Sensor Network for Animal Husbandry

Software for Embedded Systems – Instituto Superior Técnico

Grupo 9 - Tagus

Alexandre Filipe Campos
alexandre.t.campos@ist.utl.pt
n.º 66354

Resumo

Os desafios enfrentados pela agricultura moderna nunca foram tão grandes. Sempre houve a necessidade de os produtores de gado terem capacidade de observar os animais o mais frequentemente possível. Mas gerir uma grande quantidade de animais, enquanto ao mesmo tempo permiti-los ser tão soltos quanto possível, não é uma tarefa fácil. Recentemente, temos visto o aumento do uso de redes de sensores sem fio para ajudar no manejo dos animais.

Ao colocar um nó sensor num animal que pode controlar a sua localização, a sua interação com os outros, ou a sua saúde, entre outros parâmetros. E podemos ter uma contínua, durante todo o dia, observação dos animais e, portanto, podemos intervir o mais cedo possível, se alguma situação ocorra.

O objetivo deste projeto é criar esses nós sensores em uma rede.

Introdução

1. Descrição geral do sistema

O sistema é descrito pelas seguintes principais funcionalidades:

- Monitorização da localização GPS dos animais
- Monitorização do consumo de comida do animais
- Capacidade de difusão de mensagens
- Configuração dos níveis de consumo de comida
- Configuração dos níveis de comida disponível

Cada nó possui uma placa de sensor que permite a anexação de um módulo GPS. Este módulo irá providenciar leituras para a constante monitorização da posição dos animais. Adicionalmente todos os nós terão um RFID (Radio-Frequency IDentification') token para permitir definir a a proximidade aos vários pontos de alimentação presentes. Em cada nó é guardada a informação sobre o consumo máximo diário de comida para cada animal e a quantidade de comida que já consumiu no dia.

Os locais de alimentação irão suportar a alimentação dos animais ao despenderem uma dose controlado de comida para o animal, caso este ainda não tenha ultrapassado o seu limite diário. Os locais de alimentação serão munidos de um leitor RFID para detetar a presença do animal e identificar o animal aquando da aproximação deste. Adicionalmente terão um módulo Rádio-Frequência para a comunicação de quanta comida a dispensar ao animal.

Todas as medidas colecionadas pelos nós serão difundidas periodicamente através do módulo RF (Rádio Frequência). Para demonstrar toda a funcionalidade do sistema e do protocolo de encaminhamento foi definida uma topologia para a representação geográfica dos animais que permite que a qualquer momento esteja sempre um nó perto do servidor que controla a rede e efectua a coleção dos dados. Os nós mais distantes do servidor terão as suas mensagens encaminhadas pelos outros nós da rede, havendo sempre mais do que uma caminho possível para efeitos de redundância.

No servidor será possível monitorizar e configurar os diversos parâmetros através de comandos. Como efetuar alterações de consumos diários de comida a nós individuais ou todos os nós.

2. Arquitetura do Sistema

2.1 Arquitectura de Hardware

A rede *wireless* de sensores é constituída por um conjunto de nós – *MICAz motes*. Estes módulos são especialmente concebidos para criar redes de sensores *wireless* de baixo consumo energético (figura 1).

Figura 1: MICAz mote

O transmissor rádio, opera na largura de banda dos 2.4GHz e tem uma taxa de transmissão de dados de 250 kbps. Para mais informações, é possível consultar respectiva *datasheet*[3].

Para poder colecionar dados de localização GPS, é necessário ligar ao *MICAz*, uma placa de sensor e uma antena GPS compatível. Para esse efeito foi escolhida a MTS420 sensor board (Figura 2) que tem disponível diversos sensores entre eles GPS.

Figura 2: MTS420 - *Sensor board*

Uma possível implementação considerada para suportar os pontos de alimentação seria o use de um Arduíno (Figura 3). Este iria ser munido de um módulo RF 2.4Ghz para ser possível comunicar com os micaz motes e de um RFID Reader para detetar a proximidade do animal.

Para auxiliar no dispenso da comida com alguma precisão de quantidade podemos ligar ao arduíno um Servo Motor (Figure 4) que com base na quantidade de comida configurada no mote do animal poderá rodar e abrir uma espécia de fúnil para a queda de comida, um certo ângulo durante um certo tempo para servir a quantidade certa.

2.2 Arquitectura de Software

O sistema foi desenhado e desenvolvido para o sistema operativo *TinyOS*, integralmente implementado na linguagem *nesC*. O *TinyOS* é um sistema operativo baseado em eventos, composto por uma arquitetura de componentes. Fazendo uso do simulador TOSSIM para simular a totalidade da aplicação.

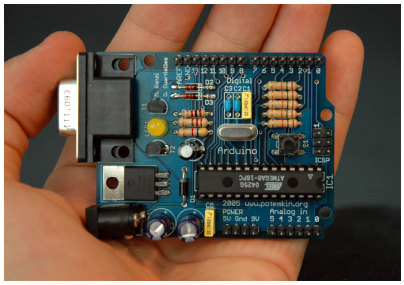

Figura 3: *Arduíno*

Figura 4: Servo motor

2.2.1 Componentes

A aplicação é suportada pelos seguintes componentes principais:

- Módulo: *AnimalC*
- Configuração: *AnimalAppC*

Toda a lógica está presente no módulo *AnimalC*, que gere todo o funcionamento do sensor desde comunicação com outros nós e *feeding spots* a interação com o sensor GPS. Este módulo faz uso de interfaces disponibilizadas pelo *tinyOS* como *ActiveMessageC* (responsável pelo controlo da comunicação rádio), *TimerMilliC* (que permite temporizar diversas acções), *DemoSensorC* (que oferece a interface de ligação a vários sensores). As ligações entres estes e o módulo são feitas pela configuração *AnimalAppC*.

Mais específicamente o módulo TimerMilliC é utilizado para temporizar diversos aspectos da aplicação, sendo estes,

a verificação de proximidade a um feeding spot. Esta proximidade é simulada através de uma variável booleana que está constantemente a ser verificada. Assim que é detectada a sua activação (esta acção simula a deteção do badge RFID pelo local de alimentação e posterior comunicação rádio com o mote), é despoletada a alimentação do animal. Outro aspecto da aplicação que é controlado é a temporização é a leitura do GPS e comunicação dos dados do nó (GPS e comida). O componente DemoSensorC representa um dispositivo sensor genérico que permite simular vários sensores, sendo usado nesta caso para simular a leitura GPS.

2.2.2 Lógica de Execução

Assim que um nó é iniciado a interface *Boot* é sinalizada sendo despoletado o event *booted()*, aqui é dada a inicialização a várias variáveis e também iniciado a componente rádio.

Assim que a componente rádio é iniciada com sucesso, são iniciados os três componentes periódicos (disponibilizados pelo componente *TimerMilliC*, que oferecem ao nó a autonomia para ler os sensores periodicamente, comunicar informação dos sensores e verificar proximidade a um ponto de alimentação.

Assim que o evento temporizado para a leitura de GPS dispara são chamadas os métodos da interface esperada mas a sua implementação é simulada com a leitura de um ficheiro. Assim que a leitura está completa, é guardada e não será enviada até que o evento de envio de informação dispare.

A mesma lógica é aplicada à alimentação do animal. Através do servidor é possível injectar um pacote na rede para definir um animal como próximo de um ponto de alimentação. Esta alteração irá despoletar uma série de eventos assim que o temporizador para alimentação disparar e verificar que a proximidade é verdade. Assim que isto acontece é simulada a comunicação rádio com o ponto de alimentação através da leitura e escrita de um ficheiro.

Através da interface rádio são definidas funções especificadas para a receção das mensagens (*ReceiveUpdateFoodDailyDosage.receive(..)* & emphReceiveUpdateFeedingSpot.receive(...)) de atualização da comida diária máxima e atualização da comida máxima disponível nos pontos de alimentação. A primeira será reencaminhada por todos os nós e com dois tipos de destino (todos ou um nó em especifico) e irá servir para atualizar os valores. A segunda será reencaminhada por todos os nós até que seja encontrado um que esteja próximo de um ponto de alimentação cuja mensagem seja destinada. Mais uma vez essa interação (nó próximo ¡−¿ ponto de alimentação) será simulada através do acesso a um ficheiro.

A lógica de receção e reencaminhamento de mensagens será explicada em mais detalhe na próxima secção.

3. Arquitectura da Rede

Os nós que compõem a rede *wireless* de sensores encontram-se distribuídos segundo uma topologia estabelecida através de um ficheiro que indica quais os nós que recebem uma dada mensagem transmitida por outro nó e o respectivo ganho, em dB. A figura 5 ilustra a topologia utilizada no contexto deste projecto.

A topologia da rede de sensores é estabelecida através de um ficheiro (topo.txt) presente no projecto. Este ficheiro representa as ligações entre nós e o respectivo ganho (em dB):

```
0  1 -53.0
1  0 -54.0
1  2 -54.0
2  1 -55.0
2  3 -55.0
3  2 -55.0
3  4 -55.0
4  3 -55.0
4  5 -54.0
5  4 -53.0
5  6 -56.0
6  5 -55.0
6  7 -53.0
7  6 -52.0
7  8 -54.0
8  7 -54.0
8  9 -54.0
9  8 -55.0
9  0 -54.0
0  9 -53.0
```

O resultado é a seguinte topologia:

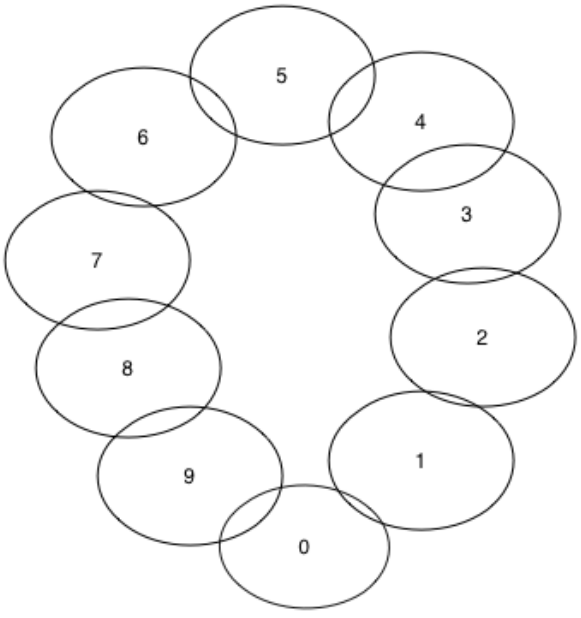

Figura 5: Topologia da rede de sensores

O protocolo definido para a comunicação entre os nós foi uma variação *broadcast flood*, em que cada nó retransmite todas as mensagens que recebe exceto alguns casos específicos: caso a mensagem seja direcionada a ele próprio, por exemplo, uma alteração da quantidade de comida máxima diária que o animal pode comer leva a que todos os cuja a mensagem não seja destinada irão fazer retransmissão em broadcast até que a mensagem chegue ao destino. Visto que todas as mensagens têm um identificador único cada nó apenas retransmite a mensagem uma vez, levando assim à conclusão que caso existam *N* nós na rede a mensagem irá ser retransmitida *N* vezes.

Embora este protocolo seja algo ineficiente em termos energéticos é extremamente eficaz em situações de perdas de mensagens causadas por colisões e/ou interferências de ruído, devido à elevada redundância. Outro ponto a favor é o facto de os animais possam estar em constante movimento levando assim a que a implementação de um algoritmo de encaminhamento mais sofisticado pode levar a resultados piores numa rede em constante alteração.

3.1 Tipos de Mensagens

Para a disseminação de informação GPS e comida dos nós foi criada a seguinte mensagem (AnimalInfo) que é disseminada periodicamente entre todos os nós com o intuito de chegar ao servidor. Cumprindo assim o objectivo ser possível aceder à informação mais recente possível sobre nós que se possam afastar da rede principal sendo impossível comunicar com eles. Desta forma também é evitado que pedidos feitos pelo servidor possam gerar uma tempestade de mensagens em resposta ao pedido.

```
typedef nx_struct AnimalInfo {
  nx_int32_t gps_la; //gps latitue
  nx_int32_t gps_lo; //gps longitude
  nx_uint16_t food_g; //food eaten
  nx_uint32_t msg_id; //unique
  nx_uint16_t mote_id; //the unique mote_id
} AnimalInfo;
```

As duas seguintes mensagens não são criadas por nós mas sim injectadas na rede pelo servidor ao nó mais próximo para disseminar. Podem ter intuito de chegar a um nó específico ou no caso de destino ser zero para chegar a todos os nós ou feeding spots.

```
typedef nx_struct UpdateFeedingSpot {
  nx_uint16_t food_g; //food in grams
  nx_uint32_t msg_id; //unique
  nx_uint8_t spot_id; //spot id
} UpdateFeedingSpot;

typedef nx_struct UpdateFoodDailyDosage{
  nx_uint32_t msg_id; //to denie rebroadcast
  nx_uint16_t mote_dest; //if 0 deliver to all
  nx_uint8_t new_food_max; //new max amount of daily food
} UpdateFoodDailyDosage;
```

4. Manual de Utilização do Simulador

Para manipular o simulador é necessário um *script* em *Python*, que recorre ao TOSSIM[1] para simular a presença de vários nós sensores e outras funcionalidades. Para executar a aplicação com sucesso basta seguir os seguintes passos:

1. Iniciar um terminal e colocar-se na diretoria do projecto

2. Compilar o programa através da Makefile. Sendo o primeiro argumento o tipo de mote e o terceiro a indicar que é uma simulação.

   ```
   make micaz sim
   ```

   Este comando deverá gerar diversos ficheiros.

3. A simulação faz uso de diversos ficheiros auxiliares que necessitam de estar presentes:

   - Topologia da rede: *topo.txt*
   - Modelo de ruído: *noise.txt*
   - Simulação GPS: *gps1 gps2 ... gpsN*
   - Simulação Feeding Spot *fs1 fs2 ... fsN*

Assim que o projecto estiver compilado com sucesso e os ficheiros todos presentes, pode-se correr o projecto com o seguinte comando:

```
python server.py
```

Assim que o simulação é iniciada será questionado sobre a quantidade de nós que quer na rede. De seguida os nós serão iniciados. Assim que a iniciação terminar será presenciado com um terminal interativo que pode receber os seguintes comandos:

```
- h
*** for help
- get <mote_id> (FOR DEBUG)
*** get state info of mote
- exit
*** to exit
- read <mote_id>
*** get last known GPS and food info
- updateFood <mote_id> <amount>
*** to update daily food. mote_id 0 for all
- getSpotFood <spot_id>
*** to get amount of food left in feeding spot.
- updateSpotFood <spot_id> <amount>
*** to update the spots food. 0 for all.
- prox <mote_id>
*** turn on proximity simulation to feeding spot.
```

[1] TinyOS SIMulator

5. Problemas/Soluções na Implementação

Um dos principais problemas ao conceber a solução foi lidar com a perda de mensagens e tentar construir algo superior ao modelo de controlo de concorrência nas comunicações radio sugerido pelo tutoriais do *tinyOS*. Este modelo sugere o uso de uma variável booleana a proteger a ocupação do rádio para transmissão levando a que possíveis envios (reacções a mensagens recebidas) fossem descartados. Foi então tentado desenvolver uma solução com locks distribuídos (usando mutex) que se revelou um falhanço devido a constantes situações de erro devido a *deadlock* distribuído. Devido ao consumo de tempo da não resolução destes problemas a solução foi abandonada e foi decido fazer uso do modelo originalmente sugerido.

Foi contemplada também a possibilidade de implementação de algorítmo de encaminhamento mais sofisticado, mas devido à natureza da rede e comportamentos imprevisíveis (possível movimento constante dos nós) poderia revelar-se mais custoso e ineficiente em várias situações.

6. Conclusões

Referências

[1] David Gay, Philip Levis, David Culler, Eric Brewer, *nesC 1.3 Language Reference Manual*, July 2009 http://nesl.ee.ucla.edu/fw/torres/home/projects/tossim_gumstix/root/nesc-1.3.1/doc/ref.pdf

[2] MDA300 Data Acquisition Board Datasheet, Memsic http://www.memsic.com/userfiles/files/Datasheets/WSN/6020-0052-04_a_mda300-t.pdf

[3] MICAz Datasheet, ZigBee™ Alliance, Crossbow, http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[4] TinyOS Tutorials - TinyOS Documentation Wiki http://docs.tinyos.net/tinywiki/index.php/TinyOS_Tutorials