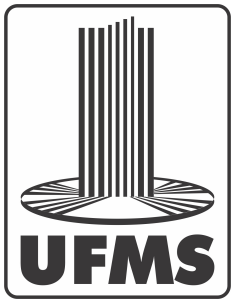
João Felipe Resende Nacer

# Aspectos de Roteamento em Redes de Sensores sem Fio para Monitoramento Bovino




Campo Grande - MS

Março de 2017

João Felipe Resende Nacer

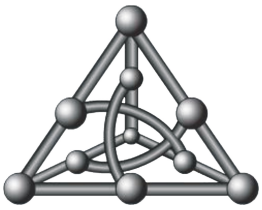
# Aspectos de Roteamento em Redes de Sensores sem Fio para Monitoramento Bovino

Dissertação apresentada como requisito para obtenção do título de Mestre em Computação Aplicada.
Prof. Orientador: Dr. Irineu Sotoma.
Coorientador: Dr. Pedro Paulo Pires.

Universidade Federal de Mato Grosso do Sul – UFMS

Faculdade de Computação

Mestrado em Computação Aplicada



Campo Grande - MS

Março de 2017

# Agradecimentos

Agradeço de forma geral à todos que contribuíram com a construção deste projeto. À minha família, pelo apoio e incentivo que me deram em todos os momentos da criação deste projeto.

Ao professor Dr. Irineu Sotoma, que me orientou neste trabalho. Pela confiança, dedicação e apoio em todas as horas, por ter se tornado um grande colega de trabalho e que contribuiu para o meu crescimento como profissional e estudante. Agradeço pela confiança e apoio.

Ao meu coorientador Dr. Pedro Paulo Pires, pelas sugestões de melhorias no trabalho.

Aos professores das disciplinas do mestrado, que contribuíram com minha formação, profissional e pessoal. Aos professores da disciplina de estudo dirigido que contribuíram com sugestões de melhorias no trabalho durante as apresentações. Aos professores da banca de qualificação e reuniões sobre o projeto.

Aos colaboradores da Embrapa, Camilo Carromeu e Quintino Izidio, que me orientaram e abriram as portas da Embrapa para e execução dos experimentos.

Aos colegas de mestrado, que me ajudaram com sugestões e informações sobre os projetos desenvolvidos anteriormente e que este trabalho veio complementar. Especialmente ao Luiz Lomba, que ajudou compartilhando o trabalho e os dados coletados, e ao Egon Dadalt pela ajuda com a criação do módulo de mobilidade no Castalia.

Agradecimentos à FUNDECT pelo apoio financeiro para a compra de equipamentos via projeto com termo de outorga Nº 102/2014.

# Resumo

Este trabalho apresenta o Projeto e Desenvolvimento de um sistema integrado de hardware/software para aquisição e transmissão de posições de GPS para monitoramento de bovinos. A proposta apresenta alguns protocolos da literatura e simulações de Redes de Sensores Sem Fio (RSSF), bem como as tecnologias utilizadas em RSSF e como podem ser configuradas. O foco do trabalho é no roteamento para a RSSF e o objetivo da pesquisa foi complementar um trabalho desenvolvido anteriormente, que consistia em coletar as informações do boi no pasto, fazendo nesta proposta o envio das informações coletadas por meio de antenas sem fio até uma estação base, onde os dados podem ser observados pelo pecuarista ou pesquisador. Este projeto consiste em mais uma parceria entre Embrapa - Gado de Corte e a FACOM/UFMS, na área de Pecuária de Precisão.

**Palavras-chaves**: pecuária de precisão; redes de sensores sem fio; protocolos de roteamento; zigbee, xbee.

# Abstract

This paper presents the project and development of a hardware and software integrated system to aquire and transmit GPS positions data for cattle monitoring. The purpose presents some literature protocols and Wireless Sensor Networks (WSN) simulators, as well the technologies used in WSN and how it can be configured. The focus of the work is on the routing for the WSN and the objective of the research was to complement a previously developed work that consisted of collecting the information of the cattle in the pasture, making in this proposal the sending of the collected information through wireless antennas to a base station, where the data can be observed by the farmer or researcher. This project consists of another partnership between Embrapa Gado de Corte and FACOM-UFMS, on Precision Livestock area.

**Key-words**: precision livestock farming; wireless sensor networks; routing protocols; zigbee, xbee.

# Lista de ilustrações


Figura 1 – Hardware básico de um nó sensor (LOUREIRO et al., 2003). . . . . 16
Figura 2 – Problema de implosão (TILAK; HEINZELMAN., 2002). . . . . 20
Figura 3 – Problema da sobreposição (TILAK; HEINZELMAN., 2002). . . . . 20
Figura 4 – Os módulos e as conexões do simulador Castalia. Adaptado de (BOULIS et al., 2011). . . . . 27
Figura 5 – Camadas de rede (SANTOS, 2007). . . . . 30
Figura 6 – Topologias ZigBee pair, star, mesh e cluster tree. (FALUDI, 2010). . . 33
Figura 7 – Módulo XBee e XBee-PRO. . . . . 34
Figura 8 – Pinagem XBee . . . . . 35
Figura 9 – Tipos de antenas dos módulos XBee . . . . . 35
Figura 10 – Estrutura de um pacote API (DIGI, 2015). . . . . 36
Figura 11 – Descrição dos pacotes API ZigBee (DIGI, 2015). . . . . 37
Figura 12 – Adaptador USB para XBee. . . . . 38
Figura 13 – X-CTU: Aba de configuração do modem XBee. . . . . 38
Figura 14 – Modos de operação do XBee (DIGI, 2015). . . . . 39
Figura 15 – Endereço MAC do XBee . . . . . 40
Figura 16 – Pinagem XBee . . . . . 46
Figura 17 – Arduino FIO . . . . . 53
Figura 18 – Painel solar flexível . . . . . 54
Figura 19 – GPS Vênus Sparkfun + Antena SMA . . . . . 55
Figura 20 – OpenLog Sparkfun . . . . . 55
Figura 21 – XBee PRO Series 2 . . . . . 56
Figura 22 – Nó sorvedouro: Raspberry PI e módulo XBee . . . . . 58
Figura 23 – Estrutura da RSSF . . . . . 59
Figura 24 – Sistema Web - Mapa com posição mais atual dos bois no pasto. . . . . 60
Figura 25 – Sistema Web - Exmeplo de lista de mensagens recebidas nos últimos 30 minutos . . . . . 60
Figura 26 – Sistema Web - Tela de caminho percorrido pelos bois. . . . . 60
Figura 27 – Equipamentos do colar com moeda para escala . . . . . 62
Figura 28 – Esquemático do colar . . . . . 62
Figura 29 – Equipamentos do nó antes de prender ao colar . . . . . 63
Figura 30 – Processo de envio de pacotes na RSSF . . . . . 66
Figura 31 – Consumo de equipamentos e estimativa de vida útil dos colares no modo recebendo . . . . . 71
Figura 32 – Área disponibilizada pela Embrapa para os experimentos em campo. . 72
Figura 33 – Colocando o colar no primeiro boi . . . . . 73

Figura 34 – Colares - Primeiro experimento . . . . . 74
Figura 35 – Teste dos colares fechados na Embrapa Gado de Corte . . . . . 74
Figura 36 – Área do experimento . . . . . 75
Figura 37 – Área do Experimento 2 no Mangueiro Digital . . . . . 77
Figura 38 – Caminho percorrido pelo animal com colar 3 . . . . . 84
Figura 39 – Número de Pacotes Criados Por cada Nó . . . . . 91
Figura 40 – Pacotes entregues no sorvedouro (por nó). . . . . 92
Figura 41 – Taxa de entrega de Pacotes . . . . . 93
Figura 42 – Latência . . . . . 94
Figura 43 – Energia consumida pelo colar em cada algoritmo. . . . . 95
Figura 44 – Energia consumida apenas pela transmissão de pacotes na rede . . . . 96
Figura 45 – Estimativa de Tempo de Vida (em horas) . . . . . 97
Figura 46 – Comparação de Estimativa de Tempo de Vida (em horas) . . . . . 97
Figura 47 – Falhas e recebimentos de pacotes na camada MAC . . . . . 98
Figura 48 – Pacotes enviados à camada MAC . . . . . 99
Figura 49 – Comparação entre tempo de vida dos colares . . . . . 101
Figura 50 – Rede com infraestrutura adicional como roteadores . . . . . 102

# Lista de abreviaturas e siglas

| | |
|---|---|
| API | *Application Programming Interface* |
| AT | *Transparent Mode* |
| ECTC | *Electronic Collars to Track Cattle* |
| EMBRAPA | Empresa Brasileira de Pesquisa Agropecuária |
| FACOM | Faculdade de Computação da Universidade Federal do Mato Grosso do Sul |
| FFD | *Full Function Device* |
| GERCOM | Grupo de Estudos em Redes de Computadores e Comunicação Multimídia |
| GPS | *Global Positioning System* - Sistema de Posição Global |
| IEEE | *Institute of Electrical and Electronic Engineers* |
| IO | *Input/Output* |
| JSON | *JavaScript Object Notation* |
| MANET | *Mobile Ad hoc Network* - Rede Móvel Ad hoc |
| NMEA | *National Marine Electronics Association* |
| OTAG | *Operational Management and Geo-decisional Prototype to Track and Trace Agricultural Production* |
| RF | Rádio Frequência |
| RFD | *Reduced Function Device* |
| RSSF | Rede de Sensores Sem Fio |
| TTFF | *Time to First Fix* |
| UFMS | Universidade Federal de Mato Grosso do Sul |
| UTM | *Universal Transverse Mercator* |
| XML | *eXtensible Markup Language* |

# Sumário


**1 INTRODUÇÃO . . . . . 10**
**1.1 Motivação . . . . . 10**
**1.2 Objetivos . . . . . 12**
**1.3 Organização do texto . . . . . 13**

**2 FUNDAMENTAÇÃO TEÓRICA . . . . . 14**
**2.1 Redes de Sensores Sem Fio . . . . . 14**
2.1.1 Características de uma RSSF . . . . . 14
2.1.2 Componentes de uma RSSF . . . . . 15
2.1.3 Consumo de energia em RSSFs . . . . . 16
**2.2 Aplicações em RSSFs . . . . . 17**
**2.3 Roteamento em RSSFs . . . . . 17**
2.3.1 Comparação entre protocolos estudados . . . . . 26
**2.4 Simulações em RSSFs . . . . . 27**
**2.5 Padrão ZigBee/IEEE 802.15.4 . . . . . 29**
2.5.1 Topologia do ZigBee . . . . . 32
**2.6 Módulos XBee . . . . . 34**
2.6.1 Modos de Comando . . . . . 36
2.6.2 Software de configuração dos Módulos . . . . . 37
2.6.3 Modos de operação do XBee . . . . . 38
2.6.4 Camadas de rede do XBee . . . . . 39
2.6.5 Endereçamento XBee Zigbee . . . . . 40
2.6.6 Roteamento no XBee . . . . . 41
2.6.6.1 AODV (*Ad-hoc On Demand Distance Vector Routing*) . . . . . 42
2.6.6.2 Roteamento Many-to-one . . . . . 43
2.6.6.3 Source routing . . . . . 44
2.6.7 Fragmentação . . . . . 44
2.6.8 Confirmações de recebimento ACKs . . . . . 45
2.6.9 Configurações de *firmware* . . . . . 45
2.6.10 Modos de utilização de *sleep* . . . . . 45
2.6.11 Limitações do módulo XBee Series 2 . . . . . 46

**3 TRABALHOS RELACIONADOS . . . . . 48**
**3.1 RSSF para monitoramento de animais . . . . . 48**
**3.2 RSSF para monitoramento Animal e Atuadores . . . . . 49**
**3.3 Trabalhos com GPS . . . . . 50**

**3.4 Comportamento Animal . . . 50**
**3.5 Detecção de doenças em animais com uso de sensores . . . 50**
**3.6 Considerações Finais . . . 51**

**4 IMPLEMENTAÇÃO DA RSSF . . . 52**
**4.1 Metodologia . . . 52**
**4.2 Dispositivos Utilizados . . . 53**
**4.3 Nós sensores e nó sorvedouro . . . 57**
4.3.1 Nó sorvedouro . . . 57
4.3.2 Nós sensores . . . 61
**4.4 Configurações da RSSF . . . 66**
4.4.1 Hibernação de recursos - Alternativas para viabilizar o modo *sleep* . . . 67

**5 EXPERIMENTOS . . . 69**
**5.1 Consumo de energia dos nós sensores . . . 69**
**5.2 Área disponível para os experimentos em campo . . . 72**
**5.3 Experimentos . . . 73**
5.3.1 Primeiro experimento . . . 73
5.3.2 Segundo experimento . . . 76

**6 SIMULAÇÕES . . . 78**
**6.1 Configurações gerais . . . 78**
**6.2 Mobilidade e pré-processamento . . . 79**
**6.3 Módulo Resource - Consumo de energia . . . 85**
**6.4 Módulo de aplicação . . . 86**
**6.5 Módulo de acesso ao meio - Camada MAC . . . 87**
**6.6 Módulo de rádio . . . 87**
**6.7 Módulos de Roteamento . . . 89**
**6.8 Resultado das simulações . . . 90**
6.8.1 Número de Pacotes Entregues no Sorvedouro - Por nó . . . 91
6.8.2 Taxa de Entrega de Pacotes . . . 92
6.8.3 Latência em nível de aplicação (em ms) . . . 93
6.8.4 Energia consumida . . . 94
6.8.5 Estimativa de vida da rede . . . 96
6.8.6 Estimativa de vida da rede com e sem GPS . . . 97
6.8.7 Considerações sobre as simulações realizadas . . . 98
6.8.8 Simulação de consumo no modo *sleep* . . . 100

**7 CONSIDERAÇÕES FINAIS . . . 104**
**7.1 Dificuldades encontradas . . . 105**
**7.2 Trabalhos futuros . . . 106**

Referências . . . . . 107

APÊNDICES 112

APÊNDICE A – ARQUIVO DE CONFIGURAÇÃO DO SORVEDOURO . . . . . 113

APÊNDICE B – ARQUIVO DE CONFIGURAÇÃO DO CENÁRIO SIMULADO NO CASTALIA . . . . . 114

ANEXOS 116

ANEXO A – BIBLIOTECAS DO ARDUINO UTILIZADAS PARA A MONTAGEM DO NÓ SENSOR . . . . . 117

# 1 Introdução

## 1.1 Motivação

A pecuária é uma atividade econômica de grande importância no cenário nacional. Segundo dados do Perfil da Pecuária no Brasil - Relatório Anual de 2016, elaborado pela ABIEC (Associação Brasileira das Indústrias Exportadoras de Carne), o Produto Interno Bruto (PIB) do Brasil chegou a R$5,9 trilhões em 2015, registrando queda de 3,85% sobre o resultado anterior. O PIB do agronegócio alcançou R$1,26 trilhão, representando 21% do PIB total brasileiro. Já o PIB da pecuária chegou a R$400,7 bilhões, 30% do agronegócio brasileiro.

A pecuária de precisão, no contexto dos ecossistemas pastoris, é a forma moderna de gerenciar os sistemas de produção animal a pasto. Consiste na medição de diferentes parâmetros dos animais, a modelagem desses dados para selecionar a informação que se quer, e o uso desses modelos em tempo real visando o monitoramento e o controle de animais e rebanhos (BERCKMANS, 2004).

Neste contexto a pecuária de precisão pode utilizar inovação tecnológica para o monitoramento dos animais e pastagens, uma das possibilidades de monitoramento conhecidas é por meio de sensores.

A Embrapa Gado de Corte e a FACOM - Faculdade de Computação da Universidade Federal do Mato Grosso do Sul possuem um histórico de parcerias desenvolvendo aplicações para a Pecuária de Precisão.

Recentemente, (JESUS, 2014) desenvolveu um projeto fruto da parceria entre Embrapa e FACOM, que é um sistema de monitoramento bovino baseado nas posições dos animais no pasto. Neste projeto, as posições dos bois são coletadas em um colar desenvolvido para cada animal composto por um sensor de GPS (*Global Positioning System* - Sistema de Posição Global) e um cartão de memória, para o armazenamento das informações. Ao final de cada experimento as informações são coletadas manualmente para análises do comportamento dos animais. O sistema pode detectar 4 comportamentos dos animais: comendo, andando, em pé e deitado.

Complementando o trabalho de (JESUS, 2014), temos o trabalho de (LOMBA, 2015) que adaptou novos sensores de luminosidade e acelerômetro no colar para identificar o comportamento bovino por meio da aceleração do animal e monitoramento da luminosidade do ambiente.

Outro projeto foi desenvolvido por (OLIVEIRA, 2013). Um sistema que utiliza um

conjunto de dados obtidos por sensores GPS para registrar o deslocamento do animal no pasto. Este projeto utiliza identificação de trajetória semântica para encontrar padrões de comportamento do animal, tais como, distância percorrida e velocidade média das trajetórias.

Segundo Helwatkar, Riordan e Walsh (2014) diversos tipos de sensores podem ser utilizados para o monitoramento do bem-estar animal, tais como, sensores de temperatura, acelerômetro, microfone, pressão, GPS e outros para descobrir aspectos do animal e possíveis mudanças no comportamento.

O trabalho proposto implementa uma Rede de Sensores Sem Fio (RSSF) utilizando o colar desenvolvido por (JESUS, 2014) visando melhorar o tempo de entrega entre a coleta e a visualização dos dados pelo pecuarista, tornando esse processo automático, ao invés de manual. Dessa forma, o pecuarista poderá saber informações sobre os bois durante o uso dos colares nos bois, não sendo necessário aguardar a retirada dos cartões de memória. Além disso, a RSSF deve ser definida para permitir a adição futura de novos tipos de sensores, como os sensores apresentados no trabalho de (LOMBA, 2015).

Uma rede de sensores sem fio (RSSF) é um conjunto de nós sensores que são dispersos dentro do fenômeno a ser observado ou próximo dele, e que trabalham cooperativamente via computação local simples e transmissão entre eles apenas dos dados parcialmente processados ou requeridos (AKYILDIZ et al., 2002b).

Os dados coletados pelos sensores poderão ser observados pelo pecuarista ou pesquisador, que poderão fazer análises de comportamentos do animal, observar padrões ou detectar sintomas de doenças durante o uso dos colares. Por meio dessas análises, pode-se melhorar a qualidade de vida do animal ou do processo de manejo.

O colar desenvolvido veio na tentativa de substituir um colar proprietário existente chamado ECTC (*Electronic Collars to Track Cattle*). Este colar é equipado de um sensor GPS e foi desenvolvido no âmbito do projeto OTAG (*Operational Management and Geodecisional Prototype to Track and Trace Agricultural Production*) para monitoramento bovino e tem como desvantagens o alto custo e o padrão fechado. Este colar comunica-se com uma estação base, que, por possuir protocolo de comunicação fechado, dificulta a integração com outras RSSF.

Segundo Akyildiz et al. (2002b), em uma RSSF, as informações coletadas pelos nós sensores são usualmente encaminhadas a um nó especial chamado nó sorvedouro. A RSSF proposta trabalha de maneira que todos os nós sensores enviam as informações a um único sorvedouro, que poderá também em trabalhos futuros, receber informações de outros sensores de animais ou do ambiente.

Para que os dados monitorados pelos nós sensores sejam enviados até o nó sorvedouro, é necessário que sejam estabelecidas rotas entre a origem e o destino. Os protocolos

que têm como objetivo determinar uma rota entre a origem e o destino são chamados protocolos de roteamento. Este trabalho tem foco no roteamento a ser escolhido para a RSSF proposta, levando em conta as possibilidades de implementação nos equipamentos escolhidos.

Pensando no contexto de monitoramento animal, para determinar o melhor algoritmo de roteamento para o trabalho, foi feito um estudo de alguns algoritmos da literatura. A proposta do trabalho era escolher o algoritmo a ser utilizado baseado em comparações realizadas com um simulador de RSSF.

Existe uma grande quantidade de simuladores que vêm sendo utilizados para redes de sensores sem fio (STEHLíK, 2011), tais como: NS-2 (NS-2, 2014), Castalia (CASTALIA, 2014), MiXiM (MIXIM, 2014), TOSSIM (LEVIS et al., 2003), Cooja (COOJA, 2014) e OMNeT++ (OMNET++, 2014). Entre os simuladores apresentados, optou-se por realizar as simulações utilizando o Castalia, que é uma extensão do simulador OMNeT++, devido ao suporte de modelos mais realísticos de rádio e de canal de comunicação sem fio.

## 1.2 Objetivos

O objetivo geral deste projeto foi a criação de uma RSSF para automatizar o processo de aquisição de dados em bovinos, pela evolução dos projetos de monitoramento bovino, desenvolvidos por (OLIVEIRA, 2013), (JESUS, 2014) e (LOMBA, 2015) para a Embrapa Gado de Corte. A criação desta RSSF permitiu a transmissão sem fio de dados de GPS dos colares até um nó sorvedouro.

Os objetivos específicos deste trabalho são:

- Desenvolvimento de um colar que possui comunicação sem fio para transmissão de posições GPS dos bois baseado no hardware XBee Series 2 Pro, que pode ser incorporado ao hardware desenvolvido por Jesus (2014) e que permita a incorporação de novos sensores em trabalhos futuros;
- Desenvolvimento de um nó sorvedouro que receba os dados da RSSF e apresente as posições em uma interface para o produtor ou pesquisador;
- Definição do protocolo de roteamento mais eficiente para o contexto de monitoramento bovino baseado em simulações dos protocolos aceitos pelo *hardware* XBee Series 2 Pro; e
- Implantação e testes em campo da RSSF.

## 1.3 Organização do texto

Os capítulos seguintes estão organizados da seguinte forma: no Capítulo 2, apresenta-se a Fundamentação Teórica de RSSF, padrões ZigBee/IEEE 802.15.4 e o hardware XBee, roteamentos e simulações em RSSF. No Capítulo 3, faz-se referência a análise dos trabalhos relacionados mais relevantes utilizados no desenvolvimento da proposta. No Capítulo 4, é apresentado o projeto da RSSF. O Capítulo 5 apresenta os experimentos realizados e o Capítulo 6 apresenta as simulações realizadas no Castalia. Por fim, no Capítulo 7, apresentamos as considerações finais, dificuldades encontradas e sugestões de trabalhos futuros.

# 2 Fundamentação Teórica

Este capítulo apresenta os conceitos fundamentais do trabalho. Na seção 2.1 são apresentadas as características e componentes de uma RSSF e na seção 2.2 algumas aplicações em RSSF. A seção 2.3 descreve e compara alguns protocolos de roteamento em RSSF. A seção 2.4 apresenta informações sobre simuladores de RSSF e a simulação dos algoritmos de roteamento estudados. Por fim, na seção 2.5 são descritos o Padrão ZigBee e o Padrão IEEE 802.15.4, bem como as características dos módulos XBee utilizados no trabalho.

## 2.1 Redes de Sensores Sem Fio

Redes de Sensores Sem Fio é um tópico que vem despertando grande interesse da comunidade científica devido a sua grande aplicabilidade em diversas áreas, como, por exemplo: saúde, agricultura, segurança, controle de fenômenos ambientais e pecuária, dentre outros.

### 2.1.1 Características de uma RSSF

Uma rede de sensores sem Fio (RSSF) é um tipo especial de rede móvel *ad-hoc* (*MANET - Mobile Ad hoc Networks*) composta por dispositivos autônomos e compactos com capacidade de sensoriamento, processamento e comunicação, denominados nós sensores (NAKAMURA, 2003).

No entanto, as RSSFs têm algumas características que as diferenciam das redes *ad-hoc* comuns, algumas delas citadas abaixo:

- Dependência da aplicação: dependendo do tipo de aplicação, fatores como tempo de vida, confiabilidade e conectividade devem ser observados para garantir a qualidade de serviço;
- Restrições de hardware: nós sensores normalmente possuem pouco poder de processamento e memória;
- Mobilidade de sensores: uma rede pode conter sensores estáticos, como por exemplo um sensor para medida de umidade do ar; ou dinâmica, como o exemplo de uma rede para monitoramento de animais;
- Número de sensores: dependendo da aplicação, o número de sensores pode ser elevado, chegando a milhares de nós. Por isso, fatores como manutenção e economia de

energia dos nós devem ser levados em consideração na implementação de um projeto de rede;

- Limitação de energia: algumas técnicas podem ser utilizadas para melhor gerenciamento de energia dos nós sensores, já que os nós sensores podem ser inacessíveis dependendo do tipo de aplicação, o que torna a reposição da bateria inviável;
- Localização dinâmica: a mobilidade dos nós é um fator que deve ser levado em consideração ao definir um tipo de roteamento para a rede;
- Nós sensores são propensos a falhas, devido à limitação de bateria.

Segundo Menezes (2004), as RSSFs têm como objetivo detectar objetos ou fenômenos químicos, físicos ou biológicos em uma determinada área ou região. Possuem dispositivos com capacidade de processamento e energia conectados por uma rede sem fio. Os nós sensores apresentam vantagens como: tamanho reduzido e baixo custo, o que possibilita a distribuição de inúmeros destes sensores em regiões onde o acesso por máquinas ou pessoas é muito difícil ou perigoso.

### 2.1.2 Componentes de uma RSSF

Uma RSSF é composta por um grupo de nós sensores, dispostos em uma área geográfica de interesse a ser monitorada. Segundo Loureiro et al. (2003), cada nó pode ser equipado com uma variedade de sensores, tais como: acústico, sísmico, infravermelho, calor, temperatura e pressão. Esses nós podem ser organizados em grupos (*clusters*), nos quais pelo menos um dos sensores deve ser capaz de detectar um evento na região, processá-lo e tomar uma decisão se deve fazer ou não uma difusão (*broadcast*) do resultado para outros nós.

Os nós sensores possuem recursos de processamento, armazenamento de informações, fonte de energia e interface de comunicação. Estes recursos são disponíveis por meio de um conjunto básico de componentes: processador, transceptor, memória, sensor e bateria, conforme observado na Figura 1 (LOUREIRO et al., 2003).

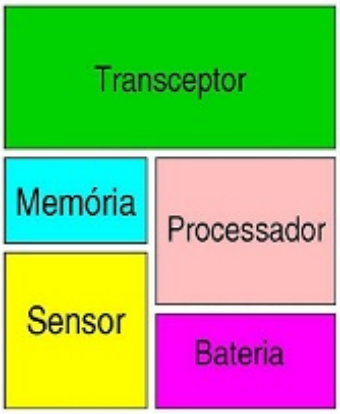



Figura 1 – Hardware básico de um nó sensor (LOUREIRO et al., 2003).

A bateria é o armazenamento de energia do dispositivo e possui capacidade limitada, com pouca possibilidade de reutilização, o que restringe a quantidade de energia da rede. A memória e o processador são utilizados para armazenamento e processamento das informações coletadas na área do monitoramento e possuem capacidade reduzida em virtude das dimensões do nó. O sensor é responsável pelo monitoramento da área e pode ser de temperatura, sísmico, detector de movimento, entre outros. O rádio inclui o sistema de transmissão, recepção, amplificador e antenas (NAKAMURA, 2003).

Um nó *sink*, ou sorvedouro, é um tipo especial de nó, responsável por coletar as informações dos outros nós sensores da rede. A principal diferença é que este nó normalmente não possui as mesmas limitações de energia dos outros nós sensores, já que este é responsável por receber todas as informações da rede.

### 2.1.3 Consumo de energia em RSSFs

Segundo Akyildiz et al. (2002a) o consumo de energia da rede, baseado nas principais funções dos nós sensores, pode ser dividido em: sensoriamento, comunicação e processamento de dados.

O maior custo energético da rede está no domínio da comunicação. As tarefas que mais consomem energia em uma RSSF são a transmissão e recepção de dados. A ocorrência de uma transmissão consome mais energia do que o processamento de dados (AKYILDIZ et al., 2002a).

A energia consumida para comunicação, entre dois pontos, está diretamente relacionada à distância entre eles. O processamento de dados, antes da comunicação, é importante para reduzir o tamanho dos pacotes a serem transmitidos. O gasto energético em processamento de dados é muito menor do que com a comunicação de dados (AKYILDIZ et al., 2002a).

## 2.2 Aplicações em RSSFs

Várias aplicações vem sendo desenvolvidas na área de RSSFs, e é grande o número de ambientes onde as RSSFs podem ser utilizadas. A seguir são citadas algumas das áreas de possível aplicação:

- **Controle ambiental**: é possível monitorar o comportamento do ambiente ou de animais de maneira não intrusiva;
- **Agricultura e pecuária de precisão**: sensores de umidade e composição do solo podem ser utilizados para melhorar a irrigação e o monitoramento de animais no pasto, proposta deste trabalho;
- **Saúde**: administração automática de medicamentos, monitoramento de pacientes em recuperação;
- **Tráfego**: monitoramento de tráfego de veículos em rodovias ou malhas ferroviárias;
- **Militar**: detecção de explosões e presença de material perigoso como radiação e monitoramento de movimentos inimigos.

De forma genérica, RSSFs podem ser usadas em segurança e monitoramento, controle, atuação e manutenção de sistemas complexos e monitoramento de ambientes internos e externos (LOUREIRO et al., 2003).

Alguns trabalhos de RSSF são apresentados no Capítulo 3.

## 2.3 Roteamento em RSSFs

Para que os dados monitorados pelos nós sensores sejam enviados até o nó sorvedouro, é necessário que sejam estabelecidas rotas entre a origem e o destino dos dados. Os protocolos que têm como objetivo determinar uma rota entre a origem e o destino em uma rede são chamados de protocolos de roteamento.

Na literatura, há inúmeros protocolos que tratam o roteamento de dados em RSSF e esta seção apresentará as principais características dos tipos de roteamento para RSSF na tentativa de definir uma melhor solução para o problema de monitoramento bovino proposto neste trabalho.

Em uma RSSF um dos pontos de maior interesse é a questão de energia dos nós sensores. Em virtude disso, a escolha do roteamento é de fundamental importância, já que com a escolha eficiente das rotas é possível balancear carga, agregar dados e utilizar outras técnicas de economia de energia.

Dependendo da densidade ou mobilidade da rede ou das extensas áreas de monitoramento, a utilização de comunicação por meio de um único salto pode não ser a solução mais adequada. Logo, é observada a necessidade de uma comunicação de múltiplos saltos.

A comunicação entre os nós sensores via rádio em uma rede de sensores sem fio permite naturalmente a difusão de uma única transmissão para mais de um nó sensor que esteja ao alcance do sinal de rádio. Essa característica é usualmente aproveitada pelos protocolos de roteamento para RSSF (AKYILDIZ et al., 2002b).

Considera-se que usualmente o nó destino é o nó sorvedouro, e que o nó origem é aquele que precisa enviar um pacote com informações obtidas de seus sensores.

Várias taxonomias são encontradas na literatura para os tipos de protocolos de roteamento para RSSF. A classificação segundo Suhonen et al. (2012) é apresentada a seguir:

1. Roteamento centrado em nós: os nós possuem identificadores globalmente únicos. Cada nó guarda uma tabela de roteamento com uma entrada para cada rota identificada pelo endereço destino e o nó para o próximo salto (*hop*) até o destino.

   A tabela de roteamento pode ser construída de forma proativa ou reativa:

   - A forma proativa é caracterizada pela descoberta de rotas a todos os destinos potenciais, o que pode exigir muito no requisito de memória em redes com grande quantidade de nós.
   - Já a abordagem reativa (sob demanda) é uma alternativa, mas pode provocar o atraso na construção de rotas no envio dos primeiros pacotes.

2. Roteamento centrado em dados: em redes com muitos sensores, pode não ser viável atribuir identificação a cada sensor. Assim, cada nó da rede envia informações diretamente ao sorvedouro, e o sorvedouro pode enviar uma solicitação de consulta para uma determinada área e aguardar que os nós daquela região respondam. Esses são alguns exemplos de roteamentos baseados em dados: *Flooding; Gossiping; SPIN; Directed Difusion e Rumor routing.*

   O roteamento centrado em dados ainda pode ser dividido em:

   - Baseado em interesse: no qual um nó sorvedouro envia uma solicitação para todos os nós da rede com o seu interesse (uma descrição dos dados que ele queira). Cada nó que contenha os dados que satisfaçam o interesse do sorvedouro inicia a transmissão dos dados;
   - Baseado em negociação: no qual há a troca de mensagens de negociação antes da transmissão dos dados reais. Isso economiza energia porque um nó pode determinar durante a negociação que os dados reais não são mais necessários.

Para esse tipo de protocolo ser útil, a sobrecarga de negociação e os tamanhos dos descritores de dados devem ser menores que os dados reais;

- Baseado em consulta: um nó realiza uma consulta expressa em uma linguagem de consulta, que é propagada pela rede de forma aleatória ou direcionada a uma determinada região. Após a consulta ser resolvida, o resultado é encaminhado de volta ao nó que originou a consulta.

3. Roteamento baseado em localização: alguns protocolos de roteamento em RSSF necessitam da localização dos nós para calcular a distância entre eles e seus nós vizinhos. Geralmente utiliza-se GPS para determinar a posição dos nós sensores na rede, o que pode aumentar o preço e o consumo da energia dos nós. Para minimizar esse custo alguns protocolos utilizam alternativas para descoberta de localização baseado em nós âncoras próximos a ele.

4. Roteamento baseado em vários caminhos (*multipath*): neste roteamento os dados podem ser encaminhados de um sensor até o destino por mais de um caminho. São exemplos desse protocolo o IRP (KWONG et al., 2012) e o *flooding* (AKYILDIZ et al., 2002b).

5. Roteamento baseado em custo: a cada nó é atribuído um valor de custo que é relativo à distância entre um nó e um sorvedouro. A vantagem é que um nó encaminha seu pacote a qualquer vizinho de menor custo, isso torna-se uma vantagem no consumo de energia. Uma desvantagem é que a criação de rotas deve ser feita proativamente.

De acordo com Suhonen et al. (2012), é importante observar que um protocolo de roteamento pode pertencer simultaneamente a mais de uma dessas classes.

Alguns protocolos de roteamento estudados são descritos a seguir e utilizados nas simulações para a RSSF proposta:

1. *Flooding* (AKYILDIZ et al., 2002b): protocolo em que os nós sensores realizam o *broadcast* das informações coletadas para todos os vizinhos. Nesse protocolo não há a preocupação com algoritmos de roteamento e manutenção de rotas. No *flooding*, descrito no Algoritmo 1, cada nó que recebe um pacote de dados repete o envio por *broadcast*, até que o número máximo de saltos do pacote seja alcançado ou que o destino de um pacote seja o mesmo nó que está enviando. Apesar da simplicidade em implementar esse protocolo, (TILAK; HEINZELMAN., 2002) cita alguns inconvenientes demonstrados na Figura 2 e Figura 3.

   Um dos problemas citados é o problema de implosão, no qual os nós sensores enviam um *broadcast* de dados aos nós vizinhos e essa informação chega mais de uma vez ao mesmo sensor por caminhos diferentes.

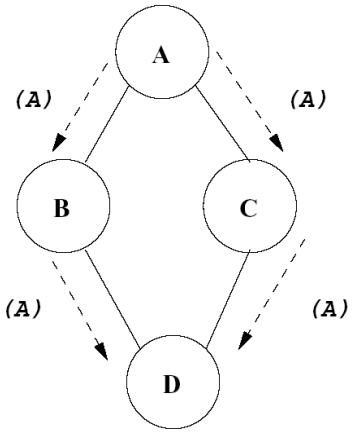



Figura 2 – Problema de implosão (TILAK; HEINZELMAN., 2002).

Na Figura 2 o problema de implosão é observado no momento em que os nós B e C reenviam para o nó D o mesmo pacote recebido pelo nó A.

Outro problema apresentado é o problema de sobreposição, no qual nós sensores próximos a uma área sensoreada captam a mesma informação e enviam aos nós vizinhos.

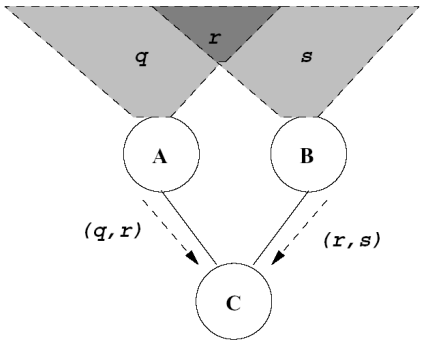



Figura 3 – Problema da sobreposição (TILAK; HEINZELMAN., 2002).

A Figura 3 apresenta o problema de sobreposição. Na figura, os nós A e B coletam o mesmo dado da região r e enviam a mesma informação ao nó C.

**Algoritmo 1:** Algoritmo Flooding

```
início
    a cada pacote de dados recebido por um nó;
    se o número máximo de saltos do pacote foi alcançado OU o pacote foi
    enviado pelo próprio nó então
        descarta o pacote;
    se o nó atual é o sorvedouro então
        recebe o pacote e envia os dados para a camada superior;
    senão
        reenvia o pacote a todos os vizinhos;
    fim
fim
```

Um outro problema deste protocolo é a cegueira de recursos, já que o *flooding* não leva em conta os recursos de energia disponíveis. Um protocolo consciente dos recursos energéticos deve levar em conta a quantidade da energia disponível em qualquer período durante sua execução.

2. *Gossiping* (AKKAYA; YOUNIS, 2005): o protocolo *Gossiping* é uma versão levemente melhorada do *Flooding*. Nesse protocolo ao invés de um sensor enviar dados para todos os vizinhos até chegar ao nó de destino, um nó escolhe aleatoriamente um dos vizinhos para enviar o pacote e isso acontece repetidamente até chegar ao nó de destino. Uma desvantagem dessa abordagem é que como a escolha é aleatória, o nó pode enviar para um vizinho que está mais distante do destino. O algoritmo *gossiping* está descrito no Algoritmo 2.

**Algoritmo 2:** Algoritmo Gossiping

```
início
    a cada pacote de dados recebido por um nó;
    se o pacote foi enviado pelo próprio nó então
        descarta o pacote;
    se o nó é o sorvedouro então
        recebe o pacote e envia os dados para a camada superior;
    senão
        reenvia o pacote para um vizinho escolhido aleatoriamente;
    fim
fim
```

3. AODV (*Ad-hoc On-Demand Distance Vector Routing*): o AODV (PERKINS; BELDING-ROYER; DAS, 2003) é um protocolo reativo, ou seja, cada nó constrói uma rota até o destino apenas quando precisa enviar informações. Sempre que esse nó precisa

enviar uma informação a algum destino no qual não possui na sua tabela de destinos inicia-se um novo processo de descobrimento de rota.

Para a construção da rota, o nó sensor envia um pacote especial chamado RREQ (*Route REQuest*) e esse pacote é encaminhado sucessivamente até o nó destino ou até algum nó que possua a rota até o nó de destino. Cada pacote de RREQ possui uma identificação da mensagem, que é gerada a cada nova requisição, chamado BroadcastID, além do número de sequência de destino, para certificar que as informações estão corretas e atualizadas.

A cada salto do pacote RREQ, os nós sensores atualizam suas tabelas com informações dos nós vizinhos, dessa forma o nó de destino pode estabelecer o caminho inverso ao receber a mensagem. Assim que o nó de destino recebe a mensagem ele retorna uma mensagem *RREP (Route REPly)* por *unicast* pelo caminho inverso até o nó que solicitou o descobrimento da rota. Quando o nó sensor recebe a mensagem RREP, ele grava a informação do próximo nó para alcançar o destino e grava também um tempo de vida, para que a rota seja excluída caso não seja utilizada nesse tempo determinado.

Quando a rota é descoberta, ela passa a ser utilizada. No entanto, devido à mobilidade da rede, alguns nós podem mudar de posição ou serem desligados. Neste caso, o nó intermediário que não conseguir encontrar o próximo nó irá enviar um pacote de erro RERR (*Route Error*) para o nó remetente. O protocolo pode ou não utilizar mensagens "*hello*" para cada nó informar aos vizinhos que está ativo.

4. HGR (*Hybrid Geographic Routing for Flexible Energy*) (CHEN et al., 2009): O HGR é um protocolo geográfico híbrido. Nesse protocolo, os sensores escolhem o nó transmissor baseados em sua posição e distância do nó destino.

   A escolha é feita considerando a menor distância de um nó sensor até o destino ou o menor ângulo de derivação entre o nó e o destino. Nesse caso, para a manutenção de rotas, esse protocolo assume que os nós saibam as localizações de seus vizinhos e do nó de destino.

   O algoritmo para a escolha do próximo salto está descrito no Algoritmo 3.

**Algoritmo 3:** Algoritmo HGR - SelectNextHop

**Entrada**: $\nu_h$: conjunto de nós vizinhos de $h$ na área *forwarding*, e nó sorvedouro $t$
**Saída**: nó $k$: vizinho que será o próximo salto
**para** *cada vizinho* $i$ *em* $\nu_h$*;*
**faça**
  **início**
    calcula $\theta_i$: ângulo de desvio entre a linha que conecta $h$ com $i$ e a linha que conecta $h$ com $t$
    $\theta_i = \arccos((D_i^h)^2 + (D_t^h)^2 - (D_t^i)^2/2D_i^h.D_t^h)$;
    calcula o $x_i = D_i^h.\cos(\theta_i)$;
    calcula o $y_i = D_i^h.\sin(\theta_i)$;
    calcula o valor $Q_i$, baseado nos critérios de direção e distância;
    $Q_i = (1 - (y_i/R))^2 + (x_i/R)^2$;
  **fim**
**fim**
seleciona o nó $k$ como próximo salto, onde $k = \text{argmax}\{Q_i | i \in \nu_h\}$;

O algoritmo usa a fórmula $\theta_i = \arccos((D_i^h)^2 + (D_t^h)^2 - (D_t^i)^2/2D_i^h.D_t^h)$ onde o $D_i^h$ é a distância entre o nó $h$ e o vizinho $i$, e $D_i^t$ é a distância entre o nó $i$ e o nó sorvedouro.

A segunda fórmula, que retorna o valor $Q_i$ é a que indica a elegibilidade do nó vizinho. Quanto maior o $x_i$, maior o valor de $Q_i$ se torna, por outro lado, quanto menor o valor $\theta_i$, maior o valor de $Q_i$.

As informações de posições dos nós vizinho e do nó sorvedouro são enviados nos pacotes de dados da rede, de modo que cada nó atualize as informações da posição dos nós vizinhos.

Uma dificuldade desse protocolo é que um nó deve saber sua posição e as posições dos nós vizinhos e do nó de destino. Em nós fixos, as posições poderiam ser adicionadas na memória como um valor constante, mas no caso de uma rede com nós móveis, pressupõe-se a utilização de um GPS para a coleta da posição de cada nó ou métodos matemáticos para o processamento da estimativa da localização por triangulação.

5. IRP (*Implicit Routing Protocol*) (KWONG et al., 2012): é um protocolo que foi desenvolvido com o propósito de monitorar o comportamento bovino.

   Esse protocolo possui um algoritmo de simples implementação baseado no número de saltos do nó destino até os nós sensores. O protocolo é dividido em duas fases.

   A primeira fase, apresentada no Algoritmo 4, é o envio de uma mensagem em *broadcast* pelo nó sorvedouro com a ID do sorvedouro e uma contagem de saltos iniciando

em 0. A cada salto a contagem é incrementada. Dessa forma, quanto mais saltos de distância do sorvedouro, maior será essa contagem de saltos.

**Algoritmo 4:** Algoritmo IRP - Etapa de configuração da rede

**início**

Nó sorvedouro envia pacote de configuração da rede com um número identificador e um número com um contador de saltos iniciando em 0;
Cada nó que recebe esse pacote incrementa o número de saltos e reenvia aos demais nós;

**fim**

Quando um nó da rede precisa enviar um pacote de dados, este envia o pacote com o número de saltos. Todos os nós que receberem o pacote e possuirem o número de saltos maior ou igual irão descartar esse pacote. Os nós, que receberem o pacote e tiverem número de saltos menor que o do sensor que está enviando reencaminharão a mensagem. Esse processo será repetido até a informação alcançar o nó sorvedouro. Esta é a fase de envio de dados, descrita no Algoritmo 5.

**Algoritmo 5:** Algoritmo IRP - Etapa de envio de dados dos sensores ao sorvedouro

**início**

um nó sensor recebe os dados enviados por outro nó sensor com os dados do sensor juntamente com o valor do contador de saltos e o número de identificação do sorvedouro;
**se** *um nó está dentro do alcance da estação base* **então**
    envia dados para a estação base;
    deleta dados desde que seja enviado com sucesso à estação base;
**se** *nível do sensor atual é menor que o do sensor recebido* **então**
    envia dados por broadcast junto com o número do nível atual;
**senão**
    descarta dados;
**fim**

**fim**

A etapa de configuração da rede é repetida periodicamente para manter os contadores sempre atualizados e permitir a mobilidade dos nós.

6. *History-based protocol* (JUANG et al., 2002): o protocolo utiliza um contador que é incrementado cada vez que algum sensor faz uma transmissão direta ao nó sorvedouro. Desse modo, quanto maior o número do contador, maior a probabilidade do sensor estar mais próximo ao sorvedouro.

Cada vez que um nó procura por nós vizinhos ele solicita o contador de cada um e depois envia os pacotes ao vizinho com o contador de maior número e isso é repetido até a informação encontrar o nó sorvedouro.

Este protocolo foi desenvolvido inicialmente para ser utilizado no projeto ZebraNet, que é um projeto de monitoramento de Zebras com RSSF. Seu objetivo era compreender alguns padrões de comportamento desses animais. Os sensores podiam detectar a posição pelo GPS e outros dados como frequência cardíaca e temperatura corporal.

O projeto ZebraNet difere-se em vários pontos da proposta deste trabalho, pois no caso do ZebraNet os pesquisadores utilizaram mais de um nó sorvedouro e estes eram móveis, sendo transportados periodicamente por carros, já que as zebras estavam em local aberto.

O algoritmo *History-based protocol* está descrito no Algoritmo 6.

**Algoritmo 6:** Algoritmo *History-based*

```
início
    a cada busca por vizinhos;
    se um nó está dentro do alcance da estação base então
        envia dados para a estação base;
        deleta dados desde que seja enviado com sucesso à estação base;
        incrementa o contador com nível hierárquico;
    senão
        verifica nível de hierarquia dos vizinhos;
        envia dados ao vizinho com maior nível, desempatando aleatoriamente;
        diminui o nível hierárquico a cada X localizações;
    fim
fim
```

7. ORW (*Opportunistic Routing in Low Duty-Cycle Wireless Sensor Networks*) (GHADIMI et al., 2014) é um protocolo oportunístico para RSSF que, difere-se de outros protocolos oportunísticos tradicionais, pois leva em conta os ciclos de trabalho (*Duty-Cycle*) [1] dos nós sensores.

   Enquanto no roteamento oportunístico tradicional no qual os dados de um sensor são enviados ao sorvedouro sempre pelo mesmo caminho entre os nós, esses sensores que fazem o roteamento podem estar no estado *sleep*, ou seja, não estão recebendo as informações. Isso causa o reenvio da informação até que o nó acorde e possa receber os pacotes para reenviar.

[1] Abordagem que os nós permanecem em operação em uma pequena parte do tempo, ficando desligados no restante do tempo (BUETTNER et al., 2006).

No ORW o protocolo seleciona uma lista de possíveis sensores encaminhadores (*forwarders*), e caso um dos sensores esteja no estado *sleep* no momento, outros sensores que estão nessa lista podem reencaminhar o pacote.

O foco deste trabalho é a comparação entre os protocolos de roteamento, por isso não explorará algoritmos que assumam períodos de *sleep*, já que os períodos de *sleep* são definidos em uma camada mais baixa. O ORW, citado anteriormente é apenas um exemplo entre vários outros disponíveis que trabalham com períodos de *sleep*.

### 2.3.1 Comparação entre protocolos estudados

Vários protocolos foram estudados e alguns selecionados para o trabalho, considerando algumas particularidades de cada um, tais como: uso de GPS para construção de rotas (método geográfico), formação de rotas de forma reativa ou proativa, opção de mobilidade dos nós, e outros.

Tabela 1 – Tabela comparativa dos protocolos de roteamento para RSSF apresentados

| Protocolo | Geográfico | Multipath | Livre de loop | Permite mobilidade | Eficiência de energia |
|---|---|---|---|---|---|
| *Flooding* | Não | Sim | Não | Sim | Baixa |
| *Gossiping* | Não | Sim | Não | Sim | Baixa |
| AODV | Não | Não | Sim | Sim | Moderada |
| HGR | Sim | Não | Sim | Sim | Moderada |
| IRP | Não | Sim | Sim | Sim | Alta |
| ORW | Não | Sim | Sim | Sim | Alta |
| *History-based* | Não | Sim | Sim | Sim | Alta |

Para fins de comparação entre os protocolos e algumas de suas características, a Tabela 1 apresenta as principais características observadas em cada algoritmo analisado.

Cada algoritmo selecionado possui suas diferenças. A escolha destes algoritmos foi para tentar simular diferentes características dos protocolos. O Flooding para simular a difusão dos dados sem nenhuma manutenção de rotas, o Gossiping para simular um algoritmo com envio aleatório. O protocolo AODV foi escolhido para estudo por ser base da implementação do protocolo ZigBee, que é o padrão utilizado no rádio utilizado no trabalho proposto e o HGR para poder aproveitar a informação que é coletada pelo colar, já que o colar utiliza um GPS. O IRP e o *History-based* foram selecionados por já terem sido utilizados em trabalhos de monitoramento de animais.

## 2.4 Simulações em RSSFs

Segundo Du et al. (2014), simulação é uma forma de avaliar, com boa aproximação, a um custo menor e usualmente em menos tempo, o desempenho de protocolos em plataformas de *hardware* específicas.

Diversos simuladores têm sido utilizados para simulação de RSSF. Neste projeto, o simulador foi utilizado para avaliação de desempenho e economia de energia dos algoritmos de roteamento selecionados. Ao final do trabalho, serão apresentados comentários e gráficos sobre as simulações realizadas e os resultados analisados para avaliar os algoritmos escolhidos.

O simulador utilizado nos experimentos foi o Castalia (CASTALIA, 2014), que é um simulador para RSSF, *Body Area Network* (BAN) e redes de dispositivos embarcados de baixo consumo.

Castalia é um simulador baseado no *framework* de simulação OMNeT++ (OMNET++, 2014), mas difere-se desse ao ser utilizado para testar algoritmos de roteamento de RSSF com o uso de modelos mais realísticos de rádio e de canal sem fio.

Demais considerações sobre o Castalia:

- O código fonte do Castalia possui estrutura modular, assim como o OMNeT++;
- É desenvolvido em linguagem C++;
- O código fonte dos módulos pode ser escrito em linguagem C++.

Segundo Boulis et al. (2011) a estrutura do Castalia é composta de nós (*node*), processo físico (*Physical Process*) e canal (*Wireless Channel*). Os nós não se comunicam diretamente e toda a comunicação é feita por intermédio dos canais. Os canais avaliam se um nó receberá a mensagem ou não. A Figura 4 apresenta a arquitetura do Castalia.

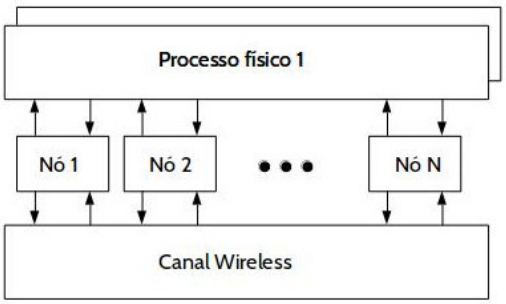



Figura 4 – Os módulos e as conexões do simulador Castalia. Adaptado de (BOULIS et al., 2011).

O Castalia utilizado no trabalho é a versão 3.3. O simulador possui 2 *scripts* para simulações e análise dos resultados. São eles, respectivamente Castalia e CastaliaResults. Além desses, existe também um *script* utilizado para a geração de gráficos, chamado CastaliaPlot.

Os protocolos de roteamento disponíveis juntamente com o Castalia são apenas o *bypassRouting* e o *multipathRings*. No primeiro protocolo, o envio dos dados é feito apenas a um salto para os nós vizinhos, sem reenvio dos pacotes. Nesse caso, se o destino está a mais de um salto do nó que está enviando, a informação é perdida.

O protocolo *multipathRings* é muito semelhante ao protocolo IRP citado neste trabalho. Nesse protocolo existe inicialmente o envio de um pacote do sorvedouro por *broadcast* para todos os nós da rede com um contador que é incrementado a cada salto. Dessa forma os nós mais distantes possuem um contador com número maior e sempre enviam os dados sensoreados aos nós com contador menor até que a informação chegue ao sorvedouro.

O Castalia conta com apenas 2 módulos de mobilidade, o primeiro, chamado *noMobilityManager*, é utilizado para redes de sensores onde não há mobilidade dos nós. O segundo módulo, chamado *lineMobilityManager*, que simula o movimento em linha reta de um nó, indicando como parâmetro as posições iniciais e finais dos nós, a velocidade de locomoção e o tempo de atualização das posições. Para que as simulações realizadas no trabalho refletissem com maior aproximação o comportamento do boi no pasto, desenvolvemos um novo módulo que utiliza as posições reais do boi no pasto. Para isso, fizemos um primeiro experimento com os colares desenvolvidos por (JESUS, 2014) a fim de obter as posições dos bois no pasto, e em seguida, foi feito um pré-processamento com as informações coletadas pelo GPS e convertidas para um formato que pudesse ser adicionado ao Castalia.

Um arquivo de configuração define valores para os parâmetros da simulação ou apenas sobrescreve valores que são padronizados no simulador. Dessa forma é possível simular uma grande variedade de cenários diferentes.

## 2.5 Padrão ZigBee/IEEE 802.15.4

Até o surgimento do padrão ZigBee, a maioria das tecnologias para RSSF oferecia altas taxas de transmissão e alto consumo de energia e as aplicações normalmente eram compostas por um número relativamente pequeno de dispositivos.

O ZigBee é um padrão desenvolvido por uma aliança de empresas (*ZigBee Alliance*) interessada em criar um padrão de comunicação sem fio de baixo custo e baixo consumo de energia. Soluções que utilizam o padrão ZigBee podem ser inseridas em dispositivos eletrônicos, automação residencial e predial, controles industriais, periféricos de computadores, brinquedos, jogos entre outras (ALLIANCE, 2012).

Diferente das tecnologias oferecidas anteriormente, o ZigBee proporciona uma infraestrutura de rede com as seguintes características (SOARES, 2012):

- Formação autônoma de redes com grandes quantidades de dispositivos em uma grande área de cobertura, em que os dispositivos pudessem se comunicar de forma confiável e segura por anos, sem a intervenção de um operador;
- Baixo consumo de energia associado a um reduzido custo de infraestrutura, complexidade e tamanho;
- Taxa de transmissão de dados relativamente baixa;
- Um protocolo padronizado e aberto que permitisse a interoperabilidade entre produtos de diferentes fabricantes para um mercado global.

Para implementar as camadas MAC (*Medium Access Control*) e PHY (*Physical Layer*) o ZigBee utiliza o padrão 802.15.4 do IEEE. O IEEE 802.15.4 trata da especificação das duas camadas inferiores, enquanto *ZigBee Alliance* provê as camadas superiores (da camada de rede à camada de aplicação) da pilha de protocolos.

As camadas superiores mostradas na Figura 5 são responsáveis por prover configuração da rede, manipulação e roteamento de mensagens definidas pelo grupo *ZigBee Alliance*. A camada de rede é responsável pelo roteamento das informações entre os sensores. Em RSSF, devido à grande quantidade de nós sensores que podem ser utilizados e também por existirem sensores que podem ser móveis, é difícil definir um protocolo de roteamento que irá atender às necessidades da aplicação.

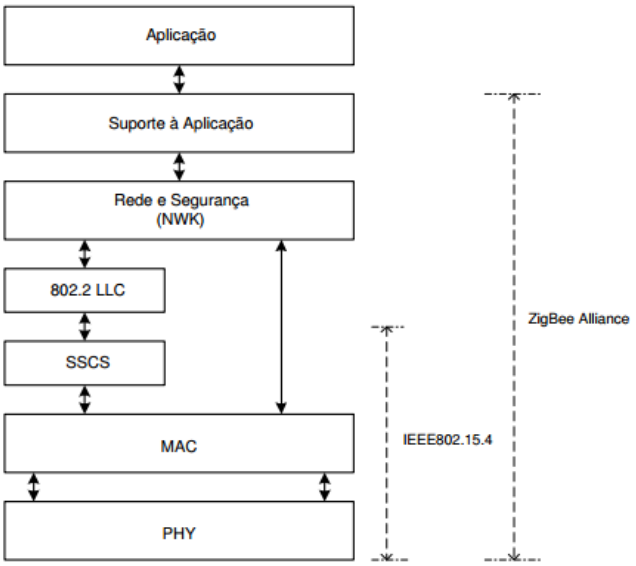



Figura 5 – Camadas de rede (SANTOS, 2007).

A rede ZigBee permite comunicações robustas e opera na frequência ISM (*Industrial, Scientific and Medical*), não necessitando de licença para funcionamento na maioria dos países. As redes ZigBee oferecem uma excelente imunidade contra interferências, e a capacidade de hospedar milhares de dispositivos numa rede (teoricamente 65.536), com taxas de transferências de dados a 250Kbps (RAMYA; SHANMUGARAJ; PRABAKARAN, 2011).

As principais razões para o uso da tecnologia ZigBee (RAMYA; SHANMUGARAJ; PRABAKARAN, 2011) são:

- Oferece suporte a um grande número de nós;
- Facilidade de implementação;
- Baixo consumo de energia;
- Segurança;
- Engloba uma comunidade com 30 ou mais fornecedores de produtos e serviços;
- Tem protocolos de padrões abertos com taxas de financiamento insignificantes ou praticamente inexistentes;
- Baixa potência;

- Não necessita de muita manutenção.

Nas redes ZigBee participam duas classes de dispositivos físicos, os FFD (*Full Function Devices*) e os RFD (*Reduced Function Devices*):

- Os FFDs são dispositivos de função completa que funcionam em qualquer topologia, capazes de desempenhar as funções de Coordenadores (*Coordinators*), Roteadores (*Routers*) ou Dispositivos finais (*End devices*), comunicam com qualquer outro dispositivo FFD ou RFD e são mais difíceis de implementar.

  A norma ZigBee atual requer que um FFD esteja sempre ativo, o que na prática significa que FFDs devem ser constantemente alimentados. Para FFDs alimentados por baterias, estas terão uma vida útil de poucos dias (KENNELLY, 1995).

- Os RFDs são dispositivos de função reduzida, limitados à topologia em estrela, comunicam-se apenas com os roteadores ou o coordenador da rede (FFDs). São destinados a aplicações simples, como sensores passivos e normalmente não tem necessidade de enviar grandes quantidades de dados. Podem se associar com um único FFD de cada vez.

  Segundo Kennelly (1995), o RFD pode ser implementado usando um mínimo de recurso e capacidade de memória.

Além das classes de dispositivos físicos existe uma classificação de dispositivos lógicos no padrão ZigBee em que cada classe define a função de um nó dentro da rede.

Segundo Ramya, Shanmugaraj e Prabakaran (2011), as classes de dispositivos lógicos que definem uma rede podem ser divididas em:

- *Coordinator* (coordenador): forma a raiz da rede em árvore e pode ser a ponte para outras redes, no intuito de expandi-la, visto que cada nó Coordenador suporta mais de 65000 dispositivos ZigBee. Há apenas um Coordenador em cada rede, único responsável por iniciar a rede e selecionar os parâmetros de rede, tais como o canal de rádio frequência, identificador da rede e outros parâmetros operacionais. O Coordenador também pode armazenar informações sobre a rede como chaves de segurança (RAMYA; SHANMUGARAJ; PRABAKARAN, 2011).

- *Router* (roteadores): funciona como nó intermediário da rede, retransmitindo dados para outros dispositivos. O roteador pode ser ligado a uma rede já existente, sendo capaz de aceitar ligações de outros dispositivos e ser uma espécie de retransmissor para a rede. A rede pode ser aumentada usando roteadores ZigBee (RAMYA; SHANMUGARAJ; PRABAKARAN, 2011).

- *End devices* (dispositivos finais): poderão ser de baixa potência ou dispositivos alimentados por baterias. Estes dispositivos podem obter informações a partir dos diversos sensores e podem se comunicar com o coordenador ou roteadores. Esta funcionalidade reduzida potencializa a redução do consumo de energia. Estes dispositivos não precisam estar continuamente ativos. Dispositivos finais não podem rotear pacotes.

  Quando um *end device* precisa se comunicar com um nó mais distante, o roteador mais próximo o ajuda, recebendo o pacote e enviando a outro nó. Por isso, roteadores não podem entrar no modo *sleep*.

  A Tabela 2 apresenta algumas das características de cada tipo lógico do padrão *ZigBee*.

Tabela 2 – Funções do ZigBee por tipos lógicos

| **F**unção da camada de rede | Coordenador | Roteador | Dispositivo Final |
|---|---|---|---|
| Estabelecer nova rede ZigBee | **X** | | |
| Definir endereço lógico de rede | **X** | **X** | |
| Permitir outros dispositivos entrarem ou saírem da rede | **X** | **X** | |
| Manter lista de vizinhos ou rotas | **X** | **X** | |
| Rotear pacotes da camada de rede | **X** | **X** | |
| Transmitir pacotes na camada de rede | **X** | **X** | **X** |
| Entrar ou sair de uma rede ZigBee | **X** | **X** | **X** |

### 2.5.1 Topologia do ZigBee

Os nós da rede ZigBee podem assumir tipos de topologias diferentes que indicam como os nós são ligados logicamente uns aos outros. Segundo Faludi (2010) são 4 os principais tipos de topologia:

1. *Pair* (par): é a forma mais simples de uma rede com apenas dois nós. Um nó deve ser configurado como coordenador, de modo que a rede possa ser formada, e o outro pode ser configurado como um roteador ou um dispositivo final.

2. *Star* (estrela): um arranjo também muito simples. O coordenador encontra-se no centro da rede e se conecta a um círculo de dispositivos finais. Toda mensagem deve passar pelo coordenador, que as encaminha para o dispositivo destinatário. Nesse caso, os dispositivos finais não se comunicam entre si diretamente.

3. *Cluster Tree* (árvore): esta é uma topologia de rede, onde os roteadores formam a parte que dá estrutura à rede. Os dispositivos finais são agrupados em torno dos roteadores. A topologia árvore não é muito diferente da *mesh*. Apresenta alcance maior que as duas topologias apresentadas anteriormente.

4. *Mesh* (malha): A topologia em malha, além de empregar o coordenador, que faz o gerenciamento da rede, emprega também roteadores. Vários dispositivos finais podem ser ligados aos roteadores ou ao coordenador. Eles podem enviar e receber informação, porém é necessária a ajuda de roteadores ou do coordenador para trocar mensagens entre si. Apresenta longo alcance e maior flexibilidade.

A Figura 6 apresenta as topologias, segundo Faludi (2010).

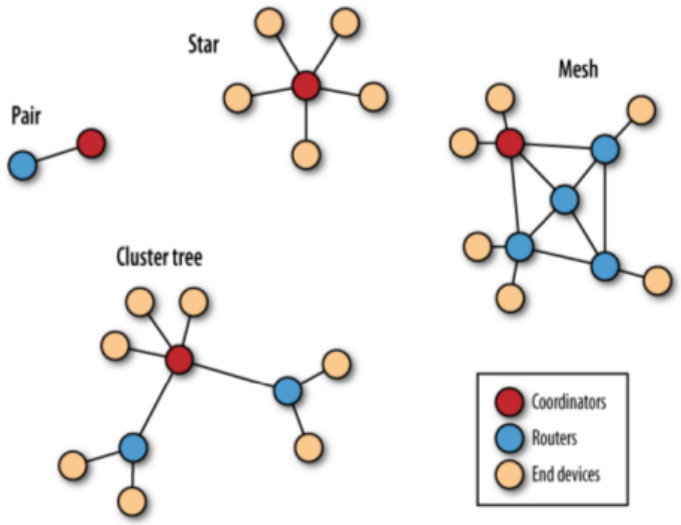



Figura 6 – Topologias ZigBee pair, star, mesh e cluster tree. (FALUDI, 2010).

Neste trabalho foi preciso adotar o tipo de topologia *mesh* (malha), e todos os nós (colares) foram configurados como roteadores da rede. Isso porquê um roteador é um tipo de dispositivo FFD que pode encaminhar pacotes de um nó para outro, e, a mobilidade dos nós torna necessária esse tipo de configuração. Outro tipo de topologia que poderia ser adotada, seria a topologia em árvore, caso existissem nós fixos no pasto, que pudessem servir como roteadores da rede. Nesta alternativa, os colares poderiam ser configurados como dispositivos RFD e poderiam entrar em período de *sleep*, o que aumentaria a vida útil de cada nó.

## 2.6 Módulos XBee

Apesar dos nomes semelhantes, é importante diferenciar o XBee e ZigBee. XBee é o nome dado pela DIGI, fabricantes dos módulos físicos.

Os módulos XBee são módulos de rádio frequência que utilizam o padrão de comunicação ZigBee para enviar informações em uma rede sem fio.

O XBee foi escolhido para este projeto pois possui o protocolo ZigBee no próprio microcontrolador e foi desenvolvido com foco em economia de energia, podendo ativar o modo *sleep*, que permite menor consumo de energia. Além disso, o XBee possui uma configuração de pinos que pode ser encaixada diretamente em um Arduino FIO, que é a placa utilizada na base do colar desenvolvido por (JESUS, 2014) e (LOMBA, 2015).

Há várias combinações de hardware para os módulos XBee, porém existem duas variedades básicas, como mostrado na Figura 7.

- XBee: Este módulo é baseado em padrões ponto-a-ponto de comunicação e redes *mesh*. Possui maior popularidade pela sua simplicidade e é utilizado em aplicações simples.
- XBee-PRO: Supera o XBee em poder computacional, tamanho e custo. Baseada em padrões de redes *mesh*, a principal topologia utilizada em redes de sensores.


Figura 7 – Módulo XBee e XBee-PRO.

A Figura 8 apresenta a pinagem de um XBee Pro. O XBee Pro é composto por 20 pinos, entre eles os pinos de alimentação de energia, entrada e saída de dados e pinos de entrada analógica, que permitem conectar alguns tipos de sensores analógicos sem a necessidade de um microcontrolador externo, como o Arduino. Pode-se ligar um XBee no modo roteador diretamente em uma bateria e configurar o *firmware* desse XBee para atuar

apenas como um roteador da rede, sem a necessidade de um microcontrolador adicional. Essas são algumas possibilidades de uso deste *hardware*.

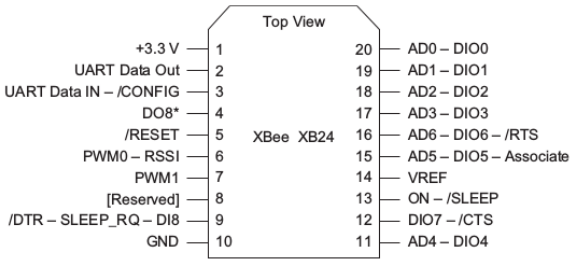



Figura 8 – Pinagem XBee

Os XBee, diferem-se também nos tipos de antenas, Figura 9.

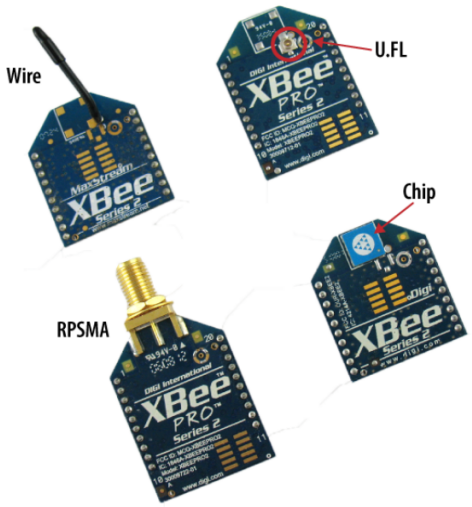



Figura 9 – Tipos de antenas dos módulos XBee

Neste trabalho foram utilizados dois tipos diferentes de antenas. Uma delas, dos colares, será o tipo de antena interna (apenas chip), este tipo foi utilizado pois era preciso encapsular todo o equipamento em uma pequena caixa que fica presa ao colar do animal. O outro tipo de antena, utilizada no sorvedouro foi o tipo RPSMA. Todo o equipamento do sorvedouro também permanece encapsulado, porém o tipo de XBee RPSMA permite incluir uma antena externa, que pode ser deixada fora do invólucro de equipamentos.

O XBee opera na frequência 2,4GHz e existe uma faixa de aproximadamente 16 canais diferentes dependendo da versão do *hardware* que podem ser escolhidas durante a gravação do *firmware*, para evitar interferências entre outros dispositivos.

O modelo XBee Series 1 implementa estritamente o protocolo IEEE 802.15.4. O segundo modelo, denominado Series 2, utilizado neste trabalho implementa o padrão ZigBee, que permite a utilização de redes do tipo *mesh* de forma facilitada.

### 2.6.1 Modos de Comando

Os módulos XBee comunicam-se utilizando dois modos de comando, AT- *Transparent Mode* (Modo Transparente) e API- *Application Programming Interface* (Interface de Programação de Aplicação).

- Modo AT - *Transparent/Command Mode*: Segundo Faludi (2010) o modo AT é o modo padrão para os módulos XBee. É chamado de modo "transparente", pois envia as informações exatamente como recebeu. Neste caso, o XBee funciona apenas como um transmissor sem fio. Toda informação que chega pelo pino de entrada da UART é transmitida pela antena, e todo dado recebido pela antena é diretamente enviado pela UART sem nenhuma alteração.

  O modo AT é muito utilizado para dispositivos *end devices* com sensores ligados diretamente à porta IO do XBee. Alguns tipos de sensores como sensor de luminosidade podem ser ligados diretamente a um dos pinos IO do XBee e coletar os dados de leitura periodicamente, conforme configurado diretamente do *firmware* do XBee.

- Modo API - *Application Programming Interface*: O modo API é uma alternativa ao modo AT. Os dados devem ser encapsulados em *frames* (pacotes) para serem enviados pelo XBee.

  Um pacote API (Figura 10) contém quatro partes principais:

  1. **Start Delimiter**: byte que indica o início do pacote;
  2. **Length**: dois bytes que indicam o tamanho do campo de dados;
  3. **Frame data**: parte que contém as informações desejadas, como por exemplo, dados coletados pelos sensores. O quadro cmdID (identificador-API) indica qual mensagem API está contida no quadro cmdData (contém dados específicos da mensagem);
  4. **Checksum**: utilizado para verificar a integridade dos dados contidos na mensagem.


Figura 10 – Estrutura de um pacote API (DIGI, 2015).

Além disso, cada pacote API recebe um número de identificação e possui uma ação específica na rede. A Figura 11 ilustra os tipos possíveis de pacotes API.

| API Frame Names | API ID |
|---|---|
| AT Command | 0x08 |
| AT Command - Queue Parameter Value | 0x09 |
| ZigBee Transmit Request | 0x10 |
| Explicit Addressing ZigBee Command Frame | 0x11 |
| Remote Command Request | 0x17 |
| Create Source Route | 0x21 |
| AT Command Response | 0x88 |
| Modem Status | 0x8A |
| ZigBee Transmit Status | 0x8B |
| ZigBee Receive Packet (AO=0) | 0x90 |
| ZigBee Explicit Rx Indicator (AO=1) | 0x91 |
| ZigBee IO Data Sample Rx Indicator | 0x92 |
| XBee Sensor Read Indicator (AO=0) | 0x94 |
| Node Identification Indicator (AO=0) | 0x95 |
| Remote Command Response | 0x97 |
| Over-the-Air Firmware Update Status | 0xA0 |
| Route Record Indicator | 0xA1 |
| Many-to-One Route Request Indicator | 0xA3 |

Figura 11 – Descrição dos pacotes API ZigBee (DIGI, 2015).

Neste projeto foi feito o uso do XBee no modo API. Neste caso, o microcontrolador do Arduino é o responsável pela montagem dos pacotes de dados conforme especificação da API. Foi utilizada a API 0x10 (*ZigBee Transmit Request*) no colar para transmitir e a API 0x90 (*ZigBee Receive Packet*) no sorvedouro para receber os pacotes da rede.

### 2.6.2 Software de configuração dos Módulos

O X-CTU é o software oficial para a configuração dos módulos XBee disponibilizado pela (DIGI, 2016a), responsável pela gravação do *firmware* nos módulos XBee.

Para configurar o XBee com o X-CTU é necessário um adaptador para ligar o módulo ao computador. Neste caso, foi utilizado o adaptador USB da SparkFun, visto na Figura 12, para acessar o módulo XBee pela porta serial do computador por meio do *software* X-CTU.

Figura 12 – Adaptador USB para XBee.

A Figura 13 apresenta a aba de configuração do *firmware* do XBee no X-CTU.

A opção *Update* é utilizada para gravar tipos diferentes de *firmwares*. Neste momento, o usuário pode definir a classe do módulo: coordenador, roteador ou dispositivo final e se será configurado no modo AT ou API.

Depois disso é possível definir demais parâmetros no *firmware*, como endereço do destino, canal de operação, PAN ID (ID da rede), períodos de *sleep* e outros.

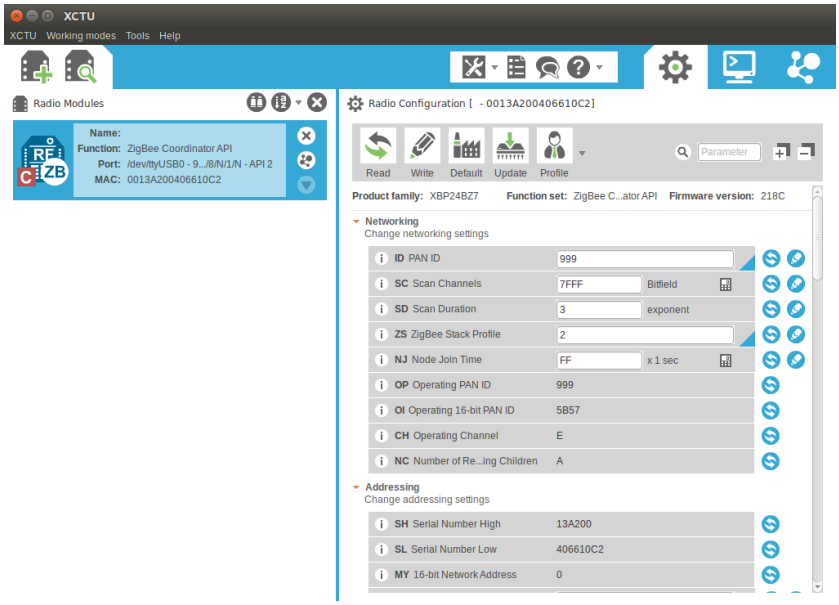



Figura 13 – X-CTU: Aba de configuração do modem XBee.

### 2.6.3 Modos de operação do XBee

Segundo o manual do fabricante, o XBee apresenta cinco modos de operação, conforme apresentado na Figura 14. O modo *idle* é quando o módulo não está enviando ou recebendo pacotes e indica que o módulo está ocioso esperando para entrar em outro modo de operação. Os modos *transmit* e *receive* indicam que o módulo está, respectivamente, transmitindo ou recebendo dados por meio de RF (rádio frequência). O modo *command*

é ativado para permitir a execução de comandos enviados aos módulos, tais como ler ou modificar os parâmetros dos módulos. E por fim, o modo *sleep* é ativado quando o módulo entra em estado de baixo consumo de energia, porém não pode ser ativado para XBees configurados como roteadores ou coordenadores de rede, apenas para *end devices*.

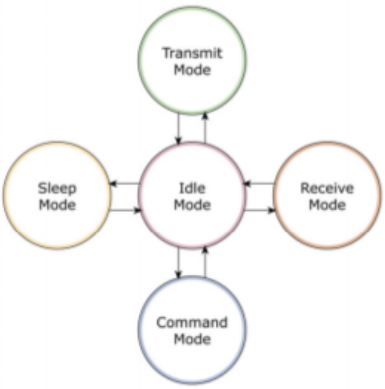



Figura 14 – Modos de operação do XBee (DIGI, 2015).

### 2.6.4 Camadas de rede do XBee

Segundo o manual do fabricante, as camadas do XBee são organizadas da seguinte maneira:

- *PHY*: define a operação física do dispositivo incluindo a sensibilidade de recepção, rejeição de canal, potência de saída, número de canais, modulação de chip, e as especificações de taxa de transmissão. A maioria das aplicações ZigBee operam na banda ISM de 2,4 GHz, a uma taxa de dados de 250 kbps.
- MAC: Gerencia transações de dados de RF entre dispositivos vizinhos (ponto a ponto). O MAC inclui serviços como repetição de transmissão e gestão de ACKs e técnicas de prevenção de colisão (CSMA-CA).
- *Network*: adiciona capacidades de roteamento que permite que os pacotes de dados de RF atravessem por vários dispositivos (múltiplos saltos "*hops*") encaminhando os dados da origem ao destino (*peer-to-peer*).
- APS (AF): camada de aplicação que define vários objetos de endereçamento, incluindo perfis, grupos e *endpoints*.
- ZDO: camada de aplicação que fornece a descoberta de dispositivos e recursos avançados de gerenciamento de rede.

### 2.6.5 Endereçamento XBee Zigbee

A documentação da Digi (DIGI, 2016a) apresenta os endereços do XBee como sendo:

- Endereço 64-bits: cada módulo XBee possui um endereço de 64 bits que o distingue de outros e previne duplicação da informação. Este endereço, também chamado de MAC, é atribuído unicamente a um dispositivo, de modo que, dois módulos não podem ter o mesmo endereço. O endereço 64 bits é uma concatenação dos parâmetros SH *(Serial Number High)* e SL *(Serial Number Low)* em qualquer módulo. O SH (*Serial Number High*) é usualmente o mesmo para todos os módulos XBee (0013A200), sendo o prefixo que identifica os dispositivos da fabricante Digi, a parte SL é diferente para cada módulo. Este endereço é impresso atrás de cada módulo (Figura 15).


Figura 15 – Endereço MAC do XBee

O endereço 64 bits 000000000000FFFF é reservado para envio de mensagens por *broadcast*.

- Endereço 16-bits: um dispositivo recebe um endereço 16-bits randômico quando ingressa em uma rede, então esse endereço é também conhecido como "endereço de rede". Este endereço pode mudar apenas se um conflito de endereços for detectado ou se o dispositivo deixar a rede e entrar novamente. O valor do endereço de 16 bits pode ser lido por meio do parâmetro (MY). O endereço 0000 é reservado ao coordenador, enquanto um valor de FFFE significa que o dispositivo ainda não ingressou em uma rede.

- *Node identifier*: NI é uma pequena string de texto que permite que um usuário nomeie um módulo com um nome mais amigável. Neste caso, a unicidade não é garantida. É possível ler ou modificar uma identificação através do parâmetro (NI-*Node identifier*).

A Tabela 3 apresenta tipos e exemplos de endereços no XBee.

De acordo com o artigo da Digi (DIGI, 2016b) sobre endereçamento do XBee-ZigBee:

Tabela 3 – Exemplo de endereçamentos XBee

| Tipo | Exemplo | Único |
|---|---|---|
| 64 bits | 0013A20012345678 | Sempre |
| 16 bits | 1234 | Único em uma rede |
| Identificador do nó | Roteador1 | Não garantido |

- O *firmware* XBee ZigBee usa o mesmo protocolo RF 802.15.4 usado em outras versões de *firmware* (não-ZigBee). Mesmo que o protocolo RF de base seja o mesmo, os sistemas de endereçamento e entrega de mensagens são muito diferentes. O *firmware* ZigBee usa a pilha ZigBee sobre o protocolo RF 802.15.4 para rede em malha.

- Assim como o *firmware* 802.15.4, o ZigBee tem duas formas de endereçamento *Broadcast* e *Unicast*. ZigBee tem uma forma muito mais robusta de transmissão de mensagens do que 802.15.4. As mensagens de *unicast*, por causa do funcionamento da rede de malha, têm o que é chamado de Endereço Estendido (64 bits) e Endereço de Rede (16 bits).

- Uma mensagem *broadcast* é uma mensagem destinada a todos os módulos em qualquer PAN (*Personal Area Network*). No ZigBee, uma mensagem *broadcast* é enviada uma vez e, em seguida, repetida pelo coordenador / roteador vizinho. Uma vez que uma mensagem *broadcast* foi repetida um certo número de vezes não será repetida mais. Em outras palavras, uma mensagem de *broadcast* só pode propagar por uma quantidade predefinida de saltos antes que ela pare de ser re-enviada.

- Uma mensagem *unicast* é enviada de um módulo para qualquer outro módulo com base no endereçamento de módulos, utilizando um dos protocolos de roteamento disponíveis.

### 2.6.6 Roteamento no XBee

A ideia inicial do projeto era comparar protocolos de roteamento para a implementação do protocolo mais eficiente para o contexto de monitoramento bovino. Infelizmente, o *hardware* XBee Serie 2, utilizado no projeto, apresenta uma limitação para sua programação. Outros tipos de *hardware* do XBee implementavam o protocolo 802.15.4 mas saíram de linha. Apesar de possuir uma API de programação aberta, este *hardware* não permite que a camada de roteamento seja alterada, toda alteração realizada pela API deve ser realizada na camada de aplicação não sendo possível alterar camadas inferiores.

Considerando essas limitações, a Tabela 4 mostra os três tipos de roteamentos que o XBee Serie 2 oferece, segundo o *datasheet* do fabricante.

Tabela 4 – Protocolos de roteamentos do XBee.

| | Descrição |
|---|---|
| *AODV* | Roteamentos são criados entre o nó fonte e o nó de destino, possivelmente por múltiplos saltos entre os nós. Cada dispositivo sabe o próximo nó para quem enviar os dados para que cheguem até o destino. |
| *Many-to-one* | Uma transmissão única por *broadcast* configura rotas reversas em todos os dispositivos para o dispositivo que enviou o *broadcast*. |
| *Source routing* | Pacotes de dados, incluindo o caminho completo que o pacote deve atravessar da fonte até o destino. |

Dos protocolos do XBee Series 2 (Tabela 4), dois são semelhantes aos listados na Tabela 1. O protocolo AODV e o protocolo IRP, que é semelhante ao protocolo *Many-to-one* do XBee.

Faludi (2010) apresenta uma tabela comparativa com pros e contras destes protocolos na Tabela 5.

Tabela 5 – Comparação entre protocolos de roteamento segundo (FALUDI, 2010)

| Roteamento | Prós | Contras |
|---|---|---|
| *AODV* | Protocolo padrão, cria rotas automaticamente, funciona em qualquer tipo de topologia. | Baixa performance para redes grandes (com mais de 40 nós) devido ao *overhead* de *route requests* repetidos. |
| *Many-to-one* | Excelente desempenho para vários caminhos de entrada para um único local central. | Não é apropriado para mensagens originando-se de um local central ou redes com mensagens de nó para nó. Requer configuração personalizada. |
| *Source routing* | Excelente desempenho para mensagens de saída de um ou mais locais centrais, especialmente de dispositivos com melhor desempenho, como computadores. Bom para redes com mais de 40 nós. | Requer pré-configuração considerável, incluindo a programação de dispositivos especiais e o armazenamento de rotas *off-board*. As rotas devem ser adquiridas, armazenadas e recuperadas programaticamente para envio. |

Os tópicos seguintes apresentam os protocolos com mais detalhes segundo o manual do XBee.

#### 2.6.6.1 AODV (*Ad-hoc On Demand Distance Vector Routing*)

O roteamento padrão utiliza o protocolo AODV (*Ad-hoc On Demand Distance Vector Routing*) como algoritmo para descobrimento de rotas.

Segundo o manual, em grandes redes onde um XBee transmite dados para muitos módulos remotos, o roteamento AODV exigiria executar uma descoberta de rota para cada nó de destino para estabelecer uma rota. Se existirem mais destinos do que entradas de tabela de encaminhamento disponíveis, rotas estabelecidas seriam substituídas por novas rotas, fazendo com que as descobertas de rota ocorram mais regularmente. Isso pode resultar em atrasos maiores e diminuição do desempenho da rede.

#### 2.6.6.2 Roteamento Many-to-one

Um cenário comum em uma rede sem fio é quando a maioria dos nós precisam se comunicar com um único nó que executa alguma função centralizada. Este nó é muitas vezes referido como um coletor ou concentrador e neste trabalho utilizamos o nome sorvedouro. Se cada XBee neste tipo de rede tivesse que descobrir uma rota antes de enviar dados para o sorvedouro, a rede poderia facilmente ser inundada com mensagens de descoberta de rota de transmissão. O roteamento *Many-to-one* é uma otimização para esse cenário. Em vez de exigir que cada XBee realize sua própria descoberta de rota, o sorvedouro envia uma transmissão por *broadcast* de pacotes *Many-to-one* para estabelecer rotas reversas em todos os dispositivos.

Em uma única operação de descoberta de rota *Many-to-one*, a rota para o sorvedouro é estabelecida em todos os dispositivos: O sorvedouro transmite uma mensagem de solicitação de rota *Many-to-one* tendo ele mesmo como o endereço de destino.

Os módulos XBee que recebem esta solicitação armazenam uma entrada de tabela de roteamento *Many-to-one* reversa para criar um caminho de volta para o sorvedouro. A pilha ZigBee em um dispositivo usa informações de qualidade de link sobre cada vizinho para selecionar um vizinho confiável para a rota reversa. O pacote de configuração da rede *Many-to-one* deve ser enviado periodicamente para atualizar as rotas reversas na rede.

O roteamento *Many-to-one* também pode ser utilizado se houver mais de um sorvedouro na rede. Se mais de um sorvedouro enviar uma transmissão *many-to-one*, os módulos do XBee criam uma entrada de tabela de roteamento reverso para cada sorvedouro.

O uso de roteamento *many-to-one* com um nó sorvedouro baseado em XBee é configurado via alteração do parâmetro *Many-to-one Route Broadcast Timeout* (AR) no nó sorvedouro. O parâmetro AR define um intervalo de tempo (medido em unidades de 10 segundos) para enviar a transmissão *broadcast Many-to-one*.

Para uma rede com nós fixos, o AR só precisa ser emitido uma vez.

#### 2.6.6.3 Source routing

*Source routing* permite que o sorvedouro armazene e especifique rotas para muitos nós. O sorvedouro armazena a rota tomada por uma mensagem de um nó até o sorvedouro para que ela possa ser utilizada para indicar a rota de uma mensagem do sorvedouro para este determinado nó.

O sorvedouro deve enviar transmissões periódicas de solicitação de rota *Many-to-one*, de modo que são criadas rotas de um para um para o sorvedouro em todos os nós. Toda vez que um módulo XBee remoto transmite dados usando uma rota *Many-to-one*, ele primeiro envia uma transmissão de registro de rota. A transmissão de registro de rota é *unicast* ao longo da rota *many-to-one* até o sorvedouro. Como o registro de rota atravessa a rota *Many-to-one*, cada nó na rota anexa seu próprio endereço de 16 bits na carga útil do registro de rota. Quando o registro de rota atinge o sorvedouro, ele contém o endereço do remetente e o endereço de 16 bits de cada salto na rota. O sorvedouro pode armazenar as informações de roteamento e recuperá-las mais tarde para enviar um pacote *source route* para o módulo remoto.

Para usar o *source routing*, o sorvedouro deve funcionar no modo API. O sorvedouro deve enviar solicitação de rota *Many-to-one* periodicamente (o valor do parâmetro AR deve ser diferente de 0xFF). É preciso um microcontrolador ou um PC conectado ao sorvedouro para capturar e armazenar automaticamente as rotas para o resto dos nós da rede. Essas rotas são usadas ao transmitir uma mensagem para um nó sensor.

### 2.6.7 Fragmentação

Cada transmissão *unicast* pode suportar até 84 bytes de carga útil. Ativar a segurança ou usar o roteamento *source routing* pode reduzir esse número. No entanto, o *firmware* XBee ZB suporta um recurso ZigBee chamado fragmentação que permite que um único grande pacote de dados seja dividido em várias transmissões de RF e remontado pelo receptor.

A estrutura de transmissão da API pode incluir até 255 bytes de dados, que serão divididos em várias transmissões e remontados no lado receptor. Se uma ou mais mensagens fragmentadas não forem recebidas pelo dispositivo de recepção, o receptor irá desprezar toda a mensagem e o remetente indicará uma falha de transmissão na estrutura de API. O *firmware* ZB só pode suportar um pacote fragmentado de cada vez no nó de recepção, devido a restrições de memória. Aplicações que não desejam usar fragmentação devem evitar o envio de mais do que o número máximo de bytes em uma única transmissão.

### 2.6.8 Confirmações de recebimento ACKs

Segundo o XBee-ZB Manual, ZigBee inclui pacotes de confirmação de recebimento (*acknowledgment*) em ambas camadas de MAC e Aplicação (APS). Quando um dado é transmitido para um dispositivo remoto, este deve atravessar múltiplos saltos até alcançar o destino.

O Ack da camada MAC funciona da seguinte forma: à medida que os dados são transmitidos de um nó para o seu vizinho, um pacote de confirmação (Ack) é transmitido na direção oposta para indicar que a transmissão foi recebida com êxito. Se o Ack não for recebido, o dispositivo transmissor irá retransmitir os dados até 4 vezes.

O Ack da camada de aplicação APS do ZigBee funciona da seguinte forma: quando existe uma transmissão entre um nó sensor e um destino, o nó que originou a transmissão espera receber um pacote de confirmação do dispositivo de destino. Este Ack irá percorrer o mesmo caminho que os dados atravessaram, mas na direção oposta. Se o nó sensor não receber este Ack, ele retransmitirá os dados, no máximo 2 vezes até que o Ack seja recebido.

### 2.6.9 Configurações de *firmware*

Para utilizar o modo API, primeiramente o *firmware* deve ser configurado como um dispositivo que oferece suporte a API.

Em versões diferentes dos módulos podem haver tipos de imagens diferentes de *firmware*, mas existem configurações recomendadas pelo fabricante para garantir a compatibilidade entre as versões diferentes.

Segundo o manual de migração da Digi (DIGI, 2016), os módulos XBee/XBee-PRO ZB (S2/S2B) possuem seis imagens de *firmware* diferentes (*Coordinator AT, Coordinator API, Router AT, Router API, End Device AT e End Device API*). Os módulos XBee/XBee-PRO ZB (S2C) combinam estes seis conjuntos de funções em uma única imagem de *firmware*. A Tabela 6 mostra quais as configurações a serem ativadas nos módulos XBee / XBee-PRO ZB (S2C) para corresponder ao conjunto de funções dos módulos XBee / XBee-PRO ZB (S2 / S2B).

### 2.6.10 Modos de utilização de *sleep*

O rádio XBee possui o parâmetro de configuração SM, que especifica o modo de operação *Sleep Mode*, período em que o XBee permanece ocioso para reduzir o consumo de energia.

Segundo o manual do fabricante, os modos de *sleep* podem ser:

- *No Sleep*: não há modo de *sleep*, o rádio fica ligado o tempo inteiro. Indicado para

Tabela 6 – Tabela de compatibilidade entre modelos de XBee S2B e S2C

| XBee/XBee-PRO ZB (S2/S2B) | XBee/XBee-PRO ZB (S2C) |
|---|---|
| ZigBee Router AT | Configurações *default* |
| ZigBee Router API | AP (API Enable) = API enabled [1] |
| ZigBee Coordinator AT | CE (Coordinator Enable) = Enabled [1] |
| ZigBee Coordinator API | CE (Coordinator Enable) = Enabled [1]<br>AP (API Enable) = API enabled [1] |
| ZigBee End Device AT | SM (Sleep Mode) = Cyclic Sleep [4] |
| ZigBee End Device API | SM (Sleep Mode) = Cyclic Sleep [4]<br>AP (API Enable) = API enabled [1] |

coordenadores ou roteadores. Não é indicado para dispositivos que utilizam bateria, devido ao alto consumo de energia.

- *Pin Hibernate*: ativa e desativa o modo *sleep* de acordo com o estado do pino 9 (Sleep-RQ) Figura 16. Este modo é ativado setando o comando SM para 1. O rádio, só entra em estado *sleep* após todos os dados em seu *buffer* serem transmitidos.

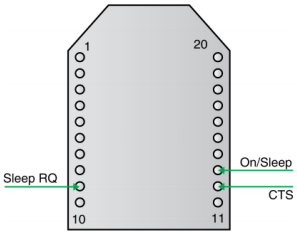



Figura 16 – Pinagem XBee

- *Cyclic Sleep*: ativa o período de *sleep* em ciclos definidos por parâmetros do *firmware*. O parâmetro SP (*Cyclic Sleep Period*) configura o período de *sleep* do módulo.
- *Cyclic Sleep Remote with Pin Wake-up*: equivalente ao *Cyclic Sleep*, entretanto, além do *sleep* cíclico, o modo *sleep* pode ser acionado pela ativação do pino 9.

Para a utilização dos períodos de *sleep* é necessário que o XBee esteja configurado como *End Device*. Os módulos configurados como roteadores devem ficar o tempo todo ativos para refazer a rota caso seja necessário ou para encaminhar dados recebidos por outros dispositivos.

### 2.6.11 Limitações do módulo XBee Series 2

Algumas limitações no XBee não permitiram realizar a ideia inicial do trabalho. O dispositivo XBee possui uma implementação proprietária, o que significa que há limitações para alterar algumas funções disponibilizadas pelo fabricante.

Não é possível implementar protocolos de roteamento customizados e embarcá-los na camada de rede do XBee. Por exemplo, não se pode implementar um protocolo de *clustering* na camada de rede do XBee, e qualquer implementação pode ser feita apenas na camada de aplicação, o que aumenta a complexidade e o processamento, tornando-se inviável em grande parte das aplicações. Outro problema é que os pacotes da camada MAC também não podem ser alterados, devido à implementação proprietária.

Uma outra limitação está relacionada à economia de energia. Os dispositivos configurados como coordenadores ou roteadores não podem ativar o modo *sleep*, por consequência, precisam ficar mais tempo ativos e por isso gastam mais energia com a transmissão dos dados dos demais sensores.

Além disso, há uma limitação no tamanho do pacote de dados, segundo Digi (2015), a carga máxima de RF por pacote de *unicast* é limitada a 66 bytes se a segurança estiver habilitada (EE) ou 84 caso contrário. O roteamento e fragmentação, se usados, podem reduzir ainda mais o tamanho máximo de carga útil de RF. Os pacotes de dados enviados por *broadcast* permitem 8 bytes a mais do que *unicast*.

# 3 Trabalhos relacionados

Além dos artigos usados como base para criar o referencial teórico sobre protocolos de roteamento em RSSF, outros trabalhos relacionados a monitoramento de animais foram analisados. Esta seção apresenta alguns dos trabalhos estudados.

## 3.1 RSSF para monitoramento de animais

Dos projetos analisados com RSSF para monitoramento bovino, observamos principalmente (GUO et al., 2006) e (GUO et al., 2009). O primeiro trabalho de (GUO et al., 2006) desenvolveu uma RSSF para compreender a movimentação e comportamentos dos bovinos no pasto, utilizando colares com sensores de GPS, acelerômetros e magnetômetros. O artigo apresenta a ideia do uso de colares com sensores FleckTM (GPS, acelerômetros e magnetômetros) para analisar o comportamento do animal, principalmente nas áreas de permanência e de trânsito do animal. No estudo, 6 animais foram analisados individualmente. Depois que os dados foram coletados, tiveram que ser agrupados para a criação de um modelo do comportamento dos animais. A coleta dos dados foi feita durante 4 dias, nos 2 primeiros dias, os dados coletados foram usados apenas para criar um modelo padrão do comportamento dos animais e definir as principais áreas de permanência e de trânsito do animal. O artigo de (GUO et al., 2009) apresenta uma pesquisa da CSIRO na Austrália que utiliza RSSF para monitoramento e controle pecuário. A ideia geral foi utilizar sensores nos animais para entender e classificar o comportamento de cada animal do estudo. O *hardware* utilizado nos nós é o Fleck2 com sensor GPS, 3 acelerômetros, 3 magnetômetros e sensor de temperatura. A pesquisa foi feita analisando 6 animais durante o período de 4 dias. Os dados eram gravados na memória interna do colar e enviados pela rede de sensores sem fio. Para validar os resultados das observações dos sensores, no dia 4 de maio 2006 foi feita uma observação humana que anotava os comportamentos dos animais em tabelas e uma observação em vídeos, com 15 gravações de movimentos dos animais. Essas observações foram usadas para comparação com os dados coletados pelos sensores.

O projeto ZebraNet (JUANG et al., 2002) apresenta uma pesquisa na área de RSSF para monitoramento de Zebras nas savanas africanas e visa entender alguns padrões de comportamento destes animais. Os sensores podiam detectar a posição pelo GPS e outros dados como taxa cardíaca e temperatura corporal. O texto também apresenta o roteamento em RSSF (*History-based protocol*) desenhado para a rede. Os colares eram compostos por sensores e um painel solar que recarregava as baterias durante o dia. O estudo foi realizado em um espaço de 40.000km$^2$ e por isso trata os desafios das distân-

cias entre os nós utilizando bases móveis em carros para receber os dados destes animais. Seguido por (LIU; MARTONOSI, 2003) que apresentou a criação de um *middleware* para fazer atualizações dos softwares dos nós da RSSF do projeto ZebraNet. A atualização é feita pela própria rede de sensores, levando em conta requisitos como desempenho, eficiência, confiança na transmissão dos dados. Basicamente dois tipos de roteamentos foram apresentados, *broadcast* e transmissão sob demanda. No mesmo projeto, (LIU CHRISTOPHER M. SADLER, 2004) é um artigo que apresenta as experiências de criação do *middleware* Impala no projeto ZebraNet, e como ele funciona, apresentando seu agendador de operações e a manipulação de eventos. Mostra também como a restrição de energia levou ao projeto das camadas de aplicação, *middleware* e hardware do programa ZebraNet. O Impala surge como uma camada para o projeto ZebraNet que faz a comunicação entre a camada de aplicações e o hardware. Esta solução foi desenhada para aprimorar a modularidade, simplicidade, adaptabilidade e reparabilidade das aplicações. Questões críticas aparecem ao implementar software para hardware. Essas questões incluem interfaces de *hardware/software*, agendamento de operação do sistema, manipulação de eventos, e o suporte de comunicação de rede. Este artigo discute essas questões no contexto das experiências de implementação do Impala, e descreve como o hardware impactou as escolhas no projeto. E Zhang et al. (2004) descreve o histórico de sensores utilizados no projeto ZebraNet e a criação de softwares de baixo nível para controlá-los. São discutidas técnicas de criação de fontes de alimentação eficientes para RSSF, os métodos para menor consumo de energia dos nós, e os métodos de gestão dos dispositivos periféricos, incluindo o rádio, memória flash, e sensores.

O projeto ZebraNet foi utilizado como referencial devido à completude da pesquisa.

## 3.2 RSSF para monitoramento Animal e Atuadores

O trabalho de (WARK et al., 2007) apresenta uma aplicação de rede de sensores e atuadores em uma criação de touros. O intuito dessa rede é coletar informações acerca do comportamento do touro e detectar os comportamentos de quando os animais apresentam intenção de luta. Os animais do estudo são touros de alto custo (aproximadamente U$$ 20.000) e a luta entre eles causa graves lesões que diminuem seu valor drasticamente. Quando o comportamento dos touros mostram uma tendência à briga, estímulos elétricos são disparados para evitar o confronto. Para que isso possa ser feito, os dados de posicionamento e a velocidade do animal devem ser transmitidos no tempo mais breve possível para aplicar o estímulo no momento certo. O hardware utilizado nos sensores é o Fleck. Este era inserido em uma caixa de plástico juntamente com duas sondas para a aplicação do estímulo elétrico.

## 3.3 Trabalhos com GPS

O intuito da pesquisa de Godsk e Kjærgaard (2011) era utilizar tecnologia de sensores GPS de baixo custo para localizar animais no rebanho que poderiam precisar de atenção ou cuidado. Os dados de posições de GPS do animal eram coletados durante o experimento, e posteriormente, processados por algoritmos que classificavam atividades específicas do animal. Os dados de posições do GPS foram coletados em 14 animais além da observação humana contínua durante esse período. Alguns comportamentos dos animais citados como normais no artigo são: caminhando, deitado, em pé e procurando comida. Neste trabalho, não existe uma RSSF, os dados são armazenados em cartões de memória e retirados manualmente ao fim do experimento.

## 3.4 Comportamento Animal

A intenção da proposta de (KILGOUR, 2012) foi obter informações a respeito do comportamento do gado no pasto baseando-se em estudos prévios de comportamento animal. Os dados foram comparados para classificar os comportamentos. O propósito deste projeto era observar o comportamento do gado de corte no pasto com o mínimo de intervenção humana. Este experimento teve foco comportamental e as observações foram apenas visuais. Neste estudo, observaram durante as horas do dia, o comportamento de seis rebanhos de gados de corte na Austrália com o mínimo de intervenção humana para determinar quais seriam os comportamentos dos animais sem intervenção humana.

O trabalho de Stricklin e Kautz-Scanavy (1984) também foi analisado para a compreensão de aspectos como mobilidade e comportamentos de liderança de bovinos e como essas características poderiam influenciar no trabalho realizado.

## 3.5 Detecção de doenças em animais com uso de sensores

Helwatkar, Riordan e Walsh (2014) classifica os tipos de sensores em animais em dois tipos:

- *Attached sensors*: sensores que podem estar montados na parte externa do corpo do boi ou no interior do corpo, como exemplo o bolus ruminal.
- *Nonattached sensors*: sensores nas imediações do animal, ou por onde o animal passa.

A proposta desta dissertação é do tipo *Attached sensors* pois está ligado diretamente ao animal pelo colar. A Embrapa Gado de Corte utiliza outros tipos de sensores em seus projetos, assim como o bolus.

Um exemplo de *Nonattached sensors* pode ser definido como a balança de passagem, que também é um projeto da Embrapa, e que coleta informações do peso do animal e envia ao pecuarista sempre que um animal passa pela balança.

Diversos sensores como: sensor de temperatura, acelerômetro, microfone, GPS, pressão podem ser utilizados na detecção de doença em animais. O texto de (HELWATKAR; RIORDAN; WALSH, 2014) também apresenta uma tabela com diversas doenças e os tipos de sensores que podem ser utilizados na detecção de cada uma delas. Entre as doenças que podem ser observadas por meio de sensores e foram citadas no texto estão: mastite, febre, cistos ovarianos, febre do leite, retenção de placenta, diarreia, pneumonia.

Estes trabalhos reforçam a importância de utilizar sensores para o monitoramento de animais, e assim tentar compreender melhor o comportamento bovino e até evitar problemas como doenças por meio da observação desses dados coletados.

## 3.6 Considerações Finais

Os trabalhos citados nesta seção, baseiam-se em observações humanas do comportamento animal ou fazem o uso de *hardwares* proprietários para a criação da RSSF, exemplo dos nós FleckTM, Fleck2. O que difere-se da proposta deste trabalho, que utiliza o Arduino FIO como placa para microcontrolador dos nós sensores. O que permite uma maior liberdade e facilidades na programação e na adição de novos sensores, por possuir pinos sobressalentes que podem ser utilizados em trabalhos futuros.

Os trabalhos de parceria entre Embrapa e FACOM/UFMS utilizados como base para este projeto foram apresentados na introdução deste trabalho, por isso não foram apresentados novamente neste capítulo.

# 4 Implementação da RSSF

Para cumprir os objetivos desse trabalho, e levando em consideração os estudos realizados na revisão bibliográfica, este capítulo apresenta a proposta para a criação de uma RSSF e continuação do projeto de monitoramento bovino iniciado por (JESUS, 2014), (OLIVEIRA, 2013) e (LOMBA, 2015).

## 4.1 Metodologia

As principais características deste projeto incluem:

- Aquisição de dados de posicionamento dos bois por meio de GPS;
- Pré-processamento dos dados: os dados são convertidos do padrão NMEA (*National Marine Electronics Association*) para posições de latitude e longitude antes da transmissão;
- Transmissão sem fio: os dados são transmitidos sem fio por meio de um ligação de rádio frequência;
- Pós-processamento: os dados são recebidos por meio de um nó denominado sorvedouro e apresentados na forma de posições em um mapa.

A metodologia desta pesquisa foi realizada da seguinte forma:

- Estudo de trabalhos relacionados à RSSF, trabalhos com sensores e monitoramento bovino;
- Desenvolvimento de algoritmos de roteamento em RSSF para simulações no Simulador Castalia;
- Compra de uma parte dos equipamentos da RSSF, o suficiente para montar 4 colares;
- Montagem de um nó sorvedouro para o recebimento dos dados da RSSF;
- Montagem de colares para primeiro experimento em campo na Embrapa Gado de Corte. Inicialmente realizado com apenas 4 colares devido a restrições orçamentárias. Neste primeiro experimento as coletas dos bois foram reservadas para serem utilizadas no simulador de RSSF. Um dos colares foi perdido no pasto.

- Desenvolvimento de um módulo de mobilidade no simulador Castalia, que recebe os dados reais de GPS das posições dos bois a partir dos dados coletados no primeiro experimento.
- Teste de cada colar e do consumo de cada equipamento para simulação no Castalia;
- Compra do restante dos equipamentos para o segundo experimento em campo;
- Montagem de todos os colares;
- Teste na rede com todos os colares montados;
- Realização da simulação com os dados coletados no experimento em campo;
- Configuração da rede com o protocolo escolhido.

## 4.2 Dispositivos Utilizados

- **Arduino FIO** (Figura 17): é uma placa microcontroladora baseada no ATmega328P e pode ser alimentado por uma conexão com bateria de polímero de lítio ou por cabo USB.

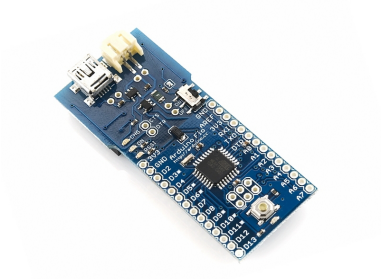


Figura 17 – Arduino FIO

O Arduino FIO que foi utilizado no primeiro colar, desenvolvido por (JESUS, 2014), apresenta uma série de vantagens que ajudaram na escolha, tais como ser um *hardware open-source*. Os pinos digitais adicionais podem ser utilizados para novos sensores em trabalhos futuros. O Arduino FIO já possui o encaixe para o módulo XBee, então não é necessário uma placa adicional para o uso dos rádios XBees que utilizamos neste trabalho.

Outra vantagem é que o Arduino FIO possui um circuito integrado chamado MAX1555, que permite a ligação de painéis solares diretamente aos pinos CHG, com uma tensão entre 3,7V e 7V, que poderá ser utilizado em projetos futuros sem a necessidade de equipamentos adicionais.

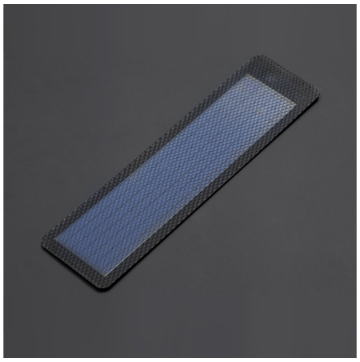


Figura 18 – Painel solar flexível

Devido ao alto consumo de energia do GPS, foram estudadas alternativas para aumentar a vida útil da bateria dos colares, foram encontrados alguns painéis solares como o da Figura 18 que poderiam ser costurados e ligados ao colar. Neste trabalho não foi feito o uso destes painéis devido ao custo.

- **Receptores de GPS Vênus da fabricante SparkFun e antena GPS com conector SMA**: segundo a fabricante, este receptor apresenta uma taxa de precisão de aproximadamente 2,5m. Além disso, o chipset possui um conector SMA, onde se pode ligar uma antena externa. Neste projeto, foi utilizada a antena presa ao colar dos animais. O modelo utilizado é da serie Venus638FLPx-L.

  Algumas informações adicionais do dispositivo segundo o manual do fabricante:

  - Tensão: 2,8V / 3,6V
  - Acurácia (velocidade): 0,1m/s
  - Acurácia (tempo): 300ns
  - *TTFF - Hot start* (a céu aberto): 1 segundo
  - *TTFF - Cold start* (a céu aberto): 29 segundos
  - Taxa de Update padrão: 1Hz
  - Protocolos NMEA-0183 V3.01, GGA, GLL, GSA, GSV, RMC, VTG (default GGA, GSA, GSV, RMC, VTG) SkyTraq Binary

  * *TTFF - Time to First Fix*, tempo necessário para o GPS adquirir a posição.

  Os dados do GPS utilizado nos colares são recebidos no padrão NMEA. Cada sentença recebida neste formato conta com um grande número de informações, como exemplo a seguir:

```
$LAGM,542,204,204,204,-16188,1527,2709,-1118,-691,-1071,208,-443,202
$GPGGA,124207.314,2026.6116,S,05443.3733,W,1,08,0.9,517.3,M,4.80,0.9,1.8*3A
$GPGSV,3,1,12,12,57,104,26,25,56,185,18,29,45,214,29,24,31,003,33*78
```

```
$GPGSV,3,2,12,05,30,085,38,21,29,318,43,02,23,142,41,26,17,219,36*72
$GPGSV,3,3,12,20,15,119,34,31,12,222,41,14,03,282,14,15,00,021,*78
$GPRMC,124207.314,A,2026.6116,S,05443.3733,W,000.0,027.5,060415,,,A*67
$GPVTG,027.5,T,,M,000.0,N,000.0,K,A*0D
```

Dos dados em NMEA são necessárias apenas as informações de latitude e longitude para montar o arquivo que será recebido no Castalia e apresentar a localização no mapa. Foi feito um filtro destes dados neste trabalho para reduzir o consumo na transmissão e gravação em cartão de memória.

O consumo do equipamento no modo *tracking*, segundo o manual do fabricante do GPS (SKYTRAQ TECHNOLOGY, INC., 2011) é de aproximadamente 29mA. No entanto, nos experimentos, foi feita a medição com o multímetro, e foi obtido um consumo médio aproximado de 82mA.

Para o funcionamento do GPS faz-se necessário o uso de uma antena com conector SMA (Figura 19). Segundo o manual do fabricante da antena, o consumo é de aproximadamente 12mA.

Figura 19 – GPS Vênus Sparkfun + Antena SMA

- **OpenLOG Sparkfun com cartão de memória:** o OpenLog (Figura 20) foi utilizado no projeto para gravar os dados coletados pelo GPS antes de enviá-los pela RSSF. Além disso, os dados ficam gravados no cartão para validação dos dados recebidos pela RSSF. O cartão pode ser utilizado também em projetos futuros, para informações que não precisam ser enviadas na RSSF. O *firmware* deste dispositivo é de código aberto e fácil alteração. A versão que foi utilizada deste *hardware* suporta FAT16, FAT32 e cartões microSD de até 16 GB. Neste projeto foram utilizados alguns cartões de 4GB e 8GB, espaço suficiente para o armazenamento dos dados coletados durante o período de duração da bateria.

Figura 20 – OpenLog Sparkfun

- **XBee PRO Series 2**

  O modelo de rádio utilizado no trabalho é o XBee-PRO Series 2, que utiliza o protocolo ZigBee, e diferente do modelo XBee Serie 1, já implementa a criação de redes *mesh*. O modelo de XBee Series 2 veio para substituir os dispositivos Serie 1, que utilizavam estritamente o protocolo IEEE 802.15.4.

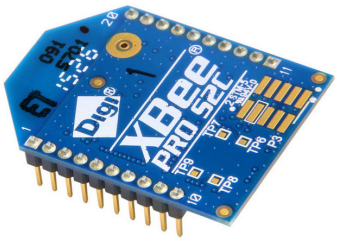



Figura 21 – XBee PRO Series 2

No trabalho, foram utilizadas duas variações de XBee da Serie 2. O XBee Pro S2B e o XBee Pro S2C (Figura 21).

O XBee Serie 2B, foi o primeiro a ser comprado e utilizado no primeiro experimento em campo. Na segunda compra, o modelo Series 2B já havia sido descontinuado, sendo substituído pelo XBee Series 2C. De acordo com o guia de migração do fabricante (DIGI, 2016), as configurações entre os dois modelos são semelhantes, com diferença no consumo de energia e alcance de envio dos dados. O XBee S2C tem um maior alcance de transmissão e um menor consumo de energia. Por isso, as diferenças entre os dois tipos foram consideradas nas simulações no Castalia.

- **Raspberry PI 2 Modelo B**

  O Raspberry PI 2 Modelo B é um computador pequeno e de baixo custo que pode ser alimentado por uma fonte de 5V por meio de um conector micro-USB. O modelo utilizado neste projeto, possui 1GB de memória RAM, uma porta Ethernet de 100mb e quatro portas USB, além disso, fornece um conector HDMI que pode ser utilizado para conectar a um monitor ou televisão.

  A escolha do Raspberry foi devido à facilidade de uso deste dispositivo; pelo baixo custo; e pelo baixo consumo de energia, sendo necessária uma fonte de 5V para a alimentação de energia. Além disso o Raspberry escolhido tem processamento de 900MHz e permitiu embarcar uma distribuição de um Sistema Operacional, que facilitou o tratamento das mensagens recebidas.

Várias distribuições de Sistemas Operacionais foram desenvolvidas para o Raspberry PI, tais como: Archlinux ARM (ARCH, 2015), OpenELEC (OPENELEC, 2015), RaspBMC (RASPBMC, 2015), Raspbian (RASPBIAN, 2015) e, mais recentemente, o Windows 10 IoT Core Dashboard.

A distribuição que foi utilizada neste trabalho é a Raspbian, que é uma distribuição oficial do Raspberry, derivada do Linux Debian e que pode ser instalada em um cartão de memória de 4GB ou superior. Esta distribuição foi desenvolvida de forma otimizada para uso no Raspberry. Nesse projeto está sendo utilizado um cartão de memória de 16GB para armazenar o Sistema Operacional Raspbian e os dados recebidos pela rede.

Em uma das portas USB foi ligado o adaptador para o XBee, responsável por receber os dados da rede. Nele, colocamos o XBee coordenador da rede, com conexão RPSMA ligado a uma antena externa.

## 4.3 Nós sensores e nó sorvedouro

### 4.3.1 Nó sorvedouro

O nó sorvedouro (Figura 22) é o nó coordenador da rede, responsável por receber e tratar as informações enviadas pelos sensores. Este nó foi desenvolvido utilizando um XBee RPSMA (no modo coordenador) acoplado a um Raspberry PI 2 (Modelo B). Por ser o coordenador da rede, este nó deve ficar ligado todo o período para receber os dados enviados pelos colares. Por isso, foi descartado o uso da bateria para este nó e assumimos que este estará ligado diretamente a uma fonte de energia no Mangueiro Digital da Embrapa. A fonte utilizada provém de uma tomada de energia e converte a energia de 110V ou 220V para os 5V necessários para a ligação. Para a utilização em áreas que não possuam fonte de energia elétrica poderão ser utilizados pequenos painéis solares conectados ao Raspberry.

Em uma das portas USB do Raspberry ligou-se o adaptador para o XBee RPSMA. Para conectar-se à rede da Embrapa foi utilizado um cabo Ethernet nos experimentos. O uso de adaptadores USB para internet sem fio pode ser utilizado em trabalhos futuros. No entanto, optou-se por utilizar a rede de dados da Embrapa nos experimentos realizados.

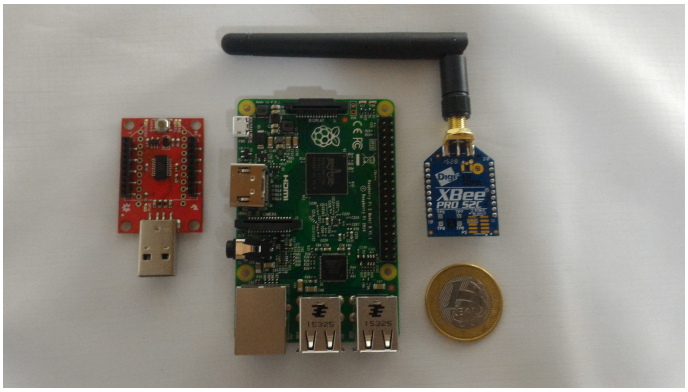



Figura 22 – Nó sorvedouro: Raspberry PI e módulo XBee

Neste projeto, o Raspberry foi programado para receber todas as mensagens do XBee em uma API desenvolvida em Java e disponibilizada pela fabricante dos módulos XBee (DIGI, 2016).

A Tabela 7 apresenta os parâmetros utilizados na configuração *firmware* do XBee do nó sorvedouro.

Tabela 7 – Configuração do *firmware* do nó sorvedouro

| ID | PAN ID | 999 |
|---|---|---|
| SC | Scan Channel | 7FFF |
| ZS | ZigBee Stack Profile | 2 |
| JV | Channel Notification | 1 |
| DH | Destination address high | 0 |
| DL | Destination address low | FFFF (representa a opção de Broadcast) |
| NI | Node identifier | COORDENADOR |
| AR | Many-to-One Route Broadcast Time | 1E (Default FF) |

O parâmetro AR representa o tempo em segundos de envio do pacote de manutenção do roteamento da rede *Many-to-one*. Nesse caso, escolheu-se o valor em hexadecimal 1E (300 em decimais), este valor é multiplicado por 10, retornando 3000 segundos, ou 5 minutos de intervalo a cada envio, o mesmo valor utilizado pela coleta dos dados do GPS. Este parâmetro pode ser facilmente alterado já que deve ser configurado apenas no nó coordenador da rede.

A Figura 23 exemplifica a estrutura da rede montada.

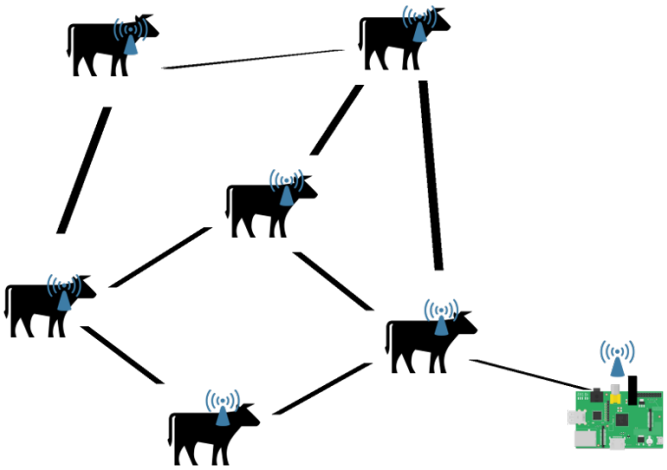


Figura 23 – Estrutura da RSSF

No sorvedouro foi criada uma aplicação que recebe as mensagens e faz o devido tratamento. Podendo inserir em um banco de dados, gravar um arquivo CSV ou enviar as informações por POST. A aplicação que recebe os dados no sorvedouro foi desenvolvida utilizando a API XBEE em JAVA. A cada posição recebida pelo sorvedouro, a aplicação em Java cria um novo registro no banco de dados com as informações recebidas. Um banco de dados em PostgreSQL foi criado no nó sorvedouro para o registro dessas posições. No entanto, um outro banco de dados na mesma rede pode ser configurado.

Foi criado um arquivo rssf.properties para configuração do nó sorvedouro, apresentado no Apêndice A.

No nó sorvedouro foi disponibilizada um servidor WEB com uma aplicação com algumas telas com os dados recebidos. Ao adicionar o sorvedouro a uma rede, basta acessar o IP designado a ele por um navegador de internet.

Para monitorar o recebimento das mensagens, foi desenvolvida uma interface web com o mapa do local das coletas e a cada recebimento de um pacote pelo sorvedouro o mapa era atualizado com as posições dos bois. A primeira tela da interface apresenta a posição de cada boi por um ponto no mapa do Google Maps, assim como apresentado na Figura 24.

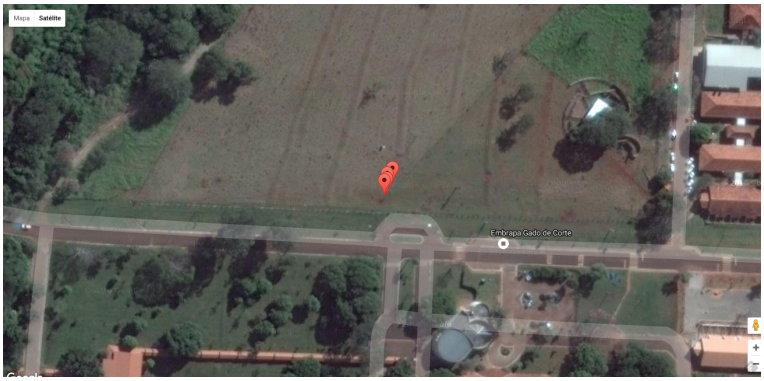



Figura 24 – Sistema Web - Mapa com posição mais atual dos bois no pasto.

Além do mapa, há uma outra tela com a lista das últimas posições recebidas nos últimos 30 minutos, mostrado na Figura 25.

Dados recebidos nos últimos 30 minutos

| Id | Timestamp do sorvedouro | Colar | Lat. | Long. | Altitude | Data | Hora |
|---|---|---|---|---|---|---|---|
| 2660 | 2015-09-15 15:34:28.026737 | 4 | -20.443228 | -54.722546 | 544.90002 | 150915 | 19:34:27 |
| 2659 | 2015-09-15 15:34:22.304944 | 2 | -20.442911 | -54.722569 | 529.09998 | 150915 | 19:30:57 |
| 2658 | 2015-09-15 15:34:20.481112 | 1 | -20.439884 | -54.725243 | 514.40002 | 150915 | 18:44:15 |

Figura 25 – Sistema Web - Exmeplo de lista de mensagens recebidas nos últimos 30 minutos

A última tela (Figura 26), apresenta o caminho percorrido por cada animal, sendo possível filtrar os dados apresentados pelos colares e período desejado.

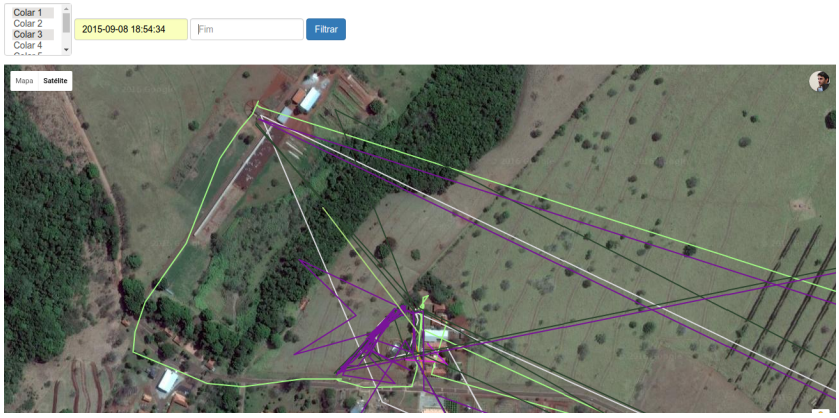



Figura 26 – Sistema Web - Tela de caminho percorrido pelos bois.

Um *Web Service* é uma solução utilizada na integração de sistemas e na comunicação entre aplicações distintas. Permite que mesmo aplicações desenvolvidas em plataformas diferentes se tornem compatíveis por meio de um sistema padronizado de comunicação. Tradicionalmente, aplicações desenvolvidas fazendo uso de *Web Services* interagem

por meio de mensagens SOAP que é um protocolo para troca de informações em plataformas distribuídas (NUNES; DAVID, 2005).

Há uma opção de envio de cada posição recebida para um *WebService* por REST JSON, como foi realizado durante experimento na Embrapa. Esta opção foi implementada pois há a ideia para trabalhos futuros que utilizem WebServices para integrar dados de outros projetos na Embrapa.

Para testar esta funcionalidade, no primeiro experimento uma versão deste sistema ficou hospedada em um servidor da Embrapa, e recebia as posições por REST do sorvedouro, sempre que uma nova posição era recebida. Assim era possível acessar o mapa ou lista de últimas posições à partir dos endereços da internet:

- Listagem das últimas posições recebidas:

  http://production.cnpgc.embrapa.br/rssf/list.php

- Mapa das últimas posições recebidas:

  http://production.cnpgc.embrapa.br/rssf/map.php

### 4.3.2 Nós sensores

Os nós sensores utilizados no trabalho são compostos pelos seguintes itens enumerados como na Figura 27:

1. XBee Pro Series 2, modelos S2B e S2C;
2. Bateria LiPo 2200mAh ou 6000mAh;
3. Arduino FIO, atuando como microcontrolador do nó sensor, criando os pacotes de dados e controlando o tempo de envio de cada pacote na rede;
4. OpenLog SparkFun e Cartão de memória, para armazenar os dados do GPS antes de enviá-los na rede;
5. GPS Vênus SparkFun; e
6. Antena para GPS com conector SMA.

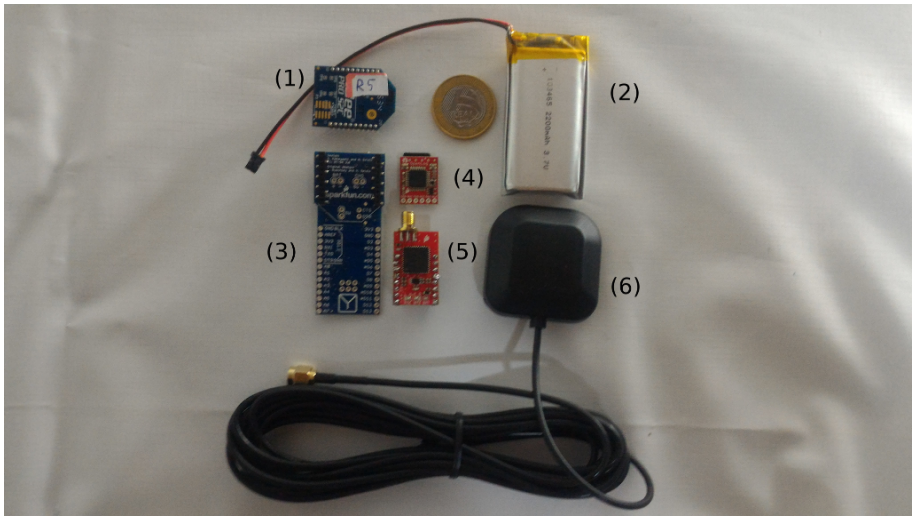



Figura 27 – Equipamentos do colar com moeda para escala

Ao colar desenvolvido por (JESUS, 2014) foram adicionados os rádios XBee modelo PRO Series 2.

Uma representação da organização dos componentes pode ser visualizada no esquemático do colar, apresentado na Figura 28.

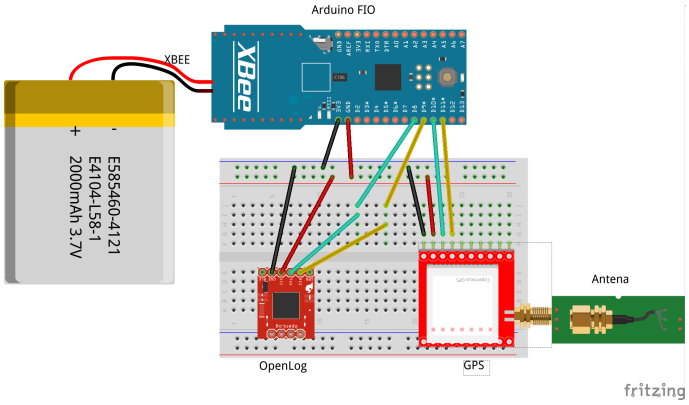



Figura 28 – Esquemático do colar

O nó sensor (colar), foi planejado para possibilitar a inclusão de novos sensores, por ser composto pelo Arduino FIO. No entanto, para este trabalho inicial, foi utilizado apenas o GPS por questões orçamentárias como o colar base. Reduzindo-se o número de sensores, foi possível comprar uma maior quantidade de colares para os testes da rede. Em trabalhos futuros, sensores adicionais poderão ser incluídos nas portas sobressalentes do Arduino FIO e os dados poderão ser enviados pela RSSF utilizando o mesmo protocolo de roteamento utilizado neste trabalho.

A Figura 29 apresenta os equipamentos de um nó antes de ser preso ao colar.

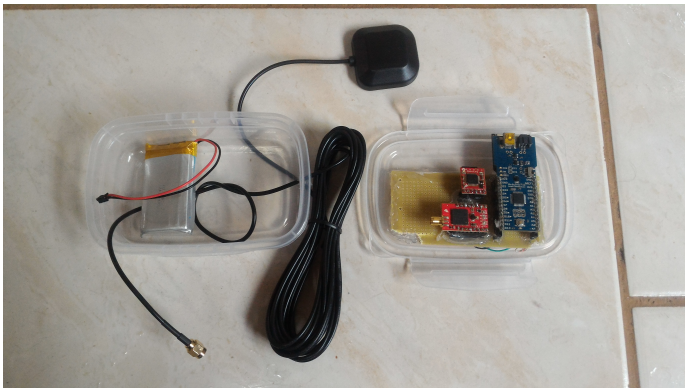


Figura 29 – Equipamentos do nó antes de prender ao colar

Todos os colares foram configurados como roteadores da rede com destino ao nó sorvedouro.

O *firmware* de cada dispositivo roteador, utilizado nos colares, foi configurado com os seguintes parâmetros no XCTU, conforme apresentados na Tabela 8.

Tabela 8 – Configuração do *firmware* nós sensores

| ID | PAN ID | 999 |
|---|---|---|
| SC | Scan Channel | 7FFF |
| ZS | ZigBee Stack Profile | 2 |
| JN | Join Notification | 1 |
| JV | Channel Notification | 1 |
| DH | Destination address high | 0 |
| DL | Destination address low | 0 |
| NI | Node identifier | RouterX<br>X = (ID do Colar) |

Observa-se que o endereço de destino está setado como 0 nos campos DH e DL ao invés do endereço físico do nó. Configurar o destino com o valor 0 é uma forma de indicar que o envio dos dados é sempre para o coordenador da rede, que neste projeto é o sorvedouro. Dessa forma, se for preciso fazer uma troca deste, não é preciso trocar o endereço de destino de todos os nós da rede, basta substituir o XBee do nó sorvedouro com a configuração de coordenador da rede.

Para configurar todos os módulos de XBee na mesma rede utilizou-se o número 999, escolhido arbitrariamente, para identificar a rede (PANID). Este número foi aplicado em todos os nós sensores para que pudessem fazer parte da mesma rede e trocar mensagens entre si. Cada novo nó a ser inserido na rede deve obrigatoriamente utilizar este mesmo identificador PANID.

No colar de (JESUS, 2014) havia a gravação de todas as informações recebidas pelo GPS no padrão NMEA, tais como: número de satélites, posições, datas, entre outras contidas no formato NMEA. Como primeira alternativa para reduzir o consumo de energia na transmissão dos dados, foi realizado um pré-processamento dos dados recebidos no formato NMEA antes de enviar para a RSSF. Ao invés de enviar pela RSSF toda a informação recebida no formato NMEA, eviou-se apenas os dados necessários para a identificação e posição de cada nó sensor, além da data e hora recebidas pelo GPS, para comparações futuras.

Uma biblioteca para Arduino chamada TinyGPS++ (HART, 2014) foi utilizada para este processamento. Além de reduzir o número de informações recebidas pelo NMEA, com o uso desta biblioteca pode-se filtrar os dados de posição e convertê-los facilmente para posições de latitude e longitude, o que torna útil para utilizar em APIs de mapas, como o GoogleMaps, utilizado nos experimentos para confirmar a posição correta dos animais. Se algum dado adicional do formato NMEA precisar ser adicionado ao pacote de dados em trabalhos futuros, o desenvolvedor poderá fazer o uso dessa mesma biblioteca para incluir novas informações do padrão NMEA.

Com o uso da biblioteca, cria-se uma sequência de caracteres com os dados separados por vírgula e cria-se um novo pacote de dados API com esta sequência para envio na rede. Desta forma, dados adicionais podem ser incluídos em trabalhos futuros.

Para cada pacote de dados enviado pela rede, a sequência contava com os seguintes dados, por exemplo:

**idcolar; latitude; longitude; altitude; data; hora**

**1,-20.443878,-54.723942,521.40002,17092015,08:23:12**

Utilizou-se a API 0x10 para envio dos pacotes de dados a partir dos colares e a API 0x90 para o recebimento dos pacotes no nó sorvedouro. A Tabela 9 apresenta um exemplo de pacote da API 0x10 enviado por um colar.

Tabela 9 – Pacote API 0x10 - Para envio de dados

| Nome do campo | Valor HEX | Valor |
|---|---|---|
| Start Delimiter | 7E | |
| Length | 00 41 | |
| Frame type | 10 | |
| Frame ID | 01 | |
| 64-bit dest address | 00 00 00 00 00 00 00 00 | |
| 16-bit dest address | 00 00 | |
| Broadcast radius | 00 | |
| Options | 01 (padrão 00) | 0x01 - *Ack* desativado |
| RF data | 31 2C 2D 32 30 2E 34 34 33 38 37 38 2C 2D 35 34 2E 37 32 33 39 34 32 2C 35 32 31 2E 34 30 30 30 32 2C 31 37 30 39 32 30 31 35 2C 30 38 3A 32 33 3A 31 32 | 1,-20.443878,-54.723942, 521.40002,17092015,08:23:12 |
| Checksum | E9 | |

As configurações apresentadas na Tabela 9 representam um exemplo de envio de um pacote de dados com a seguinte sequência de caracteres hexadecimais: **7E 00 41 10 01 00 00 00 00 00 00 00 00 00 00 00 01 31 2C 2D 32 30 2E 34 34 33 38 37 38 2C 2D 35 34 2E 37 32 33 39 34 32 2C 35 32 31 2E 34 30 30 30 32 2C 31 37 30 39 32 30 31 35 2C 30 38 3A 32 33 3A 31 32 E9**.

O XBee permite fazer o envio de um pacote com ou sem confirmação de envio (ACK). Para reduzir o envio de pacotes na rede e por consequência o consumo de energia, a opção de confirmação de recebimento de pacotes (ACK) no campo *options* foi desativada, conforme mostrado no exemplo da Tabela 9.

Para a criação e envio dos pacotes do XBee utilizou-se a biblioteca para Arduino chamada XBee (RAPP, 2016) que facilita a transmissão de dados em formato API pelo Arduino, passando apenas a sequência de caracteres criada, a biblioteca se encarrega de encapsular o pacote e fazer o envio.

A Figura 30 representa o processo desde a conversão de NMEA para posições em latitude e longitude pelo Arduino, criação do pacote na API 0x10 e envio pela rede, entre os colares até o nó sorvedouro, que o recebe e apresenta a localização em um mapa.

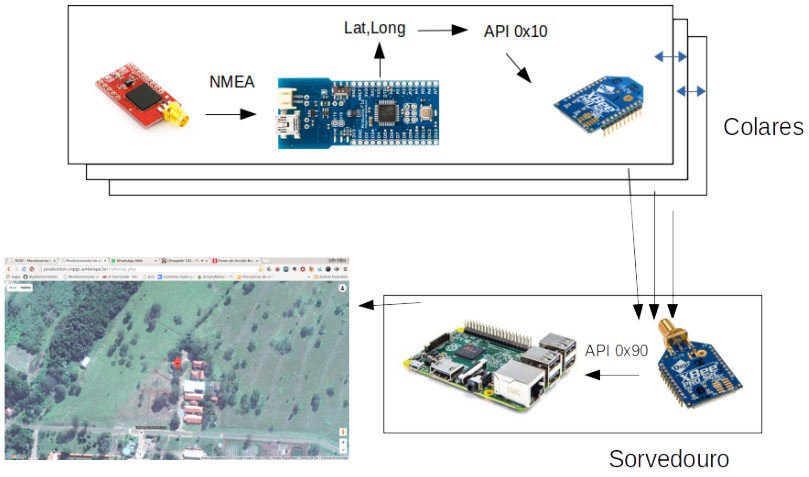



Figura 30 – Processo de envio de pacotes na RSSF

Nos experimentos, o ciclo de coleta de cada posição do GPS foi configurado a cada cinco minutos, e este intervalo foi escolhido utilizando trabalhos de monitoramento animal como o de (MEZZALIRA et al., 2011), que sugere que a escala de observação de cinco minutos permitiu detectar as alterações diárias que compõem a dinâmica das refeições no comportamento ingestivo de animais em pastejo.

Apesar de ter sido utilizado o período de 5 minutos a cada coleta, é possível alterar este período conforme as necessidades do pesquisador. Em trabalhos futuros estas configurações poderão ser alteradas.

## 4.4 Configurações da RSSF

Uma RSSF que possui nós fixos é por vezes mais fácil de implementar do que uma RSSF com nós móveis, visto que em redes com nós móveis, a cada movimentação de um nó, a conexão pode ser interrompida devido ao afastamento. Isso faz com que os nós tenham que verificar constantemente a conexão e os caminhos para a transmissão de um pacote.

Na RSSF proposta, todos os nós são móveis, pois estão alocados nos colares dos bois, exceto o sorvedouro, que permanece fixo para receber os dados da rede. Por este motivo, os sensores dos colares devem ficar o tempo todo se comunicando para não perderem a conectividade entre os demais colares e alcance ao nó sorvedouro.

Para que os colares fiquem sempre se comunicando sem haver falhas, todos os XBees devem ficar "ativos" o tempo todo, ou seja, não podem entrar no modo *sleep*, já que este modo não permite a retransmissão de dados entre os nós.

Ou seja, para um nó servir de intermediário entre um outro nó mais afastado, este

não pode estar no modo *sleep*. Por isso, teve-se que configurar todos os nós dos colares no modo roteador (*router*), que aumentou bastante o consumo da bateria de cada colar. Essa configuração permite que todos os nós possam se movimentar livremente.

A maior desvantagem da configuração dos colares como roteadores é o aumento do consumo de bateria, já que o nó roteador não suporta o modo *sleep*, o que poderia reduzir o consumo de energia. Esse fato aumenta consideravelmente o consumo da bateria dos nós sensores, já que cada nó precisa ficar o tempo todo "ativo" para que possa receber e retransmitir as informações adiante, até chegarem ao seu destino. No entanto, essa configuração garante uma maior escalabilidade na rede.

Nós configurados como *end devices* devem estar ligados diretamente a um roteador ou coordenador e não permitiriam a redefinição de rotas, tornando-se inviável neste projeto. Nesta rede, apenas o nó sorvedouro foi configurado como nó coordenador da rede (*coordinator*), pois este é o nó responsável por receber todas as informações dos demais nós.

### 4.4.1 Hibernação de recursos - Alternativas para viabilizar o modo *sleep*

Conforme apresentado anteriormente neste trabalho, a ideia deste trabalho teve foco na comparação de roteamentos para RSSF, sem o uso de infraestrutura adicional, ou seja, os colares deveriam comunicar entre si e reencaminhar os pacotes até o sorvedouro utilizando-se do melhor protocolo para o cenário estudado.

Devido a mobilidade dos bois e a tendência da mobilidade em grupos, esta é uma tarefa complicada, pois independente do protocolo escolhido pode haver a quebra de enlace por conta da distância entre os bois e o sorvedouro.

Além deste trabalho, outros trabalhos com sensores estão sendo desenvolvidos para a Embrapa Gado de Corte, e alguns poderão utilizar sensores para monitoramento do ambiente. Estes sensores, se forem fixos no pasto, poderiam servir como roteadores da rede. Dessa forma, os colares poderiam ser configurados como *End devices* podendo assim entrar em modo *sleep*, e pela utilização do mesmo protocolo de roteamento, mas reduziria o consumo de bateria dos colares. Existem diversas ideias de projetos na Embrapa que poderão fazer uso de sensores para coletar dados de animais e do ambiente.

Uma possível solução para que os colares pudessem entrar no período de *sleep* e assim enviar pacotes de dados apenas periodicamente, seria a implantação de nós adicionais no pasto com uma fonte constante de energia, como painéis solares, para servir como roteadores, enquanto os nós dos colares seriam dispositivos finais com possibilidade de *sleep*.

Tendo em vista as limitações de *sleep* do XBee, pensou-se em outras estratégias para reduzir o consumo de energia dos colares. Uma delas foi utilizar o modo *sleep* na

configuração *Pin Hibernate*. Dessa forma, a cada intervalo de tempo, o modo *sleep* seria ativado por meio do microcontrolador Arduino no pino *sleep* do XBee.

Os sensores de GPS podem captar diversas informações, dentre elas, o horário atual, baseado em satélites. Segundo o manual do GPS, essa informação de tempo tem uma precisão de poucos milissegundos. Com isso pensou-se na ideia de reduzir o período em que os XBEEs ficam ativos e acordar os dispositivos apenas durante o período de transmissão, sincronizando os colares pelo tempo do GPS para a configuração dos períodos de hibernação (*sleep*) dos nós.

Infelizmente, o modo *Pin Hibernate* também não pode ser utilizado no XBee configurado como roteador, sendo utilizado apenas em dispositivos configurados como *End Devices*. As opções de *sleep* cíclico do XBee ou *sleep* por *Pin Hibernate* poderão ser exploradas em trabalhos futuros se alguma infraestrutura adicional for implementada no pasto, para reduzir boa parte do consumo de bateria.

# 5 Experimentos

Este capítulo está dividido em seções com as atividades realizadas e demais informações da proposta, começando com a seção 5.1 que apresenta a parte de consumo de energia dos equipamentos e como essas informações foram utilizadas na simulação e experimentos.

Em seguida apresentamos a área disponibilizada pela Embrapa na seção 5.2. Por fim, a seção 5.3 apresenta os experimentos realizados.

## 5.1 Consumo de energia dos nós sensores

A preocupação com o consumo de energia dos nós é algo importante em uma RSSF. Neste projeto foi utilizado o GPS, que é um dispositivo que consome boa parte da bateria comparado ao consumo total do colar. Por isso, pensou-se em estratégias para reduzir parte do consumo do colar.

Além da simulação da RSSF por meio do simulador, que fornece o consumo dos nós, coletamos informações do consumo de energia dos equipamentos com base nos manuais do fabricante (*datasheet*) e comparamos com o consumo obtido por meio de multímetro. Dessa forma, utilizou-se essas informações para estimativas de tempo de vida da bateria e simulações do consumo de energia do colar.

A Figura 31 apresenta em cada linha, o consumo de energia de alguns equipamentos do colar. Não foi encontrado nos manuais do fabricante do Arduino FIO o consumo de energia, por isso coletamos o consumo manualmente por meio de um multímetro. O *datasheet* do GPS Venus SparkFun informa que o consumo do equipamento é de 29mA, no entanto, medindo em um multímetro obteve-se o consumo de aproximadamente 82mA. O consumo dos demais equipamentos foi retirado do *datasheet* dos fabricantes.

Os modelos do XBee utilizados neste trabalho são da Série 2, entre eles o XBee S2B e o XBee S2C. A mudança dos tipos ocorreu porque o recurso para a compra chegou em dois períodos diferentes, e no segundo momento não se encontrou componentes do mesmo modelo comprado anteriormente. Apesar disso, os dois modelos são compatíveis, porém, com diferença no consumo e alcance da transmissão, segundo (DIGI, 2016).

Os dados de consumo de energia de cada equipamento foram utilizados para a realização da estimativa de consumo do colar e para as simulações em Castalia.

Enquanto alguns sensores, como o sensor de luminosidade, acelerômetro e temperatura, utilizados no trabalho de (LOMBA, 2015) possuem um consumo reduzido de

energia, o dispositivo de GPS exige um consumo maior comparado aos sensores citados anteriormente.

Conforme dito anteriormente, todos os colares tiveram que ser configurados como roteadores na rede para formar a topologia em malha, e por isso, não puderam entrar no modo *sleep*, permanecendo em sua maior parte do tempo no modo ativo, e periodicamente, nos modos recebendo ou enviando.

Utilizou-se o consumo de energia do colar no modo ativo para fazer uma estimativa simples do tempo de vida de cada colar alimentado a partir da bateria.

Uma estimativa simples do tempo de vida de um nó sensor foi realizada utilizando-se os cálculos aproximados do consumo de energia de cada dispositivo fornecido apresentados na Figura 31. Utilizamos as baterias dos colares, de 2200mAh e 6000mAh como fonte de energia para a estimativa.

Para estes cálculos, seguimos o tutorial da SparkFun (DEE, 2016), fabricante da maioria dos equipamentos utilizados no colar, que sugere a equação que segue:

$$\frac{BatteryCapacity(AmpHours)}{CurrentDraw(Amps)} = BatteryLife(Hours) \tag{5.1}$$

A Figura 31 apresenta uma tabela com os colares com os módulos S2B Pro e S2C Pro com o cálculo de estimativa de vida útil para cada tipo de bateria respectiva. As informações de consumo foram obtidas dos respectivos manuais de usuário (DIGI, 2015) e (DIGI, 2016b).

| Consumo dos equipamntos do colar (sem Xbee) – em mA | Consumo mA |
|---|---|
| Arduino FIO | 4 |
| GPS Venus Sparkfun | 82 |
| Open Log | 2 |
| Antena GPS 3V SMA | 12 |
| Total | 100 |

(Tabela 1)

| Consumo do Xbee (Sem demais equipamentos) – em mA | Xbee S2B Pro | Xbee S2C Pro |
|---|---|---|
| Transmitindo | 205 | 120 |
| Recebendo | 47 | 31 |
| Sleep | 0,0035 | 0,001 |

(Tabela 2)

| Consumo colar (Equipamentos + Xbee) em mA | Xbee S2B Pro | Xbee S2C Pro |
|---|---|---|
| Transmitindo | 305 | 220 |
| Recebendo | 147 | 131 |
| Sleep | 100,0035 | 100,001 |

(Tabela 3)

| Equipamentos + Xbee | Xbee S2B Pro | Xbee S2C Pro |
|---|---|---|
| Consumo Recebendo (mA) | 147 | 131 |
| Bateria (mAh) | 6000 | 2200 |
| Estimativa de vida útil do colar em horas | 40,8163265306 | 16,7938931298 |

(Tabela 4)

Figura 31 – Consumo de equipamentos e estimativa de vida útil dos colares no modo recebendo

Para este exemplo utilizou-se o consumo de cada equipamento do circuito no modo padrão. Ou seja, no caso do GPS, utilizou-se o consumo no modo track, que ocorre após a aquisição das conexões com os satélites, e no caso do XBee, utilizou-se o modo ocioso (*idle*), em que o XBee estaria apenas ligado, sem receber ou enviar pacotes.

Segundo o manual do fabricante do OpenLog, o consumo constante de energia é 2mA, aumentando para 6mA durante a gravação de dados, para facilitar a estimativa, como a gravação é feita periodicamente a cada 5 minutos, utilizou-se o valor constante de 2mA nesses primeiros cálculos. O consumo do OpenLog no modo gravação é explorado melhor nas simulações em Castalia.

Essas primeiras estimativas foram estimativas simples, que são utilizadas como *baseline* de consumo do circuito nas simulações. Temos que levar em conta o consumo variável do XBee, nos diferentes modos de operação, aumentando drasticamente o consumo no modo *transmit*, por exemplo. Utilizando como base esse consumo no simulador, pode-se notar a diferença dos consumos de energia apenas entre os protocolos diferentes, que poderão ter consumos variados ao transmitir devido ao número de pacotes enviados.

A preocupação com o consumo de energia foi direcionada apenas aos colares, carregados por bateria. Não houve a mesma preocupação com nó sorvedouro, pois assumiu-se

que este deve ser carregado constantemente por uma fonte ligada à rede elétrica do Mangueiro Digital da Embrapa.

## 5.2 Área disponível para os experimentos em campo

A Embrapa Gado de Corte disponibiliza uma área de aproximadamente 78,169 ha denominada Mangueiro Digital, que está dividida nas três subáreas conectadas e apresentadas na Figura 32:

- **Área 1** com aproximadamente 31,320 ha, com seus lados medindo, 627 m, 121 m, 290 m, 628 m, 102 m, 120 m, 204 m, 81 m, 122 m e 107 m.
- **Área 2** com aproximadamente 23,518 ha, com seus lados medindo, 247 m, 690 m, 622 m, 191 m, 204 m, 112 m, 206 m e 118 m.
- **Área 3** com aproximadamente 23,331 ha, com seus lados medindo, 880 m, 292 m, 442 m, 222 m, 62 m, 111 m, 148 m, 156 m, 181 m, 91 m, 390 m e 144 m.


Figura 32 – Área disponibilizada pela Embrapa para os experimentos em campo.

Devido ao pequeno número de colares desenvolvidos, foi preciso limitar um pouco a área de experimentos disponibilizada pela Embrapa. O projeto Mangueiro Digital e o bebedouro estão presentes próximo a área 1 (Figura 32), na qual há fornecimento de energia elétrica. Por este motivo e pelo fato do boi sempre se locomover até o bebedouro para tomar água, escolheu-se a área próxima ao bebedouro como o local para o nó sorvedouro.

## 5.3 Experimentos

A subseção 5.3.1 apresenta o primeiro experimento em campo com os primeiros 4 colares montados na Embrapa Gado de Corte.

A subseção 5.3.2 apresenta o segundo experimento realizado em campo na Embrapa Gado de Corte com um número maior de colares.

### 5.3.1 Primeiro experimento

No primeiro momento, por questão orçamentária, foram comprados equipamentos para o desenvolvimento de 4 colares.

Foi realizado um primeiro experimento em campo utilizando o protocolo AODV, padrão do XBee. A intenção deste primeiro experimento foi testar o protocolo AODV e coletar informações de GPS para que pudessem ser adicionadas ao Simulador Castalia.

A Figura 33 apresenta o momento que o primeiro colar foi colocado no boi. A Figura 34 mostra os 4 bois com os colares, logo depois de colocados.

Figura 33 – Colocando o colar no primeiro boi

Figura 34 – Colares - Primeiro experimento

A criação de apenas 4 colares no primeiro experimento foi por uma questão meramente econômica. Um novo experimento foi realizado posteriormente, quando foi recebida uma segunda parte dos recursos para a compra de novos equipamentos.

Cada coleta de posição dos bois era realizada a cada 5 minutos e os dados eram enviados assim que recebidos para o nó sorvedouro. Cada pacote enviava um *payload* composto com os seguintes dados: id do colar, latitude, longitude, altitude, data, hora, número de satélites, *timestamp*.

Figura 35 – Teste dos colares fechados na Embrapa Gado de Corte

Antes de colocar os colares no boi, todos foram ligados e fechados. A Figura 35 mostra os colares fechados com o sistema web aberto para verificar que os colares estavam transmitindo.

A coleta teve início no dia 16/09/2015 no período matutino. O experimento foi realizado em uma área de aproximadamente 10788,603 $m^2$ (1,079 Hectares) (Figura 36) dentro do mangueiro digital da Embrapa Gado de Corte.


Figura 36 – Área do experimento

A escolha do intervalo das coletas, de 5 minutos, foi realizada com base nas referências sobre monitoramento animal estudadas e citadas no trabalho de (MEZZALIRA et al., 2011).

Um *WebService* foi desenvolvido para os testes neste projeto. As posições recebidas por ele eram apresentadas em uma interface web que apresentava a lista das últimas posições recebidas na rede nos últimos 30 minutos e um mapa com as últimas posições de cada nó. Para essa interface foi utilizada a linguagem PHP com API do GoogleMaps. A cada 5 minutos, automaticamente fazia um *reload* na página, e as informações mais recentes dos bois eram apresentadas no mapa. Este *WebService* e a interface web com os mapas e listas de posições ficaram hospedados em um domínio da Embrapa durante o primeiro experimento. Dessa forma, era possível acompanhar a movimentação dos bois por meio dessa página em qualquer navegador com acesso à internet.

Neste experimento, o *WebService* fez o uso de JSON (*JavaScript Object Notation*), e não era fundamental para o teste da rede, portanto, foi desenvolvido para testes na infraestrutura da Embrapa, já que havia um estudo paralelo que poderia fazer o uso dessas informações.

Neste primeiro experimento, além do teste de recebimento dos colares e a apresentação das posições no mapa, era pretendido utilizar as posições coletadas para a simulação em Castalia. Estas posições seriam coletadas diretamente do colar. Durante o experimento, houve períodos de desconexão, pois algumas vezes os animais andavam em grupo para um canto do pasto que não tinham alcance ao sorvedouro. Além disso, um dos colares foi perdido neste primeiro experimento.

### 5.3.2 Segundo experimento

O segundo experimento iniciou no dia 17/10/2016. Neste experimento, 8 colares foram montados.

Antes de montar os colares nos bois foi feito um teste de envio utilizando a ferramenta *range test* do XCTU para verificar a perda de pacotes ao fechar o recipiente com o XBee.

O primeiro teste foi realizado com o colar 4, escolhido aleatoriamente. Esse teste consistia em enviar um número de mensagens entre o colar e o sorvedouro para medir taxa de recebimento dos pacotes. O teste foi feito inicialmente com o colar aberto e depois de fechado o colar com a fita adesiva.

Dos 100 pacotes enviados com o recipiente aberto, todos foram recebidos. O mesmo teste foi repetido em seguida, com o recipiente fechado com a fita adesiva, na mesma distância do teste anterior. Todos os 100 pacotes enviados foram recebidos.

Na montagem do colar tomou-se cuidado para evitar que peças de metal bloqueassem o XBee, que é uma recomendação da fabricante para evitar bloqueio de sinal.

A área do segundo experimento foi um pouco maior que a do primeiro por conta do número de colares.

Após o teste de recebimento do colar fechado, os colares foram colocados nos animais. Às 09:37, os animais foram liberadas no pasto para o experimento.

O período de coleta das posições de GPS neste experimento permaneceu a cada 5 minutos. Cada coleta armazenava a identificação do boi, posição do boi no pasto e data/hora.

O protocolo de roteamento escolhido para este experimento foi o *Many-to-one* e o nó sorvedouro ficou próximo do bebedouro. A posição do nó sorvedouro foi medida com o GPS de um dos colares para ser utilizada nas simulações.

No dia 19/11 (quarta-feira), 6 dos colares foram removidos dos bois, os outros 2 colares não puderam ser removidos por conta do stress dos bois, sendo removidos apenas na segunda-feira da semana seguinte.

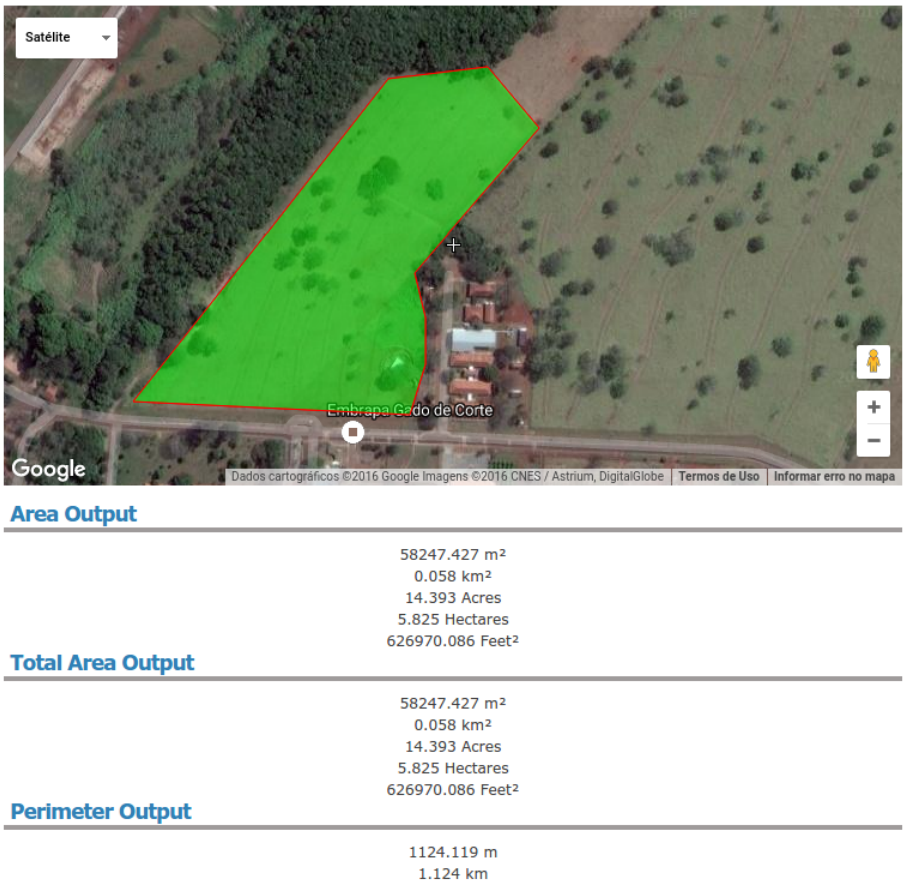



Figura 37 – Área do Experimento 2 no Mangueiro Digital

Com os dados de todos os colares realizou-se a etapa de pré-processamento dos dados de posições para a simulação. Foi preciso converter as posições de latitude e longitude para metros, para ser aceita no Castalia, e as informações sobre este processo estão detalhadas no Capítulo 6.

# 6 Simulações

Este capítulo descreve como as simulações foram realizadas, parâmetros e métricas analisadas para a comparação dos protocolos de roteamento.

Todos os colares deste projeto são abastecidos por uma bateria, por isso a primeira preocupação na montagem da rede foi a escolha de um roteamento eficiente, que enviasse a maior quantidade de dados necessárias e com o menor consumo de bateria. Para isso, foram implementados alguns dos protocolos de roteamento estudados para comparação no simulador Castalia, para definir o que melhor se adequava a este projeto.

As simulações realizadas em Castalia buscaram uma boa aproximação da realidade. Nas configurações utilizadas no simulador, foram utilizados parâmetros como a quantidade de bois que fariam uso do colar; dados da área do experimento; localização onde os bois estariam distribuídos e as características do nó sensor, tais como o protocolo de roteamento, consumo base do colar (sem transmissões), camada de acesso ao meio, e configurações como o tempo de envio das mensagens foram levadas em conta nas simulações.

Os protocolos apresentados na revisão bibliográfica (Tabela 1) (*flooding, gossiping*, AODV, IRP, *history-based*, HGR) foram implementados para uso no Castalia, (em linguagem C++ e linguagem NED). Esta foi a parte mais trabalhosa do estudo, principalmente pela falta de documentação do simulador. O protocolo AODV é um dos mais complexos e foi obtido no grupo de discussões do Simulador Castalia, implementado pelo grupo de pesquisa GERCOM (GERCOM, 2003), além de ser utilizado nas simulações, auxiliou na compreensão para a criação dos demais algoritmos.

Depois de analisados todos os algoritmos da Tabela 1 e entender o funcionamento do roteamento do XBee percebeu-se uma limitação do modelo de XBee utilizado. Conforme explicado anteriormente, não é possível alterar o algoritmo da camada de roteamento do XBee utilizado, optou-se por comparar os protocolos já existentes em XBee e definir qual deles seria o mais eficiente para a utilização no projeto.

Algumas métricas que foram analisadas são a quantidade de pacotes recebidos pelo nó sorvedouro e a energia consumida pelos nós sensores em diferentes configurações, para maximizar o recebimento de pacotes e prolongar o tempo de vida da rede.

## 6.1 Configurações gerais

Em cada modelagem no Castalia pode-se usar a opção collectTraceInfo para coletar detalhes da simulação. Por padrão, um arquivo chamado Castalia-Trace.txt é criado com

as informações de traces de cada módulo.

O módulo que compõe a rede de sensores (SN) no Castalia contém vários sub-módulos de nós. Estes estão endereçados em forma de um *array*. Dessa forma, é possível selecionar um nó e atribuir apenas a ele um parâmetro. Por exemplo, o comando que informa ao nó 0 que ele era o sorvedouro da rede.

```
SN.node[0].Application.isSink = true
```

Também é possível atribuir parâmetros em vários nós ao mesmo tempo, utilizando as formas a seguir de endereçamento:

```
[*] todos
[3..5] nós 3,4,5
[..4] nós 0, 1, 2, 3, 4
[5..] nós 5 até o último nó
```

Como configuração geral, temos os seguintes parâmetros utilizados em nossas simulações:

```
SN.numNodes = 9

sim-time-limit = 72245s #Colar7

SN.field_x = 678
SN.field_y = 1060
```

Alguns parâmetros adicionais serão mostrados nas seções seguintes, mas farão parte de um módulo específico, como por exemplo, o módulo de mobilidade:

```
SN.node[0..8].MobilityManagerName = "GpsMobilityManager"
```

De forma geral, este é o modo que os parâmetros são atribuídos aos nós sensores em um cenário de simulação do Castalia.

As seções seguintes detalharão os demais módulos utilizados, como por exemplo: mobilidade, roteamento, aplicação.

## 6.2 Mobilidade e pré-processamento

O Castalia não disponibiliza um módulo de mobilidade que pudesse ser utilizado para simular a movimentação dos bois no pasto. O módulo de mobilidade mais aprimorado disponibilizado no Castalia simula apenas uma mobilidade em linha reta, dada uma velocidade. Para isso, foi necessário desenvolver um novo módulo de mobilidade em C++ e NED para o Castalia.

A mobilidade dos bois é algo determinante na RSSF, pois devido a ela, pode haver a "quebra" do caminho entre o nó sensor e o nó sorvedouro, impedindo a entrega dos pacotes caso algum boi se movimente para fora do alcance de uma transmissão entre um nó e outro.

Para melhor aproximar a simulação do cenário real, foi desenvolvido um módulo de mobilidade dos bois que utilizou os dados de posições GPS dos bois da Embrapa. Dados coletados no segundo experimento realizado foram utilizados. As principais preocupações da movimentação são exatamente a quebra dos enlaces e/ou a reconstrução de rotas.

Criar este módulo foi um dos desafios de implementação no simulador. O Castalia não aceita coordenadas geográficas nos cenários de simulações, por isso, após coletar as posições de GPS do boi no pasto, foi feito um pré-processamento dos dados coletados convertendo as coordenadas geográficas para metros para inserir no simulador.

Conforme dito no capítulo de experimentos, os cartões de memória gravaram as posições coletadas do GPS no formato de latitude e longitude. As posições eram gravadas juntamente com a data/hora de cada coleta. Foi criado um script em Python para ler os arquivos dos cartões de memória e criar um novo arquivo para cada nó com as informações em metros que pudessem ser lidas pelo simulador Castalia. Para isso, converteu-se as posições de latitude e longitude para posições UTM (*Universal Transverse Mercator*).

Segundo Leonardi (2016), UTM é um sistema de coordenadas baseado no plano cartesiano (eixo x,y) e usa o metro (m) como unidade para medir distâncias e determinar a posição de um objeto. Diferentemente das Coordenadas Geodésicas, o sistema UTM não acompanha a curvatura da Terra e por isso seus pares de coordenadas também são chamados de coordenadas planas. Os fusos do sistema UTM indicam em que parte do globo as coordenadas obtidas se aplicam, uma vez que o mesmo par de coordenadas pode se repetir nos 60 fusos diferentes.

Para cada arquivo lido foi criado um novo arquivo para uso no Castalia, cada arquivo gerado no cartão de memória pelo colar foi gravado no formato de texto com o nome SEQLOG00.TXT. O script criado fazia a leitura de cada arquivo do colar e gravava um novo arquivo com o nome colarX.ini onde o X era o número do colar. As pastas foram organizadas com os nomes colarX e dentro de cada pasta estava o arquivo SEQLOG00.TXT.

O arquivo SEQLOG00.TXT tinha as informações coletadas no experimento no seguinte formato:

Número do colar,Latitude,Longitude,Altitude,Data,Hora, por exemplo:

```
$1,-20.44252400,-54.72272100,306.50000,171016,11:51:22
```

O script contém duas funções: `lercolar` e `gravacolar`.

A função `lercolar` lia as linhas de um colar e retornava a menor latitude, longitude e *timestamp*. O *timestamp* era obtido por meio da função datetime do python e utilizado para sincronizar o tempo dos colares. Um laço era executado chamando a função `lercolar` para cada arquivo, depois novos arquivos era gravados pela função `gravacolar` no formato colarX.ini.

Para converter as posições de latitude e longitude para metros, utilizamos a biblioteca UTM do python, por exemplo:

```
utmval = utm.from_latlon(float(lat),float(long))
x = utmval[0]
y = utmval[1]
```

O arquivo colarX.ini era composto pelo timestamp, posição X e posição Y em metros, por exemplo:

```
1425,239.810100417,286.463776956
```

Após a criação dos arquivos, foram criados outros dois scripts para validar os dados e conferir se estavam refletindo as posições reais.

Para isso, o primeiro script lia as posições do arquivo SEQLOG00.TXT no formato latitude e longitude e fazia um cálculo do intervalo entre uma coleta e outra, a distância percorrida e a velocidade entre um ponto e outro no período entre as coletas. Para calcular o intervalo entre uma coleta e outra apenas converteu-se de data/hora para *timestamp* e diminuiu-se o tempo entre uma coleta e outra. A distância percorrida entre uma coleta e outra foi calculada utilizando uma outra função em python chamada vincenty.

A função vincenty retorna a distância entre posições de GPS em Km, que foi multiplicada por 1000 para obter a distância em metros.

```
distancia = vincenty(anterior,atual)*1000
tempo = (timestamp-timestamp_anterior)
velocidade = distancia/tempo.total_seconds()
```

A velocidade foi calculada apenas dividindo a distância pelo tempo entre as coletas.

O resultado do script retornava as informações, tais como listadas a seguir:

```
-----
Horario da coleta anterior: 2016-10-19 03:55:39
Posicao da coleta anterior: (-20.44289, -54.72289)
Horario atual: 2016-10-19 04:00:40
Posicao atual: (-20.44269, -54.72288)
Tempo despendido: 0:05:01 Tempo em s: 301.0
Distancia percorrida (m): 22.167
Velocidade m/s: 0.0736445182724
```

```
-----
Horario da coleta anterior: 2016-10-19 04:00:40
Posicao da coleta anterior: (-20.44269, -54.72288)
Horario atual: 2016-10-19 04:05:41
Posicao atual: (-20.44276, -54.72284)
Tempo despendido: 0:05:01 Tempo em s: 301.0
Distancia percorrida (m): 8.802
Velocidade m/s: 0.0292425249169
-----
Horario da coleta anterior: 2016-10-19 04:05:41
Posicao da coleta anterior: (-20.44276, -54.72284)
Horario atual: 2016-10-19 04:10:42
Posicao atual: (-20.4425, -54.72301)
Tempo despendido: 0:05:01 Tempo em s: 301.0
Distancia percorrida (m): 33.812
Velocidade m/s: 0.112332225914
-----
Horario da coleta anterior: 2016-10-19 04:10:42
Posicao da coleta anterior: (-20.4425, -54.72301)
Horario atual: 2016-10-19 04:15:42
Posicao atual: (-20.44226, -54.72375)
Tempo despendido: 0:05:00 Tempo em s: 300.0
Distancia percorrida (m): 81.664
Velocidade m/s: 0.272213333333
```

Os mesmos cálculos foram feitos utilizando os arquivos gerados anteriormente com posições de eixo XY para o Castalia.

Para o cálculo do tempo despendido entre uma coleta e outra, apenas subtraiu-se os segundos entre as coletas.

Para a distância entre os pontos, utilizou-se a fórmula de distância euclidiana.

$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$

O resultado foi muito próximo, como visto nas linhas abaixo, que podem ser comparadas com os cálculos em posições geodésicas.

```
-----
Horario da coleta anterior: 145682
Posicao da coleta anterior: (221.608268533, 246.180802362)
Horario atual: 145983
Posicao atual: (222.959574839, 268.312733236)
Tempo despendido: 301.0
Distancia percorrida (m): 22.1731457611
```

```
Velocidade m/s: 0.0736649360835
-----
Horario da coleta anterior: 145983
Posicao da coleta anterior: (222.959574839, 268.312733236)
Horario atual: 146284
Posicao atual: (227.026815949, 260.503499923)
Tempo despendido: 301.0
Distancia percorrida (m): 8.80491767047
Velocidade m/s: 0.0292522181743
-----
Horario da coleta anterior: 146284
Posicao da coleta anterior: (227.026815949, 260.503499923)
Horario atual: 146585
Posicao atual: (209.683345461, 289.540277214)
Tempo despendido: 301.0
Distancia percorrida (m): 33.8220402107
Velocidade m/s: 0.112365582095
-----
Horario da coleta anterior: 146585
Posicao da coleta anterior: (209.683345461, 289.540277214)
Horario atual: 146885
Posicao atual: (132.816630839, 317.188423077)
Tempo despendido: 300.0
Distancia percorrida (m): 81.6878925327
Velocidade m/s: 0.272292975109
```

Com a leitura desses dados, definiu-se o tamanho do cenário de simulação utilizando o maior eixo X e Y. Este tamanho foi adicionado às configurações das simulações como mostrado a seguir:

```
SN.field_x = 678
SN.field_y = 1060
```

Além disso, foram utilizados os dados coletados para obter a velocidade máxima atingida pelos bois no pasto neste experimento.

```
Colar (colar1) Maior velocidade (m/s): 0.877897009967
Colar (colar2) Maior velocidade (m/s): 1.11675415282
* Colar (colar3) Maior velocidade (m/s): 1.16278737542
Colar (colar4) Maior velocidade (m/s): 1.0731627907
Colar (colar5) Maior velocidade (m/s): 0.914906976744
Colar (colar6) Maior velocidade (m/s): 1.01769435216
Colar (colar7) Maior velocidade (m/s): 0.72492358804
Colar (colar8) Maior velocidade (m/s): 1.09004651163
```

```
Horario da coleta anterior: 2016-10-17 21:54:28
Posicao da coleta anterior: (-20.441475, -54.721851)
Horario atual: 2016-10-17 21:59:29
Posicao atual: (-20.443878, -54.72382)
Tempo despendido: 0:05:01 Tempo em s: 301.0
Distancia percorrida (m): 336.143
Velocidade m/s: 1.11675415282
```

A maior distância percorrida no período de 5 minutos (calculada em linha reta entre um ponto e outro) foi percorrida pelo animal com o colar 3 (Figura 38), marcado com asterisco.


Figura 38 – Caminho percorrido pelo animal com colar 3

O módulo de mobilidade desenvolvido para simular as posições foi o GpsMobilityManager que recebeu os parâmetros de mobilidade. Nesse caso, cada nó da simulação aponta para um arquivo de posições, criados após os experimentos.

```
SN.node[0..8].MobilityManagerName = "GpsMobilityManager"
SN.node[0].MobilityManager.PositionsFile = "sorvedouro.ini"
SN.node[1].MobilityManager.PositionsFile = "colar1.ini"
SN.node[2].MobilityManager.PositionsFile = "colar2.ini"
SN.node[3].MobilityManager.PositionsFile = "colar3.ini"
SN.node[4].MobilityManager.PositionsFile = "colar4.ini"
```

```
SN.node[5].MobilityManager.PositionsFile = "colar5.ini"
SN.node[6].MobilityManager.PositionsFile = "colar6.ini"
SN.node[7].MobilityManager.PositionsFile = "colar7.ini"
SN.node[8].MobilityManager.PositionsFile = "colar8.ini"
SN.node[*].MobilityManager.collectTraceInfo = true
```

## 6.3 Módulo Resource - Consumo de energia

Para simular o consumo de energia dos nós foi utilizado o módulo ResourceManager do Castalia.

Este módulo permite incluir informações da bateria dos colares nas simulações. Ao inserir a carga inicial da uma bateria conseguimos obter o consumo aproximado de um nó sensor e o tempo de vida da rede.

O Castalia oferece um parâmetro chamado *baselineNodePower* para definir uma base constante de consumo de energia do nó. Para este parâmetro utilizou-se o valor do cálculo realizado do consumo básico do colar, sem a transmissão de dados pelo XBee. Utilizou-se o valor da Figura 31 (Tabela 1) como base do consumo de bateria.

Este parâmetro recebe valores em mW, para tal, foi feita a conversão de 100mA para mW considerando a bateria de 3,7V utilizada nos experimentos.

$$100mA * 3,7V = 370mW \tag{6.1}$$

```
SN.node[1..].ResourceManager.baselineNodePower = 370 # in mW
```

Além do consumo básico do nó *baselineNodePower*, este módulo permite definir a carga inicial da bateria a partir do parâmetro *initialEnergy*.

Os parâmetros de carga de bateria recebidos pelo Castalia devem ser informados em Joules e as baterias compradas apresentam informações em mAh, para fazer o cálculo dos Joules de cada bateria foi utilizada uma tabela de referência que foi sugerida dentro do código do Castalia em (BATTERIES, 2016).

Inicialmente foi calculado o valor em Wh com os valores mAh e V obtidos pela descrição das baterias. Em seguida, foi transformado para Joules. Os cálculos são apresentados a seguir:

Bateria de 2200mAh:

$$Power = Current * Volts$$
$$(mAh) * (V)/1000 = (Wh)$$
$$2200mAh/1000$$
$$2,2Ah * 3,7v = 8,14wh$$
$$1watt/hour = 3600Joules$$
$$Power = 8,14 * 3600 = 29304J$$

Bateria de 6000mAh:

$$Power = Current * Volts$$
$$(mAh) * (V)/1000 = (Wh)$$
$$6000mAh/1000$$
$$6Ah * 3,7v = 22,2wh$$
$$1watt/hour = 3600Joules$$
$$Power = 22,2 * 3600 = 79920J$$

```
SN.node[1..5].ResourceManager.initialEnergy = 29304 # em joules
SN.node[6..8].ResourceManager.initialEnergy = 79920 # em joules
```

A simulação do tempo de vida da RSSF também fez o uso deste módulo. Neste caso, o valor considerado para definir quando o tempo de vida é quando o primeiro nó fica sem bateria.

Segundo o manual do Castalia, o módulo Resource periodicamente realiza o cálculo do consumo da bateria utilizando a constante definida no parâmetro baselineNodePower, subtraindo o consumo da bateria disponível. Além deste cálculo, o módulo Radio também pode enviar mensagens ao módulo Resource para reduzir o consumo da bateria. Isto é feito sempre que o Radio envia um pacote na rede.

## 6.4 Módulo de aplicação

Para simular a camada de aplicação foi desenvolvido um módulo de aplicação denominado *ToSink*. Neste módulo, cada nó envia sua posição atual a cada 5 minutos para o sorvedouro, semelhante ao intervalo utlizado nos experimentos realizados.

Neste módulo também definiu-se o tamanho do pacote de dados e o nó sorvedouro.

```
SN.node[0].Application.isSink = true
```

Em uma RSSF é comum que os nós sejam ligados um de cada vez e não todos ao mesmo tempo.

No módulo de aplicação adicionamos o tempo de início de cada nó em segundos, baseado no tempo que foram ligados nos experimentos:

```
SN.node[1].Application.startupDelay = 822
SN.node[2].Application.startupDelay = 588
SN.node[3].Application.startupDelay = 712
SN.node[4].Application.startupDelay = 389
SN.node[5].Application.startupDelay = 470
SN.node[6].Application.startupDelay = 909
SN.node[7].Application.startupDelay = 33162
SN.node[8].Application.startupDelay = 1
```

Isso torna mais real as simulações, já que se em uma simulação todos os nós iniciam ao mesmo tempo, pode haver muita interferência que não haveria em um cenário real.

## 6.5 Módulo de acesso ao meio - Camada MAC

As simulações ocorreram sem *duty-cycle* por considerar que os colares estão todos configurados como roteadores.

Além disso, foi utilizado o protocolo MAC que mais se aproxima do CSMA-CA, utilizado pelo XBee no modo roteador.

```
SN.node[*].Communication.MACProtocolName = "TunableMAC"
SN.node[*].Communication.MAC.dutyCycle = 0
SN.node[*].Communication.MAC.collectTraceInfo = true
```

Nas simulações não estamos assumindo tentativas de retransmissão de pacotes na camada MAC. O pacote é enviado apenas uma vez.

O *duty-cycle* nas simulações iniciais foi desativado, já que os colares não fazem uso deste modo.

## 6.6 Módulo de rádio

Os parâmetros de Radio devem ser o mais próximo do equipamento real para a simulação.

Para configurar o módulo de Radio com as informações de consumo do XBee criou-se um arquivo de parâmetros para refletir o consumo do XBee nos variados estados, estes dados foram obtido a partir dos manuais de usuário dos XBees S2B Pro (DIGI, 2015) e

S2C Pro (DIGI, 2016b). Dados, como potência de transmissão, e energia consumida nos diferentes modos de operação.

Nos parâmetros de rádio pode-se configurar algumas informações como: potência de transmissão, consumo de bateria ao transmitir ou receber pacotes.

Os parâmetros da Tabela 10 apresentam os dados obtidos dos manuais dos XBees.

Tabela 10 – Tabela de dados dos XBee obtidos no *datasheet*

| | XBee Pro S2B | XBee Pro S2C |
|---|---|---|
| *Transmit power output* | (+18 dBm)<br><br>*(+10 dBm) for International variant* | +18 dBm |
| *Receiver sensitivity* | -102 dBm | -101 dBm |
| *Operating current (transmit, max output power)* | 205 mA, up to 220 mA with *programmable variant (@3.3 V) 117 mA, up to 132 mA with programmable variant (@3.3 V), International variant* | 120 mA @ +3.3 V |
| *Operating current (receive)* | 47 mA, up to 62 mA with programmable variant (@3.3 V) | 31 mA |
| *Power-down current* | 3.5 $\mu$A typical @ 25 °C | $< 1$ $\mu$A @ 25 °C |

Os arquivos que definem os parâmetros do Radio ficam na pasta Parameters, dentro da pasta Simulations, onde ficam os cenários de simulações.

Na pasta Parameters, existe uma pasta chamada Radio onde criamos dois novos arquivos com as configurações do XBee S2B e S2C, assim como obtidas nos manuais respectivos:

```
SN.node[0].Communication.Radio.RadioParametersFile =
                    "../Parameters/Radio/XBEES2C.txt" # sorvedouro
SN.node[1..5].Communication.Radio.RadioParametersFile =
                    "../Parameters/Radio/XBEES2C.txt"
SN.node[6..8].Communication.Radio.RadioParametersFile =
                    "../Parameters/Radio/XBEES2B.txt"
```

## 6.7 Módulos de Roteamento

Esta seção apresenta as informações sobre detalhes da implementação dos algoritmos no Castalia que são importantes para compreender os resultados obtidos na simulação. Todos os algoritmos citados aqui foram apresentados na seção 2.3.

O protocolo *Flooding* faz o *broadcast* de pacotes de dados para os nós vizinhos e assim por diante. Para não enviar os pacotes de dados indefinidamente e assim inundar a rede com os pacotes enviados por *broadcast* adicionamos uma configuração de número máximo de saltos. Definida pelo parâmetro maxHop, que define quantos saltos um pacote pode percorrer até que a mensagem seja descartada. Neste trabalho, utilizamos o número 7 como número máximo de saltos, que é a quantidade de colares dos experimentos, menos um salto.

O protocolo *Gossiping* envia o pacote de dados para um vizinho aleatório na rede, e o pacote é reenviado até alcançar o destino ou até voltar ao nó que iniciou o envio, sendo descartado nesse caso.

Para enviar o pacote para um vizinho, o nó deve manter uma tabela atualizada dos vizinhos naquele momento. Por isso, antes de um nó fazer o envio dos dados é preciso que este tenha uma tabela de vizinhos atualizada. Para isso, foi implementada uma busca por vizinhos que foi a mesma utilizada nos protocolos *Gossiping*, e *History-based*. A busca por vizinhos funciona da seguinte maneira: o nó envia um pacote por *broadcast* com a sua identificação. Todos os nós que receberam este pacote, o respondem por *unicast* informando que receberam. Quando o nó recebe este retorno, ele consegue identificar de quem recebeu e o adiciona como vizinho em uma tabela de vizinhos.

Com a tabela de nós vizinhos preenchida, o protocolo *Gossiping* seleciona aleatoriamente um dos vizinhos para enviar a mensagem. Isso se repete até a mensagem encontrar o nó de destino ou até voltar para o nó que iniciou o envio, que neste caso descarta o pacote.

A implementação do protocolo *History-based* protocol (ZebraNet) foi baseada no pseudocódigo apresentado na seção 2.3 e utiliza a mesma forma de busca por vizinhos utilizada no *Gossiping*. Segundo o algoritmo, a cada busca por vizinhos, se o nó está no alcance do destino, envia a mensagem e aguarda o recebimento para aumentar seu nível na rede. Um nó saberá se o sorvedouro está próximo, atualizando e verificando sua tabela de vizinhos. Assim que o nó sorvedouro recebe um pacote de dados de um dos colares, este envia ao colar um pacote que na implementação demos o nome de ACK por *unicast*. Sempre que um colar recebe o ACK destinado a ele, este aumenta seu nível, este nível é um parâmetro de cada nó, que se inicia com 0 e é aumentado sempre que recebe um ACK. Se um nó precisa enviar um pacote e não encontra o sorvedouro em sua lista de vizinhos, este nó envia uma mensagem ao vizinho com maior nível hierárquico e assim por

diante. Após um número determinado de busca por vizinhos, o nível hierárquico de um nó é diminuído. Neste trabalho, a implementação realiza uma busca por vizinhos sempre que precisa enviar um pacote de dados, ou seja, a cada 5 minutos a tabela é atualizada.

O protocolo HGR recebe dois parâmetros no arquivo de configuração referentes à posição do nó sorvedouro. São eles, a posição no eixo X, sinkPosX, e a posição no eixo Y, sinkPosY. O algoritmo de busca de próximo salto, conforme descrito na seção 2.3 realiza os cálculos para selecionar o salto de acordo com a distância e direção entre o nó que pretende enviar a mensagem, os vizinhos e o sorvedouro. Utilizou-se neste trabalho apenas o algoritmo de seleção de próximo salto. Segundo o artigo utilizado como base para esta implementação, os dados de direção são carregados no pacote de dados. Na implementação realizada, os nós utilizam essas informações do pacote de dados para atualizar as tabelas com informações das posições dos vizinhos. Utilizamos a seleção de próximo salto combinada com um *broadcast*. Funcionando da seguinte forma: sempre que o nó precisa enviar um pacote com dados de posições, ele verifica se possui algum vizinho na sua tabela de vizinhos. Se não possui, ele envia esse pacote por *broadcast*, que irá atingir todos os vizinhos mais próximos e não será retransmitido. Com esses dados enviados, os vizinhos podem atualizar suas tabelas com as posições do nó que enviou o pacote, aproveitando as informações de posições. Se o nó que deseja enviar o pacote de dados tem vizinhos em sua tabela de vizinhos, verifica primeiro se tem o nó destino como vizinho, e se tiver, envia o pacote diretamente ao destino por *unicast*. Se o nó destino não estiver na lista de vizinhos, a escolha do vizinho é feita utilizando o algoritmo descrito na seção 2.3.

O algoritmo AODV utilizado no trabalho está baseado na implementação do protocolo realizada pelo grupo de pesquisa (GERCOM - Grupo de Estudos em Redes de Computadores e Comunicação Multimídia) e disponível para download em (GERCOM, 2003). Esta implementação em Castalia já foi utilizada em outros trabalhos como (KASHNIYAL; MANDORIA, ), (YAGOUTA; GOUISSEM, 2016) e (MACHADO et al., 2013). A implementação foi baseado na especificação do algoritmo RFC 3561. Os parâmetros do AODV utilizados foram o padrão da especificação RFC 3561.

Os protocolos de roteamento IRP e *Many-to-one* foram configurados com o tempo de envio de configuração da rede em 5 minutos.

## 6.8 Resultado das simulações

Esta seção apresenta os resultados das simulações. As métricas utilizadas para comparar os protocolos foram: consumo de energia, taxa de entrega de pacotes no sorvedouro e latência.

### 6.8.1 Número de Pacotes Entregues no Sorvedouro - Por nó

Para realizar o cálculo de pacotes entregues, primeiramente definimos o total de pacotes enviados por cada colar na rede. O gráfico da Figura 39 apresenta o quantitativo de pacotes enviados na rede por colar.

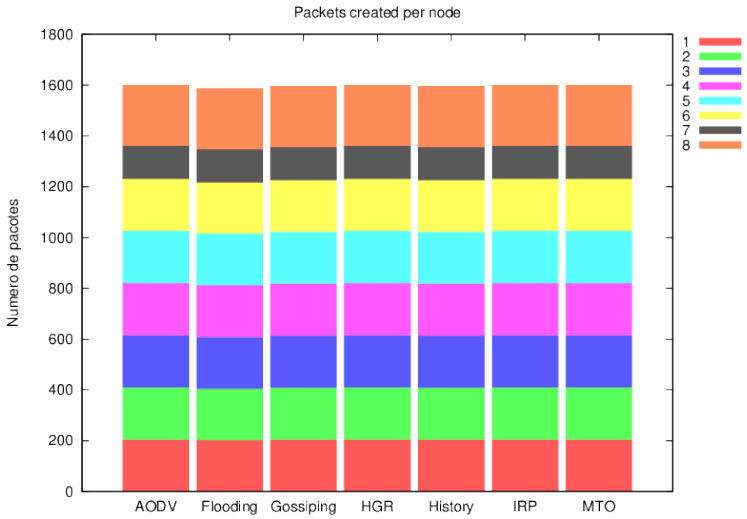



Figura 39 – Número de Pacotes Criados Por cada Nó

No experimento de coleta de dados, houve um problema físico no colar 7 que teve que ser ligado tempos depois dos demais colares, o gráfico apresenta um número menor de pacotes criados por este nó. Portanto, a mesma quantidade de pacotes foi enviada na rede em todos os protocolos simulados. Com este número foi feito o cálculo da taxa de entrega para cada protocolo.

O gráfico da Figura 40 apresenta a quantidade de pacotes entregues no sorvedouro por cada nó. Este valor refere-se ao número de pacotes não duplicados, desprezando-se os repetidos.

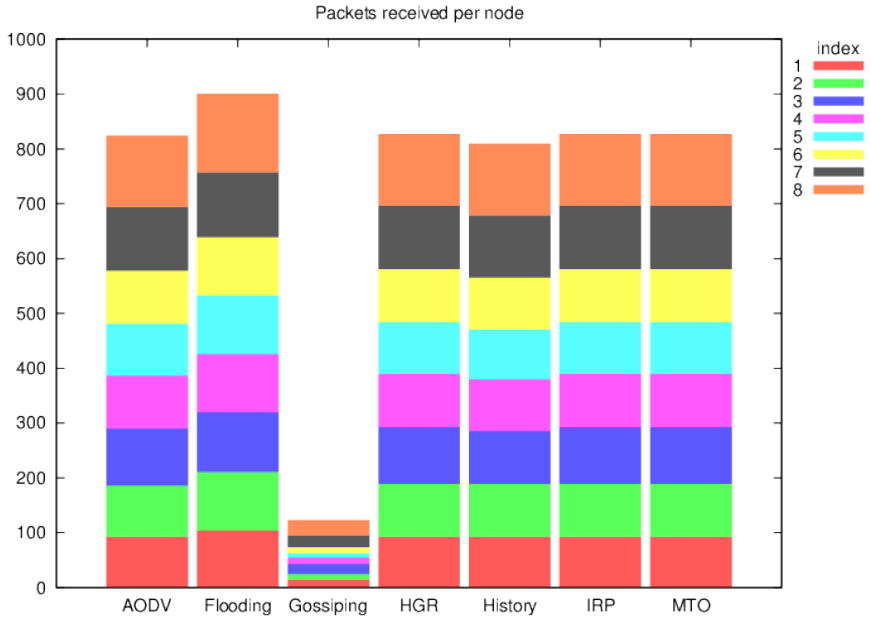



Figura 40 – Pacotes entregues no sorvedouro (por nó).

O protocolo *Gossiping* foi o protocolo com menor taxa de entrega devido a característica de protocolo de escolha aleatória do próximo salto. Mais informações sobre a implementação no simulador Castalia são apresentados na subseção 6.8.7.

### 6.8.2 Taxa de Entrega de Pacotes

O gráfico da Figura 41 apresenta a taxa de pacotes entregues ao sorvedouro por protocolo. Assim como no gráfico de pacotes recebidos, neste gráfico estão contabilizados apenas os pacotes não duplicados recebidos pelo sorvedouro.

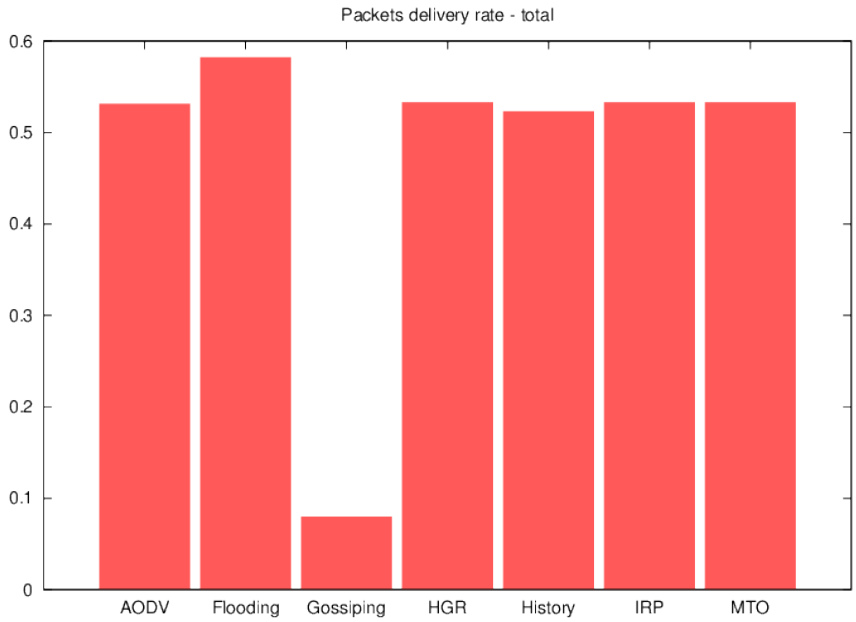



Figura 41 – Taxa de entrega de Pacotes

O quantitativo apresentado no gráfico é dado pela razão dos pacotes de dados não duplicados recebidos na camada de aplicação do nó sorvedouro pela quantidade de pacotes criados pelos nós sensores.

### 6.8.3 Latência em nível de aplicação (em ms)

A Figura 42 apresenta a latência da camada de aplicação, estimativa de tempo que um pacote leva desde o colar até o sorvedouro.

O gráfico apresenta no eixo X os protocolos comparados e no eixo Y a quantidade de pacotes recebidos. A legenda de cores apresenta a faixa de tempo em milissegundos.

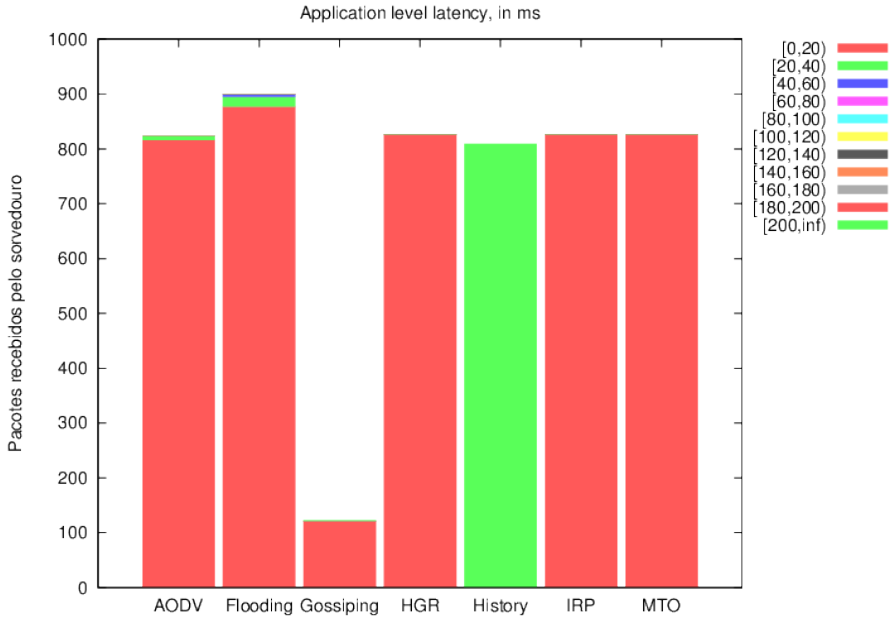



Figura 42 – Latência

Observa-se neste gráfico que os algoritmos que utilizam uma configuração prévia de rota tiveram melhor desempenho. Por exemplo, os algoritmos que implementam buscas de vizinhos sempre que precisam enviar pacotes tiveram pior desempenho comparados aos demais. Os protocolos em que o sorvedouro envia um pacote de manutenção de rota periodicamente tiveram melhor desempenho. O AODV, que é um protocolo reativo, e o *History* que na implementação refaz uma busca por vizinhos sempre que precisa enviar uma mensagem, tiveram atrasos no envio e latência um pouco maior que os demais.

Com apenas 8 colares, não houve grande atraso nas entregas. Considerando o período de envio de dados a cada 5 minutos, a latência apresentada nos resultados não se mostrou um fator determinante para a escolha do protocolo para o cenário estudado. Outros fatores, como o consumo de energia tiveram um resultado com mais divergências e por isso tem maior impacto na escolha.

### 6.8.4 Energia consumida

O Castalia calcula a energia consumida pelo rádio com base no tempo de envio da mensagem e potência de transmissão do rádio. O tempo de transmissão da mensagem é calculado pela fórmula a seguir, extraída do código do simulador, onde a variável txTime é o tempo de transmissão:

$$double txTime = ((double)(end- > getByteLength() * 8.0f))/RXmode- > datarate;$$

Sempre que um pacote é enviado no Castalia, o módulo de rádio entra no estado TX e muda o consumo do nó durante o intervalo calculado em txTime.

O Castalia não calcula consumo nos pacotes recebidos, e assume apenas os estados do radio. Para o Castalia, se o rádio está no estado RX, estará gastando uma quantidade determinada de energia independente de estar recebendo pacotes ou não. E no estado de transmissão, consome a quantidade específica para a potência utilizada pelo transmissor.

O gráfico da Figura 43 mostra a estimativa da energia consumida pelos colares. Este gráfico apresenta o consumo total dos colares, somando o consumo do circuito e o consumo dos XBees.

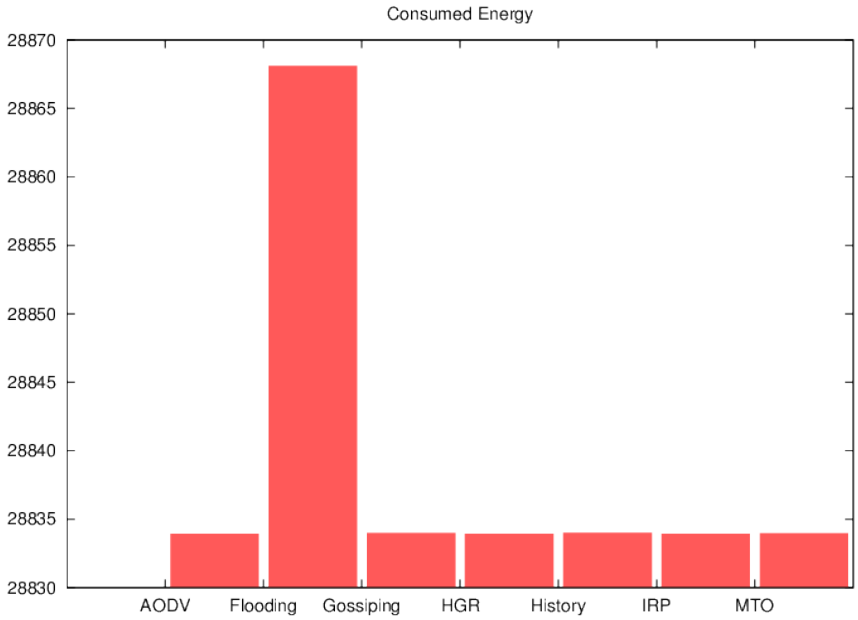



Figura 43 – Energia consumida pelo colar em cada algoritmo.

O consumo de energia apresenta pouca diferença entre os protocolos devido o consumo base do colar que é alto, devido ao elevado consumo do GPS, e o intervalo de envios dos dados na rede que é de 5 minutos. Portanto, alterou-se a faixa de apresentação do consumo no gráfico.

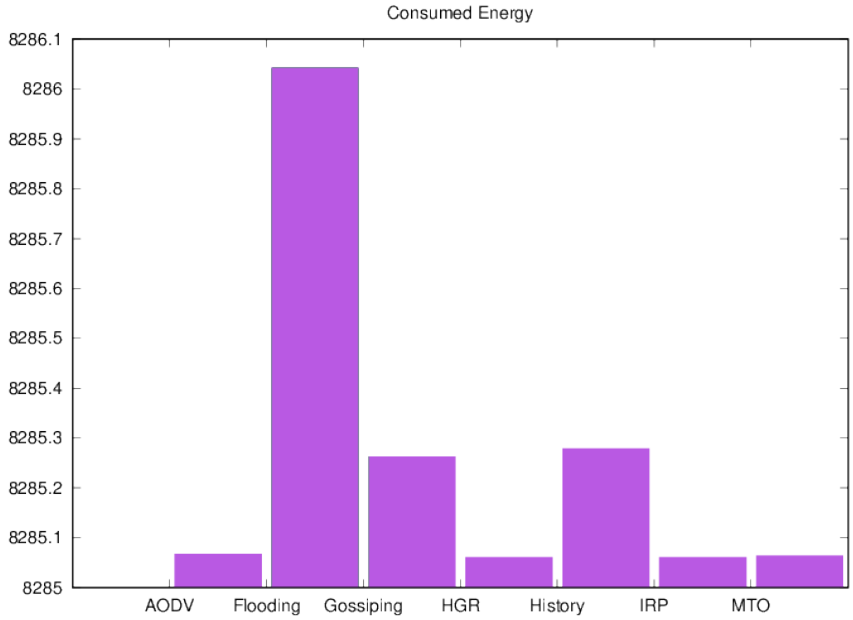



Figura 44 – Energia consumida apenas pela transmissão de pacotes na rede

Um outro gráfico (Figura 44) foi criado apenas com os dados das transmissões dos pacotes de dados, excluindo o consumo do GPS.

### 6.8.5 Estimativa de vida da rede

A tempo de vida no Castalia utiliza a seguinte fórmula:

$$((initialEnergy * simulation_time)/((initialEnergy - remainingEnergy) * 86400.0));$$

Esta fórmula retorna o tempo de vida em meses. Para as simulações realizadas, foi feita uma alteração no Castalia para apresentar o tempo de vida em horas.

O gráfico da Figura 45 apresenta a estimativa em horas do tempo de vida da rede. Neste gráfico, assim como no gráfico de energia consumida, observa-se que os valores são muito próximos entre os protocolos simulados, devido ao alto consumo base do colar e ao intervalo de envio dos pacotes a cada 5 minutos.

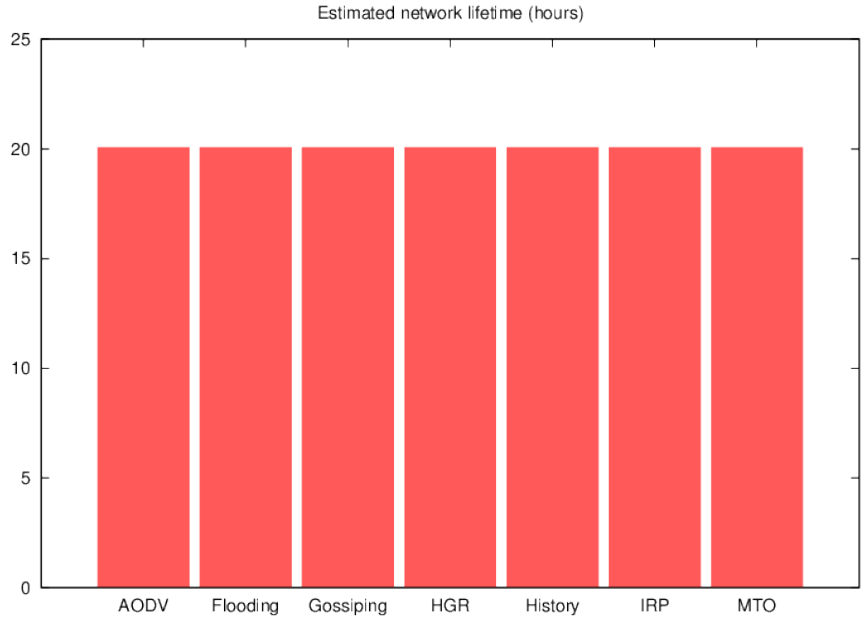



Figura 45 – Estimativa de Tempo de Vida (em horas)

### 6.8.6 Estimativa de vida da rede com e sem GPS

Para apresentar melhor a diferença entre o consumo dos protocolos e ao mesmo tempo comparar o consumo entre um colar com e sem GPS, foi gerado um novo gráfico.

O gráfico da Figura 46 apresenta a estimativa de vida da rede, com e sem o uso do GPS no colar.

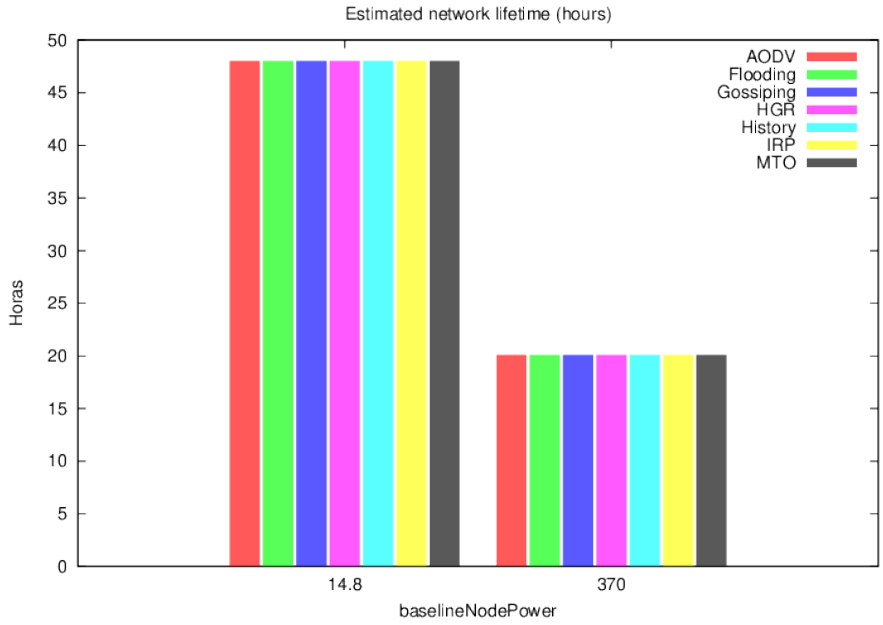



Figura 46 – Comparação de Estimativa de Tempo de Vida (em horas)

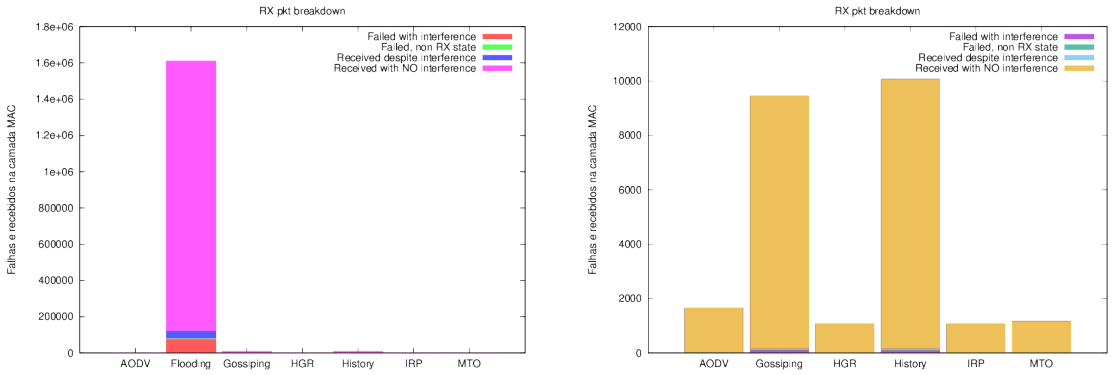



Figura 47 – Falhas e recebimentos de pacotes na camada MAC

A comparação na Figura 46 leva em conta apenas o uso ou não do GPS. Nesta simulação, os colares ainda permanecem no modo roteador, e é possível reduzir ainda mais o consumo de energia alterando os colares para dispositivos *end devices* em trabalhos futuros. Para isso é preciso mais estudo para definir as posições dos roteadores da rede, por isso, não foi realizada esta simulação neste trabalho.

Utilizamos o valor aproximado do consumo do Arduino sem o GPS e antena, nas medições por multímetro obteve-se um valor próximo a 4mA com a bateria de 3,7V utilizada nos experimentos. Convertemos este valor para mW, para obter o valor de 14,8mW. Sem o uso do GPS, outras alternativas podem ser estudadas para reduzir ainda mais o consumo do microcontrolador.

### 6.8.7 Considerações sobre as simulações realizadas

Para entender por que alguns protocolos tiveram desempenho parecidos, verificamos a Figura 47 que apresenta o estado dos pacotes transmitidos e recebidos na camada MAC. Neste gráfico pode-se verificar os pacotes recebidos ou não na camada MAC e as causas. Como o objetivo era comparar os protocolos de roteamento, as transmissões na simulação não tiveram tentativa de reenvio de pacotes.

Devido a grande quantidade de pacotes enviados e recebidos pelo *Flooding*, foram feitos dois gráficos separados, um deles sem o *Flooding* com todos os protocolos.

O Castalia não calcula drenagem de energia dos pacotes recebidos ou falhas de recebimento na camada MAC.

O gráfico da Figura 48 apresenta a quantidade de pacotes enviados à camada MAC para cada protocolo. Para o Castalia, a bateria é drenada apenas no período que o rádio está no estado de transmissão, enquanto os pacotes estão sendo enviados.

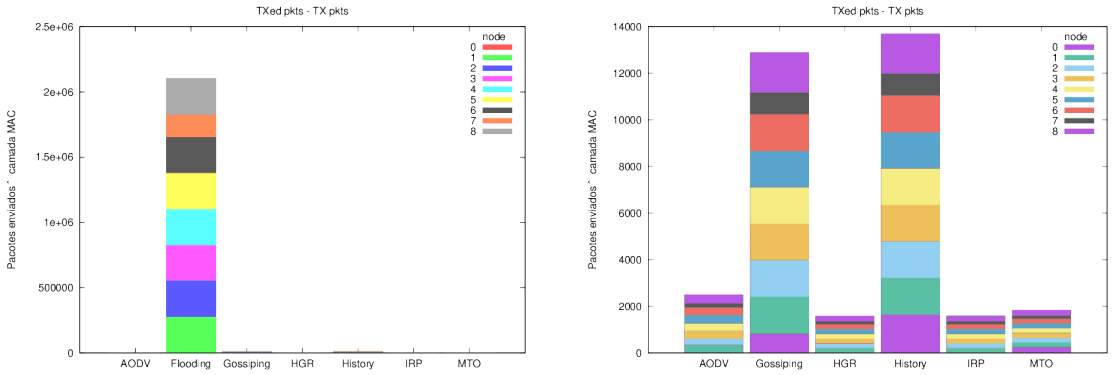



Figura 48 – Pacotes enviados à camada MAC

Pode-se observar por estes gráficos que em alguns dos protocolos há um grande número de pacotes enviados, como exemplo do AODV, *Flooding* , *Gossiping* e *History-based*. O que era esperado no protocolo *Flooding* por conta dos problema de implosão deste protocolo. Já os protocolos *History-based* e *Gossiping* enviam um número grande de pacotes porque da maneira que foram implementados, realizam uma busca por vizinhos sempre que precisam enviar um pacote. O protocolo AODV também tem um grande número de envios, dado a implementação que emite pacotes *hello* periodicamente para confirmar a disponibilidade dos *links*. Estes fatores aumentam o consumo de energia e reduzem o tempo de vida da rede.

Nos protocolos comparados, vimos que o MTO, IRP e HGR enviaram número menor de pacotes na rede, em relação aos demais protocolos. Isto porque os nós destes protocolos não precisam criar rotas ou mecanismos de descobertas por vizinhos. No MTO e IRP, se o sorvedouro estiver enviando os pacotes de manutenção de rede periodicamente, basta os colares armazenarem os caminhos ou números do próximos saltos, depois disso, a mensagem pode ser enviada por *unicast*. Observa-se que não houve interferência na transmissão dos pacotes na rede (Figura 47). O protocolo HGR também não realizou busca por vizinhos, por isso não houve interferência nos pacotes.

Mesmo em simulações com um intervalo menor entre as transmissões, os protocolos AODV, MTO, HGR e IRP apresentam um menor número de interferências comparados aos demais.

Neste trabalho foram comparados os protocolos de roteamento, portanto não adicionamos confiabilidade na camada MAC como há no XBee. O protocolo AODV teve taxa de recebimento de pacotes de aplicação pouco menor que o *many-to-one*, dado provavelmente a interferência ou não recebimento por estar com o rádio transmitindo. No entanto, o XBee permite adicionar confiabilidade da transmissão por meio da camada MAC e pela camada de aplicação, o que em um cenário real poderia aumentar um pouco o consumo

devido aos pacotes retransmitidos, mas provavelmente aumentaria a taxa de entrega. Quanto ao consumo de energia, nota-se que o *many-to-one* consumiu menos energia que o AODV. Entre os dois protocolos para o cenário estudado o protocolo *many-to-one* é o que se saiu melhor e o que é sugerido para utilização neste projeto.

Após a realização das simulações, considera-se que para ao cenário analisado, os protocolos IRP, HGR e *Many-to-one* apresentam bom desempenho no tempo de entrega das mensagens (*delay*), o menor consumo de energia e maior tempo de vida da rede. Estes protocolos juntamente com o *Flooding* tiveram o melhor desempenho na quantidade de pacotes recebidos pelo sorvedouro.

Os demais protocolos apresentaram falhas de recebimento de pacotes, seja por interferência ou pelo rádio não estar no estado de recebimento no momento que deveria receber um pacote (Figura 47).

A finalidade inicial deste trabalho era comparar os protocolos de roteamento baseando em posições reais coletadas em experimentos reais na Embrapa Gado de Corte.

Inicialmente, pensou-se em criar simulações adicionais com mobilidade aleatória utilizando uma velocidade máxima definida pelos resultados dos experimentos dos bois. A velocidade máxima no experimento foi calculada, portanto, observando o comportamento dos bois durante os experimentos, observou-se quebras nos enlaces pois os bois tendiam a permanecer em grupos nos experimentos realizados, o que é também refletido nas simulações. Por isso, julgou-se que implementar um módulo com posições aleatórias para simular os bois não refletiria um comportamento próximo ao comportamento dos bovinos, já que não haveria a mobilidade em grupos e tempos de pausa dos animais.

Um outro aspecto do comportamento social dos bovinos é a liderança, que muitas vezes resulta na atividade sincronizada dos bovinos. Um rebanho de vacas se comporta como uma unidade, na qual a maioria dos membros apresenta o mesmo comportamento ao mesmo tempo. Há sempre um animal que inicia o deslocamento ou as mudanças de atividade, quando ele é seguido pelos outros, trata-se do líder. Geralmente são as vacas mais velhas que lideram os rebanhos, que não estão no topo da ordem de dominância. Isto faz sentido se considerarmos que a estrutura social dos bovinos é originalmente matrilinear (STRICKLIN; KAUTZ-SCANAVY, 1984).

### 6.8.8 Simulação de consumo no modo *sleep*

Uma simulação adicional (Figura 49) foi realizada para comparar o consumo da bateria dos colares atuais com o consumo dos colares caso estes fizessem o uso do *sleep*. Neste caso, não foi levado em conta o protocolo utilizado, pois o XBee configurado como *End device* não executa roteamento, apenas envia os pacotes ao roteador ou coordenador que está ligado na rede.

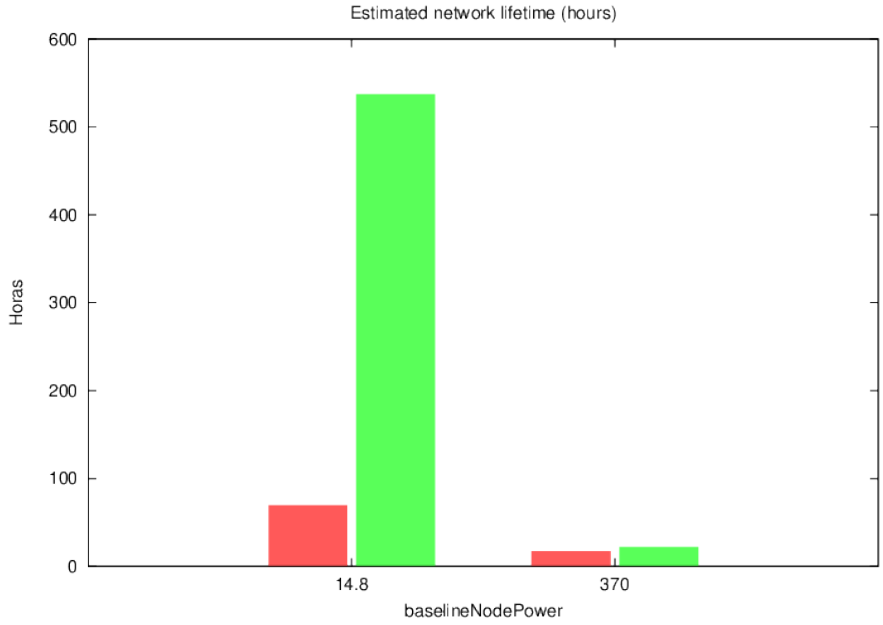



Figura 49 – Comparação entre tempo de vida dos colares

Para esta simulação, foi utilizado um novo cenário com apenas um nó. Este nó simulou um colar com o XBee S2C, bem como a mesma bateria de 2200mAh, utilizou também o mesmo tempo de simulação que as demais simulações realizadas. O tamanho dos pacotes também permaneceu o mesmo. O módulo de aplicação utilizado foi o ThroughputTest, fornecido pelo Castalia, para simular o envio de um pacote para entre um nó e outro a cada 5 minutos.

Neste mesmo gráfico temos duas *baselines*, uma delas sem o GPS (14,8mW) e a outra, do colar com o GPS (370mW).

As barras de cor avermelhada representam o tempo de vida em horas do colar sem utilização do modo *sleep*.

O modo *sleep* é apresentado nas barras de cor esverdeada. Nestas simulações, durante o intervalo de 5 minutos entre as transmissões, dedicamos um tempo de 10 segundos, nos quais o colar pode variar nos estados RX/TX.

A Figura 49 faz uma estimativa da diferença entre o tempo de vida do colar atual e do colar que estivesse fazendo uso do recurso *sleep*. Várias tentativas foram feitas neste trabalho para tentar reduzir o consumo de energia sem infraestrutura adicional, no qual apenas os colares dos bois fariam a retransmissão dos dados. No entanto, não há como garantir que não tenha desconexão entre os nós devido ao comportamento dos bovinos.

Pensando nisso, foi visto como opções de melhorias solucionar o problema da confiabilidade da entrega dos dados ou a criação de infraestrutura adicional no pasto para funcionar como roteadores da rede.

Um desafio para o problema da confiabilidade é que o nó teria que gravar os dados e fazer novas tentativas até conseguir fazer o envio para o sorvedouro, o que aumentaria o tráfego de pacotes na rede. Uma solução alternativa é se o colar sempre gravasse as informações e tentasse transmitir quando estivesse próximo do sorvedouro, o que exigiria alguma forma de saber a localização e sempre verificar sua distância, no entanto o tempo de entrega dos pacotes poderia demorar, pois dependeria do boi se locomover até próximo do sorvedouro.

Para a instalação de infraestrutura adicional, seria necessário realizar um estudo para garantir uma cobertura de nós dentro do pasto, para que esses nós consigam replicar os dados até o sorvedouro. Para isso, seria necessário fazer um calibração para saber o alcance de transmissão do colar no pasto. Neste caso, garantindo que os nós de infraestrutura estejam sempre alimentados de energia elétrica ou painéis solares. É mais fácil definir o protocolo utilizado para esta topologia, com os roteadores fixos. Com isso, a entrega dos pacotes não dependeria da locomoção do bovino até o sorvedouro. Com esta configuração, os colares se comunicariam diretamente a um desses roteadores e poderia fazer o *sleep*.

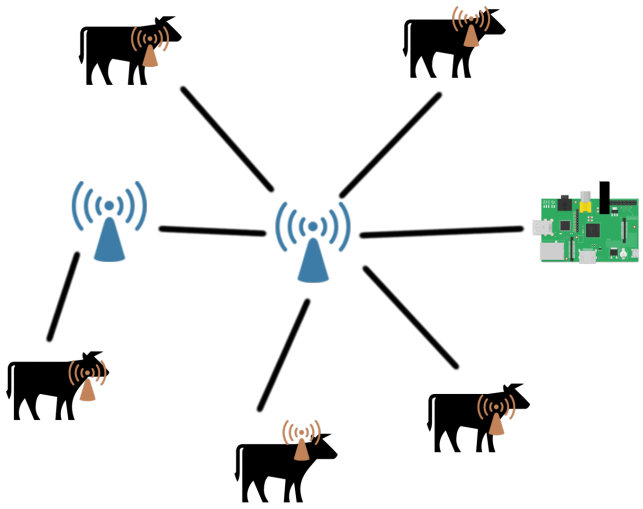


Figura 50 – Rede com infraestrutura adicional como roteadores

A Figura 50 representa a sugestão de rede para trabalhos futuros. Na imagem, os colares são nós *end devices* da rede. Na cor marrom, e enviam os pacotes diretamente aos locais de antenas, em azul.

Nesta topologia, que facilitaria a escalabilidade da rede, diminuiriam as quebras por falta de alcance desde que estude pontos para cobertura da área analisada, e diminuiriam muito o consumo das baterias dos colares, aumentando o tempo de vida da rede. Neste novo cenário com nós fixos, o protocolo *many-to-one* pode ser utilizado, e o envio de manutenção de rota pelo sorvedouro poderia ser feito apenas quando um novo nó fosse

ligado. No entanto para garantir que os nós não sejam desconectados, um período de envio de manutenção de rotas pode ser ativado no sorvedouro, já que os nós fixos poderia fazer o uso de painéis solares, não precisando se preocupar com o consumo de energia.

# 7 Considerações finais

O trabalho escolhido exigiu a solução de uma série de desafios. A restrição de bateria dos nós sensores e a mobilidade dos animais foram os maiores desafios deste projeto.

Diversos protocolos de RSSF disponíveis na literatura foram analisados no contexto de monitoramento bovino, considerando por exemplo, aspectos como a mobilidade dos bois e requisitos de tempo de entrega das mensagens.

Os algoritmos de roteamento selecionados foram testados no simulador de RSSF Castalia, considerando a aplicação de monitoramento bovino. Atualmente existem vários simuladores de RSSF, a escolha pelo Castalia fez com que fosse necessário adaptar o simulador para as necessidades do projeto.

O uso do simulador auxiliou a escolha do protocolo utilizado no trabalho, apesar das limitações do XBee, e foi possível selecionar entre o melhor algoritmo disponível.

O nó sorvedouro teve que ser desenvolvido de forma diferente dos demais sensores, com um *hardware* diferente, alimentado de energia durante todo o tempo para suportar uma maior quantidade de transmissões, pois este é o nó que recebe todos os dados da rede.

Neste trabalho, foram utilizadas algumas estratégias para reduzir o consumo de energia durante as transmissões, como o envio de amostras em intervalo de tempo e minimizando a quantidade de informações transmitidas na rede, convertendo do formato NMEA e transmitindo apenas as informações necessárias para a identificação da posição dos colares. Não foi possível fazer a hibernação de recursos devido a mobilidade dos colares.

O que pode se perceber nos resultados das simulações e observações deste trabalho é que o GPS é um equipamento que consome bastante bateria do colar. O foco do trabalho foi a escolha do melhor protocolo de roteamento visando boa taxa de entrega e menor consumo de energia. Porém, somente com a escolha do protocolo de roteamento com o menor consumo não há grande impacto na economia de energia do colar sem solucionar o problema do consumo de energia do GPS. Desenvolver um colar sem o GPS, fazendo cálculos de posição à partir da potência de transmissão dos demais colares pode ser uma alternativa viável para trabalhos futuros, o que exigirá a pesquisa de soluções para calibrar e definir uma estimativa de distância a partir da potência recebida pelo sinal. Ao reduzir o consumo do GPS, o tempo de vida da RSSF poderá ser aumentado.

Apesar de ter escolhido o melhor protocolo para este projeto baseado nas simulações e experimentos, observou-se que devido à mobilidade dos bois não foi possível

utilizar os modos de *sleep* do XBee, o que reduziria o consumo de energia. Como é citado no manual do XBee, é recomendável que os XBees configurados como roteadores sejam alimentados por energia durante todo o período que permanecem ligados. A existência de nós adicionais na rede permitiria que os colares utilizassem o modo *sleep* caso pudessem se comunicar com nós roteadores. Uma vez que este projeto teve foco no roteamento, sugerimos melhorias simples na RSSF desenvolvida na seção 7.2.

A RSSF desenvolvida permitiu a recepção dos dados de forma automática para o sorvedouro, que apresenta as posições atualizadas dos animais sempre que recebe uma nova posição. Com o monitoramento dos bois a partir do sorvedouro, o produtor pode saber a última informação recebida pelo boi, com o horário e localização, também pode buscar uma lista de posições pelo ID de um ou mais colar, ou apresentar no mapa o caminho percorrido por um ou mais colar dentro de um determinado intervalo de tempo.

## 7.1 Dificuldades encontradas

Em uma RSSF com nós fixos, é mais fácil programar os nós sensores para enviar os pacotes de dados a um determinado destino, pois podemos saber antes da implantação da rede sempre quem será o próximo nó. Já em RSSF móveis, a mobilidade pode fazer com que um determinado nó sensor não consiga encontrar o próximo salto se este sair de seu alcance, e dessa forma, não alcançar o nó de destino. Essa é uma das maiores dificuldades deste trabalho já que os bois podem se mover.

Um outro desafio é preservar a bateria dos nós sensores pelo maior tempo possível. A bateria tem uma carga finita e pode ser inviável financeiramente incluir um painel solar em cada boi, além dos problemas de peso e tamanho do painel. É possível utilizar painéis solares para alimentar o nó sorvedouro, caso não seja possível alimentar o nó sorvedouro pela rede elétrica, mas esta funcionalidade foi deixada para trabalhos futuros.

Demais dificuldades encontradas:

- Pouca documentação e exemplos de códigos na internet para o simulador Castalia, o que exigiu a implementação da maioria dos protocolos de roteamento estudados.
- O simulador não possui interface gráfica, o que dificulta a implementação e testes dos algoritmos.
- Desenvolvimento de um módulo de mobilidade com os dados de posições reais dos bois no pasto. Como dito durante a fundamentação teórica, o Castalia disponibiliza apenas um módulo de mobilidade, no qual podemos definir a posição de início e fim de cada nó no arquivo omnet.ini e o outro módulo é utilizado apenas para redes fixas.

- Encontrar a melhor forma de montagem do colar de forma que o boi ou o próprio colar não sejam obstáculos para a transmissão das mensagens.

## 7.2 Trabalhos futuros

Algumas ideias de trabalhos futuros compreendem:

- Melhoria do software aplicativo para recebimento de dados de outros sensores dos animais, além das posições, podendo apresentar dados históricos do animal, como data de nascimento ou data de vacinação;
- Execução de ações baseadas nos dados fornecidos pelos sensores, como emitir alertas por email ou SMS para o pesquisador sempre que o sistema detectar um comportamento anormal de algum animal;
- Utilização de algoritmos de compactação das mensagens entre os sensores, para diminuir o consumo de energia durante as transmissões de dados;
- Utilização de nós sensores fixos no pasto que pudessem detectar variáveis do ambiente, como umidade e temperatura;
- Utilização de nós sensores fixos no pasto que pudessem servir como "roteadores" para a transmissão dos dados;
- Substituição da placa Arduino FIO por uma placa impressa com um microcontrolador para a redução de custo do colar;
- Desenvolvimento de colares sem o GPS, que calculassem a posição aproximada do animal por aproximação dos demais nós e enviasse a informação pela rede.

# Referências


AKKAYA, K.; YOUNIS, M. A survey on routing protocols for wireless sensor networks. *Ad hoc networks*, Elsevier, v. 3, n. 3, p. 325–349, 2005. Citado na página 21.

AKYILDIZ, I. et al. Wireless sensor networks: a survey. *Computer Networks*, v. 38, n. 4, p. 393 – 422, 2002. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128601003024>. Citado na página 16.

AKYILDIZ, I. F. et al. A survey on sensor networks. *IEEE communications magazine*, IEEE, v. 40, n. 8, p. 102–114, 2002. Citado 3 vezes nas páginas 11, 18 e 19.

ALLIANCE, Z. 2012. Disponível em: <http://zigbee.org/non-menu-pages/zigbee-specification-download/>. Citado na página 29.

ARCH. 2015. Disponível em: <http://en.wikipedia.org/wiki/Arch_Linux_ARM>. Citado na página 57.

BATTERIES. *All about batteries*. 2016. (Accessed on 11/15/2016). Disponível em: <http://www.allaboutbatteries.com/Energy-tables.html>. Citado na página 85.

BERCKMANS, D. Automatic on-line monitoring of animals by precision livestock farming. *Proceedings of the ISAH Conference on Animal Production in Europe: The Way Forward in a Changing World - Saint-Malo, France*, p. 27–31, 2004. Citado na página 10.

BOULIS, A. et al. Castalia: A simulator for wireless sensor networks and body area networks. *NICTA: National ICT Australia*, 2011. Citado 2 vezes nas páginas 4 e 27.

BUETTNER, M. et al. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In: ACM. *Proceedings of the 4th international conference on Embedded networked sensor systems*. [S.l.], 2006. p. 307–320. Citado na página 25.

CASTALIA. 2014. Disponível em: <https://castalia.forge.nicta.com.au/index.php/en/>. Citado 2 vezes nas páginas 12 e 27.

CHEN, M. et al. Hybrid geographic routing for flexible energy delay tradeoff. *IEEE Transactions on Vehicular Technology*, IEEE, v. 58, n. 9, p. 4976–4988, 2009. Citado na página 22.

COOJA. 2014. Disponível em: <http://www.contiki-os.org/>. Citado na página 12.

DEE, J. *How to Power a Project*. 2016. Disponível em: <https://learn.sparkfun.com/tutorials/how-to-power-a-project>. Citado na página 70.

DIGI. *ZigBee RF Modules - XBEE2, XBEEPRO2, PRO S2B User Guide*. 2015. Disponível em: <http://www.digi.com/resources/documentation/digidocs/PDFs/90000976.pdf>. Citado 7 vezes nas páginas 4, 36, 37, 39, 47, 70 e 87.

DIGI. *Digi XBee Application Note*. 2016. Migration from XBee/XBee-PRO ZB (S2/S2B) to XBee/XBee-PRO ZB (S2C). Disponível em: <http://www.digi.com/pdf/xbee%20zigbee%20migration%20guide.pdf>. Citado 3 vezes nas páginas 45, 56 e 69.

DIGI. *DIGI XCTU*. 2016. Disponível em: <http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu>. Citado na página 37.

DIGI. *Knowledge Base Article: XBee ZigBee Addressing*. 2016. <http://knowledge.digi.com/articles/Knowledge_Base_Article/XBee-ZigBee-Addressing/?l=en_US&fs=RelatedArticle>. (Accessed on 11/14/2016). Citado na página 40.

DIGI. *XBee Java Library*. 2016. Disponível em: <https://docs.digi.com/display/XBJLIB/XBee+Java+Library>. Citado na página 58.

DIGI. *XBee ZigBee Mesh Kit - Addressing*. 2016. <http://www.digi.com/resources/documentation/Digidocs/90001942-13/concepts/c_addressing.htm>. (Accessed on 11/11/2016). Citado na página 40.

DIGI. *XBee/XBee-PRO S2C ZigBee RF Module User Guide*. 2016. Disponível em: <http://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf>. Citado 2 vezes nas páginas 70 e 88.

DU, W. et al. Modeling and simulation of networked low-power embedded systems: a taxonomy. *EURASIP Journal on Wireless Communications and Networking*, Springer, v. 2014, n. 1, p. 1–12, 2014. Citado na página 27.

FALUDI, R. *Building wireless sensor networks: with ZigBee, XBee, arduino, and processing*. [S.l.]: "O'Reilly Media, Inc.", 2010. Citado 5 vezes nas páginas 4, 32, 33, 36 e 42.

GERCOM. Implementação do protocolo aodv. July 2003. (Accessed on 11/15/2016). Disponível em: <http://www.gercom.ufpa.br/index.php?option=com_osdownloads&view=item&id=9&Itemid=346>. Citado 2 vezes nas páginas 78 e 90.

GHADIMI, E. et al. Opportunistic routing in low duty-cycle wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, ACM, v. 10, n. 4, p. 67, 2014. Citado na página 25.

GODSK, T.; KJÆRGAARD, M. B. High classification rates for continuous cow activity recognition using low-cost gps positioning sensors and standard machine learning techniques. In: SPRINGER. *Industrial Conference on Data Mining*. [S.l.], 2011. p. 174–188. Citado na página 50.

GUO, Y. et al. Animal behaviour understanding using wireless sensor networks. In: IEEE. *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. [S.l.], 2006. p. 607–614. Citado na página 48.

GUO, Y. et al. Using accelerometer, high sample rate gps and magnetometer data to develop a cattle movement and behaviour model. *Ecological Modelling*, Elsevier, v. 220, n. 17, p. 2068–2075, 2009. Citado na página 48.

HART, M. *TinyGPS++ / Arduiniana*. 2014. <http://arduiniana.org/libraries/tinygpsplus/>. (Accessed on 08/22/2016). Citado na página 64.

HELWATKAR, M. A.; RIORDAN, D.; WALSH, J. Sensor technology for animal health monitoring. In: *Proceedings of the 8th International Conference on Sensing Technology. Liverpool, UK*. [S.l.: s.n.], 2014. Citado 3 vezes nas páginas 11, 50 e 51.

JESUS, L. d. Identificação do comportamento bovino por meio do monitoramento. *Dissertação de Mestrado, Universidade Federal de Mato Grosso do Sul*, 2014. Citado 9 vezes nas páginas 10, 11, 12, 28, 34, 52, 53, 62 e 64.

JUANG, P. et al. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In: ACM. *ACM Sigplan Notices*. [S.l.], 2002. v. 37, n. 10, p. 96–107. Citado 2 vezes nas páginas 24 e 48.

KASHNIYAL, J.; MANDORIA, H. Energy efficiency analysis between aodv, leach and leach-e using castalia. *International Journal of P2P Network Trends and Technology (IJPTT)*, v. 1, n. 3, p. 240–244. Citado na página 90.

KENNELLY, R. J. Ieee standards for physical and data communications. *Biomedical instrumentation & technology/Association for the Advancement of Medical Instrumentation*, v. 30, n. 2, p. 172–175, 1995. Citado na página 31.

KILGOUR, R. J. In pursuit of "normal": A review of the behaviour of cattle at pasture. *Applied Animal Behaviour Science*, Elsevier, v. 138, n. 1, p. 1–11, 2012. Citado na página 50.

KWONG, K. H. et al. Practical considerations for wireless sensor networks in cattle monitoring applications. *Computers and Electronics in Agriculture*, Elsevier, v. 81, p. 33–44, 2012. Citado 2 vezes nas páginas 19 e 23.

LEONARDI, I. *Coordenadas topográficas X Coordenadas UTM*. 2016. (Accessed on 11/15/2016). Disponível em: <http://www.apeaesp.org.br/index.php/artigos-mobile/292-coordenadas-topograficas-x-coordenadas-utm?start=2>. Citado na página 80.

LEVIS, P. et al. Tossim: Accurate and scalable simulation of entire tinyos applications. In: ACM. *Proceedings of the 1st international conference on Embedded networked sensor systems*. [S.l.], 2003. p. 126–137. Citado na página 12.

LIU CHRISTOPHER M. SADLER, P. Z. M. M. T. Implementing software on resource-constrained mobile sensors: Experiences with impala and zebranet. In: *Proceedings of the 2nd international conference on Mobile systems, applications, and services*. [S.l.: s.n.], 2004. p. 256–269. Citado na página 49.

LIU, T.; MARTONOSI, M. Impala: A middleware system for managing autonomic, parallel sensor systems. In: ACM. *ACM SIGPLAN Notices*. [S.l.], 2003. v. 38, n. 10, p. 107–118. Citado na página 49.

LOMBA, L. F. D. *Identificação do Comportamento Bovino a partir dos Dados de Aceleração do Animal e Monitoramento da Luminosidade do Ambiente*. Dissertação (Mestrado) — FACOM - UFMS, 2015. Citado 6 vezes nas páginas 10, 11, 12, 34, 52 e 69.

LOUREIRO, A. A. et al. Redes de sensores sem fio. In: SN. *Simpósio Brasileiro de Redes de Computadores (SBRC)*. [S.l.], 2003. p. 179–226. Citado 4 vezes nas páginas 4, 15, 16 e 17.

MACHADO, K. et al. A routing protocol based on energy and link quality for internet of things applications. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 13, n. 2, p. 1942–1964, 2013. Citado na página 90.

MENEZES, G. *Modelo e Algoritmos para a Definição da Densidade, Cobertura e Conectividade em uma Rede de Sensores sem Fio*. Tese (Doutorado) — Dissertação de mestrado, Instituto de Ciência da Computacão, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil, 2004. Citado na página 15.

MEZZALIRA, J. C. et al. Aspectos metodológicos do comportamento ingestivo de bovinos em pastejo. *Revista Brasileira de Zootecnia*, scielo, v. 40, p. 1114 – 1120, 05 2011. ISSN 1516-3598. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1516-35982011000500024&nrm=iso>. Citado 2 vezes nas páginas 66 e 75.

MIXIM. 2014. Disponível em: <http://mixim.sourceforge.net/>. Citado na página 12.

NAKAMURA, F. G. *Planejamento dinâmico para controle de cobertura e conectividade em Redes de Sensores sem Fio Planas*. Dissertação (Mestrado), 2003. Citado 2 vezes nas páginas 14 e 16.

NS-2. 2014. Disponível em: <http://www.isi.edu/nsnam/ns/>. Citado na página 12.

NUNES, S.; DAVID, G. Uma arquitectura web para serviços web. *XATA 2005-XML: Aplicações e Tecnologias Associadas*, 2005. Citado na página 61.

OLIVEIRA, M. T. P. de. Análise comportamental de bovinos baseada em trajetórias semânticas aplicada à pecuária de precisão. *Dissertação de Mestrado, Universidade Federal de Mato Grosso do Sul*, 2013. Citado 3 vezes nas páginas 10, 12 e 52.

OMNET++. 2014. Disponível em: <http://www.omnetpp.org/>. Citado 2 vezes nas páginas 12 e 27.

OPENELEC. 2015. Disponível em: <http://en.wikipedia.org/wiki/OpenELEC>. Citado na página 57.

PERKINS, C.; BELDING-ROYER, E.; DAS, S. *Ad hoc On-Demand Distance Vector (AODV) Routing*. [S.l.], 2003. (Accessed on 11/15/2016). Disponível em: <https://www.ietf.org/rfc/rfc3561.txt>. Citado na página 21.

RAMYA, C. M.; SHANMUGARAJ, M.; PRABAKARAN, R. Study on zigbee technology. In: IEEE. *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*. [S.l.], 2011. v. 6, p. 297–301. Citado 2 vezes nas páginas 30 e 31.

RAPP, A. *andrewrapp/xbee-arduino: Arduino library for communicating with XBee radios in API mode*. 2016. (Accessed on 11/15/2016). Disponível em: <https://github.com/andrewrapp/xbee-arduino>. Citado na página 65.

RASPBIAN. 2015. Disponível em: <https://www.raspbian.org/>. Citado na página 57.

RASPBMC. 2015. Disponível em: <https://www.raspberrypi.org/blog/tag/raspbmc/>. Citado na página 57.

SANTOS, S. T. dos. *REDES DE SENSORES SEM FIO EM MONITORAMENTO E CONTROLE*. Dissertação (Mestrado), 2007. Citado 2 vezes nas páginas 4 e 30.

SKYTRAQ TECHNOLOGY, INC. *Venus638FLPx GPS Receiver - Data Sheet*. 2011. Disponível em: <http://cdn.sparkfun.com/datasheets/Sensors/GPS/Venus/638/doc/Venus638FLPx_DS_v07.pdf>. Citado na página 55.

SOARES, S. A. F. Rede de sensores sem fio para localização e monitoramento de pequenos ruminantes. *Monografia Curso de Graduação em Engenharia da Computação*, 2012. Citado na página 29.

STEHLíK, M. Comparison of simulators for wireless sensor networks. *MASARYK UNIVERSITY - FACULTY OF INFORMATICS*, 2011. Citado na página 12.

STRICKLIN, W.; KAUTZ-SCANAVY, C. The role of behavior in cattle production: a review of research. *Applied Animal Ethology*, Elsevier, v. 11, n. 4, p. 359–390, 1984. Citado 2 vezes nas páginas 50 e 100.

SUHONEN, J. et al. *Low-Power Wireless Sensor Networks: Protocols, Services and Applications*. [S.l.]: Springer Science & Business Media, 2012. Citado 2 vezes nas páginas 18 e 19.

TILAK, N. B. A.-G. S.; HEINZELMAN., W. Infrastructure tradeoffs for sensor networks. *In Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, ACM Press, p. 49–58, 2002. Citado 3 vezes nas páginas 4, 19 e 20.

WARK, T. et al. The design and evaluation of a mobile sensor actuator network for autonomous animal control. In: ACM. *Proceedings of the 6th international conference on Information processing in sensor networks*. [S.l.], 2007. p. 206–215. Citado na página 49.

YAGOUTA, A. B.; GOUISSEM, B. B. Compromises between quality of service metrics and energy consumption of hierarchical and flat routing protocols for wireless sensors network. *Sensors & Transducers*, IFSA Publishing, SL, v. 206, n. 11, p. 15, 2016. Citado na página 90.

ZHANG, P. et al. Hardware design experiences in zebranet. In: ACM. *Proceedings of the 2nd international conference on Embedded networked sensor systems*. [S.l.], 2004. p. 227–238. Citado na página 49.

# Apêndices

# APÊNDICE A – Arquivo de configuração do sorvedouro

```
serial=/dev/ttyUSB0
baud=9600
writecsv=true
exportjson=false
savetodb=true
db_driver=org.postgresql.Driver
db_user=postgres
db_senha=postgres
db_url=jdbc:postgresql://localhost:5432/rssf
```

# APÊNDICE B – Arquivo de configuração do cenário simulado no Castalia

```
SN.numNodes = 9
sim-time-limit = 72245s


SN.field_x = 678
SN.field_y = 1060


SN.node[0..8].MobilityManagerName = "GpsMobilityManager"
SN.node[0].MobilityManager.PositionsFile = "sorvedouro.ini"
SN.node[1].MobilityManager.PositionsFile = "colar1.ini"
SN.node[2].MobilityManager.PositionsFile = "colar2.ini"
SN.node[3].MobilityManager.PositionsFile = "colar3.ini"
SN.node[4].MobilityManager.PositionsFile = "colar4.ini"
SN.node[5].MobilityManager.PositionsFile = "colar5.ini"
SN.node[6].MobilityManager.PositionsFile = "colar6.ini"
SN.node[7].MobilityManager.PositionsFile = "colar7.ini"
SN.node[8].MobilityManager.PositionsFile = "colar8.ini"
SN.node[*].MobilityManager.collectTraceInfo = true


SN.node[1..].ResourceManager.baselineNodePower = 370 # in mW

SN.node[1..5].ResourceManager.initialEnergy = 29304 # em joules
SN.node[6..8].ResourceManager.initialEnergy = 79920 # em joules


SN.node[0].Application.isSink = true
SN.node[1].Application.startupDelay = 822
SN.node[2].Application.startupDelay = 588
SN.node[3].Application.startupDelay = 712
SN.node[4].Application.startupDelay = 389
SN.node[5].Application.startupDelay = 470
```

```
SN.node[6].Application.startupDelay = 909
SN.node[7].Application.startupDelay = 33162
SN.node[8].Application.startupDelay = 1


SN.node[*].Communication.MACProtocolName = "TunableMAC"
SN.node[*].Communication.MAC.dutyCycle = 0
SN.node[*].Communication.MAC.collectTraceInfo = true


SN.node[0].Communication.Radio.RadioParametersFile =
"../Parameters/Radio/XBEES2C.txt" # sorvedouro
SN.node[1..5].Communication.Radio.RadioParametersFile =
"../Parameters/Radio/XBEES2C.txt"
SN.node[6..8].Communication.Radio.RadioParametersFile =
"../Parameters/Radio/XBEES2B.txt"
```

# Anexos

# ANEXO A – Bibliotecas do arduino utilizadas para a montagem do nó sensor

- Time.h (http://playground.arduino.cc/code/time): Biblioteca de tempo que adiciona funcionalidade de cronometragem ao Arduino. Utilizada para a formatação de data e hora à partir dos dados do *timestamp* do GPS.
- XBee.h (https://code.google.com/p/xbee-arduino/): Biblioteca Arduino para se comunicar com XBees em modo API, com suporte tanto para Série 1 (802.15.4) e Série 2 (ZB Pro / ZNet).
- TinyGPS++.h (http://arduiniana.org/libraries/tinygpsplus/): TinyGPS ++ é uma biblioteca Arduino para analisar fluxos de dados NMEA fornecidos por módulos GPS.