Instituto Superior Técnico

MEIC

Aplicações para Sistemas Embebidos

Sensor Network for Fire Detection

Authors:
Leonardo Natal
Samuel Coelho

May 1, 2015

Contents

1 Introdução

Incêndios representam um grande risco tanto para os seres humanos quanto para animais e estruturas físicas. Quando florestal, pode trazer graves consequências para o ecossistema local, levando a um desequilíbrio ambiental.

Neste cenário, é de extrema relevância uma ferramenta que possibilite antever tais fatos ou que pelo menos os detecte de maneira prematura. E este é o propósito do trabalho aqui descrito: uma rede de sensores sem fio composta por módulos espalhados por uma grande área que se deseja monitorar. Cada módulo mede humidade, temperatura, detecta fumo e informa sua localização para um servidor que então é capaz de armazenar tais informações de maneira a facilitar o monitoramento e gestão do espaço em questão.

No entanto, não basta ter um sistema que detecta fumo e informa um servidor. Para tornar o sistema confiável e robusto foi necessário implementar não só a lógica básica de obter os dados e construir as mensagens a enviar ao servidor, como também teve que ser implementado um protocolo de comunicação que evitasse a duplicação de mensagens. No entanto, muitas falhas podem ocorrer na rede de sensores. De todos os desafios de implementação que foram enfrentados, a tolerância a faltas foi o maior de todos.

No tópico a seguir são apresentadas em detalhe as arquitecturas de *Hardware*, de *Software* e de Rede do sistema. Na secção 3 é descrito o simulador e as suas principais funcionalidades. Na secção 4 são explicadas as recomendações para a colocação e instalação dos vários módulos que constituem a rede de sensores. Por fim, em 5 é feita uma reflexão sobre todo o projecto relatado neste documento.

2 Arquitectura

A rede de sensores implementada neste projecto é constituída por três tipos de nós:

Nós sensores compostos pelos sensores de temperatura, humidade, detector de fumo e GPS, são responsáveis por gerar e transmitir dados obtidos através dos seus sensores;

Nós *routers* responsáveis por reencaminhar as mensagens dos nós sensores até ao nó servidor;

Nó servidor responsável por colectar todas as informações geradas pelos nós sensores e organizá-las em um ficheiro.

2.1 Software

É adoptada uma arquitectura de *software* que inclui o Sistema Operativo TinyOS[1], o que proporciona um gerenciamento básico dos recursos disponíveis. Junto a isso é utilizada a linguagem nesC[2], que proporciona uma boa abstracção dos detalhes de implementação por meio de sua biblioteca, ao mesmo tempo que é suficientemente flexível permitindo implementar de maneira optimizada as partes críticas do sistema. Para testar o software sem ser necessário colocá-lo de imediato no hardware, foi utilizado o TOSSIM[3], que se trata de um simulador para o TinyOS. No entanto, o referido simulador tem a limitação de não permitir simular mais do que um tipo de nó ao mesmo tempo. Como tal, o código correspondente aos três tipos de nós (nós sensores, *routers* e servidor) foi escrito no mesmo ficheiro de código. O comportamento diferenciado para cada um foi implementado através de cláusulas *if*, tirando partido da especificação que atribuía uma gama de identificadores a cada um dos tipos. A estrutura deste módulo genérico que implementa os vários tipos de nós é ilustrada na figura 1. Como resultado do processo de compilação para o hardware real temos um programa com os seguintes requisitos (excluíndo o código do nó servidor que lê e escreve em ficheiros):

- ROM: 15084 bytes
- RAM: 347 bytes

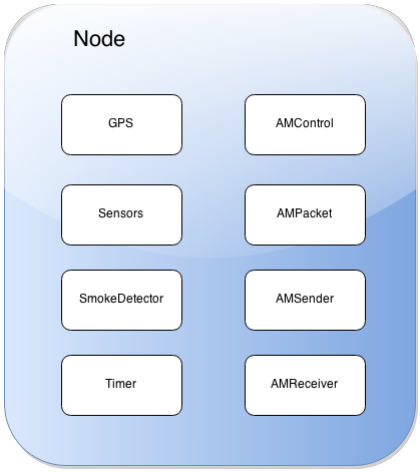

Figure 1: Módulo Genérico desenvolvido

2.2 Hardware

Toda a transmissão e recepção de dados é realizada por meio de Transmissores e Receptores RF que estão presente em todos os 3 tipos de módulo apresentados. O hardware para implementar a solução encontra-se resumido na tabela 1.

Componente	**Modulo de hardware**	**Características**
Nó wireless	MEMSIC XM2110CB	CPU: Atmel ATMega128 8 Mhz 128 KB Flash RAM: 8KB Frequência rádio: 2.4 Ghz
Sensores	MEMSIC MTS420	Sensores: - Temperatura - Humidade - Pressão atmosférica - Luz - GPS
Detector de fumo	Sensor de gases (ex: CO)	

Table 1: Tabela resumo com os componentes de hardware necessários

O nó sensor seria resultado de combinar todos os componentes de *hard-*

ware (Nó wireless, sensores e detector de fumo). Já o nó *router* apenas necessitaria de utilizar o Nó wireless pois apenas é necessário um módulo de radio-frequência para receber e enviar mensagens na rede de sensores.

2.3 Rede

Como referido anteriormente, esta rede de sensores é constituída por três tipos de nós: Nós sensores, que obtém dados acerca do ambiente envolvente; nós routers, que são responsáveis por encaminhar as mensagens até que estas cheguem a um outro nó; o servidor, que, centraliza os dados obtidos a partir dos sensores. Os nós sensores enviam os dados recolhidos, através de um módulo de comunicação rádio, aos nós routers. Estes nós, vão fazendo *broadcast* destas mensagens até as mesmas chegarem ao destino, isto é, ao nó servidor.

Tal como mostrado na figura 2 uma mensagem que circula na rede é constituída pelos seguintes campos:

type : Constante que indica o tipo de mensagem, como por exemplo, medidas de temperatura e humidade, detecção de fumo, etc;

nodeId : Identificador do nó sensor que originou a mensagem (quando aplicável)

timestamp : Instante de tempo em que a mensagem foi criada (usada em mensagens de medidas de temperatura e humidade, coordenadas GPS e alerta de fumo)

rank : Valor que indica a distância ao nó servidor (explicado em maior detalhe no parágrafo seguinte)

value1 e value2 : Estes campos dependem do tipo de mensagem. Por exemplo, se a mensagem tiver medidas de temperatura e humidade o *value1* e *value2* correspondem a temperatura e humidade respectivamente.

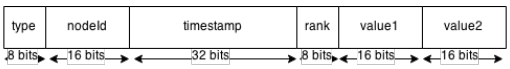

Figure 2: Estrutura de uma mensagem que circula na rede de sensores

A mensagem tem um total de 96 bits, o que corresponde a 12 bytes.

Inicialmente, o nó sensor envia uma mensagem (JOIN), em modo *broadcast* para se juntar à rede. Os nós *router* à volta, perto desse nó sensor respondem de volta com uma mensagem do tipo (JOIN ACK). De seguida, o nó sensor irá enviar uma mensagem de volta (JOIN ACCEPT) ao primeiro nó *router* do qual recebeu um JOIN ACK e esse nó *router* decrementa uma variável que indica o número de nós sensores já ligados a ele. Quando um nó *router* recebe uma mensagem JOIN, só responde com JOIN ACK apenas se esta variável não estiver ainda com valor zero. Esta variável é inicializada com valor 100, de acordo com a especificação, que corresponde ao número máximo de nós sensores que podem estar "ligados" a um nó *router*. Na figura 3 é ilustrado um exemplo de execução deste protocolo inicial em que o nó sensor 100 tenta juntar-se à rede, os nós *router* 1 e 2 respondem de volta e o nó sensor 100 responde com JOIN ACCEPT ao nó *router* 1 por ter recebido primeiro a mensagem JOIN ACK deste. De seguida, o nó sensor 100, cada vez que obtiver dados a partir dos seus sensores, irá enviar mensagens apenas para o nó *router* 1.

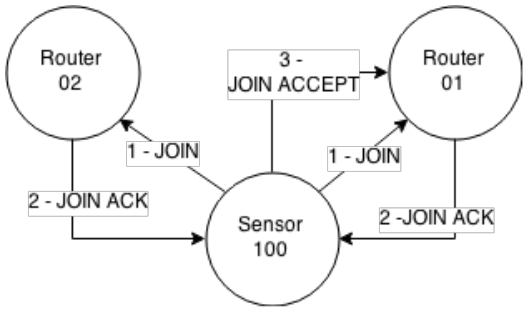

Figure 3: Protocolo inicial executado pelo nó sensor

Como já foi referido, os nós *router* fazem chegar a mensagem ao servidor fazendo *broadcast*. No entanto, foi necessário implementar um protocolo para evitar que existam mensagens a circular indefinidamente. Ou seja, um nó *router* deve "saber" quando não deve retransmitir uma mensagem recebida. Este problema é resolvido atribuindo um valor de *rank* ao nó *router* quando o mesmo ligado. Quando o nó *router* é ligado, envia uma mensagem do tipo GET RANK a todos os outros nós *router* à volta. Todos os nós *router* que tiverem um valor de *rank* definido irão enviar de volta uma mensagem do tipo RANK com o valor do seu *rank*. O valor de *rank* do servidor é zero. O

nó servidor também responde a mensagens do tipo GET RANK. Ou seja, o valor de *rank* corresponde à distância a que o nó se encontra do servidor. A figura 4 ilustra um exemplo de atribuição de valor de *rank* a um nó *router* (Router 02). O nó *router* fica sempre com o valor de *rank* máximo de entre os valores recebidos, incrementado em uma unidade. Consequentemente, este valor refere-se à distância, pelo caminho mais longo, ao servidor.

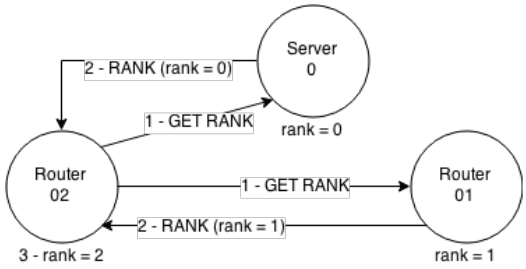

Figure 4: Protocolo para atribuição de valor de *rank* a um nó router

Desta forma, um nó *router*, quando recebe uma mensagem apenas a reencaminha se o valor de *rank* for maior que o valor do nó que está a processar a mensagem. Antes de reencaminhar a mensagem, o nó *router* substituí o valor de *rank* na mensagem pelo seu próprio valor.

3 Simulador

Para testar a solução implementada foi escrito um pequeno programa em *python*[1] que utiliza o módulo de TOSSIM gerado no processo de compilação para simulação. Este simulador foi implementado como um programa que espera input do utilizador e executa um comando com base nesse input. O simulador fornece as seguintes funcionalidades:

- Carregar uma topologia de rede a partir de um ficheiro;
- Carregar um modelo de ruído a partir de um ficheiro;
- Iniciar o *boot* de todos os nós da rede;
- Correr um dado número de eventos;
- Imprimir no ecrã a topologia, o estado de cada nó (ligado/desligado), o *log* do servidor, o nó *router* associado a cada nó sensor, e o valor de *rank* para cada nó;
- Ligar/Desligar um dado nó;
- Mostrar o valor de uma variável de um dado nó;
- Ligar/Desligar um dado canal de *debug*;
- Carregar comandos de simulação a partir de um ficheiro;
- Mostrar todos os comandos possíveis.

Para simplificar a simulação foram consideradas as seguintes condições:

- Ruído do ambiente: foi adotado um ruído constante.
- Topologia de rede: por se tratar de uma rede sem fios, a transmissão é realizada por *broadcast*. Entretanto, para simulação, foram consideradas apenas duas situações possíveis para dados dois módulos: ou estão ou não estão um no raio de alcance do outro.

[1] http://www.python.org/

- Atenuação: nível de atenuação do sinal na comunicação entre cada um dos pares de módulos que se comunicam. Este parâmetro foi assumido constante de forma a satisfazer a condição binária de comunicação citada anteriormente.

No entanto, com o simulador implementado é possível fornecer um modelo de ruído real. Devido ao facto de poder carregar ficheiros é possível simular as mais diversas condições e testar características do projecto, como por exemplo, a tolerância a faltas.

4 Deployment

Para proceder ao *deployment* dos vários nós, não é necessário seguir uma ordem específica. No entanto, é recomendado que se proceda da seguinte forma:

1. Ligar o nó servidor no local pretendido;

2. Colocar e ligar os vários nós *router* espalhados por vários pontos mas com uma distância relativamente curta uns dos outros. Apesar de serem toleradas alguns tipos de faltas é recomendado que estejam pelo menos dois nós deste tipo a uma curta distância do servidor e dois também colocados a uma curta distância de cada nó sensor. Desta forma, a rede de sensores torna-se mais robusta pois se algum dos nós *router* falhar existe sempre um alternativo e será possível chegar ao servidor através de vários caminhos;

3. Colocar e ligar os nós sensores junto dos nós *router*. Aqui também é recomendado que os nós sejam colocados de forma a existir redundância. Isto é, no mínimo dois nós sensores a monitorizar uma área muito pequena. Assim, se for detectado fumo num dado ponto, mais do que um nó vai enviar essa informação para a rede e, caso uma das mensagens se perca, outras irão conseguir chegar ao servidor.

Apesar do procedimento acima descrito, é possível colocar mais um nó *router* ou sensor em qualquer momento. Por exemplo, é possível ligar um nó *router* e só depois o servidor pois o nó *router* vai tentando continuamente obter um valor de *rank* até que algum outro nó lhe envie uma resposta de volta.

5 Conclusão

Neste projecto foi implementada uma rede de sensores *wireless* capaz de detectar fogo numa floresta e enviar essa informação a um servidor central. A rede é constituída por três tipos de nós: Nós sensores, que detectam fumo e enviam essa informação para a rede; nós *routers* que apenas difundem mensagens na rede, mensagens essas com dados vindos dos nós sensores; e o nó servidor, que recebe as mensagens e as escreve num ficheiro.

Devido ao facto de todos os nós *routers* enviarem as mensagens em *broadcast* foi implementado um algoritmo de *routing* baseado em valores de *rank*, explicado em maior detalhe na secção 2.3. Porém, o maior desafio foi tornar o sistema robusto de forma a conseguir tolerar faltas de alguns nós. Isto é, o sistema deve continuar a funcionar mesmo que alguns nós falhem. Nesta implementação são suportadas faltas em nós *routers*, isto é, quando um nó sensor envia uma mensagem, se o seu nó *router* não responder durante um certo limite de tempo, o nó sensor volta a executar o protocolo para seleccionar outro nó *router*. Algo semelhante acontece quando um nó *router* tenta encaminhar uma mensagem e passado algum tempo, nenhum outro nó manda a mensagem de volta. Neste caso, é detectado que este nó *router* não consegue fazer chegar uma mensagem ao servidor. De seguida, é mandada, para trás, uma mensagem de erro que irá ser recebida pelo nó sensor que tentou enviar algo ao servidor. Este nó sensor assume que o nó *router* ao qual está ligado não está a conseguir fazer com que as suas mensagens cheguem ao servidor. À semelhança do caso anterior, o nó sensor volta a executar o protocolo para "escolher" um novo nó *router*, mas desta vez, tendo o cuidado de forçar a que seja um diferente do anterior. Um sistema com uma maior tolerância a faltas implicaria uma maior utilização de recursos. Por se tratar de uma rede de sensores para ser instalada numa grande área, a utilização de recursos deve ser a mínima possível para evitar um elevado custo final.

Para testar a rede foi implementado um simulador que permite testar qualquer topologia, qualquer modelo de ruído e ainda as mais variadas situações, como por exemplo, desligar nós para testar a tolerância a faltas.

Por fim, os nós podem ser instalados e ligados por qualquer ordem. No entanto, é recomendável que se instale o nó servidor, de seguida os nós *routers* e por fim os nós sensores. Se forem instalados primeiros os nós *routers*, estes irão continuamente enviar mensagens para os outros à volta para tentar obter o seu valor de *rank*. Devido à forma como o protocolo está implementado, nenhum nó vai obter este valor até o servidor ser ligado.

A solução proposta pode ser considerada uma boa alternativa à vigilância com recurso a apenas vigilantes humanos, para a prevenção de incêndios. No entanto, a intervenção humana será sempre necessária, pois apesar de serem suportadas falhas em alguns nós e ser recomendado que haja redundância de nós, podem ocorrer falhas catastróficas, como por exemplo, numa dada área, todos os nós falharem ou até mesmo o servidor falhar, que se assumiu sempre ligado por razões de simplificação.

References

[1] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, *et al.*, "Tinyos: An operating system for sensor networks," in *Ambient intelligence*, pp. 115–148, Springer, 2005.

[2] D. Gay, P. Levis, D. Culler, and E. Brewer, "nesc 1.3 language reference manual," *TinyOS documentation, July*, 2009.

[3] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 126–137, ACM, 2003.