suelo en una altura próxima: pastos, matorrales, repoblaciones jóvenes, pequeños arbustos, restos de cortas, despojos, etc.

Estos combustibles son los que propagan el fuego de unos árboles a otros y normalmente no afecta a los árboles que se encuentran en medio. Con esto no queremos decir que si hay continuidad vertical no se quemen las copas, pero cuando desaparezca esa continuidad el fuego volverá a bajar al suelo y se seguirán quemando los combustibles bajos. Se trata del fenómeno conocido como coronamiento, y no debe confundirse nunca con el fuego de copas.

### *5.3.2 Incendios de Copas*

Todo incendio que se propague a través de las copas de los árboles se conoce como incendio de copas.

Este tipo de incendios suele tener dos frentes de avance. Primero se queman las copas, ya que el viento es más intenso que a nivel de suelo y las hojas son combustibles finos (ligeros) que arden bien, especialmente si contienen aceites o resinas como ocurre en las especies de Eucalipto y Pino respectivamente. Después, por detrás, se van quemando los combustibles superficiales, como matorrales o pastos.

Se trata de un tipo de incendio de muy alta intensidad y que se produce en masas arboladas cerradas con continuidad de combustible, tanto horizontal como vertical, en los que es necesaria la presencia de fuerte viento ya que en cuanto amaina el fuego suele dejar las copas y pasar a los combustibles superficiales para seguir avanzando.

### *5.3.3 Incendios de Subsuelo*

Son fuegos que progresan por debajo del suelo, quemando raíces y tallos subterráneos. Se trata de incendios muy lentos debido a la poca disponibilidad de oxígeno, que no tienen llama y que prácticamente no desprenden humo, lo que los hace extremadamente difíciles de localizar.

No son muy frecuentes pero hay especies que debido a sus características tienen una propensión a presentar este tipo de fuego, como las turberas o la gayuba, por ejemplo.

Son fuegos incómodos, ya que su control es difícil porque están bajo tierra y en cuanto afloran a la superficie encuentran oxígeno y cambian su comportamiento, imitando al incendio de superficie.

## *5.4 Comportamiento del Incendio*

Hay ciertas variables a tener en cuenta en el comportamiento de un incendio que podremos extraer de las muestras que nos lleguen de la red de sensores. El objetivo es proporcionar información a los equipos de extinción para facilitarles el trabajo e intentar reducir el impacto del fuego.

### *5.4.1 Velocidad de Propagación*

La velocidad de propagación es la tasa de incremento del incendio y se mide en función de tres variables:

- Propagación lineal: es la medida de avance lineal en una sola dirección. Es interesante porque nos permite calcular el tiempo que tardará en llegar el frente del incendio desde donde se encuentra a puntos alejados de él. Es una medida sobre una distancia en un tiempo determinado: metros por minuto, kilómetros por hora, ...
- Propagación perimetral: es el aumento de perímetro. Lo necesitamos para estimar la longitud de las líneas que tenemos que construir y la longitud que tenemos que apagar. Se mide también en metros por minuto o kilómetros por hora.

- Propagación en superficie: es la superficie afectada en hectáreas por hora. Nos indica el ritmo de crecimiento del daño.

De todas estas variables, la más apreciada en la lucha contra incendios es la velocidad de avance o propagación lineal.

| Apreciación de Velocidad | Velocidad real (m/min) |
|---|---|
| Lenta | 0 - 2 |
| Media | 3 - 10 |
| Alta | 11 - 70 |
| Extrema | > 70 |

Fig 20. Categorización de la velocidad de avance de un incendio.

### *5.4.2 Forma del Incendio*

Al mismo tiempo que el fuego avanza hacia delante también se propaga por los flancos y por detrás, de tal forma que área y perímetro cambian constantemente. Así pues, información precisa sobre la forma del incendio es necesaria para decidir la mejor forma de organizar los recursos para su extinción.

Una vez empieza el fuego, las llamas van extendiéndose a su alrededor formándose la línea perimetral, encerrando en su interior la superficie quemada.

Si el terreno fuese llano, no hubiera viento y la vegetación fuese la misma la forma del incendio sería circular, ya que las llamas avanzarían por igual en todos los sentidos.

Cuando hay una pendiente claramente marcada o hace un viento con dirección constante el incendio adquiere forma elíptica y el fuego tiene distinta intensidad y velocidad según el punto del perímetro en que nos situemos, por lo que es importante conocer en qué parte nos encontramos.

Debemos distinguir las siguientes partes:

- Borde: línea perimetral, que en algunos lugares presentará llama y en otros no.
- Cabeza: extremo más activo, donde el fuego avanza con más intensidad y más rápidamente.
- Flancos: bordes laterales, con llamas.
- Cola: extremo donde el fuego progresa con más lentitud.

Cuando la topografía del terreno es irregular y/o la vegetación es variable, o la dirección del viento es errática, el incendio adopta formas variables, progresando más en aquellas zonas donde las condiciones de propagación son más favorables y adoptando una forma compleja o irregular. Entonces pueden darse situaciones en las que aparezcan dedos o lenguas de frente en los que el avance es mayor que en bolsas y entrantes, en los que el fuego se desenvuelve con mayor dificultad.

La velocidad de propagación y la forma que adquiere nos va indicando si las labores de extinción permiten controlar el incendio, o si por el contrario el fuego progresa más deprisa de lo que se apaga y por tanto requiere otro tipo de equipos más rápidos y potentes, o más medios que ayuden a contener su velocidad de avance, …

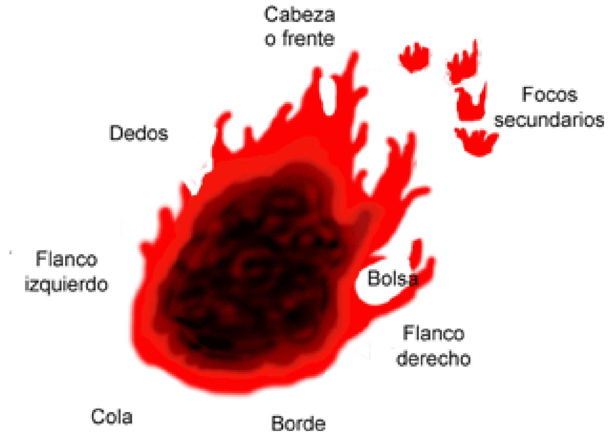



Fig 21. Partes de un incendio.

## 5.5 Primeras Conclusiones

Una vez terminada la caracterización detallada de los incendios forestales debemos extraer conclusiones y adaptar la red de sensores para que sea lo más útil posible.

Dependiendo del tipo de bosque que queramos monitorizar tendremos una configuración u otra. Por ejemplo, si queremos instalar la red en un bosque sin demasiada continuidad horizontal, pero con mucho matorral y plantas herbáceas, podremos centrarnos en la detección de incendios de superficie, porque serán los más frecuentes. Ahora bien, si por el contrario trabajamos con un bosque frondoso con continuidad horizontal deberemos concentrarnos en incendios de copas.

La diferencia entre especializar la red para detectar un tipo de incendio u otro es la ubicación de los sensores. En el caso de que trabajemos con incendios de superficie lo ideal sería colocar los nodos a un metro y medio del suelo y beneficiarnos así del calor transmitido por radiación así como por las columnas de convección que pudieran levantarse en su proximidad. Por el contrario, si nuestro objetivo es detectar incendios de copas y colocamos los sensores por debajo de la altura de la copa del árbol el sensor recibirá el calor transmitido por radiación, pero quedará fuera de la columna de convección, con lo que tardaremos más en detectar el fuego. Así pues, lo conveniente sería colocarlo a una altura media de la copa, dependiendo del tamaño cada árbol.

Tal y como acabamos de ver, el proceso de distribución de los nodos en el bosque es algo que se debe hacer con atención y con minuciosidad, completamente contrapuesto a la idea de una siembra aleatoria desde dispositivos aéreos, por ejemplo.

Además, para poder interpretar mejor los datos que recibamos de la red sería útil llevar un registro de los sensores instalados y de su posición, ya que las placas

MTS400 no cuentan con dispositivo de posicionamiento global. Naturalmente, no es necesario registrar todos y cada uno de los sensores, todo depende de la precisión con la que queramos trazar el perfil del incendio. Una medida aproximada sería tomar nota de un nodo cada 500 metros o cada kilómetro.

Con ese registro también sería posible conocer la distancia real que hay entre dos nodos y eso nos daría la posibilidad de calcular con precisión la velocidad de propagación lineal del incendio.

Al margen de todas las bondades que tiene la red, debemos tener muy presente que los sensores no son ignífugos, concretamente, recordemos que la placa deja de funcionar cuando llega a los 60° C. Por tanto, la red no podrá detectar rebrotes de incendios (los combustibles con potencial de retorno), aunque esto no resulta un verdadero problema porque la atención y los medios del Cuerpo de Bomberos ya están puestos en la zona.

## *5.6 Combustión de Combustibles*

La combustión de la materia orgánica no es más que una oxidación muy rápida, en la que la energía se desprende en forma de luz y calor. En términos generales lo que diferencia la combustión de la simple descomposición aerobia de la materia orgánica es la velocidad con la que se lleva a cabo la oxidación.

La madera se compone principalmente de celulosa y lignina, además de una gran cantidad de resinas y aceites esenciales, que tienen comparativamente un alto valor calorífico. En su combustión podemos distinguir cinco etapas:

- En la primera fase se produce el calentamiento de la partícula debido a la combustión de partículas adyacentes.
- Al llegar a los 100º C empieza a evaporarse el agua contenida en el combustible.

- A partir de los 150º C aumenta el proceso de descomposición de la madera con la consiguiente liberación de gases inflamables y carbonización.
- Entre los 300 y los 500º C se produce la reducción del proceso de descomposición en la partícula carbonizada.
- Finalmente, entre 500 y 1000º C se da la combustión de carbón con la consiguiente liberación de CO y $CO_2$.

Está distribución en etapas resulta especialmente interesante porque podemos ver que la liberación de $CO_2$ no se produce hasta llegar a la última fase de la combustión. Creemos que es interesante porque en el diseño de la red de sensores hubo momentos en que dudamos de la efectividad de los sensores de temperatura y nos planteamos cambiarlos por sensores de $CO_2$, aunque nos vimos disuadidos de inmediato por su elevado coste y su voluminosidad. Además, ahora deberíamos añadir a la lista de inconvenientes que si un combustible libera $CO_2$ es porque antes ha experimentado un incremento de temperatura, con lo que para minimizar el tiempo de detección es mejor detectar temperaturas altas que niveles de $CO_2$ elevados.

También debemos tener presente que no todos los combustibles forestales contienen la misma energía, e incluso comparando ejemplares de la misma especie y de similar tamaño nos encontramos con grandes diferencias. Si nos fijamos en la especie Pinus Silvestris, las acículas tienen un poder calorífico de 5190 Kcal/kg y la resina de 8300 Kcal/kg, por lo que su contenido influye en el poder calorífico de la madera. Ezupov (1934) descubrió que el poder calorífico de la madera de pinos sanos, con un contenido de resina normal (3.4%), era de 4872 Kcal/kg mientras que en individuos afectados por el hongo Peridermium pini, que provoca que el árbol secrete más resina como medida defensiva, situó el nivel de resina en un 45.3% y el poder calorífico aumentó hasta 6253 Kcal/kg.

## *5.7 La Humedad en los Combustibles*

### *5.7.1 Efectos en la Combustión*

El fuego puede iniciarse y establecerse sobre los vegetales debido a su capacidad para inflamarse y desencadenar el proceso de combustión. Tanto la inflamabilidad como la combustibilidad dependen de una serie de propiedades térmicas y características físicas y químicas de las partículas individuales de combustible y del lecho en su conjunto, algunas de las cuales fueron mencionadas anteriormente. Entre estas características hay una que destaca por encima de las demás: la humedad. En los combustibles el agua se puede encontrar de tres formas distintas:

- agua de constitución: forma parte de los componentes químicos del combustible.
- agua higroscópica (de imbibición, de impregnación o agua absorbida): se encuentra unida a las paredes celulares mediante enlaces químicos.
- agua libre: rellena poros, conductos y cavidades internas e impregna la superficie externa de los combustibles.

Las dos últimas son las de mayor interés para valorar la influencia de la humedad en el comportamiento del fuego y en el peligro de incendio ya que varían a lo largo del tiempo mientras que el agua de constitución permanece constante. Así pues, cuando hablemos de la humedad del combustible nos referiremos a la proporción que supone el agua libre y la higroscópica, que se expresa habitualmente en porcentaje respecto a peso seco.

Cuando un foco externo de calor actúa de manera continua sobre una partícula combustible la temperatura de su superficie se va incrementando conforme pasa el tiempo y se puede llegar a alcanzar el punto de ignición. En ese preciso momento, los

gases inflamables que el combustible ha ido liberando durante la pirólisis reaccionan con el oxígeno del aire y pueden producir llama. Ahora bien, cuanto mayor sea la humedad del combustible, mayor será la cantidad de calor necesaria para alcanzar la temperatura de ignición. Así pues, La humedad retarda la ignición de formas muy diversas pero fundamentalmente actúa incrementando la capacidad de calor y la conductividad térmica del combustible.

El incremento de la capacidad de calor implica aumentar la resistencia del combustible a elevar su temperatura. El calor necesario para que un combustible consiga elevar su temperatura, desde la temperatura ambiente hasta alcanzar su punto de inflamación, consta de los siguientes componentes:

- calor absorbido por el agua contenida en el combustible hasta alcanzar los 100º C.
- calor de vaporización del agua.
- calor de calentamiento de la materia anhidra o seca.

Vemos que si se incrementa la humedad del combustible al mismo tiempo se estarán incrementando los dos primeros componentes.

El hecho de que aumente la conductividad térmica implica que el calor que está entrando en el combustible se difunde más fácilmente hacia el interior en vez de concentrarse en la superficie. En consecuencia, se ha de absorber más calor para que la capa superficial del combustible alcance la temperatura de ignición. Por tanto, cuanto mayor es la humedad del combustible el foco de calor ha de estar actuando durante más tiempo sobre el mismo hasta lograr la inflamación. Además, no sólo conseguimos retardar el tiempo de ignición sino que también la probabilidad de ignición disminuye porque puede ser que el foco externo de calor se apague antes de inflamarse el combustible, con lo que no se produciría el incendio.

Si al suprimirse el foco externo de calor el proceso de pirólisis se mantiene a sí mismo

la combustión está en marcha. La combustión se puede interpretar como una cadena de igniciones de capas cada vez más profundas del combustible de forma que la humedad afecta a la combustión de manera similar a la forma en que actúa sobre la ignición. Sin embargo hay que añadir el efecto del vapor de agua como diluyente de los gases inflamables que se están liberando durante la pirolisis. Si los gases están demasiado diluidos en vapor de agua no se pueden inflamar y no prosigue la combustión. Se denomina humedad de extinción al contenido en humedad máximo que puede tener un combustible para que se pueda mantener la combustión con llama. La humedad de extinción depende del estado del combustible (vivo o muerto), el grosor y la composición química del combustible pero también de la intensidad del incendio y de la disponibilidad de oxígeno. En cuanto al grosor, la humedad de extinción es mayor cuanto más grueso es el combustible y también es mayor cuanto más rico sea el vegetal en sustancias volátiles. Para la mayoría de los combustibles vivos varía entre el 120 y 160%, mientras que para los muertos oscila entre el 25 y el 40%. Naturalmente, hay excepciones: en pastos secos se han registrado humedades de extinción del 12%, es decir, que por encima de un 12% de contenido en agua ya no se puede mantener la llama. En el otro extremo, la humedad de extinción de acículas vivas de algunas coníferas muy resinosas alcanza el 200%. La combustión sin llama se puede mantener con humedades de los combustibles muertos de hasta un 135%.

Cuanto mayor sea la humedad, mayor será la proporción del calor desprendido durante el proceso de combustión que se invierte en evaporar agua y que por tanto no se emplea en calentar el propio combustible.

### *5.7.2 Transmisión de Calor*

Para el desarrollo y propagación del fuego no es suficiente con que se produzca la ignición y se establezca la combustión sobre un elemento individual, es necesario además que el calor se transmita hacia otros combustibles cercanos. De las tres formas fundamentales de transmisión de calor la humedad afecta sobre todo a la radiación y a la conducción.

A partir del 10% de humedad en los combustibles muertos el vapor de agua reduce la transmisión de calor por radiación. Esta pérdida de capacidad del aire para transmitir calor por radiación tiene especial importancia en los primeros minutos del desarrollo de un incendio, cuando todavía no se ha establecido la convección y puede ser la causa de que un fuego incipiente es apague.

En cuanto a la transmisión de calor por conducción es cierto que la conductividad del material se incrementa con la humedad. Sin embargo, al mismo tiempo aumenta la resistencia del combustible a aumentar su temperatura (capacidad de calor). En general, cuanta más agua tiene el combustible mayor será el calor necesario transmitir entre dos puntos de un elemento o de elementos adyacentes para mantener la combustión por conducción.

### *5.7.3 Disponibilidad del Combustible*

Cuanto menor sea la humedad del combustible mayor será la carga disponible para arder y consumirse en un lugar y momento determinado.

Para que un combustible leñoso arda debe tener un contenido de humedad menor o igual al 25%. Cuando un combustible está en posibilidad de arder y por tanto por

debajo de ese porcentaje se le conoce como combustible disponible. Por el contrario, si el contenido en humedad es elevado recibe el nombre de no disponible.

### *5.7.4 Variación de la Humedad*

La humedad de los vegetales evoluciona de forma muy distinta y alcanza niveles muy diferentes en función del estado de la planta. En los restos se dan variaciones mucho más rápidas que en los vegetales vivos y el rango de variación es mayor. No se pueden interpretar las consecuencias que un determinado valor de humedad de combustible puede tener en el comportamiento del fuego si no especificamos el estado. Por ejemplo, un 70% de humedad sería un valor muy alto para combustibles muertos pues superaría con creces la humedad de extinción y sin embargo sería una cifra muy favorecedora para la propagación del fuego si se refiere a combustibles vivos.

#### *5.7.4.1 Combustibles Muertos*

Para la elaboración de factores de riesgo de incendio nos centraremos en el análisis de combustibles muertos. Principalmente, porque el contenido en humedad de los restos muertos está muy influenciado por las condiciones atmosféricas y es posible calcularlo a partir de ellas. Sin embargo, todas las técnicas que determinan la humedad en combustibles vivos requieren una recogida de muestras previa con lo que el sistema perdería parte de su autonomía.

Los combustibles forestales muertos, ya sean leñosos o herbáceos, están inmersos en un proceso continuo de variación de humedad en el que se alternan ciclos de humedecimiento y de secado. Esta variación es tan importante que a lo largo de un sólo día los combustibles finos y muertos de la superficie pueden recorrer todo el rango de humedades acontecidas a lo largo de una estación completa.

Los cambios de humedad de los combustibles forestales muertos son controlados principalmente por los siguientes procesos naturales:

- la precipitación en forma de lluvia, nieve o niebla que actúa humedeciendo directamente el combustible, pero también indirectamente la humedad ambiental y la del suelo.
- la condensación de agua sobre la superficie del combustible.
- el intercambio de agua en estado de vapor entre la atmósfera y el combustible.

Aparte de estos tres fenómenos naturales hay que añadir que el suelo puede ser una fuente importante de humedad hacia el combustible, ya sea porque están en contacto o bien por su proximidad. La humedad del suelo influye en la del combustible del entorno porque al irse evaporando incrementa la humedad ambiental. Además el descenso nocturno de temperatura puede llegar incluso a provocar condensación de agua sobre la superficie del combustible. De hecho, se han identificado dos formas distintas de condensación sobre los vegetales, el rocío procedente de la corriente descendente de humedad atmosférica y la llamada destilación originada por la corriente ascendente de humedad procedente del suelo.

El agua puede entrar en el combustible por adsorción de vapor de agua de la atmósfera o por absorción del agua líquida que pudiera haber en su superficie y cuyo origen está en la condensación o en la precipitación. En cuanto a la salida tenemos el drenaje, la evaporación y la desorción de vapor.

Adsorción y desorción son mecanismos complementarios que permiten el establecimiento del intercambio de vapor entre el combustible y el aire que lo rodea. La magnitud y la dirección del intercambio dependen de la diferencia entre la presión de vapor de la atmósfera en el entorno del combustible y la presión de vapor en el propio combustible. Así que cuando la presión de vapor en el interior del combustible es inferior a la de su entorno adsorbe humedad del mismo tratando de alcanzar un equilibrio. De forma contraria, si la presión de vapor en el combustible supera a la del

ambiente, éste cederá humedad en la búsqueda de dicho equilibrio. Es decir, cuando la humedad ambiental es elevada el combustible muerto capta humedad de la misma y cuando el ambiente está seco el combustible cede humedad a la atmósfera, y todo este intercambio se puede realizar debido a la naturaleza higroscópica del combustible. De las diferentes formas que existen de medir o cuantificar la humedad ambiental recurriremos, al hablar de humedad del combustible, a la humedad relativa ya que según reputados expertos es la variable que mejor indica o refleja la habilidad de la atmósfera para intercambiar vapor de agua con el combustible.

Ahora bien, la adsorción de agua no es ilimitada; cuando el contenido en humedad del combustible alcanza el punto de saturación de la fibra, que para la mayoría de los combustibles forestales se encuentra en torno al 30-35%, ya no se puede adherir más agua a las paredes celulares. La humedad de los restos puede seguir incrementándose enormemente por encima de ese porcentaje, pero a partir de él los incrementos de humedad se deben a la absorción del agua líquida que se va depositando sobre su superficie y que va rellenando poros, conductos y cavidades internas en forma de agua libre.

En sentido inverso, el agua libre drenará o se irá evaporando de la superficie de los combustibles cuando cese la lluvia o la condensación. El proceso de secado al principio es bastante rápido. El agua de la superficie pasa a la atmósfera a la velocidad potencial de evaporación marcada por el déficit de presión de vapor (diferencia entre la presión de vapor de saturación a la temperatura del agua de la superficie y la presión de vapor en el aire del entorno). Sin embargo, a medida que disminuye el contenido en humedad la pérdida de humedad por evaporación se hace más lenta ya que se ve limitada por la velocidad de difusión del agua en el interior del combustible.

En los vegetales muertos, el agua se desplaza por capilaridad a lo largo de los mismo canales y poros utilizados en vida, de forma que la velocidad de difusión se ve muy condicionada por la estructura interna del combustible y por las fuerzas de tensión

superficial.

Una vez que el combustible ha perdido toda el agua líquida, el principal proceso de secado es la desorción. El agua que se encuentra químicamente ligada a la madera va saliendo del combustible a una velocidad también regulada por su estructura y las condiciones atmosféricas. Una vez más, a medida que va disminuyendo el agua higroscópica el proceso se hace más lento pues se va incrementando la energía necesaria para eliminarla.

Se producen períodos del año en que los procesos de intercambio de vapor de agua con la atmósfera regulan completamente la variación de la humedad de los restos combustibles, más concretamente aquellos en los que se registra una ausencia de precipitaciones y de condensación. Así resulta lógico pensar que en épocas de incendios, caracterizadas en general por llevar muchos días sin llover, la humedad que poseen los combustibles puede estar totalmente supeditada a la variación de la humedad relativa a lo largo del día. Sin embargo, hay que recordar que si la condensación nocturna de agua sobre los combustibles es importante puede enmascarar el intercambio de vapor, o bien limitarlo a las horas más secas y cálidas del día.

##### *5.7.4.1.1 Humedad de equilibrio y tiempo de respuesta*

Según lo visto hasta ahora, si quisiéramos establecer un mapa de riesgo de incendio basándonos en la humedad de los combustibles sería tan fácil o difícil como tomar la humedad relativa del aire y suponer que ese mismo valor corresponde al contenido de humedad de los combustibles.

Con esta idea en mente tomamos la definición de Viney (1991) del concepto de humedad de equilibrio:

“El contenido en humedad de equilibrio de un elemento combustible bajo unas condiciones ambientales dadas es el contenido en humedad que dicho elemento alcanzaría si se dejase suficiente tiempo en esas condiciones ambientales constantes.”

Similarmente, Pyne et al. (1991) lo define como:

“el valor que alcanza la humedad real del combustible si se deja durante un intervalo de tiempo infinito expuesto a condiciones atmosféricas constantes de temperatura y humedad”.

Es sencillo apreciar que ambas definiciones destacan la necesidad de mantener constantes las variables ambientales. En realidad, en el monte, el estado de equilibrio o de balance entre el vapor de agua del combustible y de la atmósfera es una situación teórica o potencial que nunca o rara vez se alcanza. El combustible no cambia instantáneamente, sino que tarda un tiempo en responder a las variaciones de la humedad relativa y la temperatura de su entorno, y lo que sucede la mayoría de las veces es que antes de que le dé tiempo a alcanzar el equilibrio ya se han vuelto a modificar las condiciones atmosféricas.

Éste tiempo de retardo es lo que se conoce como tiempo de respuesta, y es directamente proporcional al grosor del combustible, es decir, cuanto más grueso sea el material más lento será el intercambio de vapor de agua con la atmósfera y por tanto más tiempo tardará en llegar al estado de equilibrio. Determinar el tiempo de respuesta real de cada combustible es algo que sólo se puede realizar en laboratorio bajo condiciones controladas, por lo que se hace la simplificación de considerar que los combustibles finos tienen un tiempo de respuesta de 1 hora, los intermedios de 10 o 100 horas y los más gruesos de 1000.

| Tiempo de respuesta (horas) | Nombre de la categoría | Diámetro equivalente (mm) |
|---|---|---|
| < 2 | 1 hora (1h) | < 6 |
| 2 - 20 | 10 horas (10h) | 6 - 25 |
| 20 - 200 | 100 horas (100h) | 25 - 75 |
| > 200 | 1000 horas (1000h) | > 75 |

Fig 22. Categorías de tiempo de respuesta y tamaño de partícula equivalente.

La conclusión que debemos extraer es que la idea con la empezamos el apartado es equivocada, y si asumiéramos que el valor de la humedad relativa es el mismo que el de la humedad del combustible estaríamos cometiendo un error lo suficientemente grave como para alterar los factores de riesgo que pudiéramos calcular basándonos en esos datos.

##### *5.7.4.1.2 Otros Factores de Variación de la Humedad*

La humedad del combustible muerto se incrementa con el agua de precipitación y de condensación y varía en respuesta a los cambios en la humedad relativa del aire. Pero además, hay otros aspectos del tiempo atmosférico que también contribuyen a esa variación de humedad:

- la temperatura.
- la radiación solar.
- el viento.

Dada una humedad absoluta en el aire, la humedad relativa cambia en sentido

inverso a la temperatura. Así, cuanto mayor es la temperatura menor es la humedad relativa ya que el aire es capaz de retener más agua en estado de vapor[1]. Podemos afirmar que la humedad del combustible se relaciona de manera directa con la humedad relativa y de manera inversa con la temperatura, y ya que ambas variables evolucionan a lo largo del día, la humedad del combustible hará lo propio. Habitualmente, la temperatura del aire se irá incrementando desde el amanecer hasta las primeras horas de la tarde, alcanzando entonces su máximo registro para a continuación empezar a descender y llegar a un nuevo mínimo en el siguiente amanecer.

La humedad relativa, al variar en sentido inverso a la temperatura, es máxima al amanecer y mínima a primeras horas de la tarde.

Anteriormente asignamos a los combustibles finos un tiempo de retardo de una hora lo que nos indica que responden con rapidez a los cambios ambientales. Por lo tanto, en general se acepta la idea de que los combustibles muertos finos se están secando desde primeras horas de la mañana hasta las primeras horas de la tarde, y se están humedeciendo desde las primeras horas de la tarde hasta las primeras horas de la mañana. Recordemos que en caso de lluvia u otro tipo de aporte de agua líquida no se cumplirá este patrón hasta que se haya drenado o evaporado toda el agua libre.

La temperatura de los combustibles se eleva cuando el sol incide en ellos y altera el microclima del entorno: la temperatura del aire en contacto con la superficie se incrementa y la humedad relativa disminuye, obteniendo como resultado una disminución de la humedad del combustible a causa del calentamiento solar.

[1:] *Esto se debe a que al aumentar la temperatura del aire, éste se expande debido a una mayor actividad molecular, y su volumen total aumenta. Aunque la cantidad de agua que hay sea la misma, ahora se encuentra en un mayor volumen y por tanto la humedad relativa habrá disminuido.*

Por último, el viento acelera la pérdida de agua en el combustible pero también puede ejercer un efecto contrario sobre aquellos que estén expuestos al sol debido a su acción refrigerante.

## *5.8 Modelo de Predicción de la Humedad*

Utilizaremos un modelo canadiense que además es el que utiliza el GRAF para estimar los factores de riesgo de incendio en Cataluña.

Se trata del Canadian Forest Fire Weather Index System (FWI), un sistema de seguimiento del peligro meteorológico de incendio forestal que existe a media tarde, momento en el que se considera el peligro es mayor. A pesar de que se refiere al entorno de las 16.00 horas, se obtiene a partir de información registrada a mediodía. Se basa en la estimación del contenido en humedad de tres clases de combustibles con diferentes velocidades de secado (superficial fino, mantillo superficial poco compacto y mantillo profundo) y en el efecto que junto con el viento provocaría dicha humedad en la velocidad de liberación de energía del frente de un incendio.

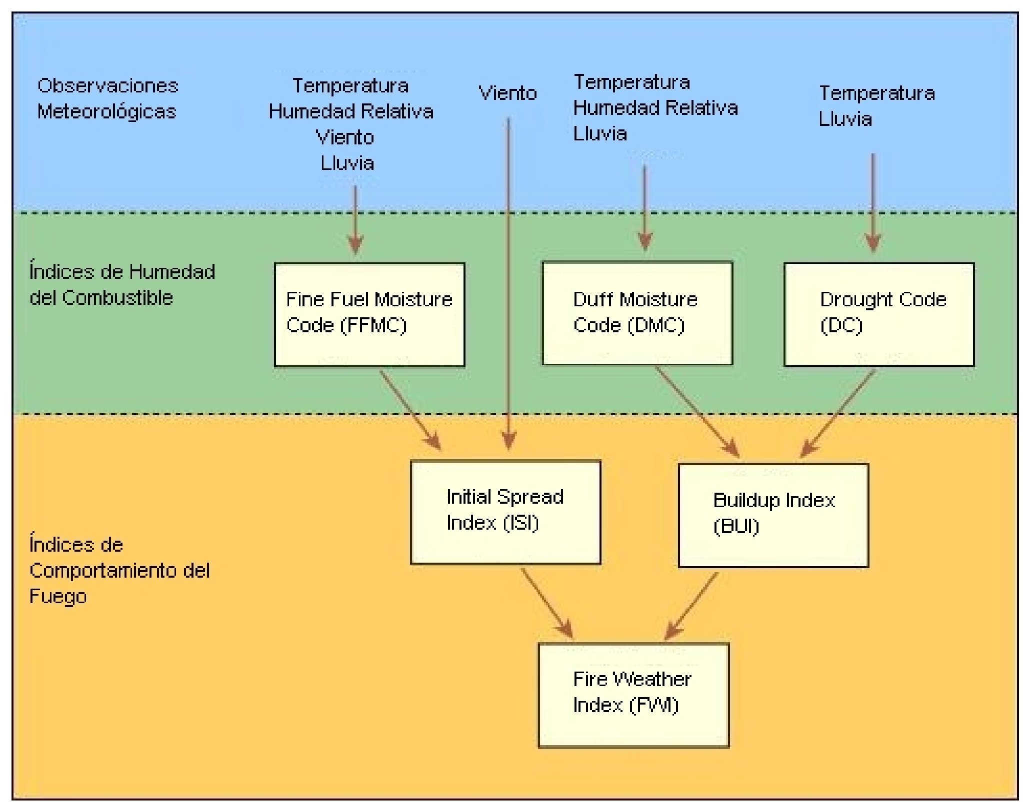



Fig 23a. Componentes del Canadian Forest Fire Weather Index System (FWI).

| | **Índices** | **Significado** |
|---|---|---|
| Índices de Humedad del Combustible | FFMC – Fine Fuel Moisture Code | Es un valor relacionado con la humedad de los combustibles finos muertos que forman parte de la hojarasca superficial umbrosa del bosque. Es un indicativo de la facilidad que hay para la ignición de dicho combustible. |
| | DMC – Duff Moisture Code | Es un valor relacionado con la humedad de la capa más superficial y menos compacta del mantillo. Refleja su disponibilidad para arder y la de los combustibles de tamaño intermedio. |
| | DC – Drought Code | Es un valor relacionado con la humedad de la capa más profunda y compacta del mantillo. Es un indicador de la importancia que puede tener la combustión sin llama en dicha capa y en la madera de grandes dimensiones. |

| | | |
|---|---|---|
| Índices de Comportamiento del Fuego | ISI – Initial Spread Index | Es un indicativo de la velocidad de propagación esperada que combina el efecto del viento y del valor obtenido para el FFMC. |
| | BUI – Buildup Index | Indica la cantidad total de combustible disponible combinando los valores del DMC y del DC |
| | FWI – Fire Weather Index | Se obtiene combinando el ISI y el BUI. Se emplea como índice general de peligro meteorológico de incendio en Canadá. |

Fig 23b. Componentes del Canadian Forest Fire Weather Index System (FWI).

Una vez calculados los índices de humedad, se obtienen, a través de ecuaciones de conversión sencillas denominadas escalas, los valores estimados para la humedad de las distintas capas de combustibles. El valor de los índices se va incrementando conforme disminuye la humedad.

| **Índice** | **Tiempo de respuesta[2] (días)** | **Profundidad de la capa (cm)** | **Carga de combustible seco (kg/m$^2$)** |
|---|---|---|---|
| FFMC | 2/3 | 1,2 | 0,25 |
| DMC | 12 | 7 | 5 |
| DC | 52 | 18 | 25 |

Fig 24. Propiedades de los combustibles asociados a los índices de humedad del FWI.

De entre todos los índices descritos trabajaremos con el Drought Code porque así nos lo recomendó el GRAF, ya que es el que ellos utilizan y realiza previsiones fiables.

*[2]: el tiempo de respuesta que figura es el correspondiente a 21,1º C de temperatura, 45% de humedad relativa, 12 km/h de velocidad de viento y al mes de julio.*

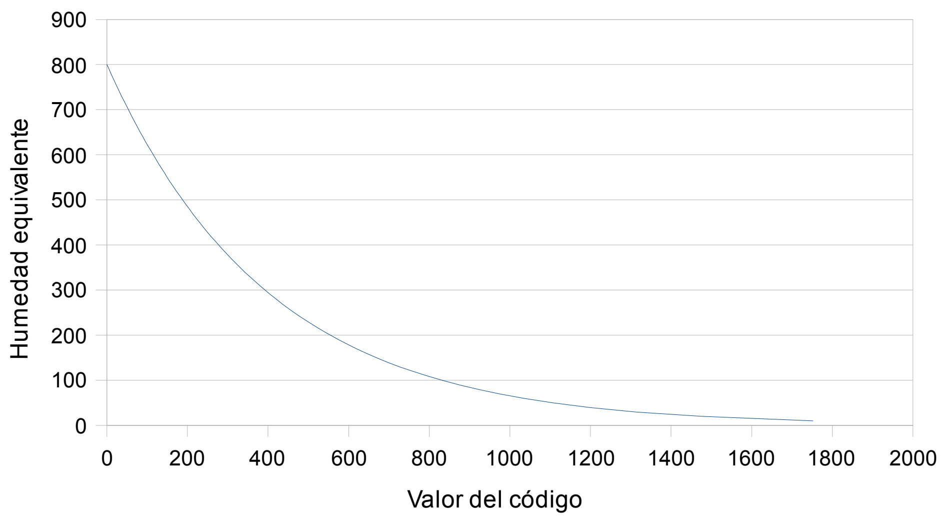



Fig 25. Escala de transformación del código DC en humedad equivalente del combustible.

En un principio el DC se ideó como un estimador de las reservas de agua de las capas superiores del suelo y no como un indicativo de la humedad de los combustibles forestales de más baja velocidad de secado. El índice, llamado inicialmente Stored Moisture Index (SMI), estimaba directamente el agua almacenada en el suelo utilizando como unidad las centésimas de pulgada y considerando un valor máximo o valor de saturación del mismo de 800 centésimas. Murano, Lawson (1970) y Van Wagner (1974) vieron que el SMI seguía, razonablemente bien, las variaciones de humedad de las capas profundas y compactas de mantillo en diversas localizaciones de Canadá y en consecuencia se decidió incorporarlo al sistema FWI. Para ello, y buscando la similitud con el FFMC y el DMC, se ideó la escala logarítmica actual en la que el valor del índice DC se incrementa al secarse el mantillo. Dicha escala queda representada gráficamente en la figura 24 y su expresión matemática es la siguiente, en la que 400 indica la humedad máxima que teóricamente puede alcanzar la capa orgánica representada por el Drought Code:

$$DC = 400 \, Ln \, (800/m)$$

La humedad del mantillo (m) se denomina en este caso humedad equivalente y

conserva las mismas unidades del SMI, es decir, las centésimas de pulgada. Como 1 pulgada equivale a 25,4 mm, cada centésima equivale a 0.254 mm y por tanto también se puede decir que la humedad equivalente se expresa en unidades de 0.254 mm.

Se considera que el mantillo asociado al DC incrementa su humedad tras la lluvia y luego se va secando día a día. Esta disminución de humedad se ha relacionado con las pérdidas por evapotranspiración[3].

A continuación presentamos las ecuaciones que nos permitirán calcular el Drought Code.

Si ha llovido con anterioridad la humedad del combustible del día anterior a la estimación ($m_0$) hay que corregirla por el efecto de la precipitación siempre que ésta supere los 2,8 mm:

$$m_r = m_0 + 3{,}937 * r_e$$

Donde $m_r$ es la humedad del combustible del día anterior corregida por la lluvia y $r_e$ es la cantidad de lluvia efectiva (la disponible para almacenarse en el suelo una vez hemos descontado la intercepción de la cubierta), se expresa en mm y se obtiene a partir de:

$$r_e = 0{,}87r - 1{,}27$$

Donde r es la cantidad de lluvia en mm acumulada durante las últimas 24 horas y medida a campo abierto a las 12:00 horas del día actual. Una vez que se ha corregido la humedad por el efecto de la precipitación se ajusta el código:

$$DC_r = 400 \, Ln(800/m_r)$$

[3]*: Se define la evapotranspiración como la pérdida de humedad de una superficie por evaporación directa junto con la pérdida de agua por transpiración de la vegetación. Se expresa en mm por unidad de tiempo.*

El código corregido por lluvia ($Dc_r$) es el nuevo valor del código del día anterior. Si resulta negativo se considerará igual a 0. Finalmente, para obtener el valor del código del día actual hay que aplicar:

$$DC=DC_r + 0,5V$$

Donde V es la evapotranspiración potencial en unidades de 0,254 mm de agua/día. En el modelo, la evapotranspiración potencial se estima a través de la siguiente relación empírica que depende de la temperatura y de la estación:

$$V = 0,36(T + 2,8) + L_f$$

T es la temperatura del aire en ºC tomada a las 12:00 horas y $L_f$ es un coeficiente que depende de la longitud del día y que por tanto varía a lo largo del año. Si $T < -2,8$ se ha de considerar $T=-2,8$.

Además, como V no puede ser negativa adoptamos la convención de si $V < 0$ consideraremos $V = 0$. Por último, ya podemos obtener la humedad del mantillo (m):

$$m=800e^{-DC/400}$$

| | Ene. | Feb. | Mar. | Abril | May. | Jun. | Jul. | Ago. | Sep. | Oct. | Nov. | Dic. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_f$ | -1,6 | -1,6 | -1,6 | 0,9 | 3,8 | 5,8 | 6,4 | 5,0 | 2,4 | 0,4 | -1,6 | -1,6 |

Fig 26. Coeficiente $L_f$ de ajuste del DC en función de la longitud del día.

Por el contrario, si no ha llovido o si $r < 2,8$ mm:

$$DC = DC_0 + 0,5V$$

y,

$$m=800e^{-DC/400}$$

# 6. Modelo Vigente de Detección de Incendios

Cuando un foco de fuego se transforma en un incendio forestal comienza una cadena de sucesos, que como vimos en el capítulo anterior, terminan convirtiéndose en una suma de tiempos.

Si las condiciones son favorables, según vaya pasando el tiempo un incendio se irá haciendo cada vez más grande y más virulento y por lo tanto más difícil de controlar. Por ello, el principal objetivo de cualquier sistema de defensa es iniciar lo antes posible las labores de extinción, para lo cual es esencial la rápida detección del fuego.

Todo sistema de vigilancia debe cumplir cuatro objetivos:

- debe ser rápido, claro y preciso, proporcionando la información necesaria para evaluar la gravedad de la alarma y poder poner en marcha y dirigir hacia ella los medios de extinción en el menor tiempo posible.
- debe proporcionar información suficiente para valorar los medios que, en principio, son necesarios para la extinción.
- debe proporcionar información periódica de cómo va evolucionando el incendio, o según se le demande.
- debe ser preventivo en las zonas de mayor tránsito de personas y de mayor sensibilidad a la aparición de incendios atribuibles a negligencias de estas personas.

Distinguimos entre vigilancia fija y vigilancia móvil. La primera es la que se realiza desde puntos de observación desde donde se divisa la superficie a vigilar, mientras que la segunda se realiza recorriendo itinerarios concretos de la zona.

## *6.1 Contenido de la Información*

Todo vigilante, fijo o móvil, cuando detecte un posible incendio debe transmitir información sobre los siguientes puntos:

- localización los más exacta posible del fuego.
- tipo de vegetación afectada.
- comportamiento del fuego si se aprecia: velocidad de propagación y longitud de las llamas u otros indicadores como la columna de humo.
- condiciones atmosféricas de la zona, indicando la dirección y la velocidad estimada del viento.
- rutas de acceso al incendio: carreteras, caminos, ...
- información de interés, como por ejemplo accidentes geográficos como roquedos, ríos o embalses en la zona.
- causa del humo si se conoce.

Que el vigilante tenga esta información siempre a mano y a punto para ser transmitida en el menor tiempo posible significa que debe conocer en detalle la zona a vigilar, así como el manejo de planos y equipos de medición como la alidada.

## *6.2 Tipos de Humos*

Todo vigilante, en el área que observa, tiene zonas que entran en su campo visual directo y otras que quedan fuera, por lo que en general es muy probable que no vea las llamas sino el humo. Por eso un buen observador debe ser capaz de determinar el tipo de humo según sea el origen, el color o la textura.

| Tipos de humos | Descripción |
|---|---|
| Por su origen: | |
| - falsos | No son humos, son, por ejemplo, polvareda por el tránsito de vehículos, de ganado, o por remolinos de aire, vapores de industrias, ... |
| - legítimos | Correspondientes a fuegos autorizados. |
| - periódicos | Como quemas de basureros o humos procedentes de chimeneas de industrias. |
| - eventuales | Quemas autorizadas de matorral, pastos, rastrojos, ... |
| -ilegítimos | Humos de procedencia desconocida, que pueden ser debidos a un incendio. |
| Por su color: | |
| - blanco | Corresponde a la quema de combustibles ligeros |
| - gris claro | El fuego afecta a combustibles de tipo medio, como matorrales pequeños |
| - gris oscuro | Se ven afectados combustibles más gruesas y pesados: matorrales grandes y árboles. |
| - amarillento | Tonalidad habitual al quemarse la resina de los árboles. |
| - negro | Indica una gran cantidad de combustible y una insuficiencia de oxígeno en el proceso de combustión. |
| Por su textura: | |
| - ligera | Poca densidad, liviano. Indica que hay poca cantidad de combustible y que además está disperso. |
| - densa | Humo espeso. Indica la presencia de gran cantidad de combustible y/o donde la combustión es muy intensa. |

Fig 27. Categorías de humo.

## *6.3 Vigilancia Fija*

Se realiza desde puntos de observación fijos en el terreno. Elevaciones desde las que se divisan grandes superficies de terreno y, por supuesto, áreas forestales.

Hay que tener en cuenta que las distancias entre unos puntos y otros son tales que se

deben ver, como mínimo, dos puntos desde cada posición.

Los puntos de vigilancia pueden ser:

- atalayas: lugar dominante sin ninguna construcción aneja. En ellas se suele situar un vigilante en algunas horas determinadas del día.
- casetas: construcción en un punto dominante con buen acceso y buena visibilidad al exterior desde ella.
- torres: construcción elevada, sobre el nivel del suelo, que se pone en lugares donde la topografía es bastante llana, o para superar la altura de la vegetación circundante.

Fig 28. Torre de Vigilancia.

Tanto en torres como en casetas, se establecen turnos de vigilancia, que pueden llegar a cubrir las 24 horas del día.

El equipamiento para el vigilante debe ser:

- prismáticos: existen modelos que incorporan una brújula para la determinación de rumbos.
- sistema de comunicación con la central y demás equipos, es decir, un radioemisor integrado en la red de comunicaciones.
- sistema de localización del fuego o el humo, como alidada o brújula.
- mapas de la zona a vigilar, con información de vegetación, carreteras y caminos, ...
- libro de registro de incidencias.

## *6.4 Vigilancia Móvil*

Consiste en recorrer, en vehículo adecuado o a pie una zona forestal concreta.

Esta vigilancia puede tener uno o varios de los siguientes objetivos:

- vigilar zonas que no se ven bien con la vigilancia fija y que son muy sensibles por sus valores excepcionales o por la presencia de personas en ellas (zonas de acampadas, áreas recreativas, ...).
- disuadir a las personas, al sentirse vigiladas y al mantenerlas informadas.
- Realizar un primer ataque a fuegos incipientes que encuentre en su recorrido.

Para ello siempre debe disponer de los elementos de vigilancia:

- prismáticos.
- emisora para las comunicaciones.
- mapas de la zona.
- brújula para tomar rumbos.

Y si tiene que realizar una primera intervención deberá disponer de:

- vehículo todoterreno, provisto de emisora.
- mapas de la zona de actuación.
- equipo de protección individual.
- herramientas para ataque a fuegos incipientes.

La vigilancia móvil tiene el inconveniente de que la observación de cada punto se realiza cuando se pasa por él, nada más, por lo que detectará el incendio sólo cuando coincida con su presencia, pero a cambio tiene la ventaja de que puede realizar otras funciones que no tiene la vigilancia fija, como por ejemplo:

- puede proporcionar información a las personas presentes en las áreas a vigilar.
- puede proporcionarse información sobre rutas de entrada y de salida a las áreas, tanto para los medios de extinción como para el resto de personas.
- puede proporcionar información muy detallada de una alarma, o del desarrollo y evolución del incendio en el que se esté realizando el primer ataque.

## *6.5 Otros Sistemas de Vigilancia*

Actualmente se efectúan tareas de vigilancia con cámaras termométricas. Consiste básicamente en observar el territorio con cámaras que tienen instalados sensores de infrarrojos, capaces de captar y detectar focos calientes en un cierto radio.

Desde hace algunos años, en Aragón se está ensayando un sistema que consiste en:

***1)*** Cámaras con sensores de infrarrojos instaladas en torres para observación.

***2)*** Sistema de transmisión de las imágenes captadas por las cámaras a un Centro de Control.

***3)*** Centro de Control con ordenadores para realizar el análisis de las imágenes y

poder determinar la gravedad de los focos detectados y activar la alarma en caso de ser necesario.

Esta tecnología se utiliza para la detección automática de incendios, con buenas o malas condiciones de visibilidad, incluso de noche, en áreas difíciles de cubrir con vigilancia fija o móvil.

## *6.6 Comparación con la Red de Sensores*

En este capítulo hemos visto como funciona un sistema de detección de incendios real, que debe ser nuestra referencia al diseñar un sistema de detección más eficaz si cabe.

En lo que se refiere a la información que obtendremos de las alarmas, si tal y como recomendamos se elabora un registro de los sensores y su localización, en el momento en que salte la alarma automáticamente tendremos la posición exacta del fuego, el tipo de vegetación de la zona, las condiciones atmosféricas de la zona e incluso las dimensiones del incendio y hacia donde avanza si la alarma procede de más de un sensor.

No tendremos problemas con alarmas por falsos humos, porque nuestro sistema trabaja con temperaturas. En cuanto a los fuegos legítimos, al conocer con suficiente precisión la localización exacta del fuego podemos saber fácilmente si justo en esa zona estaba programada una quema autorizada.

Además, gracias a la red de sensores tendremos monitorización continua del bosque, con lo que los turnos de 24 horas en cada torre de vigilancia se harán innecesarios y serán sustituidos por un sólo turno de 24 horas en una central de alarma, donde se recojan todos los datos de la red.

La vigilancia móvil se haría innecesaria, aunque su presencia en el bosque sería útil para transmitir la sensación de que los sensores están vigilados y protegidos. Se

trataría de una función disuasoria y preventiva de actos vandálicos.

Quizá deberíamos explicar porque no hemos utilizado sensores infrarrojos para la elaboración del proyecto en lugar de sensores de humedad y temperatura, tal y como comentábamos en el apartado anterior (cámaras térmicas). La razón es que los sensores de infrarrojos tienen una zona de visión y una zona muerta, es decir, detectan luz infrarroja cuando esta luz golpea en el sensor con un cierto ángulo, y si, por ejemplo, la luz fuera a dar en el reverso del sensor, que en este caso sería la zona muerta, entonces no se podría detectar el incremento de radiación. Sin embargo un sensor de temperatura toma una característica que está presente en todo el aire que le rodea, por lo que no tiene puntos ciegos. Si quisiéramos cubrir la zona muerta tendríamos que instalar tantos sensores infrarrojos como fueran necesarios, con el consiguiente incremento de coste.

Además, con los sensores de humedad y temperatura no sólo detectamos incendios, sino que también nos facilitan los datos necesarios para elaborar factores de riesgo. Es cierto que una vez se detecte un incendio es importante actuar lo antes posible, pero si, como añadido, tenemos la oportunidad de actuar incluso antes de que el incendio se produzca debemos aprovecharla.

# 7. Prometeo

## *7.1 Detección de Incendios*

En la detección de incendios mediante una red de sensores hay dos factores que resultan clave: su distribución y su calibración.

La distribución de los sensores en el bosque ha de buscar un compromiso entre cubrir la máxima extensión de terreno y detectar un fuego lo antes posible. Así que tendremos que tener en cuenta la distancia a la que los colocamos y además la configuración espacial.

Partimos de la base de que queremos que el sistema sea capaz de detectar fuegos de tamaños superiores a 1 área (100 $m^2$), y de la suposición fundada en entrevistas con expertos de que el calor que desprende un fuego se puede percibir a una distancia de 30 metros, por lo que podemos concluir que entre dos sensores debe haber unos 40 metros aproximadamente.

Todas las formas que evaluaremos tienen en común que los sensores se corresponden con los vértices de las figuras.

En lo que a la calibración del sistema, tendremos que tener en cuenta que la temperatura es lógico que varíe a lo largo del día, así que deberemos distinguir entre incrementos de temperatura provocados por el Sol e incrementos provocados por el fuego.

### *7.1.1 Disposición Octogonal*

No es una buena disposición ya que el punto más alejado (el centro) se encuentra a 52,26 m. Si tenemos en cuenta que queremos detectar incendios de 10 x 10 m

tenemos que la frontera de un fuego centrado exactamente en el centro del octógono queda a unos 47 metros (52,26 menos 10/2) del sensor más próximo, bastante por encima de los 30 metros deseables.

| Disposición | Medidas | Parrilla |
|---|---|---|
| | a = 40m<br>(consideraremos a/2 para realizar los cálculos.)<br>G = 1080/(8*2) = 67,5º<br>c = 20/cos 67,5º = 52,26 m<br>b = sin 67,5º*c = 48,28 m | |

Fig 29. Octógono Regular.

Vemos que cuando unimos diversas formaciones octogonales se produce un cuadrado que a primera vista podría parecer sin vigilancia. Sin embargo, observando con más detenimiento la figura podemos apreciar que es un cuadrado cuyos lados miden 40 metros.

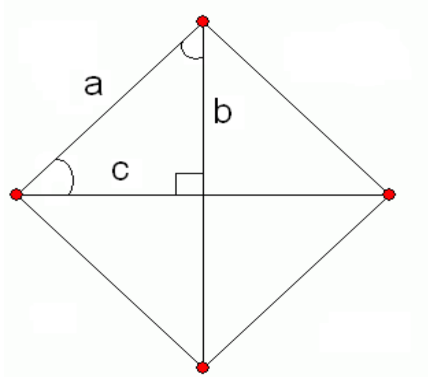



Fig 30. Zona Descubierta.

Haciendo los cálculos necesarios:

$$a^2 = b^2 + c^2;$$

$$b = c;$$

$$a = 40;$$

Llegamos a la conclusión que 'b' es igual 28,28 metros. Retomando los 10 metros de lado del incendio, tenemos que el fuego queda a unos 23 metros de los sensores. Con lo que esa zona que a primera vista pudo parecer descubierta no es tal.

Se han utilizado 24 sensores, uno por vértice. Sin embargo, para hacer esta estructura viable nos vemos obligados a colocar un sensor más en el centro de cada octógono, tal que si hay 52 metros entre un sensor del borde y el sensor del centro, y el incendio se localiza precisamente en el medio, ahora tendremos el frente del fuego a 21 metros de los sensores, 26 al centro del incendio menos los 10 metros de lado del incendio. Ahora debemos establecer una relación entre sensores utilizados y el área protegida.

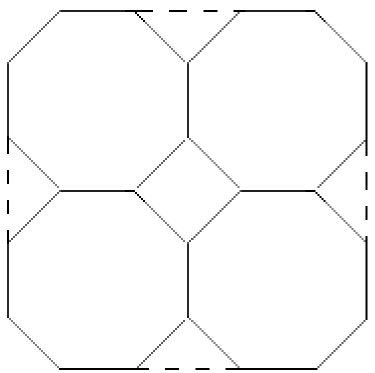


Fig 31. Zona Cubierta.

Apotema = 48,22 m (se corresponde con 'b' en la figura 28)

Perímetro = 40 metros / lado * 8 lados = 320 m

Área de un octógono = (perímetro x apotema) / 2 = 7725,48 $m^2$

Área de un cuadrado = $lado^2$ = 1600 $m^2$

Área de la zona = 4 x $A_{octógono}$ + $A_{cuadrado}$ + 4 x $A_{cuadrado}$ / 2 = 35701,93 $m^2$

Relación sensores / superficie = 28 / 35701,93 = 7,842e-4 sensores/$m^2$

La calidad del sistema es inversamente proporcional a la relación sensores / superficie. Este índice debe verse como una medida de redundancia, y cuanto menor sea la redundancia mejor, porque se cubrirá mayor terreno y necesitaremos menos sensores.

### *7.1.2 Disposición Hexagonal*

En principio no es una buena disposición ya que el punto más alejado (el centro) se encuentra a 40 m. Si tenemos en cuenta que queremos detectar incendios de 10 x 10 m tenemos que la frontera del fuego queda a 35 metros exactos del sensor más próximo, que es justamente 5 metros más de la distancia máxima a la que los incrementos de temperatura debidos al fuego son percibibles. Así que tal y como hicimos en la formación octogonal, añadiremos un sensor en el centro. Con un razonamiento similar al utilizado en el apartado superior, un fuego en el punto medio entre el sensor en el centro del hexágono y un sensor en un vértice quedaría a 15 metros como mínimo de cualquiera de los dos.

| Disposición | Medidas | Parrilla |
|---|---|---|
| | a = 40m; (consideraremos a/2 para realizar los cálculos.)<br>G = 720/(6*2) = 60º;<br>c = 20/cos 60º = 40 m<br>b = sin 60º*c = 34,64 m | |

Fig 32. Hexágono Regular.

Vemos que cuando unimos varias formaciones hexagonales no queda ningún área sin cubrir, los hexágonos encajan a la perfección.

En esta ocasión hemos utilizado 28 sensores, uno por vértice más el del centro de la figura. Calculemos ahora la relación entre sensores utilizados y el área protegida.

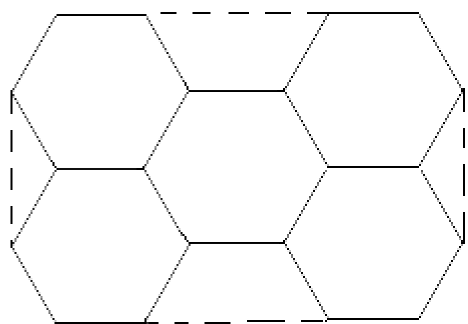


Fig 33. Parrilla de Hexágonos.

Apotema = 34,64 m (se corresponde con 'b' en la figura 31)

Perímetro = 40 metros / lado * 6 lados = 240 m

Área de un hexágono = (perímetro x apotema) / 2 = 4156,92 $m^2$

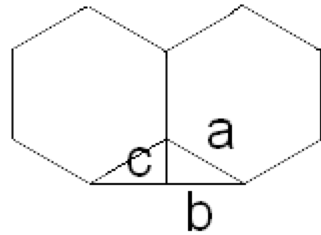



Fig 34. Triángulo Lateral de un Hexágono.

a = 40 m;

b = sin (60/2) * 40 = 20;

c = cos (60/2) * 40 = 34,64;

Área de un triangulo = (base x altura) / 2 = 2*b x 34,64 / 2 = 692,82 $m^2$

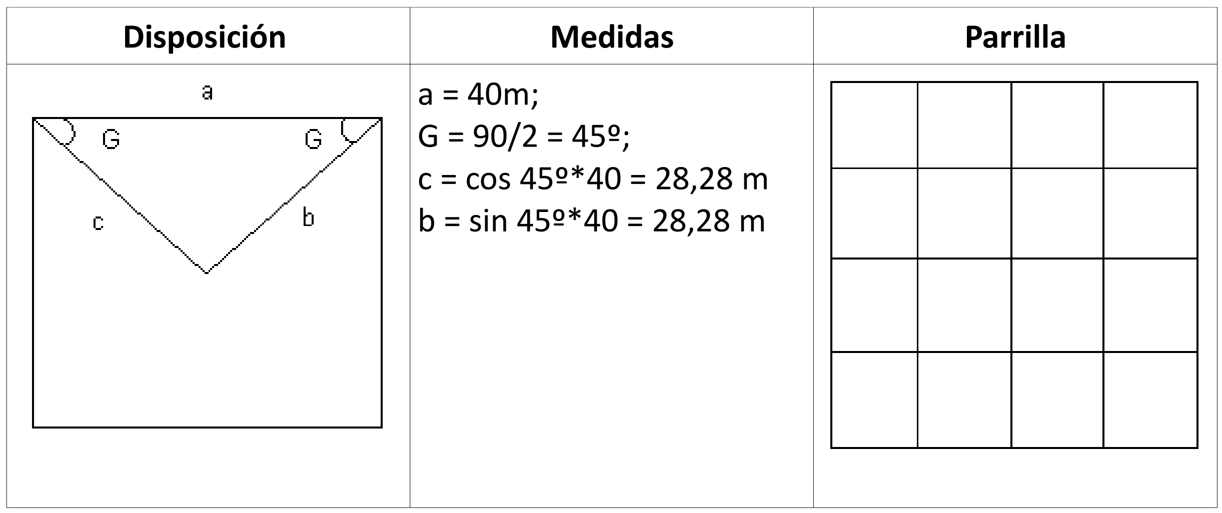
Área de la zona = 5 x $A_{hexágono}$ + 2 x $A_{hexágono}$ / 2 + 2 x $A_{triangulo}$ = 26327,12 $m^2$

Relación sensores / superficie = 28 / 26327,12 = 0,001 sensores/$m^2$

La relación sensores/superficie contradice la suposición inicial de que habíamos encontrado una formación excelente en el hexágono. La formación hexagonal no mejora la octogonal ya que necesitamos más sensores por unidad de superficie.

### *7.1.3 Formación Cuadrada*

Se trata de una buena disposición, ya que el punto más alejado (el centro) se encuentra a 28,28 m. Si tenemos en cuenta que queremos detectar incendios de 10 x 10 m tenemos que la frontera del fuego queda a unos 23 metros del sensor más próximo.




| Disposición | Medidas | Parrilla |
|---|---|---|
| | a = 40m;<br>G = 90/2 = 45º;<br>c = cos 45º*40 = 28,28 m<br>b = sin 45º*40 = 28,28 m | |

Fig 35. Cuadrado.

Vemos que cuando unimos varias formaciones cuadradas no queda ningún área sin cubrir, igual que ocurre con los hexágonos, los cuadrados encajan a la perfección.

En esta ocasión hemos utilizado 25 sensores, uno por vértice. Calculemos ahora la relación entre sensores utilizados y el área protegida.

Área de un cuadrado = $lado^2$ = 1600 $m^2$

Área de la zona = 16 x $A_{cuadrado}$ = 25600 $m^2$

Relación sensores / superficie = 25 / 25600 = 9,76e-4 sensores/$m^2$

Con este resultado queda en evidencia que la mejor disposición de los sensores es la octogonal, seguida del cuadrado y con la hexagonal en último lugar. El motivo parece ser aquella zona que en un primer momento calificamos como descubierta. Tal y como ha quedado en evidencia después de realizar los cálculos, esa zona que queda repartida entre cuatro octógonos es un área vigilada añadida para la cual no tenemos que instalar ningún otro sensor, es decir, cada cuatro octógonos de sensores que instalamos en el bosque la geometría nos obsequia con vigilancia gratuita de un cuadrado de 160 $m^2$.

Somos totalmente conscientes de que la diferencia entre la formación hexagonal y la cuadrada es mínima, y que podría atribuirse a errores en el cálculo del área bajo protección octogonal. Sin embargo, la estimación del área del cuadrado es exacta y precisa, proporcionándonos una referencia, mientras que el área de los octógonos es una aproximación mínima y aún así la relación sensores/superficie es menor.

### *7.1.4 Zona de Fresnel*

El coeficiente sensores/superficie no debe ser el único motivo que nos impulse a escoger una formación u otra, también debemos tener en cuenta las interferencias que producen los árboles y los demás elementos del bosque, es decir, debemos asegurarnos de que los sensores sean capaces de comunicarse, y para ello debemos tener en cuenta lo que se conoce como zona de Fresnel.

Cuando no hay obstáculos, las ondas de radio viajan en linea recta del transmisor al receptor. Ahora bien, cuando sí los hay, otras ondas de radio se reflejan en esos objetos y es probable que lleguen desfasadas con respecto a la señal que viaja directamente, con lo que terminan por reducir la potencia de la señal recibida. Sin embargo, también es posible que se produzca el efecto contrario si la señal reflejada y la que viaja directamente llegan en fase.

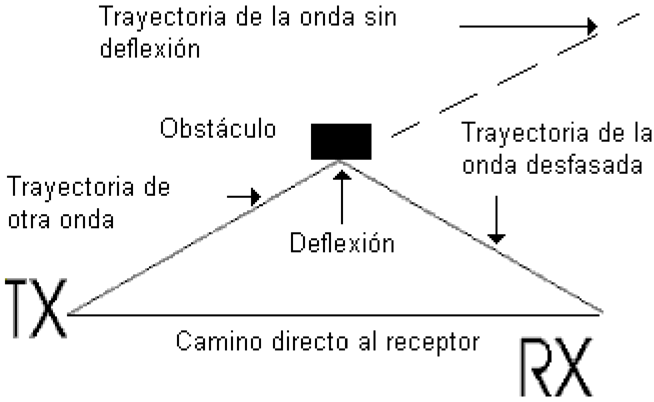



Fig 36. Influencia de los obstáculos.

Concretamente, se conoce con el nombre de Zona de Fresnel cada uno de los elipsoides concéntricos cuyo volumen define el patrón de radiación de las ondas electromagnéticas que se producen entre un emisor y un receptor.

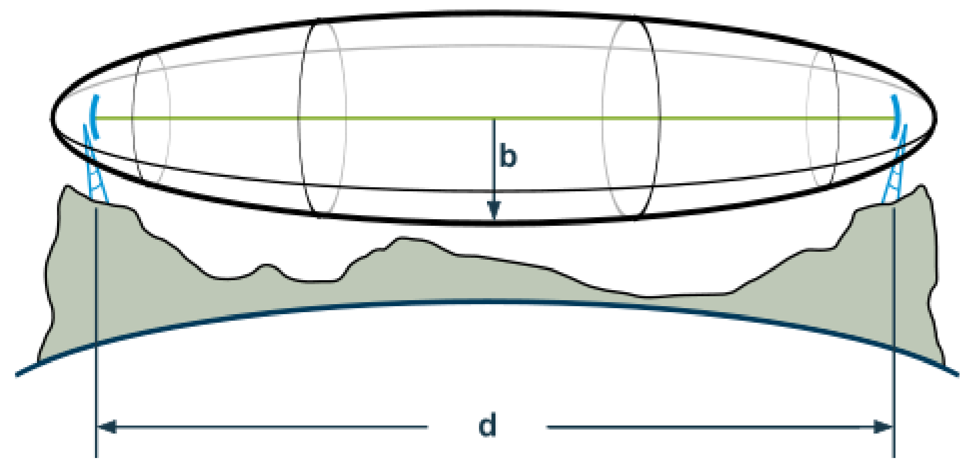



Fig 37. Zona de Fresnel. 'd' es la distancia entre el transmisor y el receptor, 'b' es el radio de la zona

El índice de la zona viene dado por el radio del elipsoide.

| Zona | Efecto |
|---|---|
| Primera | Provoca señales desfasadas entre 0 y 90º. |
| Segunda | Provoca señales desfasadas entre 90 y 270º. |
| Tercera | Provoca señales desfasadas entre 270 y 450º. |

Fig 38. Efectos de las tres primeras Zonas de Fresnel.

Las zonas de número par se las conoce como zonas destructivas, ya que el efecto de cancelación de fase es máximo, mientras que las de número impar son constructivas, es decir, es posible que se produzca una adición de ondas. Además, la fuerza de la señal, y por tanto la magnitud del efecto de cancelación, es mayor en la primera zona y decrece conforme tomamos zonas más exteriores.

La cuestión es que para maximizar la señal emitida debemos minimizar el efecto de las señales desfasadas. Algo que podemos conseguir si nos aseguramos que las señales más fuertes no topen con obstáculo alguno, es decir, la primera zona de Fresnel entre dos antenas debe estar libre de obstáculos como mínimo en un 60%, aunque lo recomendable es a partir de un 80%.

La red de sensores estará desplegada en un bosque, con lo que los obstáculos serán algo más que habituales. Por este motivo, cuantos más vértices tenga la figura en la que instalamos los sensores más improbable será que un nodo no pueda transmitir su información hacia la base, porque aunque entre él y su nodo más próximo haya una barrera de árboles aún tiene otros compañeros dentro del alcance de la radio (100 metros en el exterior) con  los que probar la comunicación.

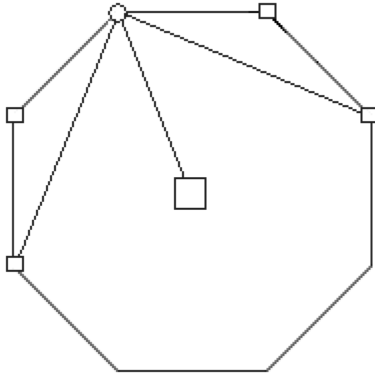


Fig 39. Alternativas de Comunicación

## *7.2 Protocolo Utilizado*

La red de sensores está pensada para ser una red ad-hoc, y por lo tanto, un nodo puede incorporarse a la red en cualquier momento durante su funcionamiento. La idea es establecer relaciones padre – hijo entre los nodos, tal que un hijo siempre

envía sus muestras y las de sus propios hijos a su padre. Veamos su comportamiento en detalle.

En un primer momento todos los nodos empiezan en el estado IHNF (I Have No Father). Entonces, si recibe un mensaje TD (Tree Discovery), que puede provenir de la estación base en primera instancia o de cualquier otro nodo, toma nota de quien lo ha enviado (a partir de ahora será el padre del nodo) y le envía un mensaje YAMF (You Are My Father). A continuación, ese nodo pasa al estado IHAF (I Have A Father) y programa un temporizador para un cierto período de tiempo; si pasado ese tiempo el nodo no ha recibido un mensaje YAMF_ACK (YAMF Acknowledge) de su padre, vuelve al estado IHNF. Ahora bien, una vez el nodo recibe el mensaje YAMF_ACK entonces envía en modo broadcast un mensaje TD, pasa al estado WFS (Waiting For Sons) y programa un temporizador. Si cuando expira el temporizador el nodo aún no ha recibido respuesta de otros nodos/hijos pueden suceder dos cosas:

- Si el nodo no ha agotado aún el número de intentos para descubrir el árbol, entonces volverá al estado IHAF, para reenviar el TD y volver a esperar hijos en estado WFS.
- Si el nodo ha superado el número máximo de intentos entonces pasa al estado SAMPLE (muestreo).

En cambio, si antes de que expire el temporizador el nodo recibe algún mensaje YAMF, responderá inmediatamente con un YAMF_ACK. Entonces, una vez expire el temporizador, pasará al estado SAMPLE. Si en estado SAMPLE, recibiera un YAMF debido a que el YAMF_ACK que envió al hijo no llegó a destino simplemente enviaría el YAMF_ACK de nuevo pero no volvería al estado WFS.

Una vez en estado SAMPLE, el nodo enviará muestras a su padre, tanto las propias como las de sus hijos, y si todo funciona como debe no volverá a cambiar de estado. No obstante, las cosas no siempre funcionan como deben, y es posible que un nodo pierda a su padre. La pérdida se detecta porque cada cierto número de muestras un

nodo espera un mensaje HEALTH (salud) de su padre. Si no lo recibe, le envía un mensaje HEALTH_REQ (Health Request) y si pasado cierto tiempo no ha recibido el mensaje de salud asume que ha perdido a su padre. Nótese que del mismo modo que el nodo espera un mensaje de salud de su padre, sus nodos hijo también lo esperan de él. Así que cada cierto tiempo deberá hacer un broadcast de un mensaje HEALTH para ellos.

Cuando un nodo pierde a su padre detiene el temporizador de muestreo y pasa al estado ILMF (I've Lost My Father). Entonces hace un broadcast de un mensaje ADOPTION y todos los nodos que lo reciben responden con un IAYF. Finalmente, una vez recibe un mensaje IAYF pasa del estado ILMF a SAMPLE y puede continuar sampleando.

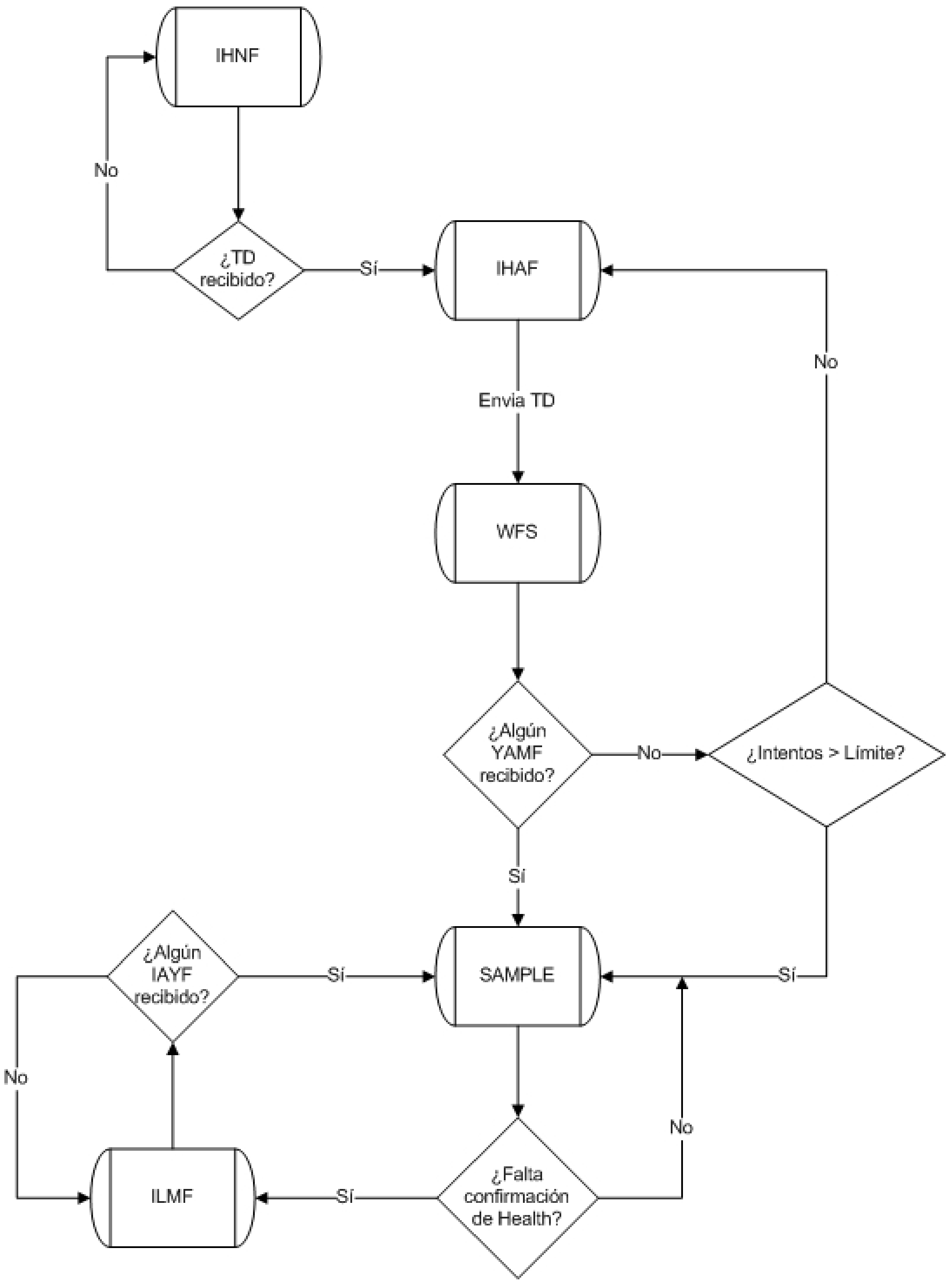



Fig 40. Diagrama de Estados del Protocolo.

## *7.3 Protocolo con Seguridad*

En este apartado tomaremos el protocolo del apartado anterior e incorporaremos mecanismos de seguridad para lograr la autenticación de paquetes así como su integridad. Utilizaremos un sistema de cadenas de hash junto con criptografia simétrica.

Cada nodo calcula una semilla, aleatoria, y le aplica una función de hash un número determinado de veces, obteniendo así una cadena de hash. Sin embargo, los nodos no cuentan con demasiada memoria, así que sólo guardamos un subconjunto de los elementos de la cadena. Este proceso lo haremos dos veces, de esta forma cada nodo tendrá una cadena para comunicarse con el padre y una cadena para comunicarse con los hijos.

La fortaleza del sistema se basa completamente en la fortaleza de la función de hash, y la fortaleza de ésta radica en la dificultad computacional de invertir la función (one-way) y en la ausencia de colisiones. Es decir, si escogemos una función de hash débil, en la que las colisiones sean frecuentes y que su inversión sea computacionalmente posible, estaremos añadiendo overhead al protocolo y no aportaremos nada.

La idea es que si tenemos una buena función de hash, computar el siguiente elemento de la cadena partiendo del elemento anterior es sencillo y el coste computacional es reducido. Por eso, las cadenas de hash se utilizan al revés.

Cada vez que un sensor desea enviar un mensaje lo hará en el siguiente formato:

| Id | $Datos_{t-1}$ | Num_sec | $h_{n+2}$ | MAC[4]($h_{n+1}$,$Datos_{t-1}$\|\|Id\|\|$h_{n+2}$\|\|Num_sec) |
|---|---|---|---|---|

Fig 41. Formato de la Trama.

[4]: *MAC-message authentication code, es una función de cifrado simétrico cuyo primer parámetro es una clave secreta y el segundo son los datos a cifrar.*

Como Id cada nodo hará público el primer elemento de la cadena invertida, es decir, el último hash calculado. A continuación enviaremos los datos, además de un número de secuencia que permitan relacionar la trama con su respuesta, seguidos del elemento n+2 de la cadena de hash para finalizar con un cifrado de toda la trama.

La idea es que un nodo al recibir este mensaje, lo guarde hasta que le llegue el siguiente y entonces le llegará la clave con el que se cifró el primer paquete totalmente en claro y podrá verificar que los datos fueron enviados por el nodo en cuestión. Vemos que no hay ningún problema de seguridad derivado de hacer pública el elemento n+2 de la clave, porque se pública ya cuando se ha dejado de utilizar para firmar los paquetes.

En realidad, el nodo que envía la trama de la figura 39 no tiene modo alguno de saber por si mismo que ya es seguro enviar en claro la clave. Por eso, debe esperar a recibir un mensaje de confirmación del nodo al que envió la trama.

| Id | ACK | Num_sec | $h'_{n+2}$ | MAC($h'_{n+1}$,ACK\|\|Id\|\|$h'_{n+2}$\|\|Num_sec) |
|---|---|---|---|---|

Fig 42. Trama de Respuesta.

Como vemos aquí es donde se utiliza la segunda cadena de hash con la que cuentan los nodos, h'. El num_sec del paquete es el mismo valor que aparece en la trama a la que está respondiendo, así nos aseguramos que no pueda haber un ataque por repetición de tramas, o un man in the middle, y que el nodo que envió la primera trama no haga pública la clave antes de que el nodo receptor reciba el paquete.

El sistema es seguro excepto cuando se está desplegando la red. Por ello, todos los nodos compartirán una clave secreta que servirá para intercambiar la identidad de los nodos, y cifrar el primer paquete. La ventaja respecto a utilizar siempre el cifrado con la clave compartida es que con la versión mixta si la clave se ve comprometida después de la fase inicial no afectará al desarrollo de la red, mientras que de la otra

forma toda la red sería vulnerable.

## 7.4 Comparativa: Seguridad vs Raw

En el caso de que aplicáramos el protocolo con seguridad a nuestra red, tendríamos que elegir entre dos modelos de validación de muestras: distribuido o central.

En el caso de la validación distribuida de las muestras, un nodo no propaga las muestras de sus hijos hacia su padre hasta que ha conseguido verificar la integridad las mismas, con lo que en el periplo de las muestras desde su origen hasta la llegada a la estación base, cada vez que se encuentra con un eslabón padre – hijo se introduce un cierto retraso. Consecuentemente, con redes de tamaño mediano la monitorización del bosque dejará de ser en tiempo real.

```
typedef nx_struct Sample
{
        nx_uint8_t id;
        nx_uint8_t humtemp;
};
typedef nx_struct TSamplesList
{
        nx_struct Sample samples[10];
        nx_uint8_t nextSample;
};
typedef nx_struct FireDetectionSample
{
        nx_uint16_t source;
        nx_uint16_t counter;
        nx_uint8_t type;
        nx_struct TSamplesList samples;
```

```
} FireDetectionSample;
```

Fig 43. Extracto de FireDetection.h.

Si tomáramos el modelo central, deberíamos añadir a nx_struct Sample una serie de campos siguiendo la estructura de trama descrita en la figura 39 para proceder a la verificación de la muestra en el momento de recepción de su sucesiva. En el mejor de los casos, si añadiéramos sólo tres campos por muestra (uno para el hash, otro para el num_sec y un tercero para el cifrado de la trama), pasaríamos de los 26 bytes actuales a 56. Desafortunadamente, existe una limitación relacionada con la radio del dispositivo que ni tan siquiera permite añadir un sólo campo a la estructura Samples, es decir, 10 bytes adicionales. Cuando decimos que no lo permite, no nos referimos a errores de compilación o excepciones graves de ejecución que provocan la terminación del programa, sino que, sencillamente, no queda garantizado que los campos que reciba el nodo padre tengan los mismos valores con los que en teoría los envió el nodo hijo.

De lo dicho hasta ahora sacamos la conclusión de que para hacer este esquema viable debemos renunciar a la agregación de muestras, o si mas no reducirla. Sin embargo, debemos tener cuidado a la hora de reducir la agregación. Entre dos muestras de temperatura se sucede un tiempo $t_{sample}$ que debe ser suficiente para que un nodo reciba las muestras que le envían sus hijos y las propague hacia su padre. Si en ese tiempo no ha conseguido pasar las muestras al siguiente nivel de la cadena se va a empezar a producir un retraso en la propagación de temperaturas. Además, cuando diseñamos el protocolo teníamos en mente minimizar el uso de la radio, para maximizar el tiempo que está disponible para recibir mensajes por lo que las muestras de temperatura no tienen mensaje de acknowledge. En consecuencia, si disminuimos la agregación de muestras tendremos que enviar más paquetes, no sólo corremos el peligro de que el tiempo de muestreo se quede corto, sino que además es posible la pérdida de muestras.

Por todo lo anterior hemos decidido implementar el protocolo en su versión sin seguridad.

# 8. Aplicación: Prometeo – Fire Detection System

Cuando hablamos de la aplicación nos referimos al programa que se deberá instalar en el ordenador del centro de control. Ésta debe ser sencilla y rápida de manejar, para reducir al máximo la formación del usuario y además evitar que una alarma por incendio se pierda entre una maraña de instrucciones y ventanas.

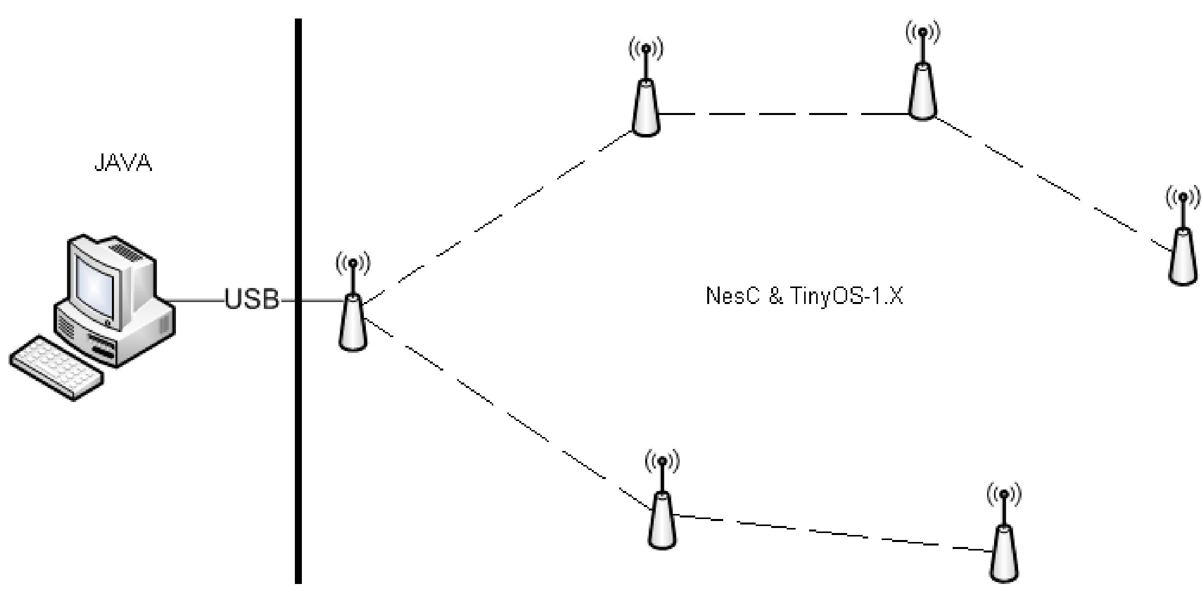



Fig 44. Estructura de la Red y Aplicación.

A grandes rasgos, debemos distinguir tres componentes:

- la clase que escucha en el puerto serie.
- la base de datos.
- la interfaz.

La clase MsgParser es la encargada de registrarse en el puerto serie para recoger todos los paquetes que lleguen al puerto a través de la estación base, tratarlos y en el caso de que se trate de muestras de los sensores insertarlos en la base de datos.

La interfaz irá a buscar a la base de datos las muestras más recientes para presentarlas al usuario y además permitirá la realización de ciertas operaciones que veremos en el apartado siguiente.

Finalmente, la base de datos se encargará de la persistencia de las muestras y hará la función de enlace entre la interfaz y MsgParser.

## *8.1 Diagrama de Casos de Uso*

Nuestro sistema debe permitir las funcionalidades descritas en el siguiente diagrama:

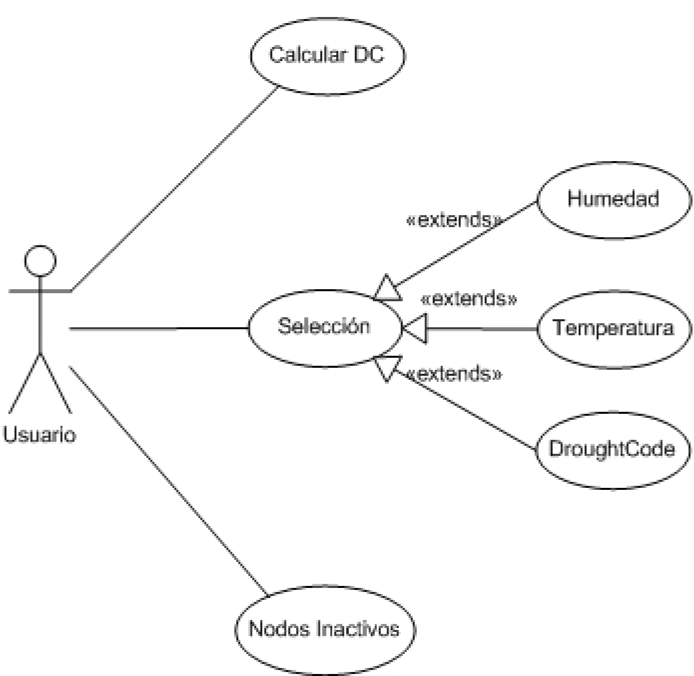



Fig 45. Diagrama de Casos de Uso.

## 8.2 Casos de Uso

### 8.2.1 *Calcular DC*

1. El Usuario clica en el menú “Acciones” y selecciona el submenú “Calcular DC”.

2. El Sistema abre un formulario en el que pregunta al Usuario por los mm de precipitación caídos en el último día.

3. Usuario entra en el formulario la cantidad de lluvia caída.

4. El Sistema calcula el DC o Drought Code para cada sensor y lo inserta en la base de datos.

5. El Sistema añade a la tabla “Muestras Recogidas” el DC calculado.

### *8.2.2 Selección por Humedad*

1. El Usuario clica en el menú “Consultas” y selecciona el submenú “Por Humedad”.

2. El Sistema abre un formulario en el que pregunta al Usuario por el rango de humedades con el que quiere delimitar la consulta.

3. Usuario entra en el formulario la cota inferior y la cota superior de humedad separadas por un guión.

4. El Sistema realiza la consulta en la base de datos.

5. El Sistema añade los resultados a la tabla “Resultado de la Consulta”.

### *8.2.3 Selección por Temperatura*

1. El Usuario clica en el menú “Consultas” y selecciona el submenú “Por

Temperatura”.

2. El Sistema abre un formulario en el que pregunta al Usuario por el rango de temperaturas con el que quiere delimitar la consulta.

3. Usuario entra en el formulario la cota inferior y la cota superior de temperatura separadas por un guión.

4. El Sistema realiza la consulta en la base de datos.

5. El Sistema añade los resultados a la tabla “Resultado de la Consulta”.

### *8.2.4 Selección por Drought Code*

1. El Usuario clica en el menú “Consultas” y selecciona el submenú “Por Drought Code”.

2. El Sistema abre un formulario en el que pregunta al Usuario por el rango de DC con el que quiere delimitar la consulta.

3. Usuario entra en el formulario la cota inferior y la cota superior de DC separadas por un guión.

4. El Sistema realiza la consulta en la base de datos.

5. El Sistema añade los resultados a la tabla “Resultado de la Consulta”.

### *8.2.5 Nodos Inactivos*

1. El Usuario clica en el menú “Acciones” y selecciona el submenú “Nodos Inactivos”.

2. El Sistema busca aquellos nodos cuya última muestra se recibió hace más de un minuto.

3. El Sistema presenta los resultados en la tabla “Resultado de la Consulta”.

## *8.3 Base de Datos.*

En la base de datos debemos poder guardar un cierto número de muestras con el objetivo de detectar incendios mediante aumentos de temperatura bruscos y de poder calcular el DC a partir de la última temperatura recibida.

Teniendo en cuenta que cada sensor envía una muestra de temperatura a la estación base cada 10 segundos, consideramos que conservando las diez últimas muestras, y por tanto trabajando sobre los últimos 100 segundos, tenemos suficiente para cumplir con los objetivos.

Básicamente, la tabla “sensores” de nuestra base de datos se compone de los siguientes campos:

id: INT || temp0: INT || temp1: INT || temp2: INT || temp3: INT || temp4: INT || temp5: INT || temp6: INT || temp7: INT || temp8: INT || temp9: INT || ultimaTemp: INT || humedad: INT || dc: INT || fecha: Time

- El campo “id” se corresponde con el identificador del nodo que ha enviado la muestra.
- ultimaTemp nos indica cual es el último campo de temperatura en el que hemos escrito (del 0 al 9) para saber cual es la muestra más reciente y donde debemos guardar la próxima temperatura.
- humedad es el campo que aloja el valor de humedad leído por el sensor. Tenemos una muestra de humedad cada 60 segundos.
- dc guarda el computo del índice de sequía una vez el usuario decida computarlo.
- el campo fecha nos indica el momento, en formato HH:MM:SS, en que se recibió

la última muestra. Se utiliza para detectar nodos de la red que hayan dejado de funcionar.

## *8.4 MsgParser.*

MsgParser implementa la interfaz MessageListener de las librerías de TinyOS, y tiene como función escuchar en el puerto serie el tráfico que proviene de la red. Llegados a este punto es conveniente recordar que los nodos han sido programados en el lenguaje nesC, y que trabajan con TinyOS-1.X, y que la aplicación del ordenador base trabaja en lenguaje Java. Precisamente, la clase MsgParser es la encargada de convertir los mensajes que llegan del puerto serie en formato nesC según las especificaciones de la clase FireDetectionSample.java (que generamos mediante la herramienta MIG, explicada en el apartado 4.3.2 Comunicación) e introducirlos en la base de datos.

Para lanzar el programa debemos teclear lo siguiente en un terminal:

*java MsgParser -comm serial@/dev/ttyUSB1 FireDetectionSample*

Lo que lanzará la función main del programa.

```
public static void main(String[] args) throws Exception
{
        ...
        MsgParser mr=new MsgParser(source);
        ...
        String className=args[i];
        Class c=Class.forName(FireDetectionSample);
        Object packet=c.newInstance();
        Message msg=(Message)packet;
        v.addElement(msg);
```

```
        ...
        Enumeration msgs = v.elements();
        Message m=(Message)msgs.nextElement();
        mr.addMsgType(m);

        mr.start();
        FireMenu fm=new FireMenu();
    }

    private void addMsgType(Message msg)
    {
        moteIF.registerListener(msg,this);
    }

    public MsgParser(String source) throws Exception
    {
        ...
        moteIF=new
            MoteIF(BuildSource.makePhoenix(source,PrintStreamMessenger.err));
        ...
    }
```

Fig 46. Fragmento de MsgParser.java

El primer paso es crear un objeto MoteIF, ya que es el objeto que se encarga de las comunicaciones entre el ordenador y la estación base conectada a él. El parámetro source es el nombre del puerto con el que trabajará la función, en nuestro caso será: "/dev/ttyUSB1".

Una vez creado el objeto, iteramos sobre la lista de argumentos que pasamos al

programa, por si en un futuro ademas de escuchar los mensajes FireDetectionSample nos interesara escuchar también FireDetectionMessage. Finalmente, después de una serie de operaciones y conversiones registramos MsgParser para escuchar el tipo de mensajes FireDetectionSample en el puerto serie llamando a la función "addMsgType(Message msg)".

A partir de ese momento cada vez que el puerto serie reciba un paquete se ejecutará la función messageReceived.

```
public void messageReceived(int to, Message message)
{
        FireDetectionSample fds;

        fds=(FireDetectionSample)message;

        Short stype=new Short(fds.get_type());
        Integer itype=(Integer)stype.intValue();

        short[] ids =fds.get_samples_samples_id();
        short[] samples=fds.get_samples_samples_humtemp();
        short nextSample=fds.get_samples_nextSample();
        .....
}
```

Fig 47. Fragmento de MsgParser.java donde se convierte el mensaje a FireDetectionSample.

En esta figura se puede apreciar que gracias al trabajo previo que realizamos con la herramienta MIG la conversión de tipo del mensaje se puede hacer con un simple cast del lenguaje Java. De igual modo, para acceder a los campos del mensaje podemos llamar a los métodos generados en FireDetectionSample.java.

Para terminar, deberemos distinguir entre muestras de temperatura y humedad mediante “itype” e introducirlas en la base de datos utilizando sentencias JDBC, añadiéndoles la hora del sistema con el fin de detectar la situación en la que un nodo haya dejado de enviar muestras.

## *8.5 Interfaz.*

Nada más arrancar la aplicación nos encontramos con la pantalla principal. En ella podemos distinguir cuatro áreas principales:

- Barra de menú: las opciones del programa se dividen en acciones y consultas. En Acciones tenemos la posibilidad de calcular el Drought Code y buscar los nodos inactivos, y en Consultas podremos hacer selecciones de las muestras recibidas por DC, por temperatura y por humedad.
- Tabla de muestras: nada más empezar la aplicación y sin necesidad de ninguna acción por parte del usuario, en esta tabla irán apareciendo las muestras que reciba la estación base del resto de nodos de la red.
- Tabla de resultados: cuando realicemos una consulta o bien utilicemos la opción de Nodos Inactivos, los resultados de la operación se mostrarán en esta tabla.
- Árbol de la red: a medida que vayan llegando mensajes, en este componente de la interfaz podremos ir visualizando como se ha estructurado la red, es decir, que relaciones padre – hijo se han establecido.

## *8.5.1 StoryBoard*

En este apartado trataremos el comportamiento de la aplicación a nivel gráfico. Sin embargo, omitiremos los detalles de las operaciones porque ya fueron explicadas en profundidad en la descripción de los casos de uso.

En el siguiente diagrama se puede observar las transiciones entre las distintas ventanas:

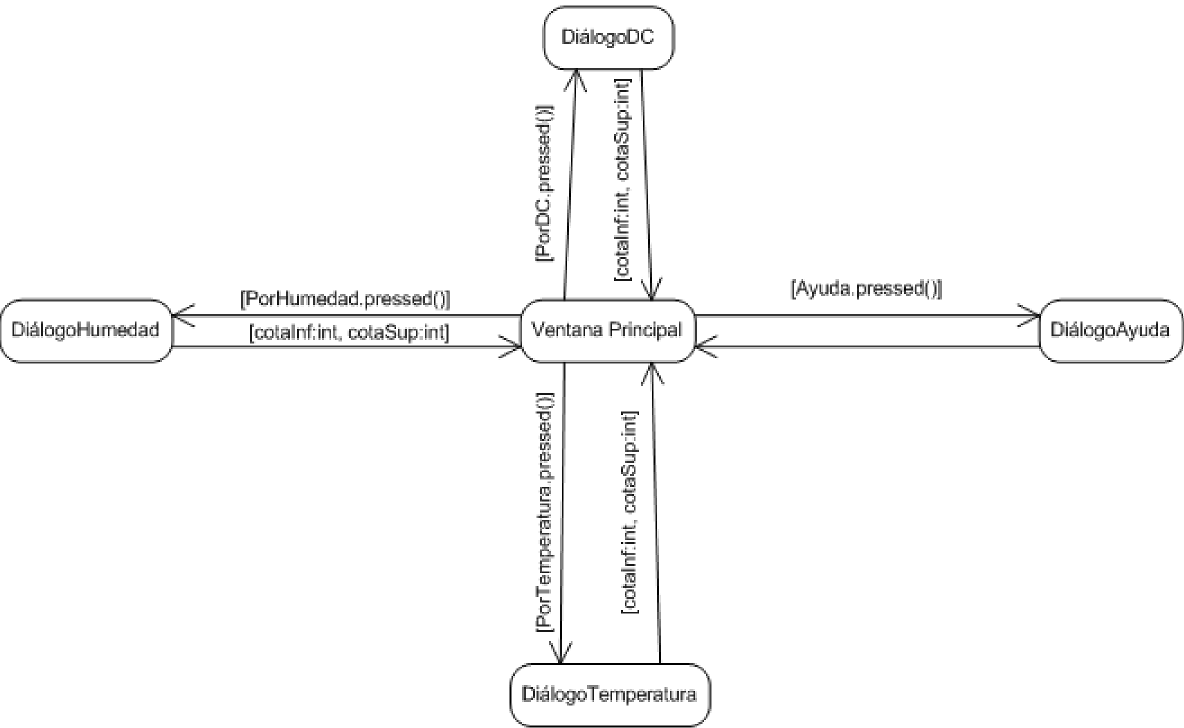



Fig 48. Diagrama de Estados de la Interfaz.

Si los parámetros no son nulos y en el correspondiente Diálogo se seleccionó la opción “Aceptar”, la Ventana Principal refrescará el componente pertinente según la descripción de los casos de uso. Por ejemplo, si queremos calcular el DC la Ventana Principal nos abrirá un diálogo donde nos pregunte por la cantidad de lluvia registrada ese día. Ese parámetro irá de vuelta a la Ventana Principal, donde se realizarán los cálculos necesarios y se refrescará la tabla de muestras.

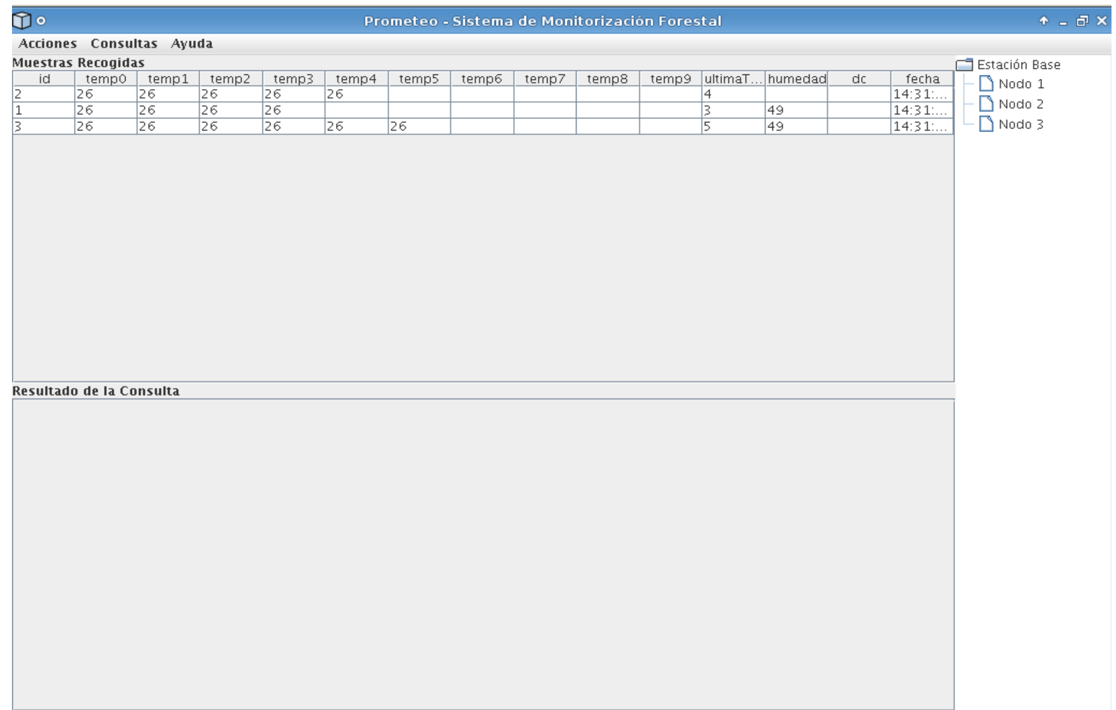



Fig 49. Ventana Principal.

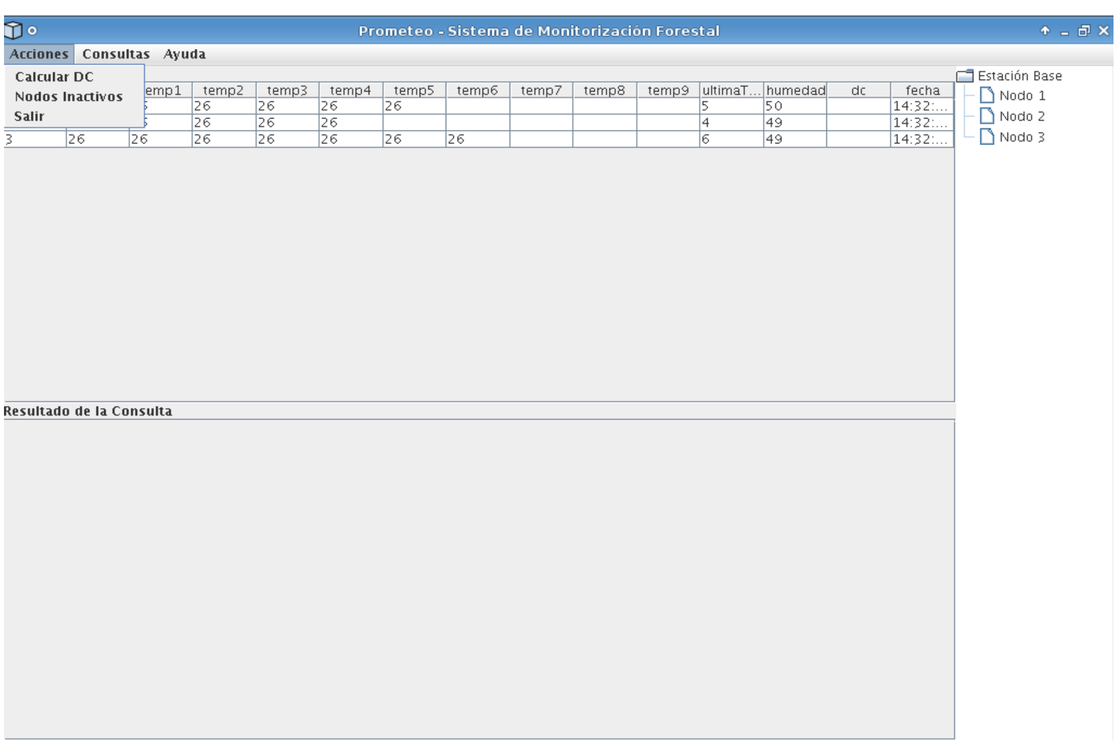



Fig 50. Ventana Principal con “Acciones”.

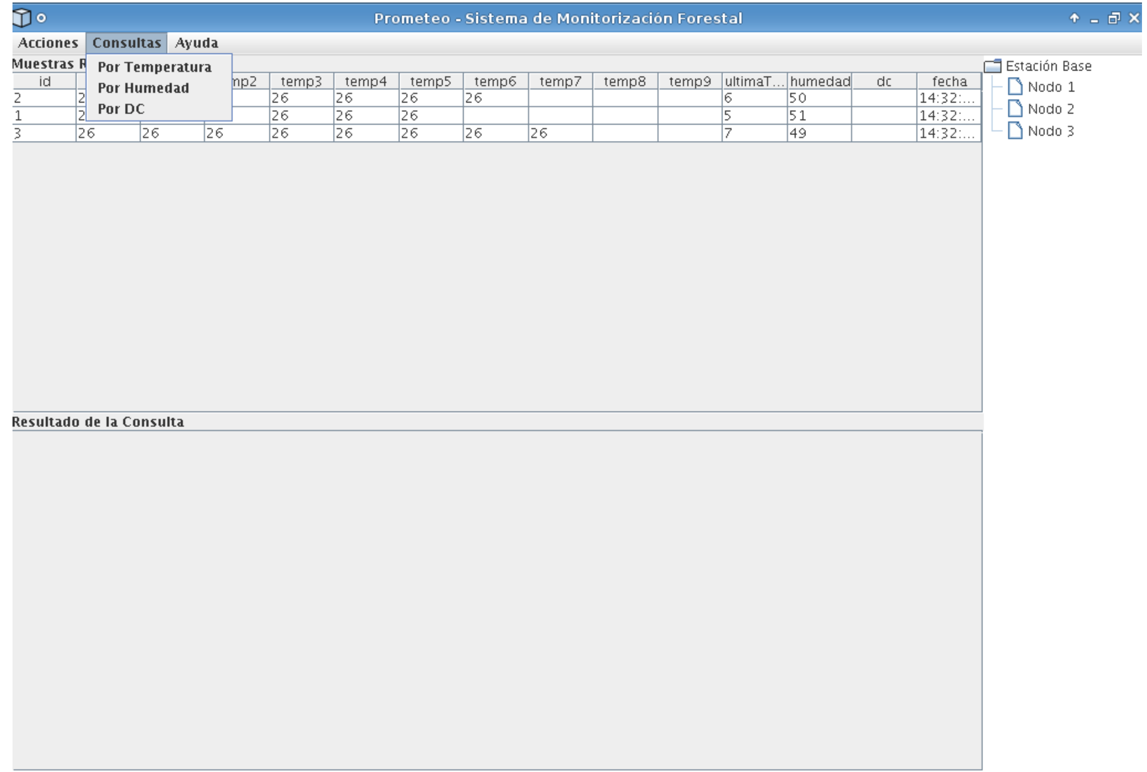



Fig 51. Ventana Principal con "Consultas".

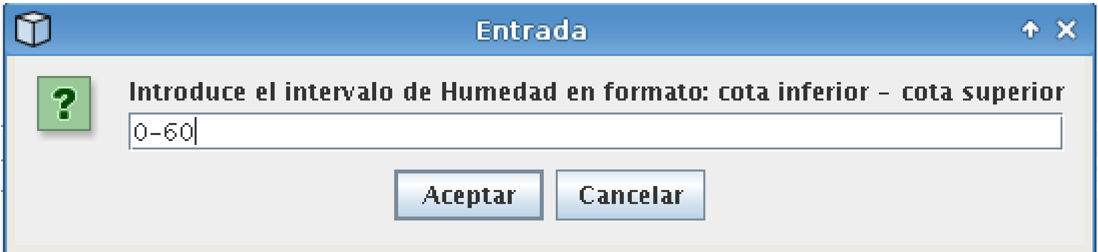



Fig 52. DiálogoHumedad.

Fig 53. DiálogoTemperatura.

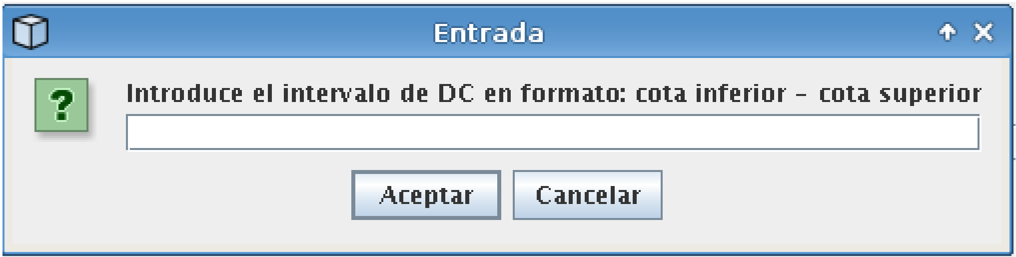



Fig 54. DiálogoDC.

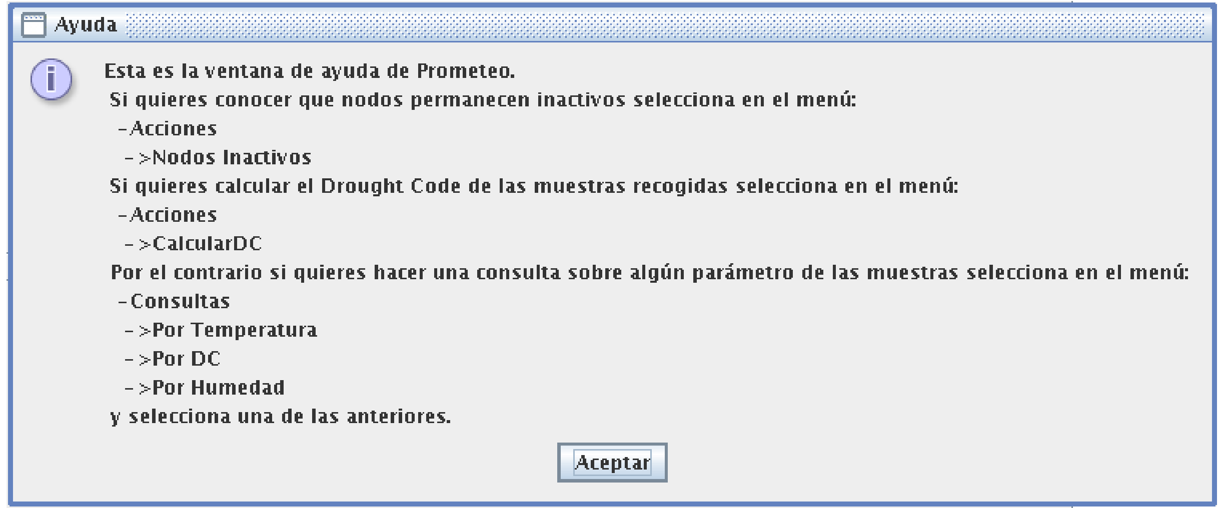



Fig 55. Ayuda.

# 9. Conclusiones Finales

Creemos que con la inversión necesaria la monitorización automática de entornos forestales sería algo posible, y rentable económicamente.

A lo largo del proyecto hemos intentado ser fieles a la realidad, buscar el consejo y la ayuda de expertos en la materia con el objetivo de llegar a conocer y comprender en profundidad el campo de los incendios forestales, y con el fin último de diseñar un sistema de detección de incendios y de elaboración de factores de riesgo que fuera útil y que pudiera servir de modelo en la automatización de procesos.

Hemos estudiado las diferentes posibilidades de sensores. Hemos visto los elementos que deben componer un buen sistema de vigilancia. Hemos estudiado como influyen las variables atmosféricas en la humedad de los combustibles, y por ende en los incendios. Hemos detallado como funciona un modelo de predicción de la humedad. Y finalmente, hemos seleccionado la mejor disposición de los nodos en la red.

Estamos contentos y satisfechos con el trabajo realizado.

# 10. Bibliografia


http://www.zytrax.com/tech/wireless/fresnel.htm

http://en.wikipedia.org/wiki/Fresnel_zone

http://es.wikipedia.org/wiki/Hex%C3%A1gono

http://es.wikipedia.org/wiki/Oct%C3%B3gono

http://en.wikipedia.org/wiki/ZigBee

http://www.zigbee.org/en/index.asp

http://nrfa.fire.org.nz/fire_weather/fwi_help/firedangerflowchart.htm

http://en.wikipedia.org/wiki/Free-space_path_loss

http://www.mma.es/secciones/biodiversidad/defensa_incendios/estadisticas_incendios/index.htm

http://docs.tinyos.net/index.php/TinyOS_Tutorials

http://en.wikipedia.org/wiki/IEEE_802.15.4

“Los Incendios Forestales en España. Año 2007” Enero 2008 – Ministerio de Medio Ambiente.

“Modelos de predicción de la humedad de los combustibles muertos: fundamentos y aplicación” A.D. Ruiz González y J.A. Vega Hidalgo, Año 2007 – Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria (INIA) – Ministerio de Educación y Ciencia.

“Manual de formación de incendios forestales para cuadrillas” Felipe Aguirre Briones, Año 2007, 1ª edición – Departamento de Medio Ambiente, Gobierno de Aragón.

“Educación e incendios forestales” Dante Arturo Rodriguez Trejo, Año 2000, 1ª edición – Universidad Autónoma de Chapingo, México.

Vélez, 2000: Los índices meteorológicos de peligro. En: La defensa contra incendios forestales. Fundamentos y experiencias. McGraw – Hill. Capítulo 8.

Viney 1991: A review of fine fuel moisture modelling. Int. J. Wildland Fire 1(4): 215-234.

Pyne S.J, Andrews P.L., Lavern R.D. 1996:Introductionto wildland fire. John Wiley & Sons, Inc USA. 769 pp.

Guerrero M., Bellalta B 2007: Secure and efficient data collection in sensor networks.

“MoteConfig User's Manual” Crossbow

“MoteWorks User's Manual” Crossbow

# Apéndice

## A. Reuniones

### A.1 Reunión con el Cuerpo de Bomberos de Mollet del Vallés

Este apartado es un resumen de la conversación mantenida con el Cuerpo de Bomberos, en el que presentaremos las ideas principales.

Las torres de vigilancia son muy efectivas en la detección de incendios. Una columna de humo de 1 $m^2$ es visible a desde un puesto de vigía a mucha distancia

Así pues, sería interesante la aplicación de redes sensores en áreas donde no haya puestos de vigía o bien enfocar el sistema a la elaboración de zonas de riesgo.

En cuanto a las variables atmosféricas, es suficiente fijarnos en tres:

- temperatura
- humedad
- viento

Para manejar todos estos parámetros es importante tener en cuenta la denominada regla del treinta:

“Todos los grandes incendios se han producido con condiciones similares: al menos 30º C de temperatura, menos del 30% de humedad relativa y con vientos con velocidades superiores a 30km/h.”

Desafortunadamente, nos encontramos con que no tenían un dato muy importante para el proyecto: a qué distancia del fuego se deja de percibir el calor que éste desprende por radiación. Sin embargo, nos remitieron al GRAF porque si alguien debía poder tener esos datos eran ellos.

## *A.2 Reunión con el Ingeniero de Montes Ramón Costa*

Ramón Costa es Ingeniero de Montes por la Universidad de Zaragoza, además de retén forestal y profesor de la asignatura “Prevención de Incendios Forestales” en el Instituto de Educación Secundaria Miguel Catalán. Estos son los puntos más relevantes de la conversación:

- La detección de incendios forestales no resulta viable por la cantidad de sensores que habría que instalar en el bosque.
- 1 área es una extensión razonable para la detección, porque es un fuego pequeño que se puede apagar con pocos medios. Si trabajamos con una superficie mayor estaremos empeorando el sistema vigente, por ejemplo, dejar que se queme una hectárea antes de que el sistema sea capaz de detectar el incendio es totalmente inaceptable.
- En el año 2007 hubo entre 3000 y 4000 incendios pequeños en Aragón, de ahí la importancia de la pronta detección.
- Orientar el sistema a la elaboración de factores de riesgo por zonas. Actualmente, Aragón cuenta con una estación meteorológica por provincia, con lo que las estimaciones no son todo lo precisas que debieran.
- A unos 30 metros de un fuego ya se puede detectar el calor que desprende.
- Existen sistemas de detección de incendios con láser y con cámaras térmicas, aunque el láser funciona detectando humos y es muy sensible a falsas alarmas, tomando por humo lo que es una simple polvareda.

## *A.3 Reunión con los Responsables del GRAF*

Las recomendaciones y observaciones del GRAF fueron las siguientes:

- No necesitan redes de sensores para monitorizar el estado del bosque. Están contentos con su método de trabajo actual.
- Tampoco necesitan factores de riesgo más precisos, sino previsiones meteorológicas acertadas.
- Ellos utilizan el Drought Code para establecer factores de riesgo de incendio.
- Estarían interesados en redes de sensores que resistieran los incendios y les ayudarán a comprender como se comportan los vientos que genera el propio incendio.

## *B. Pruebas y Resultados*

Para demostrar el correcto funcionamiento de los sensores, hemos sometido el código de TinyOS a una serie de pruebas en el simulador Tossim. Las pruebas consisten en un fichero donde se describe la topología de la red y un script en Python que será ejecutado por el simulador Tossim.

*1 2 -54.0*

*2 1 -55.0*

*1 3 -60.0*

*3 1 -60.0*

*2 3 -64.0*

*3 2 -64.0*

Fig. Apéndice 1. Topología de una Red.

En la figura superior vemos el típico formato del fichero descriptor de la topología. Por ejemplo, la primera línea nos esta diciendo que cuando el nodo 1 transmita el nodo 2 estará escuchando a -54 dBm.

Con el objetivo de hacer las simulaciones más realistas, hemos realizado un programa que crea una topología de red, trabajando con distancias aleatorias y por tanto con niveles de ruido aleatorios, y nos lo guarda en el fichero “topoexample.txt”. Se basa en el fenómeno conocido como pérdida de potencia en espacios abiertos (FSPL o Free Space Path Loss), que modela la pérdida de potencia de señal como proporcional al cuadrado de la distancia entre el emisor y el receptor.

```
#! /usr/bin/python
from random import *
from Numeric import *
import sys

frequency=2400;
Nnode=25;
MaxX=100; #en metros
MaxY=100; #en metros
xarray=zeros(Nnode, Int);
yarray=zeros(Nnode, Int);

def computeDistance(i,j):
	y=yarray[j]-yarray[i]
	y=y**2
	x=xarray[j]-xarray[i]
	x=x**2
	return sqrt(x+y)
def computeFreeSpaceLoss(d):
	return (20*log(d/1000.0)/log(10))+(20*log(frequency)/log(10))+32.44
for i in range(0,Nnode):
	yarray[i]=random()*MaxY;
	xarray[i]=random()*MaxX;
print yarray
print xarray
print " "
distance=zeros((Nnode,Nnode),Int);
```

```
for i in range(0,Nnode):
	for j in range(0,Nnode):
		distance[i][j]=computeDistance(i,j);

f=open("topoexample.txt","w");
for i in range(0,Nnode):
	for j in range(0,Nnode):
		if j>i:
			s=str(j)+" "+str(i)+" "
			+str(computeFreeSpaceLoss(distance[i][j]))+"\n";
			f.write(s);
			print j,i,distance[i][j],computeFreeSpaceLoss(distance[i][j]);
f.write("\n");
f.close();
```

Fig. Apéndice 2. Contenido del Fichero RandomSetup.py.

Una vez contamos con un fichero de topología aleatorio ya podemos realizar las pruebas.

## B.1 Test1

En este primer test tenemos como objetivo cerciorarnos de que los nodos son capaces de comunicarse y de seguir los pasos más básicos del protocolo sin realizar agregación de muestras, es decir, ver si con capaces de establecer relaciones padre – hijo entre ellos.

Lo primero que hace el programa es abrir un objeto de radio e instanciarlo con los valores descritos en el fichero de topología. A continuación, redirige la salida de los canales de comunicación Boot y FireDetection hacia un fichero y añade un modelo de

ruido a cada nodo. Finalmente, arranca los nodos y empieza a ejecutar sus eventos.

```
#! /usr/bin/python
from TOSSIM import *
import sys


t=Tossim([])
r=t.radio()
f=open("topo1.txt","r")
res=open("resultados1","w")
lines=f.readlines()
for line in lines:
	s=line.split()
	if(len(s) > 0):
		print " ",s[0]," ",s[1]," ",s[2];
		r.add(int(s[0]), int(s[1]), float(s[2]))

t.addChannel("FireDetection", res)
t.addChannel("Boot", res)

noise=open("/opt/tinyos-2.x/tos/lib/tossim/noise/meyer-heavy.txt","r")
lines=noise.readlines()
for line in lines:
	str=line.strip()
	if(str!=""):
		val=int(str)
		for i in range(1,11):
			t.getNode(i).addNoiseTraceReading(val)
for i in range(1,11):
```

```
        print "Creating noise model for ",i;
        t.getNode(i).createNoiseModel();
for i in range(1,11):
        t.getNode(i).bootAtTime(0);
for i in range(0,2000000):
        t.runNextEvent()
f.close();
res.close();
noise.close();
```

Fig. Apéndice 3. Contenido del Fichero Test1.py.

Tal y como se puede apreciar en el código, para este primer test contaremos con 10 nodos, que seguirán la siguiente topología:

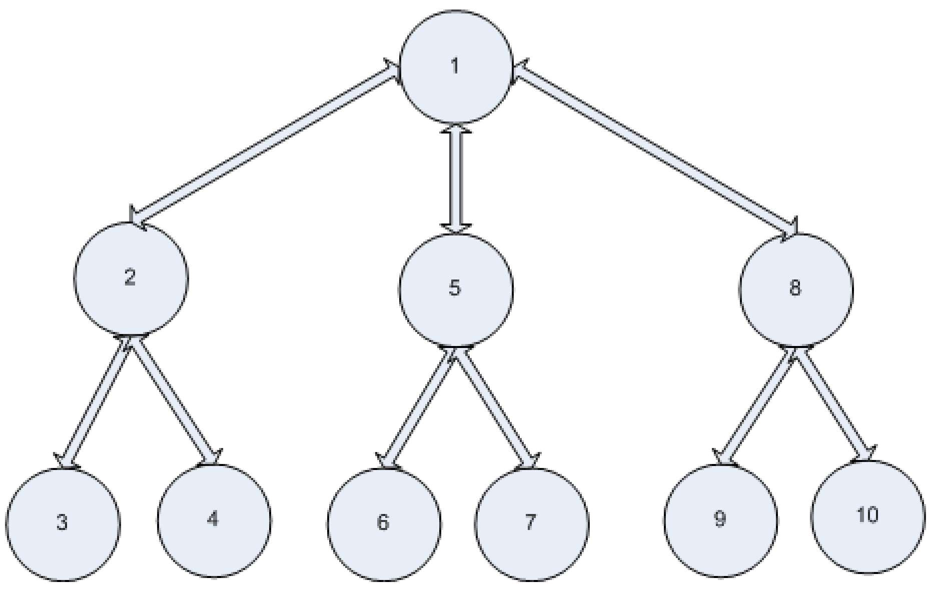



Fig. Apéndice 4. Representación Gráfica del Contenido del Fichero Topo1.txt.

El resultado de la ejecución lo podemos ver en el siguiente gráfico:

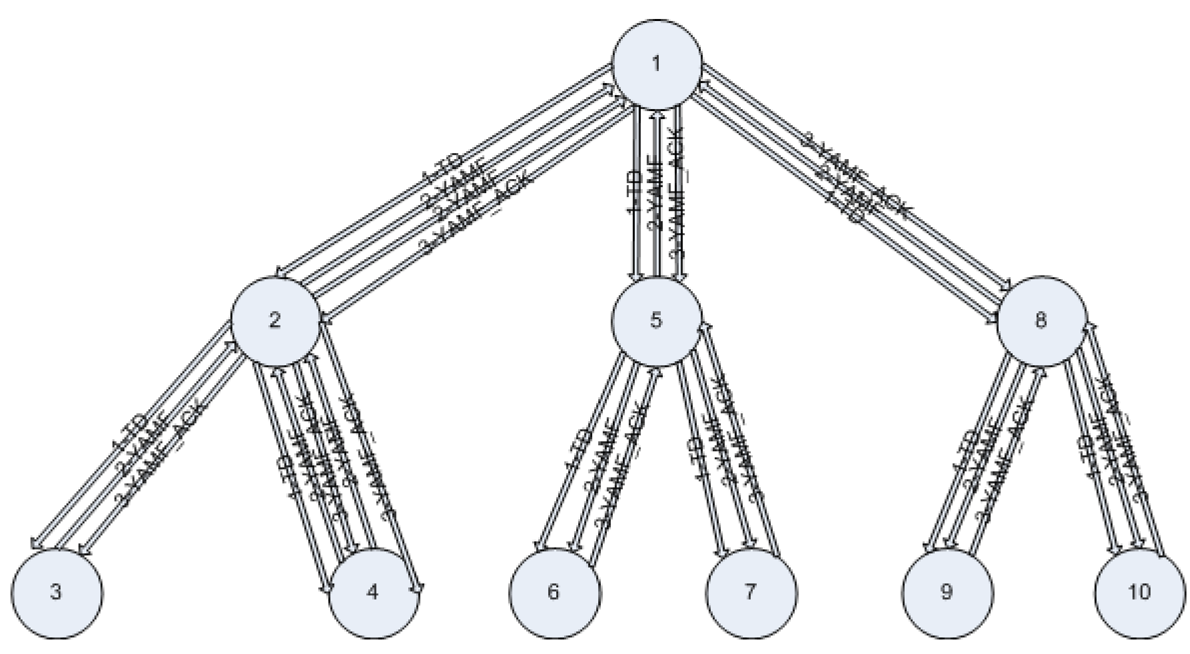



Fig. Apéndice 5. Representación Gráfica del Contenido del Fichero Resultados1.txt.

Tal y como se puede apreciar en el gráfico de resultados, la disposición de los nodos coincide a la perfección con la topología específicada en Topo1.txt (Figura Apéndice 4), con lo que podemos concluir el primer test con un resultado satisfactorio.

Las interacciones interesantes están entre los nodos 1 y 2 y entre el 2 y el 4. Entre el nodo 1 y el nodo 2 se produce una pérdida del mensaje YAMF, y al no recibir YAMF_ACK en un tiempo determinado, el nodo 2 retransmite el mensaje. En el caso de los nodos 2 y 4 lo que se produce es una pérdida del mensaje YAMF_ACK, por lo que una vez más el nodo 2 deberá retransmitir el mensaje YAMF.

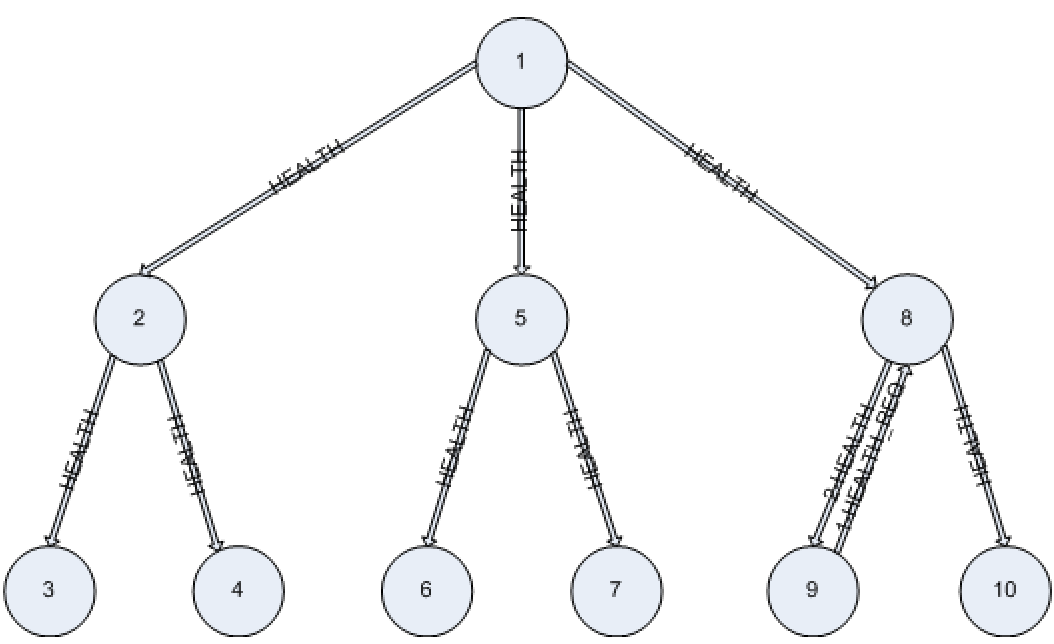



Fig. Apéndice 6. Representación Gráfica del Contenido del Fichero Resultados1.txt.

En esta ocasión nos hemos fijado en los paquetes HEALTH, cuya misión es mantener a los nodos en contacto para detectar la pérdida de padres. Por lo general, cada cieto tiempo un nodo envía un mensaje HEALTH a todos sus hijos, y en el caso de que pase ese tiempo y el hijo no haya recibido nada, enviará una petición HEALTH_REQ a la que el padre responderá con un mensaje HEALTH sólo para ese hijo.

## *B.2 Test2*

La finalidad de este segundo test es establecer que los nodos son capaces de recuperarse de la pérdida de su padre y ser adoptados por un nuevo nodo.

Para empezar, deberemos cambiar la topología, para que algunos nodos tengan varios candidatos a padres.

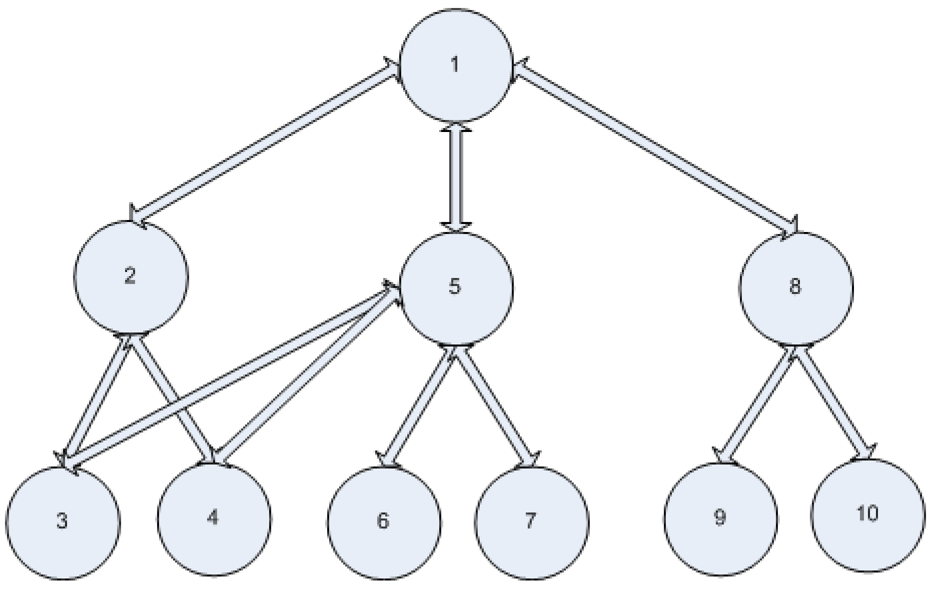



Fig. Apéndice 7. Representación Gráfica del Contenido del Fichero Topo2.txt.

En este caso, los nodos 3 y 4 podrían escoger entre ser hijos del nodo 2 o del nodo 5, aunque nosotros los forzaremos a ser hijos del nodo 2.

Repitiendo el procedimiento del test anterior, una vez hayamos arrancado los nodos y ejecutado eventos hasta que la red ya se haya formado, procederemos a desconectar los nodos 2 y 6, un padre y un hijo, para ver la respuesta de la red.

```
#! /usr/bin/python
from TOSSIM import *
import sys

t=Tossim([])
r=t.radio()
f=open("topo2.txt","r")
res=open("resultados2","w")
lines=f.readlines()
```

```
for line in lines:
	s=line.split()
	if(len(s) > 0):
		print " ",s[0]," ",s[1]," ",s[2];
		r.add(int(s[0]), int(s[1]), float(s[2]))

t.addChannel("FireDetection", res)
t.addChannel("Boot", res)

noise=open("/opt/tinyos-2.x/tos/lib/tossim/noise/meyer-heavy.txt","r")
lines=noise.readlines()
for line in lines:
	str=line.strip()
	if(str!=""):
		val=int(str)
		for i in range(1,11):
			t.getNode(i).addNoiseTraceReading(val)
for i in range(1,11):
	print "Creating noise model for ",i;
	t.getNode(i).createNoiseModel();
for i in range(1,11):
	t.getNode(i).bootAtTime(0);
for i in range(0,2000000):
	t.runNextEvent()
t.getNode(2).turnOff();
t.getNode(6).turnOff();
for i in range(0,200000):
	t.runNextEvent()
```

```
f.close();
res.close();
noise.close();
```

Fig. Apéndice 8. Contenido del Fichero Test2.py.

El resultado de la ejecución lo podemos ver en este gráfico:

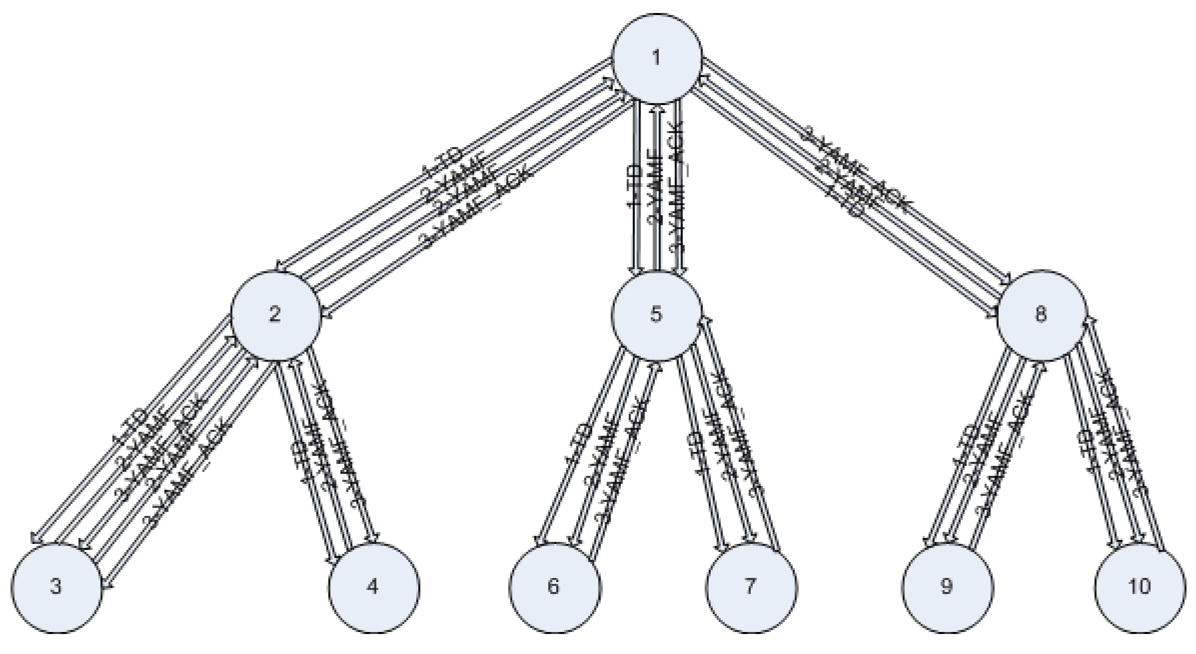



Fig. Apéndice 9. Representación Gráfica del Contenido del Fichero Resultados2.txt [Antes de la desconexión de nodos].

Vemos que de momento la ejecución es muy parecida a la del primer test.

Veamos que ocurre cuando desconectamos los nodos 2 y 6.

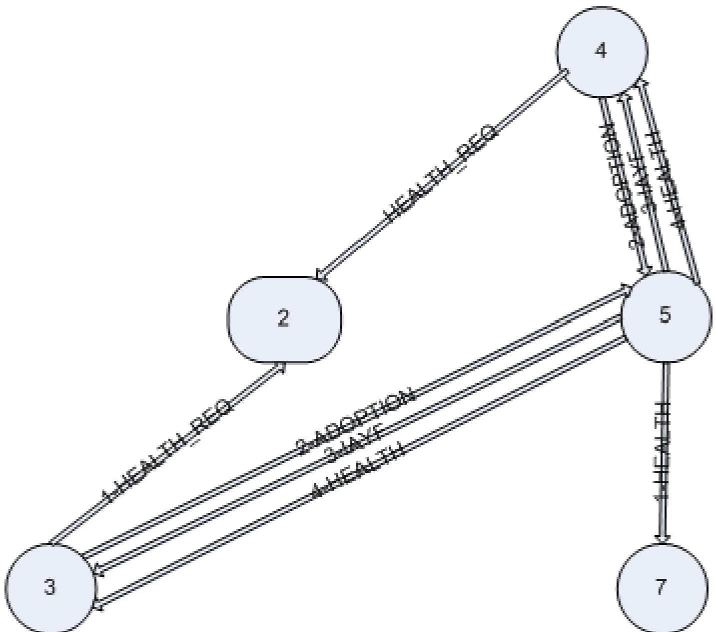


Fig. Apéndice 10. Representación Gráfica del Contenido del Fichero Resultados2.txt [Después de la desconexión de nodos].

Tal y como se aprecia en la figura, vemos que antes de desconectar los nodos 2 y 6, los nodos 3 y 4 eran hijos del nodo 2 y el nodo 6 era hijo del nodo 5, y que después de la desconexión el nodo 6 simplemente dejó de transmitir, y en los nodos 3 y 4 se desencadenó una serie de mensajes que culminaron en la adopción de ambos por parte del nodo 5.

Así que una vez más podemos dar el test por concluido satisfactoriamente.

## *B.3 Test3*

El último de los tests del simulador tiene como objetivo comprobar el correcto funcionamiento de la agregación y propagación de muestras.

En este caso el programa del simulador será prácticamente igual al del primer test, nos limitaremos arrancar los nodos y a ejecutar sus eventos.

```
#! /usr/bin/python
from TOSSIM import *
import sys

t=Tossim([])
r=t.radio()
f=open("toposamples.txt","r")
res=open("resultadossamples","w")
lines=f.readlines()
for line in lines:
	s=line.split()
	if(len(s) > 0):
		print " ",s[0]," ",s[1]," ",s[2];
		r.add(int(s[0]), int(s[1]), float(s[2]))

t.addChannel("FireDetection", res)
t.addChannel("Boot", res)

noise=open("/opt/tinyos-2.x/tos/lib/tossim/noise/meyer-heavy.txt","r")
lines=noise.readlines()
for line in lines:
	str=line.strip()
	if(str!=""):
		val=int(str)
		for i in range(1,7):
			t.getNode(i).addNoiseTraceReading(val)

for i in range(1,7):
```

```
        print "Creating noise model for ",i;
        t.getNode(i).createNoiseModel()
t.getNode(1).bootAtTime(0);
t.getNode(2).bootAtTime(0);
t.getNode(3).bootAtTime(0);
t.getNode(4).bootAtTime(0);
t.getNode(5).bootAtTime(0);
t.getNode(6).bootAtTime(0);
for i in range(0,200000):
        t.runNextEvent()

f.close();
res.close();
noise.close();
```

Fig. Apéndice 10. Contenido del Fichero TestSamples.py.

El cambio más significativo se produce en la topología de la red. En este caso, hemos buscado maximizar los niveles de propagación.

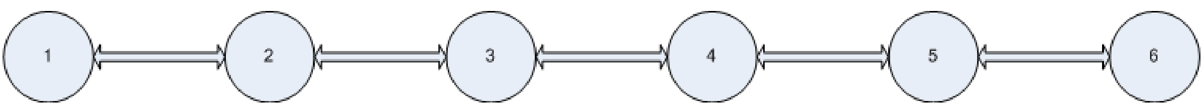



Fig. Apéndice 11. Representación Gráfica del Contenido del Fichero TopoSamples.txt.

Esta vez para comprobar si el resultado es el esperado hemos colocado una tabla en el nodo 1, y si todo funciona como debe en esa tabla veremos las muestras propagadas de todos los nodos que forman la red. Veamos como evoluciona la tabla:

| | Tiempo t0 | Tiempo t1 | Tiempo t2 | Tiempo t3 | Tiempo t4 | Tiempo t5 | Tiempo t6 |
|---|---|---|---|---|---|---|---|
| Procedencia de la muestra | | | | | | | |
| 1 | X | X | X | X | X | X | X |
| 2 | | X | X | X | X | X | X |
| 3 | | | X | X | X | X | X |
| 4 | | | | X | X | X | X |
| 5 | | | | | X | X | X |
| 6 | | | | | | | X |

Fig. Apéndice 12. Representación Gráfica del Contenido del Fichero ResultadosSamples.txt.

Tal y como se puede apreciar en la tabla, el primer nodo acaba recibiendo muestras de todos los nodos, incluso del más lejano, por lo que podemos dar el test por satisfactorio.

## *B.3 Test4*

Finalmente, sometemos al sistema a una prueba real que queda registrada en video y se puede encontrar en el cd con el código fuente del sistema. La idea sería unir todos los tests anteriores en uno y además sacarlo del entorno controlado del simulador.

# C Código

En esta sección presentaremos las clases mas importantes del sistema, así como las dos versiones del código (en TinyOS-1.X y en TinyOS-2.X) para que se pueda observar en detalle como varía de un sistema a otro.

## *C.1 Código TinyOS-1.X*

### *C.1.1.1. FireDetection.h*

```
#ifndef FIREDETECTION_H
#define FIREDETECTION_H

enum{
	AM_BLINKTORADIO=6,
	AM_FIREDETECTIONSAMPLE=6,
	IHNF=0,
	IMHF=1,
	IHAF=2,
	SAMPLE=3,
	MAX_CHILD=10,
	MAX_MESSAGES=10,
	WFS=8,
	ILMF=9,
	SONS_RETRY=1
};

enum{
```

```
        TD=4,
        YAMF=5,
        YAMF_ACK=6,
        HEALTH=7,
        HEALTH_REQ=8,
        ADOPTION=9,
        IAYF=10,
        TEMPERATURE_SAMPLE=11,
        HUMIDITY_SAMPLE=12
};

enum{
        TIMER0_PERIOD_MILLI=10000,
        TIMER1_PERIOD_MILLI=10000,
        TIMER2_PERIOD_MILLI=10000,
        SAMPLE_TIMER_PERIOD_MILLI=10000,
        PROTECTTIMER_PERIOD_MILLI=120000,
        HUMIDITY_PERIOD_MILLI=3600000,
        SAMPLES_FOR_BCAST=1,
        HSAMPLES_FOR_BCAST=5,
        HEALTHMARGIN=SAMPLES_FOR_BCAST,
        NUM_SAMPLES=10
};

typedef nx_struct Child
{
        nx_uint16_t id;
};
```

```
typedef nx_struct Sample
{
	nx_uint8_t id;
	nx_uint8_t humtemp;
};

typedef nx_struct TSamplesList
{
	nx_struct Sample samples[10];
	nx_uint8_t nextSample;
};

typedef nx_struct FireDetectionMsg
{
	nx_uint16_t source;
	nx_uint16_t counter;
	nx_uint8_t type;
} FireDetectionMsg;

typedef nx_struct FireDetectionSample
{
	nx_uint16_t source;
	nx_uint16_t counter;
	nx_uint8_t type;
	nx_struct TSamplesList samples;
} FireDetectionSample;
#endif
```

## *C.1.1.2. FireDetectionApp.nc*

```
#include "FireDetection.h"
configuration FireDetectionAppC
{
}

implementation{
        components Main,LedsC;
        components FireDetectionC as App;
        components TimerC as Timer0,TimerC as Timer1, TimerC as Timer2, TimerC as
SampleTimer,TimerC as ProtectTimer;
        components GenericComm as Comm;
        components SensirionHumidity;
        components new QueueC(nx_struct Child,MAX_CHILD) as Sons;
        components new SamplesListC(nx_struct Sample,10) as List0;
        components new SamplesListC(nx_struct Sample, 10) as List1;
        components new SamplesListC(nx_struct Sample, 10) as ListH;

        Main.StdControl->Timer0.StdControl;
        Main.StdControl->Timer1.StdControl;
        Main.StdControl->Timer2.StdControl;
        Main.StdControl->ProtectTimer.StdControl;
        Main.StdControl->SampleTimer.StdControl;
        Main.StdControl->App.StdControl;
        Main.StdControl->Comm.Control;
        App.Leds->LedsC.Leds;
        App.AMSend->Comm.SendMsg[AM_FIREDETECTIONSAMPLE];
```

```
	App.ReceiveMsg->Comm.ReceiveMsg[AM_FIREDETECTIONSAMPLE];
	App.Timer0->Timer0.Timer[unique("Timer")];
	App.Timer1->Timer1.Timer[unique("Timer")];
	App.Timer2->Timer2.Timer[unique("Timer")];
	App.SampleTimer->SampleTimer.Timer[unique("Timer")];
	App.ProtectTimer->ProtectTimer.Timer[unique("Timer")];
	App.Sons->Sons;
	App.List0->List0;
	App.List1->List1;
	App.ListH->ListH;
	App.TempHumControl -> SensirionHumidity;
	App.Humidity -> SensirionHumidity.Humidity;
	App.Temperature -> SensirionHumidity.Temperature;
	App.HumidityError -> SensirionHumidity.HumidityError;
	App.TemperatureError -> SensirionHumidity.TemperatureError;
}
```

### C.1.1.3 FireDetectionC.nc

```
#include "FireDetection.h"
module FireDetectionC
{
	provides interface StdControl;
	uses
	{
		interface Timer as Timer0;
		interface Timer as Timer1;
		interface Timer as Timer2;
```

```
            interface Timer as SampleTimer;
            interface Timer as ProtectTimer;
            interface SendMsg as AMSend;
            interface ReceiveMsg;
            interface Leds;
            interface Queue<nx_struct Child> as Sons;
            interface SamplesList<nx_struct Sample> as List0;
            interface SamplesList<nx_struct Sample> as List1;
            interface SamplesList<nx_struct Sample> as ListH;
            interface SplitControl as TempHumControl;
            interface ADC as Humidity;
            interface ADC as Temperature;
            interface ADCError as HumidityError;
            interface ADCError as TemperatureError;
      }
}
implementation
{
      uint16_t counter=0;
      uint8_t state,sons,father,timer1,samples,son, sensordata,activeList;
      uint32_t hsamples;
      bool busy,yamf_ack, healthbcast, waitingforhealth, missedhealth,
health_ackmissing, adopting, sending_tsample, sending_hsample, sensor_ready,
data_ready, mode_protected;
      TOS_Msg pkt;

      command result_t StdControl.init()
      {
```

```
        dbg("Boot","app booted\n");
        atomic state=IHNF;
        timer1=0;
        activeList=0;
        sons=0;
        father=90;
        busy=FALSE;
        atomic sensor_ready=FALSE;
        atomic data_ready=FALSE;
        mode_protected=FALSE;
        call Leds.init();
        call TempHumControl.init();
        return SUCCESS;
}

command result_t StdControl.start()
{
        call HumidityError.enable();
        call TemperatureError.enable();
        call TempHumControl.start();
        return SUCCESS;
}

command result_t StdControl.stop()
{
        call TempHumControl.stop();
        return call Timer0.stop();
}
```

```
task void sendMessage()
{
	if(busy)
	{
		post sendMessage();
	}
	else
	{
		FireDetectionMsg* packet = (FireDetectionMsg*)(pkt.data);
		packet->source=TOS_LOCAL_ADDRESS;
		packet->counter=counter;
		counter++;
		switch(state)
		{
			case IHNF:
			packet->type=YAMF;
			if(call AMSend.send(father, sizeof(FireDetectionMsg),
								&pkt)==SUCCESS)
			{
				dbg("FireDetection", "Message yamf sent to %d\n",
										father);
				busy=TRUE;
				atomic state=IHAF;
				yamf_ack=FALSE;
				call Timer2.start(TIMER_ONE_SHOT,
							TIMER2_PERIOD_MILLI);
			}
			else
```

```
{
        dbg("FireDetection","Message yamf NOT sent\n");
        post sendMessage();
}
break;

case IHAF:
packet->type=TD;
if(call AMSend.send(TOS_BCAST_ADDR,
            sizeof(FireDetectionMsg), &pkt)==SUCCESS)
{
        dbg("FireDetection","Message td sent to all\n");
        call Leds.greenToggle();
        busy=TRUE;
        atomic state=WFS;
        call Timer1.start(TIMER_ONE_SHOT,
                            TIMER1_PERIOD_MILLI);
}
else
{
        dbg("FireDetection","Message td NOT sent\n");
        post sendMessage();
}
break;

case WFS:
sons++;
packet->type=YAMF_ACK;
```

```
dbg("FireDetection","WFS %d\n",call Sons.size());
if(!(call Sons.empty()))
{
        nx_struct Child child;
        child=call Sons.dequeue();
        if(call AMSend.send(child.id,
                sizeof(FireDetectionMsg), &pkt)==SUCCESS)
        {
                dbg("FireDetection", "Message yamack sent
                                        to %d\n", child.id);
                busy=TRUE;
        }
        else
        {
                call Sons.enqueue(child);
                dbg("FireDetection","Message yamack NOT
                                                        sent\n");
        }
}
else
{
        dbg("FireDetection","WFS else\n");

}
break;

case SAMPLE:
if(adopting)
```

```
{
        packet->type=IAYF;
        if(call AMSend.send(son,
                sizeof(FireDetectionMsg),&pkt)==SUCCESS)
        {
                dbg("FireDetection","Message IAYF sent to
                                                        SON\n");
                busy=TRUE;
                adopting=FALSE;
        }
        else
        {
                dbg("FireDetection","Message IAYF NOT
                                                        sent\n");
                post sendMessage();
        }
}
else if(healthbcast)
{
        packet->type=HEALTH;
        if(call AMSend.send(TOS_BCAST_ADDR,
                sizeof(FireDetectionMsg),&pkt)==SUCCESS)
        {
                dbg("FireDetection","Message health sent to
                                                        all\n");
                busy=TRUE;
        }
        else
```

```
		{
			dbg("FireDetection","Message health NOT
							sent\n");
			post sendMessage();
		}
	}
	else if(sending_hsample)
	{
		uint8_t i;
		nx_struct Sample samp;

		FireDetectionSample* spacket =
			(FireDetectionSample*)(pkt.data);
		spacket->source=TOS_LOCAL_ADDRESS;
		spacket->counter=counter-1;
		spacket->type=HUMIDITY_SAMPLE;
		call Leds.greenToggle();
		spacket->samples.nextSample=0;

		i=0;
		while(i < call ListH.getNextSample())
		{
			samp=call ListH.getSample(i);
			spacket->samples.samples[i].id=samp.id;
			spacket->samples.samples[i].humtemp=
						samp.humtemp;
			spacket->samples.nextSample++;
			i++;
```

```
        }
        if(call AMSend.send(father,
        sizeof(FireDetectionSample), &pkt)==SUCCESS)
        {
                dbg("FireDetection","Message humidity sent
                                        to father %d\n",father);
                busy=TRUE;
                sending_hsample=FALSE;
                call ListH.reset();
        }
        else
        {
                dbg("FireDetection","Message health NOT
                                                        sent\n");
                post sendMessage();
        }
}
else if(sending_tsample)
{
        uint8_t i,size0,size1,list;
        nx_struct Sample samp;
        FireDetectionSample* spacket =
                        (FireDetectionSample*)(pkt.data);
        spacket->source=TOS_LOCAL_ADDRESS;
        spacket->counter=counter-1;

        spacket->type=TEMPERATURE_SAMPLE;
        i=0;
```

```
spacket->samples.nextSample=0;
size0= call List0.getNextSample();
size1= call List1.getNextSample();

if(size0 == call List0.maxSize() || size0 > size1)
{
        while(i < size0)
        {
                samp=call List0.getSample(i);
                spacket->samples.samples[i].id=
                                                samp.id;
                spacket->samples.samples[i].
                        humtemp=samp.humtemp;
                spacket->samples.nextSample++;
                i++;
        }
        list=0;
}
else
{
        while(i < size1)
        {
                samp=call List1.getSample(i);
                spacket->samples.
                                samples[i].id=samp.id;
                spacket->samples.samples[i].
                        humtemp=samp.humtemp;
                spacket->samples.nextSample++;
```

```
                    i++;
                }
                list=1;
            }
            dbg("FireDetection","justo al enviar %d\n",
                                spacket->samples.nextSample);

            if(call AMSend.send(father,
            sizeof(FireDetectionSample),&pkt)==SUCCESS)
            {
                dbg("FireDetection","Message with samples
                        sent to father %d\n",father);
                busy=TRUE;
                sending_tsample=FALSE;
            }
            else
            {
                dbg("FireDetection","Message with samples
                                        NOT sent\n");
                post sendMessage();
            }
        }
        else if(health_ackmissing==TRUE)
        {
            packet->type=HEALTH_REQ;
            if(call AMSend.send(father,
                sizeof(FireDetectionMsg),&pkt)==SUCCESS)
            {
```

```
                    dbg("FireDetection","Message health_req
                                                    sent to father\n");
                    busy=TRUE;
          }
          else
          {
                    dbg("FireDetection","Message health_req
                                                    NOT sent\n");
                    post sendMessage();
          }
}
break;

case ILMF:
packet->type=ADOPTION;
if(call AMSend.send(TOS_BCAST_ADDR,
                    sizeof(FireDetectionMsg),&pkt)==SUCCESS)
{
          father=255;
          call Leds.redToggle();
          call Timer0.start(TIMER_ONE_SHOT,
                                          TIMER0_PERIOD_MILLI);
          dbg("FireDetection","Message adoption sent to
                                                            all\n");
          busy=TRUE;
}
else
{
```

```
                    dbg("FireDetection","Message adoption NOT
                                                    sent\n");
                    post sendMessage();
                }
                break;
            }
        }
        return;
    }

    event result_t AMSend.sendDone(TOS_MsgPtr msg, result_t success)
    {
        dbg("FireDetection","sendDone\n");
        busy=FALSE;
        call Leds.yellowToggle();
        if(healthbcast)
        {
            healthbcast=FALSE;
            sending_tsample=TRUE;
            post sendMessage();
        }
        return SUCCESS;
    }

    void iSampleTimer()
    {
        atomic state=SAMPLE;
        samples=0;
```

```
        hsamples=0;
        sending_hsample=FALSE;
        healthbcast=FALSE;
        waitingforhealth=FALSE;
        health_ackmissing=FALSE;
        missedhealth=0;
        sending_tsample=FALSE;
        call List0.reset();
        call List1.reset();
        call ListH.reset();
        call SampleTimer.start(TIMER_REPEAT,SAMPLE_TIMER_PERIOD_MILLI);
}

event result_t Timer1.fired()
{
        timer1++;
        if(timer1<=SONS_RETRY)
        {
                if(sons==0)
                {
                        atomic state=IHAF;
                        post sendMessage();
                }
                else
                {
                        iSampleTimer();
                }
        }
```

```
        else
        {
                call Leds.redToggle();
                iSampleTimer();
        }
        dbg("FireDetection","Timer1 fired father is %d\n",father);
        return SUCCESS;
}

task void sendHealth()
{
        FireDetectionMsg* packet = (FireDetectionMsg*)(pkt.data);
        packet->source=TOS_LOCAL_ADDRESS;
        packet->counter=counter;
        counter++;
        packet->type=HEALTH;
        if(call AMSend.send(son,sizeof(FireDetectionMsg),&pkt)==SUCCESS)
        {
                dbg("FireDetection","Message health sent to son\n");
                busy=TRUE;
        }
        else
        {
                dbg("FireDetection","Message health NOT sent\n");
                post sendHealth();
        }
        return;
}
```

```
task void sendDelayedAck()
{
	FireDetectionMsg* packet = (FireDetectionMsg*)(pkt.data);
	packet->source=TOS_LOCAL_ADDRESS;
	packet->counter=counter;
	counter++;
	packet->type=YAMF_ACK;
	if(call AMSend.send(son,sizeof(FireDetectionMsg),&pkt)==SUCCESS)
	{
		dbg("FireDetection","Message YAMF_ACK sent to %d\n",son);
		busy=TRUE;
	}
	else
	{
		dbg("FireDetection","Message YAMF_ACK NOT sent\n");
		post sendHealth();
	}
	return;
}

event result_t Timer2.fired()
{
	if(yamf_ack==FALSE)
	{
		atomic state=IHNF;
		post sendMessage();
	}
	return SUCCESS;
```

```
}

event result_t SampleTimer.fired()
{
        samples++;
        hsamples++;

        if(waitingforhealth)
        {
                if(missedhealth <= HEALTHMARGIN)
                {
                        missedhealth++;
                }
                else
                {
                        if(!health_ackmissing)
                        {
                                health_ackmissing=TRUE;
                                missedhealth=0;
                                post sendMessage();
                        }
                        else
                        {
                                call SampleTimer.stop();
                                atomic state=ILMF;
                                post sendMessage();
                        }
                }
```

```
	}
	if(samples==SAMPLES_FOR_BCAST)
	{
		bool aux;
		atomic aux=sensor_ready;
		if(aux==TRUE)
		{
			call Temperature.getData();
			sensor_ready=FALSE;
		}
		else
		{
			call Leds.yellowToggle();
		}
	}
	return SUCCESS;
}

event result_t Timer0.fired()
{
	if(father==255)
	{
		post sendMessage();
	}
	return SUCCESS;
}
```

```
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr msg)
{
	if(msg->length==sizeof(FireDetectionMsg))
	{
		FireDetectionMsg* packet = (FireDetectionMsg*)(msg->data);
		dbg("FireDetection","Message received %d from %d \n",state,
								packet->source);
		switch (packet->type)
		{
			case TD:
			if(state==IHNF && !mode_protected)
			{
				dbg("FireDetection","TD received\n");
				father=packet->source;
				mode_protected=TRUE;
				call ProtectTimer.start(TIMER_ONE_SHOT,
						PROTECTTIMER_PERIOD_MILLI);
				post sendMessage();
			}
			else
			{
				if(!mode_protected)
				{
					father=packet->source;
					mode_protected=TRUE;
					call ProtectTimer.start(TIMER_ONE_SHOT,
						PROTECTTIMER_PERIOD_MILLI);
					state=IHNF;
```

```
                    post sendMessage();
          }
}
break;
case YAMF:
if(state==WFS)
{
          nx_struct Child child;
          child.id=packet->source;
          call Sons.enqueue(child);
          post sendMessage();
          dbg("FireDetection","YAMF received\n");
}
else
{
          son=packet->source;
          post sendDelayedAck();
          dbg("FireDetection","YAMF received\n");
}
break;
case YAMF_ACK:
if(state==IHAF)
{
          yamf_ack=TRUE;
          dbg("FireDetection","YAMF_ACK received\n");
          post sendMessage();
}
break;
```

```
case HEALTH:
if(state==SAMPLE)
{
	if(packet->source==father)
	{
		dbg("FireDetection","HEALTH from %d
						received\n",father);
		waitingforhealth=FALSE;
		missedhealth=0;
		health_ackmissing=FALSE;
	}
}
break;
case HEALTH_REQ:
if(state==SAMPLE)
{
	son=packet->source;
	post sendHealth();
}
break;
case ADOPTION:
if(state==SAMPLE)
{
	if(packet->source != father)
	{
		son=packet->source;
		adopting=TRUE;
		post sendMessage();
```

```
                    }
                }
                break;
                case IAYF:
                if(state==ILMF)
                {
                    father=packet->source;
                    iSampleTimer();
                    dbg("FireDetection","adoptado por %d\n",father);
                }
                break;
            }
        }
        else
        {
            if(msg->length==sizeof(FireDetectionSample))
            {
                FireDetectionSample* packet = (FireDetectionSample*)
                                                        (msg->data);
                dbg("FireDetection","Message received %d from
                                        %d \n",state, packet->source);
                switch (packet->type)
                {
                    case TEMPERATURE_SAMPLE:
                    if(state==SAMPLE)
                    {
                        uint8_t size;
                        uint8_t i=0;
```

```
uint8_t lastElem;
if(activeList==0)
{
	size=call List0.maxSize();
	lastElem =
		packet->samples.nextSample;
	while(call List0.getNextSample()
			< size && i < lastElem)
	{
		call List0.addSample(
		packet->samples.samples[i]);
		i++;
	}
	while(i<lastElem)
	{
		call List1.addSample(
		packet->samples.samples[i]);
		i++;
	}
	if(call List0.getNextSample() == size)
	{
		activeList=1;
		sending_tsample=TRUE;
		post sendMessage();
	}
}
else
{
```

```
                size=call List1.maxSize();
                lastElem=
                        packet->samples.nextSample;
                while(call List1.getNextSample()
                                < size && i < lastElem)
                {
                        call List1.addSample(
                        packet->samples.samples[i]);
                        i++;
                }
                while(i<lastElem)
                {
                        call List0.addSample(
                        packet->samples.samples[i]);
                        i++;
                }
                if(call List1.getNextSample() == size)
                {
                        activeList=0;
                        sending_tsample=TRUE;
                        post sendMessage();
                }
        }
}
break;
case HUMIDITY_SAMPLE:
if(state==SAMPLE)
{
```

```
                        uint8_t i=0;
                        while(i < packet->samples.nextSample)
                        {
                            call ListH.addSample(
                                packet->samples.samples[i]);
                            i++;
                        }
                        sending_hsample=TRUE;
                        post sendMessage();
                    }
                    break;
                }
            }
            else
            {
                dbg("FireDetection","ERROR RECEIVING SIZE\n");
            }
        }
        return msg;
    }

    async event result_t Temperature.dataReady(uint16_t data)
    {
        nx_struct Sample s;
        atomic sensordata=-38.4+(data*0.0098);
        atomic s.humtemp=sensordata;
        s.id=TOS_LOCAL_ADDRESS+father*16;
        if(activeList==0)
```

```
        {
                call List0.addSample(s);
                dbg("FireDetection","size of list0 %d\n",call
                                        List0.getNextSample());
        }
        else
        {
                call List1.addSample(s);
                dbg("FireDetection","size of list1 %d\n",call
                                        List1.getNextSample());
        }
        atomic healthbcast=TRUE;
        atomic samples=0;
        post sendMessage();
        atomic waitingforhealth=TRUE;
        atomic sensor_ready=TRUE;

        if(hsamples==HSAMPLES_FOR_BCAST)
        {
                call Humidity.getData();
        }
        return SUCCESS;
}

async event result_t Humidity.dataReady(uint16_t data)
{
        nx_struct Sample s;
        atomic sensordata=-4.0+0.0405*data-0.0000028*data*data;
```

```
        s.id=TOS_LOCAL_ADDRESS+father*16;
        atomic s.humtemp=-4.0+0.0405*data-0.0000028*data*data;
        atomic hsamples=0;
        atomic sending_hsample=TRUE;
        call ListH.addSample(s);
        post sendMessage();
        return SUCCESS;
}

event result_t TempHumControl.startDone()
{
        atomic sensor_ready=TRUE;
        call Leds.yellowToggle();
        return SUCCESS;
}

event result_t TempHumControl.initDone()
{
        return SUCCESS;
}

event result_t TempHumControl.stopDone()
{
        return SUCCESS;
}
```

```
        event result_t TemperatureError.error(uint8_t token)
        {
                call Leds.yellowToggle();
                return SUCCESS;
        }

        event result_t HumidityError.error(uint8_t token)
        {
                call Leds.yellowToggle();
                return SUCCESS;
        }

        event result_t ProtectTimer.fired()
        {
                mode_protected=FALSE;
                return SUCCESS;
        }
}
```

## C.2 Código JAVA

### C.2.1 MsgParser.java

```
import java.util.*;
import java.sql.*;

import net.tinyos.message.*;
import net.tinyos.packet.*;
```

```
import net.tinyos.util.*;

public class MsgParser implements net.tinyos.message.MessageListener
{
	private MoteIF moteIF;
	static Statement st;
	static ResultSet rs;
	static Connection conn = null;
	public static final int TEMPERATURE_TYPE=11;
	public static final int HUMIDITY_TYPE=12;
	int paqs=0;

	public MsgParser(String source) throws Exception
	{
		if(source != null)
		{
			moteIF=new MoteIF(BuildSource.makePhoenix
							(source,PrintStreamMessenger.err));
		}
		else
		{
			moteIF=new MoteIF(BuildSource.makePhoenix
							(PrintStreamMessenger.err));
		}
	}


	public void start()
```

```
{
}

public void messageReceived(int to, Message message)
{
	FireDetectionSample fds;
	java.util.Date hoy;
	long lnMilisegundos;
	java.sql.Time horat;
	String hora;
	Short stype;
	Integer itype,temps[],cid,csample,oldcid;
	short[] ids;
	short[] samples;
	short nextSample;

	fds=(FireDetectionSample)message;
	hoy=new java.util.Date();
	lnMilisegundos = hoy.getTime();
	horat = new java.sql.Time(lnMilisegundos);
	hora="'"+horat.toString()+"'";
	stype=new Short(fds.get_type());
	itype=(Integer)stype.intValue();
	temps = new Integer[10];

	ids =fds.get_samples_samples_id();
	samples=fds.get_samples_samples_humtemp();
	nextSample=fds.get_samples_nextSample();
```

```
for(int i=0; i < nextSample; i++)
{
	cid=(Integer)((new Short(ids[i])).intValue());
	oldcid=cid;
	csample=(Integer)((new Short(samples[i])).intValue());
	if(itype.intValue()==TEMPERATURE_TYPE)
	{
		try
		{
			rs=st.executeQuery("select * from sensores");
			Integer actid;
			boolean found=false;
			while(rs.next() && !found)
			{
				actid=rs.getInt("id");
				if(actid.intValue()%16==cid.intValue()%16)
				{
					found=true;
					oldcid=actid;
				}
			}

			rs=st.executeQuery("select * from sensores where
									id="+oldcid+"");
			if(rs.next())
			{
				Integer temp=rs.getInt("ultimaTemp");
```

```
temp++;
switch(temp-1)
{
        case(0):
                st.executeUpdate("update
sensores set temp1="+csample+", ultimaTemp="+temp+",id="+cid+",
        fecha="+hora.toString()+" where id="+oldcid);
        break;

        case(1):
                st.executeUpdate("update
sensores set temp2="+csample+", ultimaTemp="+temp+",id="+cid+",
        fecha="+hora.toString()+" where id="+oldcid);
        break;

        case(2):
                st.executeUpdate("update
sensores set temp3="+csample+", ultimaTemp="+temp+",id="+cid+",
        fecha="+hora.toString()+" where id="+oldcid);
        break;

        case(3):
                st.executeUpdate("update
sensores set temp4="+csample+", ultimaTemp="+temp+",id="+cid+",
        fecha="+hora.toString()+" where id="+oldcid);
        break;

        case(4):
```

```
st.executeUpdate("update
sensores set temp5="+csample+", ultimaTemp="+temp+",id="+cid+",
fecha="+hora.toString()+" where id="+oldcid);
break;

case(5):
st.executeUpdate("update
sensores set temp6="+csample+", ultimaTemp="+temp+",id="+cid+",
fecha="+hora.toString()+" where id="+oldcid);
break;

case(6):
st.executeUpdate("update
sensores set temp7="+csample+", ultimaTemp="+temp+",id="+cid+",
fecha="+hora.toString()+" where id="+oldcid);
break;

case(7):
st.executeUpdate("update
sensores set temp8="+csample+", ultimaTemp="+temp+",id="+cid+",
fecha="+hora.toString()+" where id="+oldcid);
break;

case(8):
st.executeUpdate("update
sensores set temp9="+csample+", ultimaTemp="+temp+",id="+cid+",
fecha="+hora.toString()+" where id="+oldcid);
break;
```

```
                    case(9):
                        st.executeUpdate("update
        sensores set temp0="+csample+", ultimaTemp="+0+",id="+cid+",
                fecha="+hora.toString()+" where id="+oldcid);
                    break;

                    default:
                    break;

                }
                rs.close();
            }
            else
            {
                st.executeUpdate("insert into sensores (id,
    temp0, ultimaTemp,fecha) values ("+cid+","+csample+",0,"+hora+")");
            }

        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    else if(itype.equals(HUMIDITY_TYPE))
    {
        try
```

```
                    {
                              st.executeUpdate("update sensores set humedad=
                                                  "+csample+" where id="+cid+"");
                    }
                    catch (Exception e)
                    {
                              e.printStackTrace();
                    }
          }
     }
}

private static void usage()
{
     System.err.println("usage: MsgParser [-comm <source>] message-class
                                                                      [message-class ...]");
}

private void addMsgType(Message msg)
{
     moteIF.registerListener(msg,this);
}

public static void main(String[] args) throws Exception
{
     String source=null;
     Vector v=new Vector();
     if(args.length > 0)
```

```
{
    for(int i=0; i < args.length; i++)
    {
        if(args[i].equals("-comm"))
        {
            source=args[++i];
        }
        else
        {
            String className=args[i];
            try
            {
                Class c=Class.forName(className);
                Object packet=c.newInstance();
                Message msg=(Message)packet;
                v.addElement(msg);
            }
            catch (Exception e)
            {
                System.err.println(e);
            }
        }
    }
}
else if(args.length != 0)
{
    usage();
    System.exit(1);
```

```
			}
		try
			{
				conn=BDManager.conectarBD();
				st=conn.createStatement();
				try
				{
					st.executeUpdate("DROP TABLE sensores");
				}
				catch (Exception e)
				{
					System.out.println("Table sensores does not exist");
				}
				st.executeUpdate("CREATE TABLE sensores (id INT,temp0
INT,temp1 INT,temp2 INT,temp3 INT,temp4 INT,temp5 INT,temp6 INT,temp7 INT,temp8
INT,temp9 INT,ultimaTemp INT,humedad INT,dc INT,fecha TIME, PRIMARY KEY(id))");
				rs=st.executeQuery("select * from sensors");
				while(rs.next())
				{
					System.out.println("id"+rs.getInt(1));
				}
				rs.close();
				System.out.println("FIN RESULTADOS");
		}
			catch (Exception e)
			{
				e.printStackTrace();
			}
```

```
		MsgParser mr=new MsgParser(source);
		Enumeration msgs = v.elements();
		while(msgs.hasMoreElements())
		{
			Message m=(Message)msgs.nextElement();
			mr.addMsgType(m);
		}
		mr.start();
		FireMenu fm=new FireMenu();
	}
}
```

## C.2.2 FireMenu.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.util.*;
import java.sql.*;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;

public class FireMenu extends JFrame
{
	Container contenedor;
	JTable tabla,tabla2;
	JLabel etiq1,etiq2;
```

```
JScrollPane scroll,scroll2;
DefaultTableModel modelo,modelo2;
DefaultMutableTreeNode estacionBase,estacionBase2;
DefaultTreeModel tmodelo,tmodelo2;
JTree nodos;
JMenuBar barraMenu=new JMenuBar();
JMenu menuAcciones=new JMenu("Acciones");
JMenu menuConsultas=new JMenu("Consultas");
JMenu menuAyuda=new JMenu("Ayuda");

JMenuItem opcionCalcularDC=new JMenuItem("Calcular DC");
JMenuItem opcionNodosInactivos=new JMenuItem("Nodos Inactivos");
JMenuItem opcionPorTemp=new JMenuItem("Por Temperatura");
JMenuItem opcionPorHum=new JMenuItem("Por Humedad");
JMenuItem opcionPorDC=new JMenuItem("Por DC");
JMenuItem opcionSalir=new JMenuItem("Salir");
JMenuItem opcionAyuda=new JMenuItem("Ayuda");

GridBagConstraints constraints;

static Statement st;
static ResultSet rs;
static Connection conn = null;

public FireMenu()
{
	super("Prometeo - Sistema de Monitorización Forestal");
	contenedor=getContentPane();
```

```
menuAcciones.add(opcionCalcularDC);
menuAcciones.add(opcionNodosInactivos);
menuAcciones.add(opcionSalir);
menuConsultas.add(opcionPorTemp);
menuConsultas.add(opcionPorHum);
menuConsultas.add(opcionPorDC);
menuAyuda.add(opcionAyuda);
barraMenu.add(menuAcciones);
barraMenu.add(menuConsultas);
barraMenu.add(menuAyuda);
setJMenuBar(barraMenu);
addWindowListener(new WindowHandler());
opcionCalcularDC.addActionListener(new VigilaMenu());
opcionPorTemp.addActionListener(new VigilaMenu());
opcionPorHum.addActionListener(new VigilaMenu());
opcionPorDC.addActionListener(new VigilaMenu());
opcionSalir.addActionListener(new VigilaMenu());
opcionNodosInactivos.addActionListener(new VigilaMenu());
opcionAyuda.addActionListener(new VigilaMenu());

etiq1=new JLabel("Muestras Recogidas");
etiq2=new JLabel("Resultado de la Consulta");

modelo=new DefaultTableModel();
tabla=new JTable();
tabla.setModel(modelo);
scroll=new JScrollPane(tabla);
modelo2=new DefaultTableModel();
```

```
tabla2=new JTable();
tabla2.setModel(modelo2);
scroll2=new JScrollPane(tabla2);

estacionBase=new DefaultMutableTreeNode("Estación Base");
estacionBase2=new DefaultMutableTreeNode("Estación Base");
tmodelo = new DefaultTreeModel(estacionBase);
tmodelo2 = new DefaultTreeModel(estacionBase2);
nodos = new JTree(tmodelo);

contenedor.setLayout(new GridBagLayout());
constraints=new GridBagConstraints();

setConstraints(constraints,0,0,1,1,0.0,0.0,GridBagConstraints.NONE,
                                        GridBagConstraints.WEST);
contenedor.add(etiq1,constraints);

setConstraints(constraints,0,1,2,1,1.0,1.0,GridBagConstraints.BOTH,
                                        GridBagConstraints.CENTER);
contenedor.add(scroll,constraints);

setConstraints(constraints,0,2,2,1,0.0,0.0,GridBagConstraints.NONE,
                                        GridBagConstraints.WEST);
contenedor.add(etiq2,constraints);

setConstraints(constraints,0,3,2,1,1.0,1.0,GridBagConstraints.BOTH,
                                        GridBagConstraints.CENTER);
contenedor.add(scroll2,constraints);
```

```
setConstraints(constraints,2,0,1,4,0.2,1.0,GridBagConstraints.BOTH,
                                        GridBagConstraints.WEST);
contenedor.add(nodos,constraints);

setSize(800,600);
setVisible(true);

conn=BDManager.conectarBD();
try
{
        st=conn.createStatement();
}
catch(Exception sqle)
{
        sqle.printStackTrace();
}
int modeloArbol=0;

while(true)
{
        try
        {
                rs=st.executeQuery("select * from sensores");
                RS2TableModel.convertir(rs,modelo);
                tabla.setModel(modelo);
                rs=st.executeQuery("select id,fecha from sensores");
                if(modeloArbol==0)
```

```
	{
		RS2TreeModel.eliminarNodos(tmodelo2);
		RS2TreeModel.convertir(rs,tmodelo2);
	}
	else
	{
		RS2TreeModel.eliminarNodos(tmodelo);
		RS2TreeModel.convertir(rs,tmodelo);
	}

	if(!RS2TreeModel.iguales(tmodelo, tmodelo2))
	{
		if(modeloArbol==0)
		{
			nodos.setModel(tmodelo2);
			modeloArbol=1;
		}
		else
		{
			nodos.setModel(tmodelo);
			modeloArbol=0;
		}
	}

	Thread.sleep(2000);
}
catch (Exception e)
{
```

```
                        e.printStackTrace();
                }
        }
}

public class WindowHandler extends WindowAdapter
{
        public void windowClosing(WindowEvent e)
        {
                System.exit(0);
        }
}

public class VigilaMenu implements ActionListener
{
        String respuesta;
        int k;

        public void actionPerformed(ActionEvent e)
        {
                String accion=e.getActionCommand();
                String[] res=new String[2];
                Integer i,j;
                if(accion.equals("Salir"))
                {
                        BDManager.desconectarBD();
                        System.exit(0);
                }
```

```
else if(accion.equals("Calcular DC"))
{
    int dc,temp0;
    Integer lluvia;
    ResultSet rs2;
    Statement st2;

    respuesta=JOptionPane.showInputDialog(contenedor,
"Introduce el volumen de precipitaciones durante las últimas 24h");
    lluvia=Integer.valueOf(respuesta);
    try
    {
        st2=conn.createStatement();
        rs2=st2.executeQuery("select * from sensores");
        while(rs2.next())
        {
            i=rs2.getInt("dc");
            if(i.intValue()==0)
            {
                i=771;
            }
            temp0=rs2.getInt("temp0");
            dc=DroughtCode.calculaDC(lluvia, i,temp0);
            j=rs2.getInt("id");
            st.executeUpdate("update sensores set
                    dc="+dc+" where id="+j);
        }
    }
```

```
				catch(Exception ee)
				{
					ee.printStackTrace();
				}
			}
			else if(accion.equals("Por DC"))
			{
				respuesta=JOptionPane.showInputDialog(contenedor,
		"Introduce el intervalo de DC en formato: cota inferior - cota superior");
				res=respuesta.split("-");
				i=Integer.valueOf(res[0].trim());
				j=Integer.valueOf(res[1].trim());
				try
				{
					rs=st.executeQuery("select * from sensores where
										dc>="+i+" and dc<="+j);
					deRSaModelo(rs,modelo2);
					tabla2.setModel(modelo2);
				}
				catch(Exception ee)
				{
					ee.printStackTrace();
				}
			}
			else if(accion.equals("Por Temperatura"))
			{
				respuesta=JOptionPane.showInputDialog(contenedor,
"Introduce el intervalo de Temperatura en formato: cota inferior - cota superior");
```

```
                        res=respuesta.split("-");
                        i=Integer.valueOf(res[0].trim());
                        j=Integer.valueOf(res[1].trim());
                        try
                        {
                                rs=st.executeQuery("select * from sensores where
                                        temp0>="+i+" and temp0<="+j);
                                deRSaModelo(rs,modelo2);
                                tabla2.setModel(modelo2);
                        }
                        catch(Exception ee)
                        {
                                ee.printStackTrace();
                        }
                }
                else if(accion.equals("Por Humedad"))
                {
                        respuesta=JOptionPane.showInputDialog(contenedor,
        "Introduce el intervalo de Humedad en formato: cota inferior - cota superior");
                        res=respuesta.split("-");
                        i=Integer.valueOf(res[0].trim());
                        j=Integer.valueOf(res[1].trim());
                        try
                        {
                                rs=st.executeQuery("select * from sensores where
                                        humedad>="+i+" and humedad<="+j);
                                deRSaModelo(rs,modelo2);
                                tabla2.setModel(modelo2);
```

```
					}
					catch(Exception ee)
					{
						ee.printStackTrace();
					}

				}
				else if(accion.equals("Ayuda"))
				{
					String mensaje="Esta es la ventana de ayuda de 
Prometeo.\n Si quieres conocer que nodos permanecen inactivos selecciona en el 
menú: \n \t -Acciones\n\t   ->Nodos Inactivos \n Si quieres calcular el Drought Code de 
las muestras recogidas selecciona en el menú: \n \t -Acciones\n\t   ->CalcularDC\n Por 
el contrario si quieres hacer una consulta sobre algún parámetro de las muestras 
selecciona en el menú: \n \t -Consultas\n\t   ->Por Temperatura\n\t   ->Por DC\n\t   
->Por Humedad\n y selecciona una de las anteriores.";
					JOptionPane.showInternalMessageDialog(contenedor, 
mensaje,		"Ayuda", JOptionPane.INFORMATION_MESSAGE);
				}
				else if(accion.equals("Nodos Inactivos"))
				{
					calcularNodosInactivos();
				}
			}
		}


		public void setConstraints(GridBagConstraints constraints , int gridx, int gridy,
```

```
int gridwidth, int gridheight, double weightx, double weighty, int fill, int anchor)
        {
                constraints.gridx=gridx;
                constraints.gridy=gridy;
                constraints.gridwidth=gridwidth;
                constraints.gridheight=gridheight;
                constraints.weightx=weightx;
                constraints.weighty=weighty;
                constraints.fill=fill;
                constraints.anchor=anchor;
        }

        public void deRSaModelo(ResultSet rs, DefaultTableModel modelo2)
        {
                Object[] etiquetas,datosFila;
                ResultSetMetaData metaDatos;
                int numColumnas,k,numFila;

                try
                {
                        metaDatos=rs.getMetaData();
                        numColumnas=metaDatos.getColumnCount();
                        etiquetas=new Object[numColumnas];
                        for(k=0; k < numColumnas; k++)
                        {
                                etiquetas[k]=metaDatos.getColumnLabel(k+1);
                        }
                        modelo2.setColumnIdentifiers(etiquetas);
```

```
		while(modelo2.getRowCount()>0)
		{
			modelo2.removeRow(0);
		}
	}
	catch (Exception e1)
	{
		e1.printStackTrace();
	}
	numFila=0;
	try
	{
		while(rs.next())
		{
			datosFila=new Object[modelo2.getColumnCount()];
			for(k=0; k < modelo2.getColumnCount();k++)
			{
				datosFila[k]=rs.getObject(k+1);
			}
			modelo2.addRow(datosFila);
			numFila++;
		}
	}
	catch(Exception e)
	{
		e.printStackTrace();
	}
```

```
	}

	public void calcularNodosInactivos()
	{
		java.util.Date hoy;
		long lnMilisegundos;
		java.sql.Time horat;
		String hora;
		Integer secs,mins;

		hoy=new java.util.Date();
		lnMilisegundos = hoy.getTime();
		horat = new java.sql.Time(lnMilisegundos);

		String[] horaminsec=horat.toString().split(":");

		mins=Integer.valueOf(horaminsec[1]);
		mins--;
		horaminsec[1]=mins.toString();

		if(mins<10)
		{
			horaminsec[1]="0"+horaminsec[1];
		}
hora=horaminsec[0].concat(":").concat(horaminsec[1]).concat(":").concat(horaminsec
[2]);
		try
```

```
            {
                rs=st.executeQuery("select id,fecha from sensores where
                                                        fecha<'"+hora+"'");
                deRSaModelo(rs,modelo2);
                tabla2.setModel(modelo2);
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }
        }
}
```

## C.3 Código TinyOS-2.X

### C.3.1.1. FireDetectionApp.nc

```
#include <Timer.h>
#include "FireDetection.h"

configuration FireDetectionAppC
{
}
implementation{
        components MainC;
        components LedsC;
        components FireDetectionC as App;
        components new TimerMilliC() as Timer0;
```

```
    components new TimerMilliC() as Timer1;
    components new TimerMilliC() as Timer2;
    components new TimerMilliC() as SampleTimer;
    components ActiveMessageC;
    components new AMSenderC(AM_FIREDETECTIONMSG);
    components new AMReceiverC(AM_FIREDETECTIONMSG);
    components new QueueC(nx_struct Child,MAX_CHILD) as Sons;
    components new SamplesListC(nx_struct Sample,10) as List0;
    components new SamplesListC(nx_struct Sample, 10) as List1;
    components new SamplesListC(nx_struct Sample, 10) as ListH;

    App.Boot->MainC;
    App.Leds->LedsC;
    App.Timer0->Timer0;
    App.Timer1->Timer1;
    App.Timer2->Timer2;
    App.SampleTimer->SampleTimer;
    App.Packet->AMSenderC;
    App.AMPacket->AMSenderC;
    App.AMSend->AMSenderC;
    App.AMControl->ActiveMessageC;
    App.Receive->AMReceiverC;
    App.Sons->Sons;
    App.List0->List0;
    App.List1->List1;
    App.ListH->ListH;
}
```

### *C.3.2.1.FireDetectionC.nc*

```
#include <Timer.h>
#include "FireDetection.h"

module FireDetectionC
{
	uses interface Boot;
	uses interface Leds;
	uses interface Timer<TMilli> as Timer0;
	uses interface Timer<TMilli> as Timer1;
	uses interface Timer<TMilli> as Timer2;
	uses interface Timer<TMilli> as SampleTimer;

	uses interface Packet;
	uses interface AMPacket;
	uses interface AMSend;
	uses interface SplitControl as AMControl;
	uses interface Receive;
	uses interface Queue<nx_struct Child> as Sons;
	uses interface SamplesList<nx_struct Sample> as List0;
	uses interface SamplesList<nx_struct Sample> as List1;
	uses interface SamplesList<nx_struct Sample> as ListH;
}
implementation
{
	uint16_t counter=0;
	uint8_t state;
```

```
uint8_t sons=0;
uint8_t father=90;
uint8_t timer1;
uint32_t hsamples;
bool busy = FALSE;
bool yamf_ack;
bool healthbcast;
message_t pkt;
uint8_t samples;
bool waitingforhealth;
uint8_t missedhealth;
bool health_ackmissing;
bool adopting;
bool sending_tsample;
bool sending_hsample;
uint8_t son;
uint8_t activeList;

event void Boot.booted()
{
	dbg("Boot","app booted\n");
	atomic state=IHNF;
	timer1=0;
	activeList=0;
	if(TOS_NODE_ID == 1)
	{
		atomic state=WFS;
		call Timer1.startOneShot(TIMER1_PERIOD_MILLI);
```

```
		}
		call AMControl.start();
	}

	event void AMControl.startDone(error_t err)
	{
		if(err==SUCCESS)
		{
			dbg("FireDetection","radio on\n");
			call Timer0.startOneShot(TIMER0_PERIOD_MILLI);
		}
		else
		{
			call AMControl.start();
		}
	}

	event void AMControl.stopDone(error_t err)
	{}

	event void Timer0.fired()
	{
		if(TOS_NODE_ID == 1)
		{	if(!busy)
			{
				FireDetectionMsg* btrpkt = (FireDetectionMsg*)(call
								Packet.getPayload(&pkt,NULL));
				btrpkt->source=TOS_NODE_ID;
```

```
                    btrpkt->counter=counter;
                    btrpkt->type=TD;
                    if(call AMSend.send(AM_BROADCAST_ADDR,&pkt,
                                    sizeof(FireDetectionMsg))==SUCCESS)
                    {
                        dbg("FireDetection","Message 1.TD sent\n");
                        busy=TRUE;
                    }
                    atomic state=WFS;
                    call Timer1.startOneShot(TIMER1_PERIOD_MILLI);
                }
            }
        }

        task void sendMessage()
        {
            if(busy)
            {
                post sendMessage();
            }
            else
            {
                FireDetectionMsg* packet = (FireDetectionMsg*)(call
                                            Packet.getPayload(&pkt,NULL));
                packet->source=TOS_NODE_ID;
                packet->counter=counter;
                counter++;
                switch(state)
```

```
{
        case IHNF:
        packet->type=YAMF;
        if(call AMSend.send(father,&pkt,sizeof(FireDetectionMsg))
                                                ==SUCCESS)
        {
                dbg("FireDetection","Message yamf sent to
                                                %d\n",father);
                busy=TRUE;
                atomic state=IHAF;
                yamf_ack=FALSE;
                call Timer2.startOneShot(TIMER2_PERIOD_MILLI);

        }
        else
        {
                dbg("FireDetection","Message yamf NOT sent\n");
                post sendMessage();
        }
        break;

        case IHAF:
        packet->type=TD;
        if(call AMSend.send(AM_BROADCAST_ADDR,&pkt,
                        sizeof(FireDetectionMsg))==SUCCESS)
        {
                dbg("FireDetection","Message td sent to all\n");
                busy=TRUE;
```

```
            atomic state=WFS;
            call Timer1.startOneShot(TIMER1_PERIOD_MILLI);
    }
    else
    {
            dbg("FireDetection","Message td NOT sent\n");
            post sendMessage();
    }
    break;

    case WFS:
    sons++;
    packet->type=YAMF_ACK;
    dbg("FireDetection","WFS %d\n",call Sons.size());
    if(!(call Sons.empty()))
    {
            nx_struct Child child;
            child=call Sons.dequeue();
            if(call AMSend.send((am_addr_t)child.id,&pkt,
                        sizeof(FireDetectionMsg))==SUCCESS)
            {
                    dbg("FireDetection","Message yamack sent
                                        to %d\n",child.id);
                    busy=TRUE;
            }
            else
            {
                    call Sons.enqueue(child);
```

```
            dbg("FireDetection","Message yamack NOT
                                                sent\n");
        }
    }
    else
    {
        dbg("FireDetection","WFS else\n");
    }
    break;

    case SAMPLE:
    if(adopting)
    {
        packet->type=IAYF;
        if(call AMSend.send(son,&pkt,sizeof
                    (FireDetectionMsg)) ==SUCCESS)
        {
            dbg("FireDetection","Message IAYF sent to
                                                SON\n");
            busy=TRUE;
            adopting=FALSE;
        }
        else
        {
            dbg("FireDetection","Message IAYF NOT
                                                sent\n");
            post sendMessage();
        }
```

```
}
else if(healthbcast)
{
	packet->type=HEALTH;
	if(call AMSend.send(AM_BROADCAST_ADDR,&pkt,
			sizeof(FireDetectionMsg))==SUCCESS)
	{
		dbg("FireDetection","Message health sent to
							all\n");
		busy=TRUE;
	}
	else
	{
		dbg("FireDetection","Message health NOT
							sent\n");
		post sendMessage();
	}
}
else if(sending_hsample)
{
	uint8_t i;
	nx_struct Sample samp;
	FireDetectionSample* spacket =
(FireDetectionSample*)(call Packet.getPayload(&pkt,NULL));
	spacket->source=TOS_NODE_ID;
	spacket->counter=counter-1;
	spacket->type=HUMIDITY_SAMPLE;
	i=0;
```

```
		while(i < call ListH.getNextSample())
		{
			samp=call List0.getSample(i);
			spacket->samples.samples[i].id=samp.id;
			spacket->samples.samples[i].
					humtemp=samp.humtemp;
			spacket->samples.nextSample++;
			i++;
		}
		if(call AMSend.send(father,&pkt
				,sizeof(FireDetectionMsg))==SUCCESS)
		{
			dbg("FireDetection","Message humidity sent
						to father %d\n",father);
			busy=TRUE;
			sending_hsample=FALSE;
			call List0.reset();
		}
		else
		{
			dbg("FireDetection","Message health NOT
								sent\n");
			post sendMessage();
		}
	}
	else if(sending_tsample)
	{
		uint8_t i,size0,size1,list;
```

```
            nx_struct Sample samp;
            FireDetectionSample* spacket =
    (FireDetectionSample*)(call Packet.getPayload(&pkt,NULL));
            spacket->source=TOS_NODE_ID;
            spacket->counter=counter-1;

            spacket->type=TEMPERATURE_SAMPLE;
            i=0;
            spacket->samples.nextSample=0;
            size0= call List0.getNextSample();
            size1= call List1.getNextSample();

            if(size0 == call List0.maxSize() || size0 > size1)
            {
                while(i < size0)
                {
                    samp=call List0.getSample(i);
                    spacket->samples.samples[i].
                                                id=samp.id;
                    spacket->samples.samples[i].
                        humtemp=samp.humtemp;
                    spacket->samples.nextSample++;
                    i++;
                }
                list=0;
            }
            else
            {
```

```
		while(i < size1)
		{
			samp=call List1.getSample(i);
			spacket->samples.samples[i].
							id=samp.id;
			spacket->samples.samples[i].
				humtemp=samp.humtemp;
			spacket->samples.nextSample++;
			i++;
		}
		list=1;
	}
	dbg("FireDetection","justo al enviar %d\n",
				spacket->samples.nextSample);

	if(call AMSend.send(father,&pkt,sizeof
			(FireDetectionSample))==SUCCESS)
	{
		dbg("FireDetection","Message with samples
				sent to father %d\n",father);
		busy=TRUE;
		sending_tsample=FALSE;
		if(TOS_NODE_ID != 1)
		{
			if(list==0)
			{
				call List0.reset();
			}
```

```
                else
                {
                        call List1.reset();
                }
        }
    }
    else
    {
        dbg("FireDetection","Message with samples
                                    NOT sent\n");
        post sendMessage();
    }
}
else if(health_ackmissing==TRUE)
{
    packet->type=HEALTH_REQ;
    if(call AMSend.send(father,&pkt,
                sizeof(FireDetectionMsg))==SUCCESS)
    {
        dbg("FireDetection","Message health_req
                                    sent to father\n");
        busy=TRUE;
    }
    else
    {
        dbg("FireDetection","Message health_req
                                    NOT sent\n");
        post sendMessage();
```

```
                    }
                }
                break;

                case ILMF:
                packet->type=ADOPTION;
                if(call AMSend.send(AM_BROADCAST_ADDR,&pkt,
                        sizeof(FireDetectionMsg))==SUCCESS)
                {
                    dbg("FireDetection","Message adoption sent to
                                                        all\n");
                    busy=TRUE;
                }
                else
                {
                    dbg("FireDetection","Message adoption NOT
                                                        sent\n");
                    post sendMessage();
                }
                break;
            }
        }
        return;
    }

    event void AMSend.sendDone(message_t* msg, error_t error)
    {
        dbg("FireDetection","sendDone\n");
```

```
if(&pkt==msg)
{
        busy=FALSE;
        if(healthbcast)
        {
                healthbcast=FALSE;
                if(TOS_NODE_ID != 1)
                {
                        sending_tsample=TRUE;
                        post sendMessage();
                }
                else
                {
                        uint8_t size0,size1,i;
                        nx_struct Sample sam;

                        i=0;
                        size0=call List0.getNextSample();
                        size1=call List1.getNextSample();
                        dbg("FireDetection","contenido de las tablas %d
                                                %d\n",size0,size1);
                        while(i<size0)
                        {
                                sam=call List0.getSample(i);
                                dbg("FireDetection","0temp %d id
                                        %d\n",sam.humtemp,sam.id);
                                i++;
                        }
```

```
                              i=0;
                              while(i<size1)
                              {
                                        sam=call List1.getSample(i);
                                        dbg("FireDetection","1temp %d id
                                                            %d\n",sam.humtemp,sam.id);
                                        i++;
                              }
                              call List0.reset();
                              call List1.reset();
                    }
          }
     }
     else
     {
          dbg("FireDetection","BIG ERROR\n");
     }
}

event void Timer1.fired()
{
     timer1++;
     if(timer1<=SONS_RETRY)
     {
          if(sons==0)
          {
                    atomic state=IHAF;
                    post sendMessage();
```

```
        }
        else
        {
                atomic state=SAMPLE;
                samples=0;
                hsamples=0;
                sending_hsample=FALSE;
                healthbcast=FALSE;
                waitingforhealth=FALSE;
                health_ackmissing=FALSE;
                missedhealth=0;
                sending_tsample=FALSE;
                call List0.reset();
                call List1.reset();
                call SampleTimer.startPeriodic
                                        (SAMPLE_TIMER_PERIOD_MILLI);
        }
}
else
{
        atomic state=SAMPLE;
        samples=0;
        hsamples=0;
        sending_hsample=FALSE;
        healthbcast=FALSE;
        waitingforhealth=FALSE;
        health_ackmissing=FALSE;
        missedhealth=0;
```

```
            sending_tsample=FALSE;
            call List0.reset();
            call List1.reset();
            call SampleTimer.startPeriodic(SAMPLE_TIMER_PERIOD_MILLI);
        }
        dbg("FireDetection","Timer1 fired father is %d\n",father);
}

task void sendHealth()
{
        FireDetectionMsg* packet = (FireDetectionMsg*)(call
                                                    Packet.getPayload(&pkt,NULL));
        packet->source=TOS_NODE_ID;
        packet->counter=counter;
        counter++;
        packet->type=HEALTH;
        if(call AMSend.send(son,&pkt,sizeof(FireDetectionMsg))==SUCCESS)
        {
            dbg("FireDetection","Message health sent to son\n");
            busy=TRUE;
        }
        else
        {
            dbg("FireDetection","Message health NOT sent\n");
            post sendHealth();
        }
        return;
}
```

```
task void sendDelayedAck()
{
	FireDetectionMsg* packet = (FireDetectionMsg*)(call
							Packet.getPayload(&pkt,NULL));
	packet->source=TOS_NODE_ID;
	packet->counter=counter;
	counter++;
	packet->type=YAMF_ACK;
	if(call AMSend.send(son,&pkt,sizeof(FireDetectionMsg))==SUCCESS)
	{
		dbg("FireDetection","Message YAMF_ACK sent to %d\n",son);
		busy=TRUE;
	}
	else
	{
		dbg("FireDetection","Message YAMF_ACK NOT sent\n");
		post sendHealth();
	}
	return;
}

event void Timer2.fired()
{
	if(yamf_ack==FALSE)
	{
		atomic state=IHNF;
		post sendMessage();
	}
```

```
}

event void SampleTimer.fired()
{
	samples++;
	hsamples++;
	if(TOS_NODE_ID != 1)
	{
		if(waitingforhealth)
		{
			if(missedhealth <= HEALTHMARGIN)
			{
				missedhealth++;
			}
			else
			{
				if(!health_ackmissing)
				{
					health_ackmissing=TRUE;
					missedhealth=0;
					post sendMessage();
				}
				else
				{
					call SampleTimer.stop();
					atomic state=ILMF;
					post sendMessage();
				}
```

```
            }
        }
    }
    if(samples==SAMPLES_FOR_BCAST)
    {
        nx_struct Sample s;
        s.id=TOS_NODE_ID;
        s.humtemp=30;
        if(activeList==0)
        {
            call List0.addSample(s);
            dbg("FireDetection","size of list0 %d\n",call
                                        List0.getNextSample());
        }
        else
        {
            call List1.addSample(s);
            dbg("FireDetection","size of list1 %d\n",call
                                        List1.getNextSample());
        }
        healthbcast=TRUE;
        samples=0;
        post sendMessage();
        waitingforhealth=TRUE;
    }
    if(hsamples==HSAMPLES_FOR_BCAST)
    {
        nx_struct Sample s;
```

```
			s.id=TOS_NODE_ID;
			hsamples=0;
			s.humtemp=100;
			sending_hsample=TRUE;
			call ListH.addSample(s);
			post sendMessage();
		}
	}

	event message_t* Receive.receive(message_t* msg, void* payload, uint8_t
										len)
	{
		if(len==sizeof(FireDetectionMsg))
		{
			FireDetectionMsg* packet = (FireDetectionMsg*)payload;
			dbg("FireDetection","Message received %d from %d \n",state,
								packet->source);
			switch (packet->type)
			{
				case TD:
				if(state==IHNF)
				{
					dbg("FireDetection","TD received\n");
					father=packet->source;
					post sendMessage();
				}
				break;
```

```
case YAMF:
if(state==WFS)
{
	nx_struct Child child;
	child.id=packet->source;
	call Sons.enqueue(child);
	post sendMessage();
	dbg("FireDetection","YAMF received\n");
}
else
{
	son=packet->source;
	post sendDelayedAck();
	dbg("FireDetection","YAMF received\n");
}
break;

case YAMF_ACK:
if(state==IHAF)
{
	yamf_ack=TRUE;
	dbg("FireDetection","YAMF_ACK received\n");
	post sendMessage();
}
break;
case HEALTH:
if(state==SAMPLE)
{
```

```
        if(packet->source==father)
        {
                dbg("FireDetection","HEALTH from %d
                                received\n",father);
                waitingforhealth=FALSE;
                missedhealth=0;
                health_ackmissing=FALSE;
        }
}
break;
case HEALTH_REQ:
if(state==SAMPLE)
{
        son=packet->source;
        post sendHealth();
}
break;
case ADOPTION:
if(state==SAMPLE)
{
        if(packet->source != father)
        {
                son=packet->source;
                adopting=TRUE;
                post sendMessage();
        }
}
break;
```

```
                    case IAYF:
                    if(state==ILMF)
                    {
                            father=packet->source;
                            atomic state=SAMPLE;
                            samples=0;
                            healthbcast=FALSE;
                            waitingforhealth=FALSE;
                            health_ackmissing=FALSE;
                            missedhealth=0;
                            sending_tsample=FALSE;
                            call List0.reset();
                            call List1.reset();
                            call SampleTimer.startPeriodic
                                            (SAMPLE_TIMER_PERIOD_MILLI);
                            dbg("FireDetection","adoptado por %d\n",father);
                    }
                    break;
            }
    }
    else
    {
            if(len==sizeof(FireDetectionSample))
            {
                    FireDetectionSample* packet = (FireDetectionSample*)
                                                                    payload;
                    dbg("FireDetection","Message received %d from
                                                %d \n",state, packet->source);
```

```
switch (packet->type)
{
        case TEMPERATURE_SAMPLE:
        if(state==SAMPLE)
        {
                uint8_t size;
                uint8_t i=0;
                uint8_t lastElem;
                if(activeList==0)
                {
                        size=call List0.maxSize();
                        lastElem=packet->samples.
                                                nextSample;
                        while(call List0.getNextSample() < size
                                                && i < lastElem)
                        {
                                call List0.addSample(packet->
                                        samples.samples[i]);
                                i++;
                        }
                        while(i<lastElem)
                        {
                                call List1.addSample(packet->
                                        samples.samples[i]);
                                i++;
                        }
                        if(call List0.getNextSample() == size)
                        {
```

```
                        activeList=1;
                        sending_tsample=TRUE;
                        post sendMessage();
                }
        }
        else
        {
                size=call List1.maxSize();
                lastElem=packet->
                                samples.nextSample;
                while(call List1.getNextSample() < size
                                        && i < lastElem)
                {
                        call List1.addSample(packet->
                                samples.samples[i]);
                        i++;
                }
                while(i<lastElem)
                {
                        call List0.addSample(packet->
                                samples.samples[i]);
                        i++;
                }
                if(call List1.getNextSample() == size)
                {
                        activeList=0;
                        sending_tsample=TRUE;
                        post sendMessage();
```

```
                        }
                    }
                }
                break;
                case HUMIDITY_SAMPLE:
                if(state==SAMPLE)
                {
                    uint8_t i=0;
                    while(i < packet->samples.nextSample)
                    {
                        call ListH.addSample(packet->
                                samples.samples[i]);
                        i++;
                    }
                    sending_hsample=TRUE;
                    post sendMessage();
                }
                break;
            }
        }
        else
        {
            dbg("FireDetection","ERROR RECEIVING SIZE\n");
        }
    }
    return msg;
}
}
```