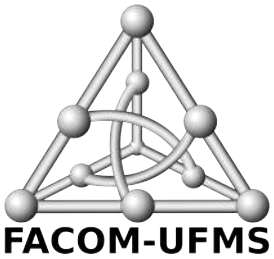
Implementação de um nó sorvedouro de uma RSSF aplicada à pecuária

André Luiz Diniz da Silva



Orientador: Luciano Gonda

Setembro de 2015

Resumo

O uso de Redes de Sensores Sem Fio (RSSF) para monitoramento de dados ambientais e fisiológicos é cada vez mais comum na área de pecuária. Além do ambiente hostil, um dos problemas enfretados é a transmissão dos dados do campo para servidores localizados em escritórios. Este trabalho trata do desenvolvimento de um nó sorvedouro para a coleta de dados de um sistema de monitoramento para pecuária. Estes dados serão coletados por meio dos nós sensores e enviados para o nó sorvedouro, e então serão transmitidos para um servidor de dados. Estes dados ficarão disponíveis para utilização em estudos de comportamento animal e influência do ambiente na vida animal. Este projeto foi desenvolvido utilizando o protocolo ZigBee e micro-computadores de baixo consumo como o Raspberry Pi.

Palavras-chaves: estação base; sensores; RSSF; Raspberry Pi.

Abstract

Use of Wireless Sensor Networks (WSN) for monitoring environmental and physiological data is increasingly common in the livestock area. Besides hostile environment, one of the problems faced is the transmission of field data to servers located in offices. This work deals with the development of a sink node to collect data from a monitoring system for livestock. These data will be collected by the sensor nodes and sent to the sink node, and then be transmitted to a data server. These data will be available for use in animal behavior studies and environment influence on wildlife. This project was developed using Zigbee protocol and low consumption micro-computers like Raspberry Pi.

Keywords: sink node; sensors; WSN; Raspberry Pi;

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Recentemente avanços em comunicações sem fio e a minituarização de componentes eletrônicos têm contribuído para implementações de Redes de Sensores Sem Fio (RSSF). Uma RSSF é composta por uma grande quantidade de nós, na qual cada nó possui um baixo consumo de energia, baixo custo de produção, um ou mais sensores, processador, memória, fonte de energia e um rádio de comunicação sem fio [39]. A facilidade da implantação deste tipo de rede em grandes áreas com baixo custo tem contribuído para o uso desta tecnologia em várias aplicações como vigilância [5], agricultura de precisão [10] [24] e pecuária. Ná área de pecuária as RSSF começam a ser usadas em controle e em aplicações de monitoramento, como cercas virtuais para sistemas de pastoreio extensivo [8], estudo do comportamento animal [2], [25], e monitoramento da saúde animal [31].

Na área agrícola, um modelo de cultura que possui grande potencial para o uso de RSSFs e vem sendo uma vertente de pesquisas na área agrícola brasileira é o sistema de Integração Lavoura-Pecuária-Floresta (ILPF). A ILPF tem como um dos objetivos a recuperação de áreas de pastagens degradadas agregando, na mesma propriedade, diferentes sistemas produtivos, como os de grãos, fibras, carne, leite e agroenergia. É uma estratégia que visa a produção sustentável por meio da integração de atividades agrícolas, pecuárias e florestais, realizados na mesma área, em cultivo consorciado, em sucessão ou rotacionado, buscando efeitos sinérgicos entre os componentes do agroecossistema, contemplando a adequação ambiental, a valorização do homem e a viabilidade econômica [9]. Uma solução para aumentar a qualidade da integração e a produtividade da área, possibilitando tomadas de decisões e verificação de dados relevantes, é a utilização de RSSFs para auxiliar a atuação automatizada e a coleta de informações do ambiente em integração. Alguns fatores chave para o sucesso de uma aplicação dedicada ao monitoramento do ambiente destinado à ILPF são, baixo custo devido ao grande número de nós para monitoramento ambiental e animal, gerenciamento eficiente de energia para o funcionamento do sensor durante um tempo aceitável e a independência de outros hardwares que podem expandir os custos e reduzir a mobilidade. A utilização da tecnologia de sensores em ILPF visa resolver problemas resultantes da integração das culturas realizadas nesta modalidade como (*i*) Manejo mais complexo por envolver um maior número de variáveis; (*ii*) Tipos diferentes de classificação do "ótimo" para cada tipo de cultura; (*iii*) Necessidade de alto investimento.

Para auxiliar no processo de coleta de dados em uma ambiente ILPF a tecnologia de RSSF pode ser utilizada para coleta em diversas regiões simultaneamente. Nesta tecnologia, existem nós especiais que podem fazer processamento denominados de *sink node*. O problema que este trabalho pretende abordar consiste em projetar e implementar um ***sink node***, também conhecido como nó sorvedouro, que faz parte de um projeto de monitoramento de ambientes e comportamento animal que visa o aumento de produtividade do gado de corte em ambientes onde está implantado a ILFP. Este ***sink node*** foi construído utilizando tecnologias de micro-computadores e tecnologia de comunicação Zigbee. Possui como objetivo obter as informações coletadas do ambiente alvo, processar algumas delas, caso necessário, e repassá-las para um servidor para o possibilitar a tomada de decisão. Os objetivos específicos do projeto são:

O1 Definição de uma arquitetura de hardware para um nó sorvedouro a ser implementada em um ambiente agrícola;

O2 Projeto de um nó sorvedouro robusto utilizando micro-computadores;

O3 Implementação, por meio de tecnologia micro-computadores, de um nó sorvedouro (*sink node*) atuando como uma interface entre a rede de sensores sem fio e a rede de computadores.

1.1 Organização do Texto

O restante deste trabalho está organizado da seguinte maneira: o Capítulo 2 é compostos pelas definições e conceitos envolvendo RSSF e sua arquitetura e comunicação, detalhamento sobre o padrão de protocolo IEEE 802.15.4 e suas camadas física e de acesso ao meio, o padrão ZigBee e principais conceitos sobre seu funcionamento. Além disso são apresentados trabalhos relacionados envolvendo a vasta utilização de RSSFs e ILPF. No Capítulo 3 são apresentados as definições de arquitetura, o modelo proposto do trabalho bem como seu desenvolvimento e construção. No Capítulo **??** são apresentadas as considerações finais do trabalho e também são elencados trabalhos futuros para o melhoramento deste projeto.

Capítulo 2

Fundamentação Teórica

A primeira aparição do tópico de redes sem fio ocorreu na década de 80 com as redes sem fio para envio de voz. Já na década de 90, especificadamente em 1999, surgiram redes sem fio para transmissão de dados. Hoje nós estamos vivendo a terceira revolução da comunicação sem fio, também conhecida como Internet das Coisas(IoT) [34]. Esta revolução utiliza Redes de Sensores sem Fio e tecnologias de controle para diminuir a distância entre o mundo físico humano e o mundo virtual dos eletrônicos. A ideia principal é que em um futuro próximo será possível monitorar e responder em tempo hábil a intempéries climáticos, problemas de tráfego nas grandes cidades, melhoramento e aperfeiçoamento da eficiência da produção de bens e o melhoramento da qualidade de vida de pessoas e animais. Tudo isso será possível através do desenvolvimento de Redes de Sensores sem Fio. Este capítulo apresenta os aspectos fundamentais para o desenvolvimento deste trabalho. Apresenta conceitos de Redes de Sensores sem Fio como sua arquitetura e seus modos de comunicação, conceitos sobre o padrão IEEE 802.14.5 e sua importância para redes de baixo consumo, o padrão ZigBee, conceitos sobre ILPF e sua importância para o desenvolvimento sustentável.

2.1 Redes de Sensores sem Fio

Redes de Sensores Sem Fio (RSSFs), conforme demonstrado na Figura 2.1, são uma especialização de uma rede *ad-hoc* (*Mobile Ad hoc Network* - MANET), e são compostas por vários dispositivos móveis autônomos e com fonte limitada de energia, e se comunicam por meio de um canal de comunicação sem fio[30].

As redes *ad hoc*, em geral, utilizam topologia *mesh*, na qual não há necessidade da existência de um elemento central responsável pela comunicação, tais como switches ou roteadores. Cada nó de rede deve realizar tarefas de transferência de dados e de gerência de comunicação, tais como iniciar seu funcionamento, executar o processo de descoberta de outros nós com os quais a comunicação é possível e estabelecer a conexão para transferência de dados. Por utilizar comunicação sem fio, existem algumas restrições tais como: limitação do raio do sinal; limitações na largura de banda; maior vulnerabilidade a inter-

ferências de outros dispositivos que utilizam comunicação sem fio e consumo de energia. A limitação de recursos de energia normalmente é ocasionada pela arquitetura adotada para os dispositivos, pois de acordo com a aplicação desejada, os componentes necessitam ser miniaturizados, reduzindo a possibilidade de acoplamento de fontes maiores de energia [30].

Dispositivos dispostos em uma MANET possuem dois tipos de transmissão: dois nós podem se comunicar entre sim diretamente se estiverem próximos o suficiente (ponto-a-ponto); ou indiretamente por meio de nós intermediários que retransmitem os pacotes (*multihop*). Para que a comunicação *multihop* seja eficiente, é necessário a existência de um algoritmo de roteamento capaz de coordenar as trocas de pacotes na rede.

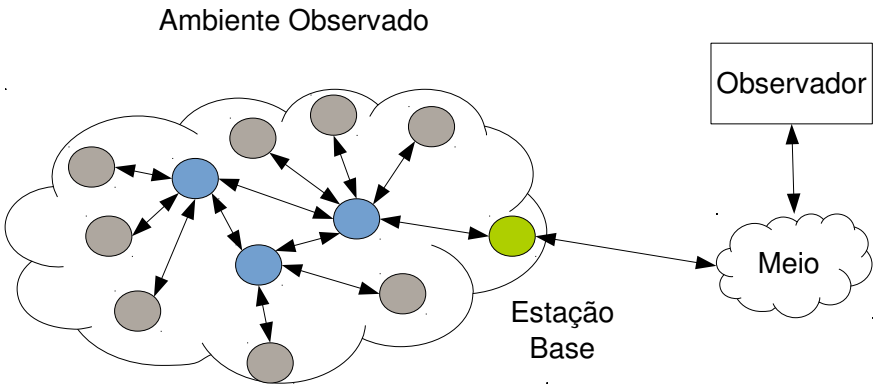


Figura 2.1: Exemplo de uma RSSF.

Os nós sensores podem executar tarefas de sensoriamento, mas também realizar tarefas de pré-processamento das informações coletadas. Devido à baixa disponibilidade de recursos, em especial de processamento e energia, o processamento de informações para tomada de decisões é realizado por um nó especial, denominado de estação base, pois esta apresenta maior disponibilidade de recursos. Podem haver também nós capazes de controlar algum parâmetro do ambiente monitorado, estes são chamados de nós atuadores.

A capacidade de sensores em uma RSSF pode variar de acordo com a capacidade do nó, ou seja, um simples sensor pode monitorar um simples fenômeno físico, enquanto dispositivos mais complexos podem combinar varias técnicas de sensoriamento (acústico, óptico, magnéticos, dentre outros). Eles podem também se diferenciar em capacidades de comunicação, isto é, podem se comunicar por meio de infravermelho, ultra-som ou tecnologias de radiofrequência com latências e taxas de transferências diferentes [36]. Enquanto sensores simples somente coletam e comunicam dados sobre o ambiente observado, sensores mais robustos podem executar processamentos e outras funções. Por exemplo, eles podem formar *backbones* de comunicação que podem ser usados por sensores mais simples para enviar dados à estação base. Por fim, alguns dispositivos da rede podem ter acesso a tecnologias de suporte, como Sistema de Posicionamento Global (*Global Positioning System*) possibilitando a obtenção da localização dos dispositivos.

O modelo funcional de uma RSSF pode ser separado em um ciclo de vida com cinco grupos de atividades: estabelecimento, manutenção, sensoreamento, processamento e comunicação [12]. Na atividade de estabelecimento são realizadas a disposição da rede e sua formação. Os sensores podem ser dispostos aleatoriamente sobre a área a ser monitorada e despertam para a formação da rede. A atividade de manutenção tem como objetivo prolongar o tempo de vida da rede, reduzir a imprevisibilidade e atender aos requisitos da aplicação. A manutenção é presente em todo ciclo de vida da rede, podendo ser reativa, preventiva, corretiva ou adaptativa de acordo com o evento ocorrido. A atividade de sensoreamento está ligada à percepção do ambiente e à coleta de dados. A atividade de processamento pode ser divida em dois tipos, o processamento de suporte, que envolve processamento de dados relacionados ao gerenciamento, comunicação e manutenção da rede, e o processamento da informação que envolve processamento sobre os dados coletados pelos sensores. A atividade de comunicação está relacionada ao envio e recebimento de dados de/para outros sensores.

2.1.1 Arquitetura de RSSF

A arquitetura de uma RSSF está diretamente ligada aos requisitos da aplicação para que será utilizada. Foram identificados uma série de características de RSSF que tem impacto direto sobre a arquitetura e decisões de projeto, que surgem naturalmente de um conjunto de requisitos da aplicação e limitações tecnológicas.

Aplicações de RSSF incluem análise de micro-organismos marinhos, monitoramento de casas, monitoramento de ambientes, monitoramento sísmicos, dentre outros. Essas aplicações mostram uma grande diversidade, porém um número significativo de características gerais é compartilhada pela maioria de aplicações em RSSFs, independente do tipo de sensores e dos objetivos da aplicação. Essas características incluem baixo custo, tamanho reduzido, consumo de energia reduzido, robustez, flexibilidade, resistência a erros e falhas, modos autônomos de operação e privacidade e segurança. Alguns objetivos de projeto e de arquitetura são descritos a seguir [12]:

- Tamanho reduzido: a intenção é fornecer componentes com um poder computacional satisfatório mantendo o tamanho mais reduzido possível de acordo com os requisitos da aplicação.
- Baixo consumo de energia: a capacidade, tempo de vida e desempenho de um sensor são limitados pela energia. Os sensores devem ser capazes de ficarem ativos por um longo período de tempo sem precisar recarregar suas baterias devido ao difícil acesso que muitas vezes torna inviável a troca do sensor.
- Operações intensivas concorrentes: a fim de obter o desempenho global satisfatório, os dados dos sensores devem ser capturados do sensor, podendo ser processados e/ou comprimidos, e então enviados pela rede simultaneamente.
- Diversidade no projeto e uso: o fato de que cada nó deve possuir tamanho reduzido, baixo consumo de energia faz com que estes sensores sejam específicos para

determinada aplicação, porém, o projeto de um sensor deve facilitar o reuso.

- Operações robustas: devido ao fato de que sensores poderão ser dispostos em ambientes grandes e, às vezes, hostis (florestas, uso militar e o corpo humano), estes sensores devem ser tolerantes a falhas e erros, portanto nós sensores necessitam de habilidades de testes, calibração e reparo.
- Segurança e privacidade: cada sensor deve possuir segurança suficiente para prevenir acessos não autorizados, ataques e danos não intencionais dentro de um nó sensor.
- Compatibilidade: o custo do desenvolvimento de software normalmente é o maior dentro do custo do sistema. Portanto, é importante o reuso de código por meio de compatibilidade binária ou tradução binária.
- Flexibilidade: é necessário acomodar mudanças funcionais e temporais. Flexibilidade pode ser atingida por meio de dois métodos: programabilidade (por meio do emprego de processadores programáveis); e reconfiguração, usando plataformas baseadas em FPGAs ou micro-computadores.

2.1.2 Arquitetura de Comunicação em RSSF

Para a comunicação entre os nós é definida uma pilha de protocolos composta pela camada de aplicação, camada de transporte, camada de rede, camada de enlace, e camada física além de conter planos de gerenciamento de energia, mobilidade e gerenciamento de tarefas, conforme apresentado na Figura 2.2.

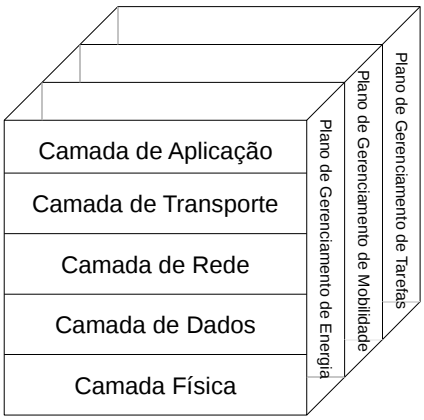


Figura 2.2: Pilha de camadas e planos gerenciais em RSSF [3].

A camada de transporte tem a função de manter o fluxo de dados caso a aplicação necessite. A camada de rede tem a função de encaminhar pacotes recebidos da camada de transporte. A camada física possui a função de modulação. Em conjunto com as camadas trabalham os planos de gerenciamento de energia, que tem como objetivo gerenciar os recursos energéticos do sensor, o plano de gerenciamento de mobilidade, que tem como

objetivo detectar e registrar o movimento dos sensores e o plano de gerenciamento de tarefas que faz o o balanço e a gerencia das tarefas de acordo com dadas diretivas.

Para garantir a interoperabilidade na comunicação entre os diversos nós sensores, foram criados padrões de comunicação pelo IEEE e pela ZigBee Alliance.

Dessa forma, as camadas inferiores da pilha de protocolos de RSSFs são formalizadas pelo padrão IEEE 802.15.4, que é mantida pelo IEEE (*Institute of Electrical and Electronics Engineers*). A especificação IEEE 802.15.4 é a base para as especificações ZigBee [4], ISA100.11a [27], WirelessHART [23] e MiWi [37], cada uma das quais estende o padrão por meio do desenvolvimento das camadas superiores que não são definidas pelo IEEE 802.15.4, mas sim, pelo ZigBee Aliance, como mostrado na Figura 2.3. O Padrão IEEE 802.15.4 e a especificação ZigBee serão descritos nas Seções 2.2 e 2.3, respectivamente.

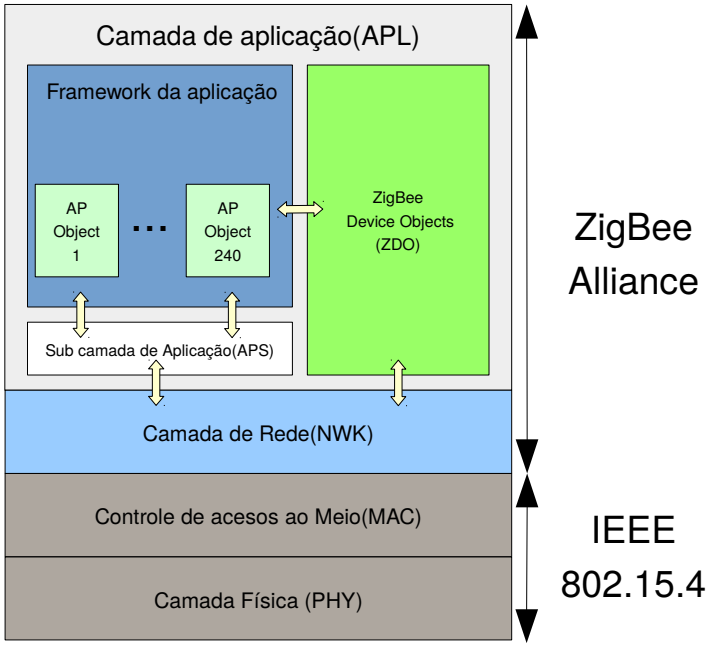


Figura 2.3: Pilha de protocolos ZigBee e IEEE 802.15.4 [4].

2.2 Protocolo IEEE 802.15.4

Com o sucesso de redes locais sem fio (WLANs), a comunidade de redes sem fio focou em melhorar as capacidades de WLANs para desenvolver novas abordagens para uma gama crescente de aplicações envolvendo redes sem fio de baixo alcance. Movidos pela necessidade de tornar possível aplicações sem fio de baixo custo o grupo de trabalho 4, dentro do grupo de trabalho 802.15 do IEEE foi formado para desenvolver os padrões referente a redes pessoais sem fio com baixa taxa de transmissão (*low-rate wireless personal area networks* - LR-WPAN) conhecido como IEEE 802.15.4 [1]. O objetivo deste grupo de trabalho era fornecer um padrão que possui como características principais baixa complexidade, baixo custo e baixo consumo para conectividade de dispositivos sem fio de

baixo custo, estacionários, portáveis ou móveis.

Favorecendo o baixo custo e o baixo consumo, o padrão IEEE 802.15.4 possibilita a criação de aplicações nas áreas industriais, de agricultura, automobilísticas, residenciais e na área médica. Sem o padrão IEEE 802.15.4 essas aplicações se tornariam inviáveis ou teriam um grande custo, pois teriam que usar soluções proprietárias. A intenção do protocolo IEEE 802.15.4 é oferecer soluções onde soluções existentes de redes pessoais sem fio (WPANs) são caras e o desempenho de tecnologias como Bluetooth não são necessárias. As LR-WPANS complementam outras tecnologias WPANs provendo capacidades de baixo consumo com um baixo custo portanto possibilitando aplicações que antes eram impraticáveis. O protocolo IEEE 802.15.4 é projetado para ser usado em uma grande variedade de aplicações que possuem como requisito principal a simplicidade na comunicação sem fio em baixa distância com pouca/limitada energia e pouca necessidade de largura de banda.

A principal característica para LR-WPANs é o baixo consumo para maximizar o tempo de vida da bateria, para isso o protocolo IEEE 802.15.4 assume que a quantidade de dados a ser transmitida é baixa e rarament transmitidos para manter um baixo ciclo de trabalho. Além disso, a estrutura do pacote foi desenvolvida para adicionar o mínimo possível de *overhead* sobre os dados transportados.

Existem dois tipos de dispositivos em uma rede sem fio IEEE 802.15.4: *Full-Function Devices* (FFDs) e *Reduced-Function Devices* (RFDs). Um FFD é capaz de executar todas as funções descritas no padrão IEEE 802.15.4 e pode executar qualquer função na rede. Um RFD, diferente do FFD, possui capacidades limitadas como reduzido poder de processamento e capacidade de memória. Além disso, um FFD pode se comunicar com qualquer outro dispositivo da rede, porém um RFD pode se comunicar somente com um FFD.

Em uma rede IEEE 802.15.4 um FFD pode executar três tipos de papéis [12]: Coordenador, Coordenador PAN (*Personal Area Network*) e dispositivo comum. O Coordenador é o dispositivo capaz de retransmitir mensagens. Se o Coordenador é também o principal controlador de uma Rede de Área Pessoal (PAN) adquire a função de Coordenador PAN. Se um dispositivo não está realizando a função de Coordenador é chamado de dispositivo comum.

Um dispositivo dentro de uma rede LR-WPAN IEEE 802.15.4 é composta pela camada física (PHY - *physical layer*) e a subcamada de acesso ao meio (MAC - *Medium Accesss Control*) que provê acesso para os canais fisicos para todos os tipos de transferência de dados de forma confiável.

2.2.1 Camada Física

A camada física do protocolo IEEE 802.15.4 suporta três frequências de radiocomunicação conforme mostrado na Tabela 2.1: a frequência de 2.450MHz (com 16 canais), a frequência de 950MHz (com 10 canais) e a frequência de 868MHz (com um canal somente). Todas usam o modo de acesso de Sequência Direta de Espalhamento do Espectro (DSSS). Para modulação, a frequência de 2.450MHz utiliza o O-QPSK (*Offset Quadrature Phase*

Shift Keying), enquanto as frequências de 950 e 868MHz utilizam o BPSK (*Binary Phase Shift Keying*). Além de operações de Ligar/Desligar, a camada física suporta funções para seleção de canal, estimativa de qualidade do enlace, avaliação de canais livres e estimativa do uso de energia.

	2.450 MHz	**950 MHz**	**868 MHz**
Taxa de transferência	250 kbps	40 kpbs	20 kbps
No de Canais	16	10	1
Modulação	O-QPSK	BPSK	BPSK

Tabela 2.1: Características da camada física do protocolo IEEE 802.15.4.

2.2.2 Camada de Controle de Acesso ao Meio

A camada de Controle de Acesso ao Meio (MAC) no protocolo IEEE 802.15.4 controla o acesso ao canal de rádio empregando o mecanismo CSMA-CA. Define dois tipos de nós [6]: Dispositivos com Funções Reduzidas e Dispositivos Totalmente Funcionais. FFDs estão equipados com total suporte a funções da camada MAC, o que possibilita a eles funcionarem como coordenadores de rede ou dispositivos finais. A camada MAC tem como objetivo promover a conectividade sem fio com dispositivos fixos, portáteis e móveis que estejam dentro ou entrem no espaço operacional pessoal das redes WPAN. A camada MAC também é responsável pelo controle de fluxo via entrega de quadros de confirmação, validação de quadros, manter a sincronização da rede, controlar associações, administrar a segurança do dispositivo e garantir o mecanismo de *slot* de tempo [15].

2.3 Padrão ZigBee

ZigBee é um padrão que define as camadas superiores da pilha de protocolos IEEE 802.15.4 [4]. A camada de rede (NWK) é responsável por organizar e prover roteamento sobre uma rede *multihop* (contruído em cima das funcionalidades do padrão IEEE 802.15.4), e a camada de aplicação (APL) provê um **framework** para o desenvolvimento e comunicação de aplicações distribuídas2.3. A APL é composta por suporte à aplicação (*Application Framework*), *ZigBee Device Objects* (ZDO), e a sub-camada de aplicação (APS). O suporte à aplicação pode ter até 240 objetos de aplicação (APOs) que são módulos de aplicações parte de uma aplicação ZigBee definidas pelo usuário. O ZDO provê serviços que permite que os objetos de aplicação descubram seus objetos vizinhos e se organizem em uma aplicação distribuída. A APS oferece interfaces para serviços de dados e segurança para os APOs e ZDOs. Um resumo da pilha de protocolos ZigBee é demostrado na Seção 2.1.2.

2.3.1 Redes ZigBee

Ligeiramente diferente das nomenclaturas de papéis utilizadas no IEEE 802.15.4, vistos na Seção 2.2, o padrão ZigBee usa terminologias diferentes para representar os tipos de dispositivos (Figura 2.4), porém com funcionalidades semelhantes. Um ZigBee *Coordinator* (ZC) é um Coordenador PAN IEEE 802.15.4, um ZigBee *Router* (ZR) é um dispositivo que pode agir como um Coordenador IEEE 802.15.4 e um ZigBee *End Device* (ZED) é um dispositivo que não é um coordenador nem um roteador, comumente com poder de processamento reduzido e menores capacidades de armazenamento porém apresenta um custo reduzido [12]. Na Figura 2.4 são apresentados os papéis dos dispositivos nos padrões IEEE 502.15.4 e ZigBee.

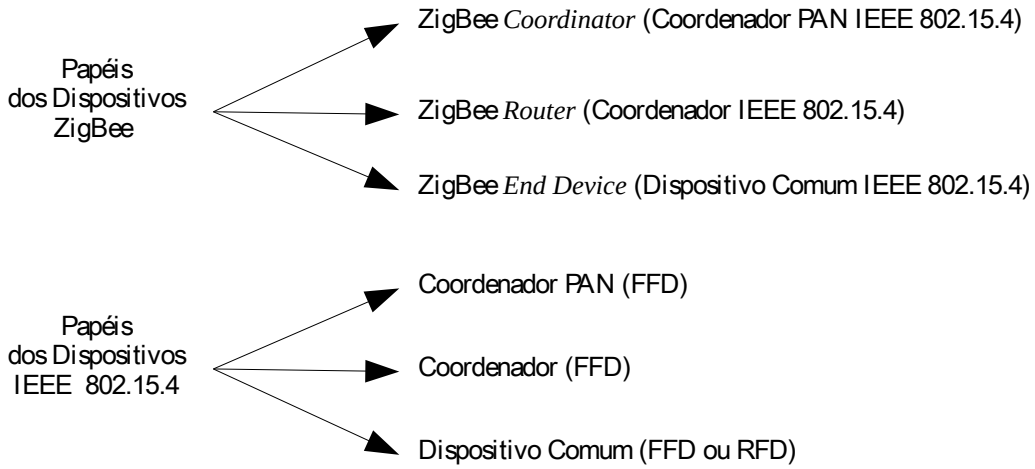


Figura 2.4: Papéis dos dispositivos nos padrões IEEE 802.15.4 e ZigBee

A formação da rede ZigBee é gerenciada pela camada de rede ZigBee e deve possuir uma das duas formações especificadas na IEEE 802.15.4: estrela ou *peer-to-peer*. As diferentes funcionalidades dos dispositivos permitem uma variedade de maneiras de se instalar a rede, de acordo com a necessidade da aplicação, podendo ser mais robusta, mais econômica, centralizadora ou distribuída. Essas características determinarão o tipo de topologia a ser empregado.

Na topologia estrela, mostrada na Figura 2.5, cada dispositivo da rede pode somente se comunicar com um coordenador central, geralmente denominado coordenador PAN. Um cenário típico na formação de uma rede estrela é um FFD, programado para ser um coordenador PAN, que é ativado e então começa estabelecendo a rede. A primeira coisa que o coordenador PAN realiza é a seleção de um identificador PAN único que não é utilizado por nenhuma outra rede na sua área esférica de influência — a região em volta do dispositivo em que seu rádio consegue comunicar com sucesso com outros dispositivos.

Na topologia *peer-to-peer* (Figura 2.6), cada dispositivo pode comunicar com qual-

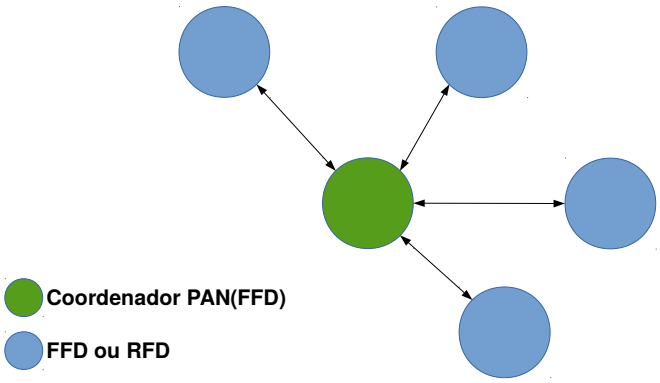


Figura 2.5: Topologia de rede estrela

quer outro dispositivo se este estiver perto o suficiente para estabelecer um canal de comunicação. Qualquer FFD em uma topologia *peer-to-peer* pode fazer o papel de um coordenador PAN. Uma maneira de decidir qual dispositivo será o coordenador PAN é selecionar o primeiro dispositivo que começa a se comunicar como coordenador PAN. Todos os dispositivos em uma topologia *peer-to-peer* que realizam a comutação de pacotes são FFD já que os dispositivos RFD não são capazes de comutar pacotes. Entretanto um dispositivo RFD pode ser parte da rede, porém só se comunicar com somente um dispositivo (um coordenador ou um roteador) na rede.

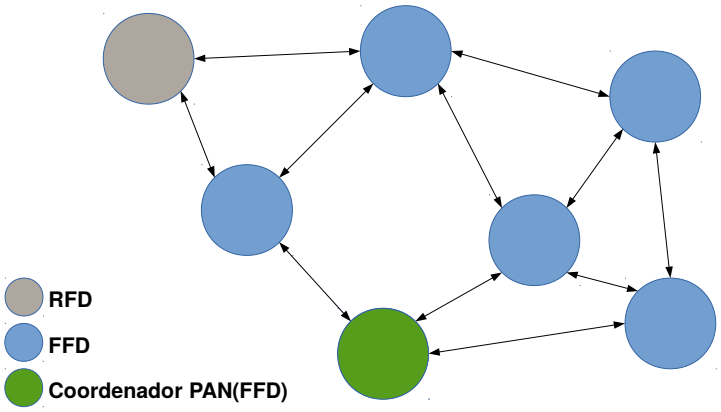


Figura 2.6: Topologia de rede *peer-to-peer*

Uma rede *peer-to-peer* pode possuir diversos formatos definindo restrições de comunicação nos dispositivos da rede. Se não existir nenhuma restrição a rede *peer-to-peer* é conhecida como uma rede *mesh*. Outra forma de rede *peer-to-peer* que o ZigBee suporta é a topologia em árvore (*tree topology*). Neste caso, conforme visto na Figura 2.6, um coordenador ZigBee (coordenador PAN) estabelece a rede inicial. Os roteadores ZigBee formam os galhos(*branches*) e comutam as mensagens. Os dispositivos ZigBee ZEDs atuam como folhas e não participam da comutação de pacotes.

Na Figura 2.7 mostra-se um exemplo de como esta topologia pode ajudar a maximizar

o alcance da rede e ultrapassar barreiras como cercas, cercas vivas, muros ou obstáculos naturais. Por exemplo o dispositivo A precisa enviar uma mensagem para o dispositivo B, porem há uma barreira entre eles que torna difícil a penetração do sinal. Esta topologia ajuda o envio da mensagem comutando os pacotes por meio dos nós roteadores até o dispositivo B (*multihopping*). Esta abordagem possui um ponto negativo, quanto maior o alcance da rede maior será a latência para a transmissão da mensagem.

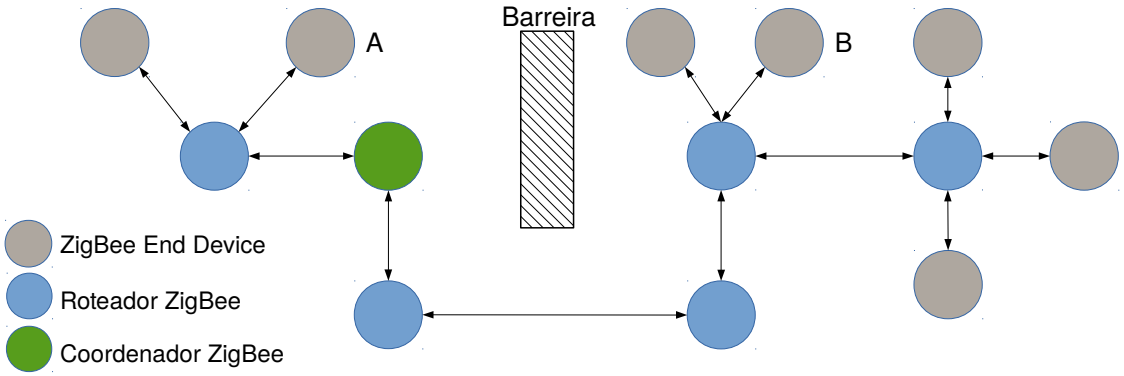


Figura 2.7: Topologia de rede árvore

2.4 Aplicaçções de RSSF

Redes de Sensores sem Fio oferecem uma combinação poderosa de sensoriamento distribuído, computação e comunicação. Eles são úteis a inúmeras aplicações e, ao mesmo tempo, oferecem inúmeros desafios devido às suas peculiaridades, principalmente as restrições energéticas rigorosas a que os nós sensores são normalmente submetidos. Várias plataformas de hardware já foram concebidos para testar as muitas ideias geradas pela comunidade de pesquisa e implementar aplicativos para praticamente todos os campos da ciência e da tecnologia. Nesta seção será apresentada soluções em que RSSF foram utilizadas para o melhoramento de soluções ou até mesmo a criação delas.

2.4.1 Redes Inteligentes de Energia

As redes de energia não são somente uma parte importante da indústria da energia elétrica, mas também uma importante parte no esforço em busca da sustentabilidade. Com o aumento gradual da dependência de energia elétrica, há também o aumento da demanda pela qualidade e confiabilidade da mesma.

Uma rede de energia inteligente possibilita a criação de aplicações com impactos profundos, provendo a capacidade de integração de mais fontes de energia renováveis com segurança; veículos elétricos e geração distribuída na rede; a entrada de energia ao consumidor final de forma confiável e eficiente; automação para reconfiguração da rede em caso de falha; tornar possível o maior controle por parte do consumidor sobre seu consumo de energia e sua participação mais ativa no mercado de energia.

Exemplos de aplicações em Redes Inteligentes de Energia são o monitoramento de linhas de transmissão [38], monitoramento inteligente e aviso prévio de problemas em subestações de energia [42], monitoramento online e aviso prévio de problemas nos sistemas de distribuição e sistemas inteligentes de consumo de energia.

2.4.2 Redes de Abastecimento

O consumo de água mais que triplicou desda década de 50 [29]. O forte crescimento da população mundial juntamente com o aumento da classe-média continuará a criar demanda para os recursos limitados do planeta. Um exemplo de recurso chave neste contexto é a disponibilidade de água tratada. Cada vez mais governos e grandes empresas estão procurando diminuir seu *footprint*[1] no ambiente através de sistemas inteligentes de monitoramento, no caso hídrico, de qualidade da água em sua fonte, eficiência de suas redes de distribuição e uso inteligente do recurso em suas instalações [14] [26].

2.4.3 Transportes Inteligentes

Sensoriamento sem fio em sistemas de transportes inteligentes diferem em diversos pontos dos conceitos tradicionais de RSSF. Na maioria dos casos, sensores podem depender de algum tipo de infraestrutura para o fornecimento de energia, ou seja o fato de eficiência energética (requisito normalmente primário em RSSF) torna-se um requisito secundário nesses sistemas.

Sistemas de transportes inteligentes podem ser subdivido em 2 categorias [40]:

- Redes de sensores estacionários: fazem parte de um veículo ou parte da infraestrutura de tráfego.
- Redes de sensores flutuantes: as entidades da rede agem como sensores. Esta categoria compreende a aplicações de monitoramento e otimização de fluxos de transportes de serviços, veículos e pessoas.

Soluções inteligentes de tráfego podem auxiliar nos problemas de locomoção urbana enfrentado pelas grandes cidades monitorando otimizando e atuando sobre as linhas de transportes para oferecer a opção mais viável para o tráfego de entidades no sistema [19] [35].

2.4.4 Ambientes, agricultura e pecuária

RSSF possuem uma grande quantidade de aplicações com requisitos e características diferentes. Por exemplo, sensores ativos, tais como, sonares necessitam um poder computacional muito mais elevado para o processamento de sinais do que sensores passivos como detectores de fumaça. Sensores para aplicações móveis precisam de um poder de

[1] *Footprint*: é uma medida da demanda humana por recursos do ecossistema terrestre.

processamento ainda maior devido aos complexos algoritmos de rede e seus protocolos. O monitoramento ambiental e animal une mobilidade com grande massa e processamento dados, tornando necessário tecnologias que possibilitam aplicações em tempo real em RSSF.

O monitoramento animal, utilizando de nós sensores construídos usando plataformas GPS pode ser visto em [17]. Neste trabalho é construído um protótipo de colar com aproximadamente 1,15 Kg para uma RSSF denominada ZebraNet. Esta rede tinha como objetivo o monitoramento da vida animal selvagem de zebras na região de Laikipia na região central do Quênia.

Primeiramente foi realizado estudo sobre o comportamento das zebras, visto que para o design do ZebraNet era necessário entender como os nós se comportariam, pois isto afetaria o projeto do hardware e do protocolo utilizado. Foram identificados três tipos de padrões de movimentação: pastando, pastando-andando e em movimento.

O protótipo inicial é composto de GPS, memória, um CPU uma antena de rádio para curta distância e uma antena de rádio para longa distância. Para minimizar a quantidade de componentes foi utilizado um modulo GPS (GPS-MS1E) que contém um processador de 32 bits Hitachi SH1, suporte a Entrada/Saída e uma memória de 1 MB dos quais 640 KB são disponibilizados para dados de usuário e restante utilizado pelo *firmware*. Usando o processador do colar a posição do animal era coletada, por meio de GPS, em intervalos de cinco minutos com uma precisão de cinco a dez metros e armazenados na memória.

Com o espaço disponível é possível armazenar até 110 dias de posições sem compressão de dados e até 300 dias com compressão. O processador também coordena o gerenciamento da comunicação através dos dois rádios disponíveis. A princípio a escolha de dois rádio foi motivada pelo controle do uso de energia. O rádio de curto alcance (aproximadamente 100 metros) tem um baixo consumo de energia e é utilizado para comunicação *peer-to-peer* entre os colares utilizados pela zebra, já o rádio de longo alcance (aproximadamente 8 quilômetros) possui um maior consumo de energia e é utilizado para comunicação com a estação base para coleta de dados.

Em [41] é realizado um trabalho de aprimoramento do hardware dos sensores da rede ZebraNet. O consumo excessivo de energia no protótipo (chamado de Versão 0.1) construído em [17] levou os projetistas a melhorarem o projeto do hardware. Na versão seguinte, denominada de Versão 1, foi substituído o microcontrolador visto que o consumo do microcontrolador utilizado na Versão 0.1, que era o disponibilizado pelo GPS, não era satisfatório. Foi adicionado um microcontrolador de baixo consumo que controla todos os periféricos além de uma memória externa de 2 Mbits. Com isso o desenvolvimento de software para os sensores foi simplificado.

Na versão 2, foi realizado somente um melhoramento da distribuição dos componentes, porém o novo *layout* consumia metade do espaço requirido na versão 1. A versão 3, considerada a versão final, é um sistema completo com integração de carregadores solares e baterias de lítio. O uso de carregadores solares é devido ainda ao alto consumo dos sensores, estimativas foram realizadas e concluíram que seria necessário uma bateria de aproximadamente 10 quilogramas para os sensores operarem por um ano seguido, o que

não é aceitável, portanto o sistema utiliza uma bateria recarregável com pequenos painéis solares. O peso adicional da bateria de dos painéis é irrisório para o animal (200 gramas) tornando aceitável para a aplicação.

No cenário brasileiro a Embrapa Gado de Corte é pioneira em pesquisas de relacionadas ao setor agropecuário. O setor agropecuário vem sofrendo grandes transformações motivadas pelo aumento nos custos de produção e maior competitividade, exigindo aumento na produtividade da atividade, qualidade e rentabilidade, sem comprometer o meio ambiente. Para atingir tais objetivos, o sistema de integração que incorporam atividades de produção agrícola, pecuária e florestal, em dimensão espacial e/ou temporal, buscando efeitos sinérgicos entre os componentes do agroecossistema para a sustentabilidade da unidade de produção, contemplando sua adequação ambiental e a valorização do capital natural [22].

A integração de lavoura com pecuária e com florestas assim como a associação de criações e cultivos é realizada pelo homem desde os primórdios da agricultura, muita vezes em situações de conflito por interesses divergentes [9]. Quando feita de forma coerente, resulta em aumento de produção por unidade de área bem, como benefícios ambientais. Agricultura Sustentável é um conceito quem vem sendo amplamente discutido e divulgado, mas para que a sustentabilidade ocorra de fato, é necessário beneficiar toda a sociedade. Ou seja, a exploração agropecuária sustentada deve manter ou melhorar a produção, com vantagens econômicas para os produtores rurais, sem prejuízo ao meio ambiente e em benefício de toda a sociedade.

Os sistemas de ILPF, com manejo adequado das culturas e pastagens, podem proporcionar substanciais aumentos na produção, principalmente em áreas degradadas e pouco produtivas. Pela adoção destes sistemas gera-se benefícios ambientais, como proteção da vegetação nativa, conservação do solo e recursos hídricos, e também promove o desenvolvimento socioeconômico regional. Assim, para que a implantação de sistemas produtivos sustentáveis seja efetivamente conduzida, é necessário desenvolver metodologias e tecnologias para monitorar e avaliar, com baixos custos, as áreas desses sistemas depois de implantados.

Nesse sentido, as geotecnologias, que envolvem imagens de satélite, fotografias aéreas, sistemas de informações geográficas e sistemas de posicionamento global por satélite (GPS), ocupam papel relevante na identificação, no monitoramento, na consolidação e na expansão de áreas que adotam sistemas de integração, seja em escala local, regional ou nacional. Outra tecnologia pouco utilizada, porém promissora, é a utilização de RSSFs para o monitoramento animal e ambiental [18]. No monitoramento animal informações sobre a saúde animal, localização e movimentação diária combinadas com informações de monitoramento ambiental como pressão, temperatura e umidade proporcionam o avanço de estudos do comportamento, de custos associados a integração, e de novas tecnologias. Além disso, RSSFs podem ser utilizadas para atuarem sobre o ambiente de integração auxiliando por exemplo a melhor eficiência de irrigação, controle de abastecimento de água para o rebanho e manejo automáticos de animais.

Em [32] é realizado o desenvolvimento de um nó sensor para coletar dados de ambiência de forma automatizada, medir os elementos climáticos e calcular os índices de conforto

térmico para bovinos a partir desses elementos. Este trabalho é realizado utilizando nós sensores Arduino em ambiente de ILFP com a transmissão de dados realizada através de rádios ZigBee.

Em [11] é proposto a criação de um nó sensor com capacidade para coletar dados por meio de uma rede de sensores sem fio, monitorar os bovinos e inferir por meio de um sistema o comportamento animal. Cada nó possui um sensor de GPS preso a um colar e colocado no bovino que circula em uma pastagem totalmente georreferenciada. Os dados gerados pelo sensor GPS são armazenados em um cartão memória e após coletados são disponibilizados em um sistema de informação. É apresentado também um sistema de apoio na observação do comportamento dos animais em campo e uma abordagem para classificação automática das atividades realizadas pelos bovinos dividida em andando, comendo/buscando, em pé e deitado. O objetivo é disponibilizar um sistema para o monitoramento de bovinos, com a finalidade de obter informações sobre o comportamento dos animais.

Em [21] é proposto o uso de sensores de aceleração e sensores de luminosidade, a fim de prover funcionalidades que auxiliem a produção e a gestão pecuária, a partir da identificação dos padrões de comportamento animal e sua relação com as características da luminosidade do ambiente.

2.5 Considerações Finais

A união de tecnologias como micro-computadores e o monitoramento simultâneo de ambientes e do comportamento animal em área de integração possibilita o aumento da produtividade e ganhos em sustentabilidade e qualidade animal. Utilizados os conhecimentos apresentados nessa seção foi construído um nó sensor, também denominado nó sorvedouro, que tem como missão o pré processamento e o envio de dados para servidores de dados. Este processo será relatado no Capitulo seguinte.

Neste capítulo foram apresentados diversos conceitos relacionados a RSSFs, ao padrão IEEE 802.15.4, ao ZigBee e ILPF. Observa-se a relevância do tema deste projeto, que poderá gerar tecnologias para o melhoramento animal e ambiental em ambientes de integração.

Capítulo 3

Implementação do Nó Sorvedouro

A análise de comportamento animal vem sendo utilizada na pecuária de precisão com o objetivo de aumentar a qualidade de vida do animal, gerando assim um ganho significativo de produtividade. O uso de tecnologias como RSSF podem ser utilizadas para analisar variáveis ambientais e comportamentais para atingir os objetivos apresentados na Seção 2.4.4. Este trabalho faz parte de um projeto maior que pretende monitorar o comportamento animal, analisando várias variáveis, como posicionamento animal, temperatura, radiação, pressão e umidade do ambiente, que podem influenciar na produtividade em um ambiente de integração Lavoura-Pecuária-Floresta. Além disso, oferece maneiras simplificadas para a automatização de ações a serem tomadas para modificação de variáveis que influenciam o ambiente.

O objetivo geral deste projeto foi o desenvolvimento de um nó sorvedouro para uma RSSF, responsável pela transmissão de dados entre o ambiente monitorado e um servidor de dados. Conforme apresentado na Figura 3.1, o produto desenvolvido é um nó sensor (representado pelo nó **GW**) que assumira a função de nó sorvedouro e será utilizado como interface de comunicação entre uma RSSF e um servidor responsável por armazenar, manipular e disponibilizar os dados.

A partir deste escopo apresentado foram definidas atividades realizadas para atingir o objetivo de criação de uma estação base que funciona como um roteador entre a rede de monitoramento e o servidor de dados: definição da arquitetura, implementação e testes.

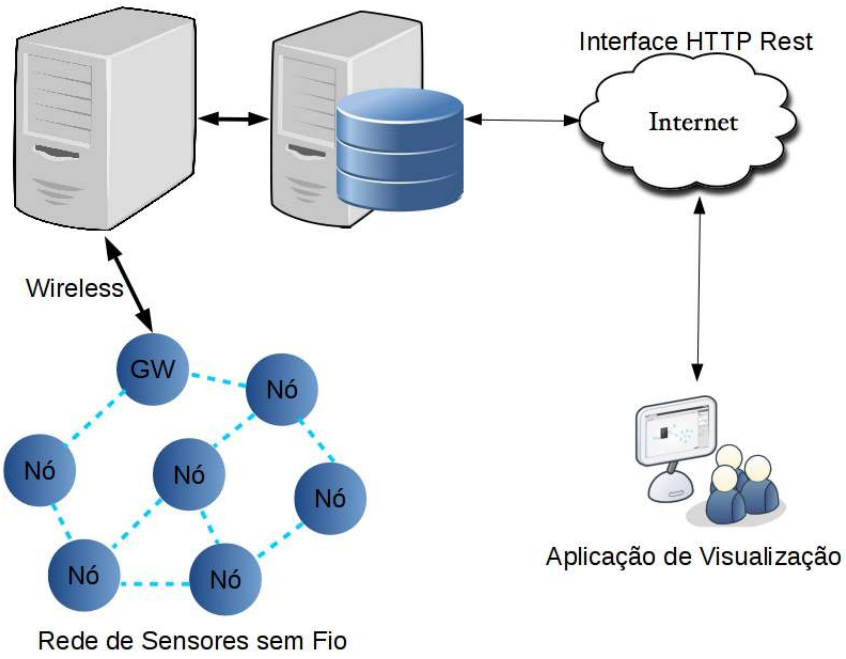


Figura 3.1: Escopo geral do trabalho realizado

A definição de arquitetura foi divida em 2 fases: fase de definição de arquitetura de hardware e fase de definição da arquitetura de software. Na fase de definição de arquitetura de hardware foram elencadas possíveis tecnologias a serem utilizadas. A primeira alternativa pesquisada foi a implementação deste projeto utilizando tecnologias de hardware reprogramável, mais especificamente o uso de *Field-Programmable Gate Array* (FPGA). A FPGA escolhida foi a FPGA da marca Altera, modelo DE2-115, mas após análises de custo constatou-se que o custo de implementação deste projeto não era viável e o ganho de performance não seria significativo. A segunda alternativa pesquisada foi a implementação deste projeto utilizando Arduino Mega 2560 que é um micro-controlador baseado no *ATMega2560* de 16Mhz, além disso possui um total de 54 pinos GPIO (*General Propose Input/Output*) digitais e 16 pinos de entrada analógico. O Arduíno foi descartado devido à baixa quantidade de memória e a não implementação nativa de *threads*. A terceira alternativa foi a busca de *credit card-sized single-board computers*, micro-computadores de tamanho reduzido. Estes dispositivos pequenos são computadores construídos em uma única placa de circuito contendo processador, memória e interfaces de entrada e saída ideais para implementação de projetos embarcados. Entre eles estão o Raspberry Pi, Intel Galileo e PandaBoard. A Tabela 3.1 mostra um comparativo entre estes dispositivos.

	Rasbperry Pi	Intel Galileo	PandaBoard
Processador	ARM 700 MHz single-core	Intel Quark X1000 400 MHz	ARM Cortex-A9 1 Ghz dual-core
Memória	512MB	256MB	1GB
USB	4 USBs	Não Nativo	2 USBs
Rede	10/100 Mbit/s Ethernet	10/100 Mbit/s Ethernet	10/100 Mbit/s Ethernet
Preço	25$	70$	220$

Tabela 3.1: Comparativo de dispositivos selecionados para o projeto.

Dentre elas, o Raspberry Pi foi escolhido para o desenvolvimento do nó sorvedouro, pois foi constatado que o mesmo apresenta características que atendem às necessidades deste trabalho: tamanho reduzido, interface de rede *onboard*, número vasto de interfaces USB para periféricos, fácil integração com sistema operacionais (Linux), baixo custo, facilidade de compra em território nacional e uma comunidade ativa para busca de melhoramentos e soluções.

Na fase de definição de arquitetura de software inicialmente foram elencados os requisitos funcionais e não funcionais da aplicação. Em seguida, foram definidas as características de implementação, conforme apresentadas na Tabela 3.2.

Requisitos funcionais e não funcionais	
RF001	O gateway deve realizar a comunicação entre uma rede ZigBee e uma rede TCP
RF002	O gateway deve suportar pacotes definidos pelo usuário
RF003	O gateway deve colher pacotes de uma rede Zigbee e encaminhá-los para uma rede TCP
RF004	O gateway deve processar os pacotes recebidos, ajustados os dados conforme a definição de cada pacote
RF005	O gateway deve possuir estrutura modular
RF006	O gateway deve possuir registro dos pacotes que transitam pelo mesmo
RNF001	O gateway deve possuir mecanismos para responder a falhas de conexão
RNF002	Em caso de falhas de conexão (por parte da rede TCP) o gateway deve armazenar os dados e depois retransmiti-los
RNF003	O gateway deve ser um módulo do SO

Tabela 3.2: Requisitos funcionais e não funcionais da aplicação

Na fase de implementação foi realizada a construção do roteador utilizando os frameworks xbee-api [28] e log4j [13] e a linguagem de programação JAVA. O Raspberry Pi possui total suporte à linguagem de programação escolhida, e também possui suporte a python, node.js, C e C++. Nas Seções 3.1 a 3.4 são descritos os processos de implementação e testes do nó sorvedouro.

3.1 Arquitetura Proposta

Baseado nos requisitos previamente citados, o primeiro passo para a construção do roteador foi a definição da arquitetura do pacote com qual o roteador irá trabalhar. Essa estrutura deve corroborar com a ideia de transmissão de múltiplos dados em um só pacote e de tipos variados. Para atender este requisito, foi criado um tipo de pacote que encapsula os sub-pacotes de dados. Basicamente, o pacote pai determina a quantidade de sub-pacotes que serão transmitidos em determinado momento, que pode variar de um sub-pacote até 8 sub-pacotes. A estrutura do sub-pacote possui um identificador único

do seu tipo, um identificador único da sua origem, a data ou variável temporal de sua transmissão e seus dados. A Figura 3.2 consta o detalhamento dos pacotes.

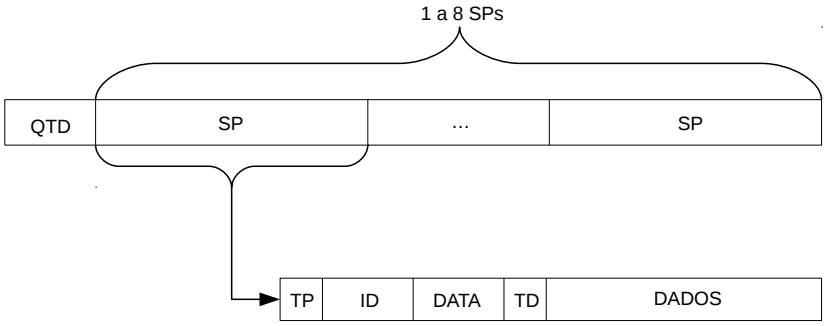


Figura 3.2: Escopo geral do trabalho realizado

Cada campo possui um tamanho pré-definido:

- **TP** ou TipoPacote: este campo é responsável por identificar o tipo de pacote para que o roteador possa efetuar o processamento de suas informações e possui o tamanho de 1 byte, possibilitando assim 2^8 tipos de pacotes.
- **ID** ou Identificação: este campo é responsável pela identificação única do nó da rede de sensores sem fio e possui tamanho de 8 bytes.
- **DATA**: este campo é responsável por determinar o tempo de coleta do dado, não necessariamente pode ser uma data mas sim algum registro temporal ou de ordem do dado coletado e possui tamanho de 8 bytes.
- **TD** ou TamanhoDado: este campo é responsável pela informação da quantidade de bytes presente no próximo campo do pacote e possui o tamanho de 1 byte.
- **DADOS**: neste campo são colocados os dados relevantes originários da extração de informações do ambiente observado.

Após a definição do pacote, iniciou-se a fase de criação da arquitetura do roteador. A criação do roteador proposto baseou-se em três princípios essenciais: simplicidade na construção de código, modularidade e eficiência na transmissão de dados. O roteador possui três categorias de componentes:

- Componente de configuração (*Configuration*): neste componente são realizadas as inicializações de arquivos de configurações e o armazenamento de filas de acessos.
- Módulos (*Module*): componente genérico que realiza algum tipo de manipulação e/ou transformação de dados.

- Inicializador (*BootLoader*): neste componente é realizada a inicialização do roteador. Neste componente a configuração é carregada, as filas são inicializadas, e os módulos criados e inicializados
- Pacotes (*Packages*): são os pacotes definidos pelo usuário

3.1.1 Classe *Module*

No desenvolvimento do roteador foi usada a ideia da modularidade. Todo componente ativo, ou seja, que realiza alguma operação com dados, deriva de um componente em comum, o *Module*. Todo *Module* pode ser descrito em uma máquina de estados e cada estado descreve uma fase da inicialização, realização de um trabalho ou encerramento do Módulo. A Figura 3.3 representa esta máquina de estados.

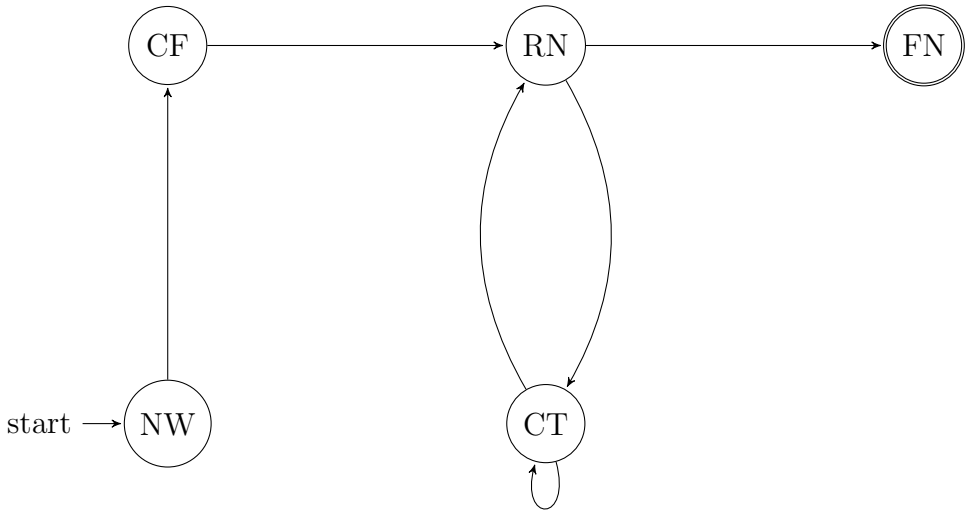


Figura 3.3: Diagrama de estados de um módulo.

O estado **NW** corresponde ao estado *NEW* ou inicial. O Módulo permanece nesse estado quando é construído, está construção é realizada pelo Inicializador. O estado **CF** corresponde ao estado *CONFIG* ou estado de configuração, neste estado deve ser realizado qualquer processo de configuração do módulo. Por exemplo, aberturas de arquivos de configuração, abertura de conexão com a interface ZigBee e abertura de conexão com a interface de rede. Além disso, cabe a este estado a configuração das fila de comunicação onde o módulo retirará dados, e colocará dados manipulados. Neste estado o módulo ainda não está apto a execução de tarefas.

O estado **RN** corresponde ao estado *RUNNING* ou estado de funcionamento. Este módulo usualmente contém um laço infinito e enquanto não houver alteração de estado provocada por estímulos externos, serão processadas as informações que lhe são cabíveis. A alteração de estado pode ocorrer em razão de fatores externos, como algum problema de comunicação, e quando isso acontece o módulo transaciona para um estado especial, se implementado, o estado de CT.

O estado **CT** corresponde ao estado *CONTINGENCY* ou estado de contingência. Este é um estado opcional, e o módulo pode ou não implementá-lo. Neste estado acontece o tratamento de dados caso o fluxo esperado do módulo não aconteça. Por último, temos o estado **FN** que corresponde ao estado *FINISHED* ou finalizado, no qual o módulo é finalizado, ou seja, suas configurações são desfeitas, recursos desalocados e suas possíveis conexões são fechadas e finalizadas.

Na Figura 3.4 observa-se a estrutura genérica do módulo implementada em Java. Esta classe abstrata funciona como um contrato, isto é, para qualquer classe de módulo derivada desta e presente no roteador, seus métodos definidos como *abstract* deverão obrigatoriamente ser implementados nas classes filhas. Todo módulo deverá chamar o construtor padrão da classe, este construtor representa o estado *NEW*. O próprio construtor já está encarregado da transição de estado por meio da chamada do método *configure()*. O método *configure* obrigatoriamente deve ser ser implementado pelas classes derivadas do módulo para que possa ser realizada a configuração individual de cada módulo. O método *run()* representa o estado *RUNNING* e sua implementação também é obrigatória por todas as classes derivadas do módulo. O método *contigency()* representa o estado *CONTINGENCY* e sua implementação não é obrigatória. A implementação *default* do módulo espera até a alteração do estado do módulo por um ator externo. Por fim temos o método *finish()* que representa o estado *FINISH* e que deve ser obrigatoriamente implementado pelas classes derivadas.

```
public abstract class GenericModule extends Thread {

    protected ModuleState state;

    protected GenericModule() {
        this.state = ModuleState.NEW;
        this.configure();
    }

    public abstract void configure();

    public abstract void finish();

    public ModuleState getModuleState() {
        return state;
    }

    public abstract void run();

    public void stopModule() {

        this.state = ModuleState.FINISHED;

    }

    public void contingency() {

        while (this.state == ModuleState.CONTINGENCY) {
            try {
                // do nothing
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }

}
```

Figura 3.4: Implementação genérica de um módulo.

3.1.2 Classe *Configuration*

O roteador possui todas as variáveis de configurações necessárias para o seu funcionamento concentrados na classe *Configuration*. Esta classe é responsável pela leitura dos arquivos de configurações e criação das estrutura necessárias. O roteador possui trés tipos de arquivos de configuração. O primeiro arquivo é o *config.properties*, que contém as configurações de ambientes como portas de instalação do modulo XBee, e configurações de endereço como, por exemplo, o IP e porta de destino dos pacotes colhidos na rede. O se-

gundo arquivo, denominado *queue.properties*, contém configurações de dimensionamento das filas que são usadas para comunicação intra-módulos. E o terceiro arquivo é onde é realizada a associação dos tipos do pacotes com seus nomes completos, ou seja, o caminho total do pacote de dados aonde será encontrado a classe que representa aquele tipo de pacote. Em todos os arquivos as configurações estão dispostas no formato <CHAVE > = <VALOR >.

A Figura 3.5 mostra exemplos de um arquivo de configuração de ambiente. As propriedades permitidas no arquivo de configuração de ambiente são as seguintes:

1. **xbee.port**: esta propriedade determina a localização dentro do sistema operacional da interface de comunicação com o módulo XBee. O Raspberry Pi utiliza o linux como sistema operacional então comumente o módulo XBee localiza-se na porta /dev/ttyUSBX onde X representa o número da porta.

2. **xbee.speed**: esta propriedade determina a velocidade de comunicação do módulo XBee, por padrão esse valor é de 9600bps.

3. **server.ip**: esta propriedade determina o endereço de IP para onde os pacotes devem ser enviados, observa-se que o servidor proveniente deste endereço de IP deverá possuir um software específico para o recebimento dos pacotes providos pelo Roteador. Este software será detalhado na Seção 3.2.1.

4. **server.port**: esta propriedade determina a porta de destino do servidor por onde as informações deverão ser trafegadas.

5. **command.port**: esta propriedade determina a porta que o roteador receberá os pacotes de comando que serão repassados para os nós da rede.

6. **timeout**: esta propriedade determina o tempo máximo de tentativa de qualquer módulo para a colocação e a retirada de dados em suas respectivas filas. Observa-se que este tempo é dado em milissegundos.

7. **processor.file**: esta propriedade determina a localização do arquivo temporário do módulo de processamento de dados caso haja problemas de conectividade com o servidor de dados. Caso esta propriedade não seja colocada no arquivo é assumido um valor padrão para o caminho do arquivo que é a pasta **ct** dentro da pasta raiz onde se encontra instalado o software.

```
xbee.port=/dev/ttyUSB0
server.ip=192.168.17.66
server.port=7100
command.port=7101
xbee.speed=9600
timeout=9600
processor.file=/opt/outer/ct.msc
```

Figura 3.5: Exemplo de arquivo de configuração.

A Figura 3.6 mostra exemplo de um arquivo de configuração de filas. As filas utilizadas pelo roteador serão detalhadas na seção 3.2. Neste arquivo, deverão ser disponibilizados os tamanhos de cada fila utilizada pelo o roteador. A forma do arquivo deverá ser $NOME\_DA\_FILA = TAMANHO$. No caso da figura apresentada temos que, por exemplo, a fila de saída do módulo XBee, determinada pela propriedade *xbee.module.out.queue*, terá 512 posições. A definição dos nomes das filhas fica a cargo do usuário, neste caso o módulo deverá saber quais filas deve consultar.

```
xbee.module.out.queue=512
xbee.module.in.queue=128
processor.module.out.queue=128
processor.module.in.queue=128
network.module.in.queue=128
```

Figura 3.6: Exemplo de arquivo de configuração de filas.

A Figura 3.7 mostra um exemplo de um arquivo de configuração de pacotes. Como os pacotes são criados serão detalhados na seção 3.1.4. Neste arquivo deverá ser disponibilizado os tipos, representado por números inteiros, de cada pacote que pode circular pelo roteador. A forma do arquivo deverá ser $TIPO\_DO\_PACOTE = NOME\_COMPLETO\_DA\_CLASSE$. No caso da figura apresentada temos a definição dos pacotes do tipo 0 para a classe *br.ufms.facom.router.packet.RawPacket*.

```
0=br.ufms.facom.router.packet.RawPacket
1=br.ufms.facom.router.packet.GpsPacket
2=br.ufms.facom.router.packet.TemperaturePacket
3=br.ufms.facom.router.packet.HumidityPacket
```

Figura 3.7: Exemplo de arquivo de configuração de pacotes.

Conforme apresentado na Figura 3.8 a classe *Configuration* foi desenvolvida usando o padrão de projeto *Singleton*, ou seja há somente uma instância desta classe quando o software é executado. Esta classe representa os arquivos de configuração previamente apresentados carregados em estruturas de rápido acesso. As propriedades de ambiente são carregadas quando o único objeto desta classe é construído, este processo é feito pelo método *loadEnvironmentProperties()*. As informações de ambiente carregadas ficam armazenadas em uma estrutura própria da Java API denominada *Properties*, esta classe nada mais é que um dicionário no qual o tempo de acesso a suas informações é constante ($O(1)$).

O carregamento das informações das filas e a criação das mesmas é realizado pelo método *loadQueueProperties()*, no qual é realizado um *parse* [1] das informações contidas no arquivo de configuração das filas. O carregamento das informações dos pacotes e a

[1] *Parsing* é o processo de analise de palavras e símbolos, podendo elas serem de linguagem natural ou computacional, em conformidade com regras de uma gramática formal.

criação das estruturas responsáveis pelo seu armazenamento é feito pelo método *loadPacketProperties()*. O processo é basicamente o mesmo realizados para arquivos de configurações previamente discutidos. Cada fila é também armazenada em uma estrutura de dicionário no qual a chave é seu nome e o valor é uma fila do tipo *BlockingQueue<Object>* com tamanho determinado pela informação extraída do arquivo.

```
public class Configuration {

    private static final Logger log = Logger.getLogger(Configuration.class)
        ;
    private static final Configuration configuration = new Configuration();
    public static final int DEFAULT_QUEUE_SIZE = 1024;

    private Properties properties;
    private HashMap<String, BlockingQueue<Object>> queueMap;

    protected Configuration() {
        loadEnvironmentPropeperties();
        loadQueueProperties();
        loadPacketProperties();
    }

    private void loadQueueProperties() { ... }

    private void loadEnvironmentPropeperties() { ... }

    private void loadPacketProperties() { ... }

    public String getProperty(String key) {
        return properties.getProperty(key);
    }

    public BlockingQueue<Object> getQueue(String queueName) {
        return queueMap.get(queueName);
    }

    public static Configuration getInstance() {

        return configuration;
    }

}
```

Figura 3.8: Classe de configuração.

3.1.3 Inicialização do roteador

Para a inicialização dos serviços do roteador(módulos) este conta com seu *BootLoader*. A classe *BootLoader* é responsável pela criação e inicialização dos módulos, inicialização da configuração e configurações de desligamento.

A Figura 3.9 apresenta as principais características que o *BootLoader* do roteador. Podemos observar que no seu método construtor[2] o *BootLoader* faz a chamada de métodos específicos de configuração e de criação. O método *configureRouterProperties()* é responsável por sensibilizar a classe *Configuration* e este fazer o carregamento das configurações previamente explicadas na Seção 3.1.2. O método *configureShutDown()* tem como objetivo determinar o conjunto de ações que devem ser executadas caso o roteador seja parado por um agente externo. O método *createModules()* é responsável pela criação dos módulos do roteador. Por fim temos o método *start()* que é responsável pela inicialização do roteador e monitoramento dos módulos.

```
public abstract class BootLoader {

    private Configuration configuration;
    private ArrayList<GenericModule> modules;


    public BootLoader() {

        configureRouterProperties();
        configureShutDown();

        modules = new ArrayList<>();
        createModules();

    }

    protected abstract void createModules();

    protected abstract void configureShutDown();

    protected abstract void configureRouterProperties();

    protected abstract void start() throws InterruptedException;

}
```

Figura 3.9: Representação abstrata da classe BootLoader.

3.1.4 Definição do pacote

O objetivo deste projeto foi o desenvolvimento de um nó responsável pela transmissão de informações providas por uma RSSF. Estas informações podem ser temperaturas do ambiente, umidade, dados de velocidade de vento e também é possível que essas informações sejam provenientes de dados coletados de animais no ambiente. Estas informações necessitam de uma abstração comum, para isso todo dado que chega ao roteador é convertido para um pacote conhecido pelos observadores. Este pacote contém as funções básicas, identificação e dados correspondentes.

[2]Método executado quando o objeto é criado

A Figura 3.10 mostra a classe desenvolvida para representar todas as estruturas de pacotes que o roteador está habilitado a processar. Quando um pacote é recebido pelo roteador, o *framework* utilizado para comunicar com o rádio XBee disponibiliza o pacote recebido em forma de um vetor de bytes. Este vetor então deve ser processado pelo módulo que retornará um novo pacote nos moldes correspondente. Observa-se que esta classe recebe em seu construtor as informações comuns a todos os pacotes que trafegam pela rede.

```
public abstract class GenericPacket implements Serializable {

    protected PacketType type;
    protected Long id;
    protected Long date;

    public GenericPacket(PacketType type, Long id, Long date) {
        this.type = type;
        this.id = id;
        this.date = date;
    }

    public abstract GenericPacket parse(int[] rawBytes);

    public abstract void toJSON();

}
```

Figura 3.10: Representação abstrata da classe GenericPacket.

Além disso, para cara adição de pacote deve ser feita a associação do mesmo a um tipo. Para isso é necessária a adição de uma associação no arquivo *packet.properties* conforme discutido na seção 3.1.2. O software realiza por meio de *reflections* a criação de objetos baseado no nome completo de sua classe.

3.2 Implementação Realizada

Na Seção 3.1 foi apresentada a proposta de arquitetura a ser utilizada no roteador e um arcabouço de funções e classes básicas para sua construção. Nesta seção, serão apresentadas implementações de software realizadas no roteador bem como seu funcionamento. Para realização da codificação deste projeto foi utilizado o ambiente integrado de desenvolvimento (IDE) Eclipse, que contém todas as ferramentas necessárias para o desenvolvimento, validação e testes de softwares em várias linguagens. A linguagem de programação utilizada foi o Java.

A partir das definições de módulo da Seção 3.1.1 foram construídos os seguintes módulos: *XBeeModule*, *ProcessorModule*, *NetworkModule*, *CProcessorModule* e *CommandModule*. Estes módulos são independentes e a comunicação entre eles é exclusi-

vamente realizada por troca de mensagens conhecidas através de filas de comunicação. A Figura 3.11 representa a implementação final realizada no roteador.

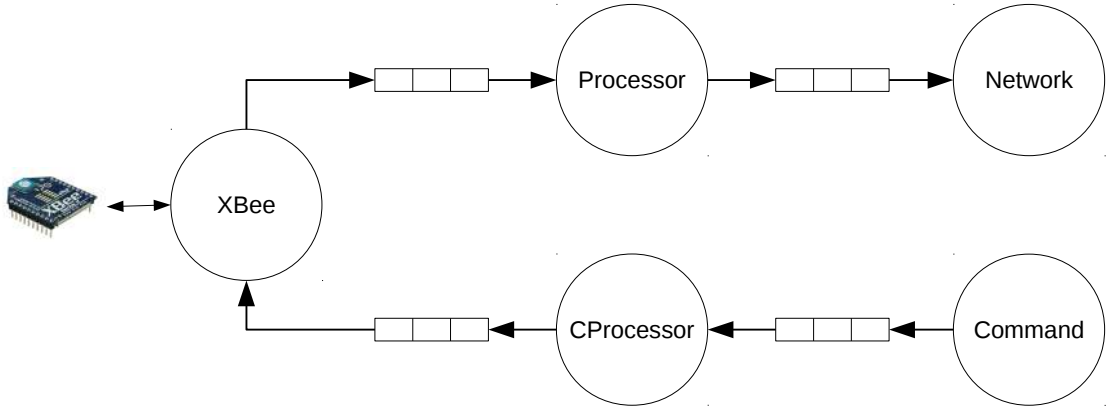


Figura 3.11: Estrutura do roteador desenvolvida.

O módulo *XBeeModule* é responsável pela comunicação com o rádio XBee. Isso é possível devido à utilização do *framework xbee-api*. Este framework é uma API Java para comunicação com os rádios XBee/Xbee-Pro (802.15.4) series 1 e series 2 (ZB/ZB Pro).

No estado de criação, são criados objetos responsáveis pela comunicação com o rádio e variáveis de controle. Além disso, o roteador requisita da classe de configuração suas filas de comunicação. No estado de configuração do módulo, o mesmo requisita uma abertura de conexão para o *framework*, passando para ele o endereço da porta onde o módulo se encontra e a velocidade desejada de comunicação. Essas informações são coletadas através de requisições para o objeto responsável por armazenar as configurações do roteador (Ver Seção 3.1.2).

Além disso, no estado de configuração o módulo requisita o tipo de rádio que está sendo utilizado para comunicação com a RSSF, para que seja possível determinar o tipo de informações que serão colocados na fila de saída do módulo.

No estado *run* o módulo entra em um processo de laço infinito até seu estado ser alterado por um ator externo. Neste momento o módulo espera um pacote da RSSF. No momento de um recebimento, é chamada uma rotina da validação do pacotes para a verificação da consistência do mesmo. Após a validação, caso erros não ocorram o pacote é colocado na fila de saída do módulo e suas informações são impressas no log.

O módulo *ProcessorModule* é responsável pelo processamento do pacote recebido do *XBeeModule* e repassa para a etapa seguinte do processo. No estado de criação variáveis de controle são inicializadas. Na fase de configuração, o módulo requisita suas filas de entrada e saída bem como requisita o caminho do arquivo para o armazenamento de informações em caso de entrada em estado de contingência. No estado *run*, o módulo aguarda a colocação de um pacote não processado na sua fila de entrada, quando esta fila é sensibilizada o módulo retira o pacote.

Neste momento, o pacote consiste em um vetor de bytes que deverá ser processado

pela classe do pacote correspondente ao seu tipo, para isso ocorrer o módulo realiza uma chamada interna de um método privado chamado *processPacket()*. Neste método é realizado o *parse* das informações contidas nesse vetor de bytes. Primeiramente é realizada a verificação da quantidade de informações legíveis (pacotes SP) contidas neste vetor, para isso é verificado o primeiro byte do vetor. Esta informação denominada **QTD** possibilita a iteração dos dados restantes, ou seja é realizado um laço que, para cada pacote **SP** dentro deste vetor de bytes, este seja processado e extraído corretamente.

A Figura 3.12 o exemplo de um pacote recebido pelo roteador em formato de vetor bytes.

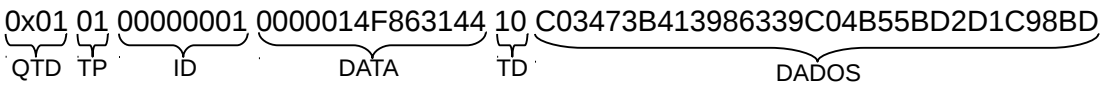


Figura 3.12: Exemplo de pacote processado pelo roteador

Observa-se que os valores dispostos neste exemplos possui valores hexadecimais. No processo de transformação de pacotes de dados a partir deste vetor de bytes, primeiramente é realizada a leitura do campo **QTD**. Este campo, como anteriormente explicado, será a quantidade de iterações para a extração dos pacotes **SP**, neste caso a quantidade é de somente um pacote **SP** para facilitar a exemplificação. Após extraída esta informação, para cada pacote **SP** é realizado uma seleção em seu tipo de pacote. Esta informação vem do próximo dado a ser lido, o campo **TP**. Neste caso, o campo **TP** contém a informação 01, que para fins de exemplificação, corresponde a um pacote de geolocalização que é denominado de *GPSPacket*.

Após identificação do tipo do pacote é realizada a leitura da informação **ID** que neste caso possui valor igual a 1. A informação seguinte é o campo **DATA** que possui informações para determinar a ordem de extração do dado do ambiente observado. Após extração destes 3 dados é realizada a criação de um objeto de pacote correspondente ao tipo extraído. Neste caso será criado um objeto da classe *GPSPacket*.

No construtor deste objeto serão passados os dados **TP**, **ID** e **DATA**. Após criação do objeto é realizada uma nova leitura do vetor de bytes da informação **TD**, esta informação é crucial para a extração do próximo dado, que são os dados propriamente ditos do pacote em questão. Neste caso o valor é 16 (10 em hexadecimal), isso significa que os próximos 16 bytes são de informações pertencentes a este pacote. Esta informação é lida em sequência e repassada para o método *parse()* do pacote, onde serão realizados processos de leitura e processamento. Estes processos variam de pacote para pacote. Após o processo de transformação de pacotes de dados a partir do vetor de bytes recebido do *XBeeModule* em objetos conhecidos estes são encaminhados para a fila de saída do módulo.

Outra funcionalidade disponibilizada neste módulo é ação de salvar dados caso aconteça algum problema de comunicação entre o roteador e o servidor que recebe os

dados. Para tal é necessária a alteração do estado do módulo por um ator externo. Quando realizado, o módulo passa a salvar os pacotes processados em um arquivo. Este processo é realizado pelo método *contingencyState()*. Neste método, para todo pacote recebido é extraída sua informação sem nenhum tipo de manipulação e colocado em um arquivo de formato CSV onde o separador é a quebra de linha. A localização deste arquivo é determinada pela propriedade *processor.file*.

Como demonstrado pela Figura 3.13 cada linha do arquivo contém um pacote que foi recebido no momento que o módulo está em estado de contingência.

```
0X010100010000014F863144E010C03473B413986339C04B55BD2D1C98BD
0X010100040000014F8BB159A010C0347AB1EFD1B493C04B4A6F170F103E
0X010100040000014F8BB5032010C0347A7C21B98E2CC04B4B0AE60D0FB3
0X010100040000014F8BB5033510C0347A7C21B98E2CC04B4B0AE60D0FB3
0X010100040000014F8BB503BB10C03480A53C650B1DC04B4E85B5B70692
0X010100040000014F8BB503E310C0347AD099E0E736C04B548F34BBC96F
0X010100040000014F8BB5032A10C0346BDE3C60D95FC04B521395083A61
```

Figura 3.13: Exemplo de arquivo de backup temporário.

O módulo, quando alterado seu estado para *run* novamente realiza a leitura deste arquivo e para cada registro faz o enfileiramento deste pacote da mesma maneira apresentada anteriormente.

O módulo *NetworkModule* é responsável pela transmissão dos pacotes processados que foram coletados pelos sensores na rede para os servidores onde estes serão armazenados. Esta comunicação é realizada através de troca de pacotes TCP realizadas por um dispositivo externo de comunicação. No estado de inicialização é realizada a criação e inicialização das variáveis de controle do módulo bem como é requisitada para o objeto de configuração sua fila. Na fase de configuração é realizada a criação de um *Socket* de comunicação. Este *Socket* utiliza as configurações *server.ip* e *server.port* disponibilizadas pelo arquivo de configuração para a criação de uma canal de comunicação com o servidor. Caso ocorra erro, o módulo notificará os demais módulos que houve um problema e tentará criar a conexão novamente a cada período determinado de tempo. Após a criação com sucesso do canal de comunicação o módulo passa para o estado *run*. Neste estado contém um laço infinito que a cada iteração é realizado a retirada de um pacote da fila de entrada, o envio deste pacote pela rede e por final é realizado uma escrita no log do status da transmissão.

O módulo *CommandModule* é responsável pelo recebimento de pacotes provenientes do servidor de dados. Estes pacotes usualmente são pacotes de configuração que tem como objetivo a configuração de variáveis nos nós sensores. No estado de inicialização é realizada a criação e inicialização de variáveis de controle, bem como é requisitada sua fila para o objeto de configuração. Na fase de configuração é realizada a criação de um servidor de rede utilizando *Sockets*. No estado *RUN* o módulo fica em um laço infinito. Neste momento para cada cliente que se conecta é realizado o procedimento de criação de uma *thread* responsável por processar as requisições do cliente. Está *thread* recebe, em

sua construção, o *socket* do cliente e a fila do módulo. Dentro é realizada o recebimento de informações do cliente e, para cada informação, é criado um pacote específico e colocado na fila do módulo. Após finalizada a comunicação com o cliente a *thread* é finalizada.

O módulo *CProcessorModule* é responsável pelo recebimento dos dados não processados enviados do módulo *CommandModule* e conversão destes em pacotes ZigBee. No estado de inicialização, é realizada a criação e inicialização das variáveis de controle do módulo. Na fase de configuração, é requisitada ao objeto de configuração sua fila correspondente. No estado *run*, o módulo aguarda a colocação de um pacote não processado na sua fila de entrada; quando esta fila é sensibilizada o módulo retira o pacote e o processa. O método responsável por este processamento é o *processPacket()*, nele é realizado a construção de um pacote intermediário que será enviado ao módulo *XBeeModule*

3.2.1 Softwares Auxiliares

Para realizar testes de funcionamento do produto construído foi necessário o desenvolvimento de softwares auxiliares para simulação de comportamentos. O primeiro foi um software que simula o recebimento de pacotes enviados pelo o nó. Foi construído um programa simples que consiste em um servidor baseado em *sockets* que espera a conexão do nó e a partir desse momento para cada pacote recebido imprime no terminal sua representação. Também foi desenvolvido um software para que possa ser enviados comandos para o nó sensor para que este possa enviar estes dados aos nós da rede. Este software consiste basicamente em um cliente *socket* que se conecta ao nó desenvolvido. Quando o software recebe um comando do usuário é realizado uma validação simples da formatação do comando neste caso o software aceita comandos na forma de chave e valor. Após validado estes comandos são enviados ao nó roteador para os demais tratamentos.

3.3 Construção do Protótipo

Após o desenvolvimento do software foi realizada a construção física do protótipo do roteador. A Tabela 3.3 demonstra os materiais utilizados.

Material	Quantidade
Raspberry Pi	1
Regulador de Tensão DC-DC LM 2596	1
Injetor Passivo POE 12-24V	1
Rádio NanoStation M5	2
Bateria 7Ah 12V	2
Rádio XBee Series 1	1
Caixa de passagem	1

Tabela 3.3: Materiais Utilizados na construção do protótipo do roteador.

A Figura 3.14 representa um diagrama de blocos do protótipo construído. Neste

diagrama temos que a fonte de energia do protótipo possui tensão de 24 V. Esta tensão é a padrão utilizada pelo rádio transmissor para comunicação com o servidor. Porém o Raspberry Pi trabalha com tensão de 5 V por padrão, então foi utilizado um regulador de tensão DC-DC LM 2596 [16] para realizar esta conversão. Temos também no diagrama um rádio XBee que se conecta ao Raspberry Pi através de suas interface USB, que provê energia e transferência de dados. A alimentação e transferência de dados para o rádio transmissor ocorre pelo mesmo cabo, para isso é necessário a utilização de um injetor Passivo PoE. Este injetor é responsável por unir a transferência de dados entre o Raspberry Pi e o rádio e a fonte de energia. A conexão de dados entre o Raspberry e o Injetor é realizada através da interface Ethernet (ETH).

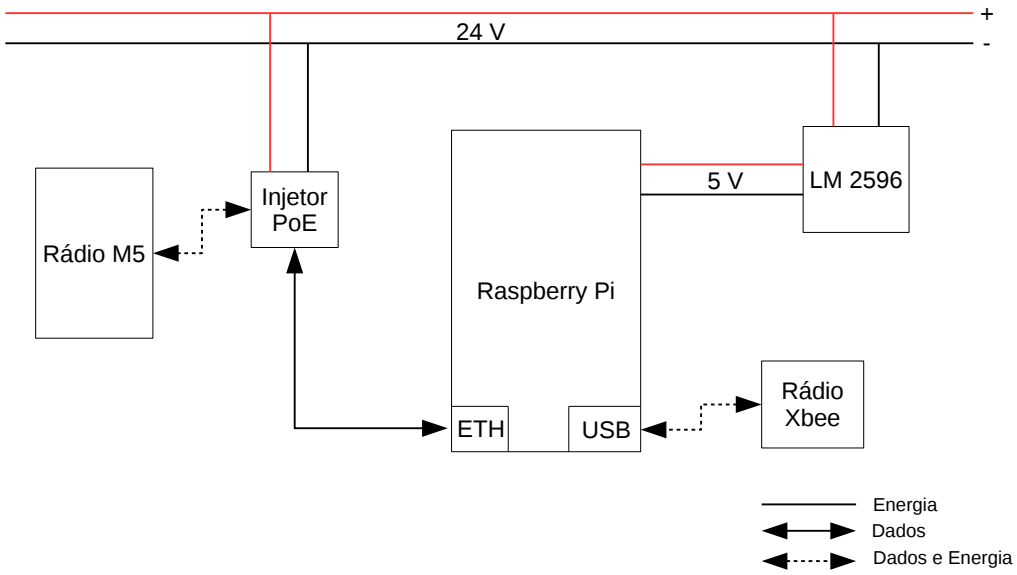


Figura 3.14: Diagrama de blocos do protótipo construído.

Para a construção, primeiramente foi realizado a construção do suporte para o Raspberry Pi dentro da caixa de passagem. Foram realizados quatro furos, utilizando uma broca tamanho 4, com distância de 49 mm de largura e 58mm de comprimento (Figura 3.15). Para a fixação foram utilizados parafusos de 12mm de largura por 25mm de altura. O próximo passo foi a adaptação do rádio XBee. Para tal foi utilizado um adaptador USB para a colocação do rádio XBee. Esse adaptador tem a função de emular uma conexão serial com o Sistema Operacional (Figura 3.16). Em relação a alimentação de energia neste projeto existem dois componentes que necessitam de alimentação de energia. Observa-se que a tensão necessária para alimentação destes componentes são diferentes, o Raspberry Pi necessita de 5V de tensão enquanto o rádio transmissor utilizado necessita de uma tensão de alimentação de 24V. Para contornar este obstáculo assume-se que a tensão padrão do protótipo construído é de 24V e para atingir as demais tensões foi utilizado o regulador de tensão LM 2596. Este regulador foi configurado para realizar a conversão de tensão de saída de 5V compatível com o Raspberry Pi. Para o fornecimento de energia do protótipo foram utilizadas duas baterias de 12V e 7A que foram conectadas em série. Para a conexão da fonte de energia ao rádio transmissor foi utilizado um

Adaptador Passivo POE Fêmea (Figura 3.17). Para conectar este adaptador à fonte de energia foi soldado um adaptador macho a fonte de energia e realizada a conexão. A Figura 3.18 mostra o estado final de construção do protótipo do roteador.

Figura 3.15: Raspberry Pi fixado na caixa de passagem.

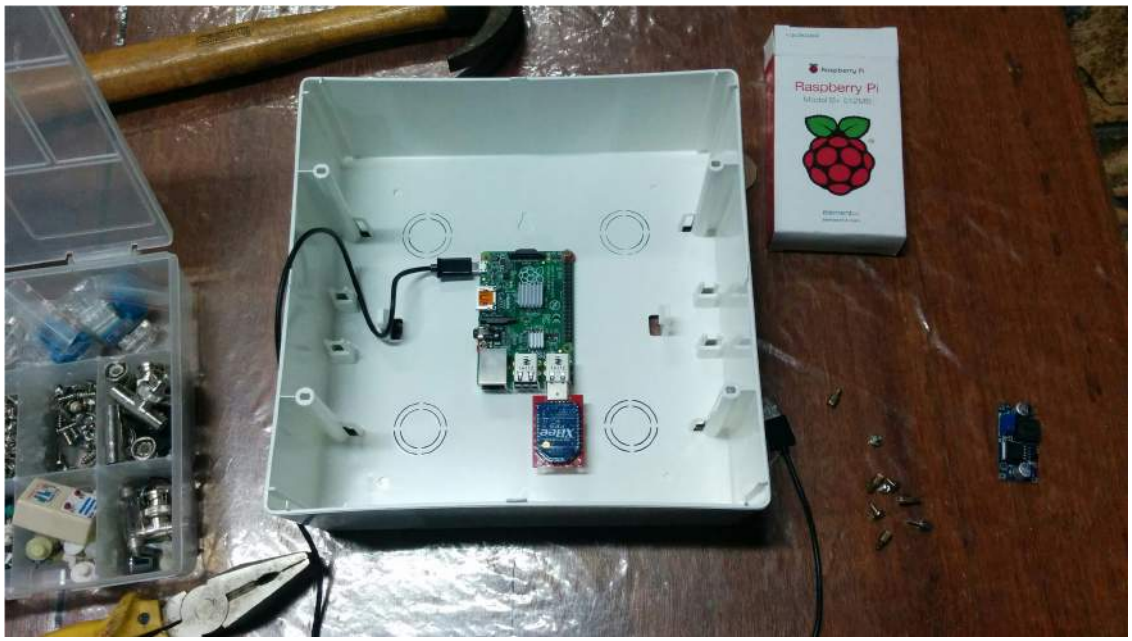


Figura 3.16: Adaptador utilizado para o rádio XBee.

Figura 3.17: Injetor POE Fêmea.

Figura 3.18: Protótipo final do roteador construído.

3.4 Testes e Resultados

Para validar a construção do protótipo construído bem como o software desenvolvido foram realizados testes de consumo de energia para a validação da utilização de baterias para o fornecimento de energia, testes de transmissão de dados para validação da completude do software desenvolvido e teste de transmissão de dados para validar a utilização

deste projeto em ambientes alvo.

3.4.1 Testes de consumo de energia

O objetivo deste teste é validar a utilização de baterias para a alimentação do protótipo construído. Primeiramente foi realizado o cálculo do consumo teórico do sistema. Para isso foram coletados dados de consumo dos itens utilizados. Estes dados são apresentados na Tabela 3.4. Observa-se que o consumo apresentado nesta tabela representa o consumo máximo que o dispositivo pode consumir. Estes dados são disponibilizados pelos fabricantes dos dispositivos.

Dispositivo	Tensão (V)	Corrente (A)	Potência (W)
Rádio XBee Series 1	3.3V	0.05A	0.17W
Raspberry Pi	5.0V	0.6A	3W
NanoStation M5	24V	0.3A	8W

Tabela 3.4: Tabela de consumo dos componentes utilizados na construção do protótipo.

Para calcularmos o consumo teórico foi necessário calcular a potência de cada componente em apenas uma tensão, no caso 24V. Para isso foi necessário o cálculo de eficiência de conversão para as demais tensões presentes no protótipo. A conversão de tensão de 24 V para 5 V é realizada pelo regulador de tensão LM 2596. Este regulador de tensão quando configurado para a redução de tensão de 24 V para 5V possui eficiência de aproximadamente 82% [16]. Para o cálculo teórico foi utilizada a Equação 3.1.

$$P = T * C \tag{3.1}$$

Nesta equação temos que a potência (P) do componente é igual a sua tensão (T) vezes a corrente (C). Se o regulador de tensão possuísse 100% de eficiência teríamos que a potência de entrada P_i seria igual a potência de saída P_0, porém sua eficiência é de de 82% então temos que a potência de entrada P_i é igual a potência de saída P_0 vezes a eficiência E (Equação 3.2) que no caso é 0,82.

$$T_i * C_i = \frac{T_0 * C_0}{E} \tag{3.2}$$

Temos que o consumo do do conjunto que utiliza a tensão de 5 V, no caso o Raspberry Pi e o rádio XBee conectado a ele, consomem aproximadamente um total de 190mAh (Equações 3.4 e 3.3) a uma tensão e 24 V. Neste ponto soma-se este resultado ao consumo do rádio Nanostation M5 que é da ordem de 300mAh ($C = 8/24$ e temos que o protótipo construído possui um consumo total aproximado de 500mAh.

$$24 * C_i = \frac{5 * 0.75}{0.82} \tag{3.3}$$

$$C_i = 0.190 \tag{3.4}$$

Nos testes realizados foram utilizadas duas baterias de 12 V e 7 Ah conectadas em série, gerando assim 24 V de tensão e 7 Ah de corrente. Considerando esta carga de bateria temos que o protótipo funcionará sem necessidade de recarregamento das baterias por aproximadamente 14 horas.

Após os cálculos realizados foram feitos testes para comprovar o consumo do protótipo. Primeiramente cada bateria recebeu carga durante 12 horas ininterruptas. Logo após foi construído um ambiente onde um nó sensor enviará uma mensagem de coleta de dados (neste caso dados aleatórios) a cada 10 segundos e o roteador encaminhara para o servidor. A cada mensagem recebida pelo servidor, era salvo a data do recebimento para posteriormente calcular o tempo de duração da bateria. Este processo foi repetido três vezes.

Foi constatado que o protótipo construído funcionou perfeitamente durante uma média de 16 horas e 23 minutos sem a necessidade de recarga de bateria. O tempo aferido é maior que o tempo teórico calculado devido aos valores utilizados no cálculo que levam em consideração o funcionamento com carga total dos componentes.

3.4.2 Testes de transmissão de dados

Para validar a transmissão de pacotes pelo roteador foi criado um conjunto de pacotes de testes que foram transmitidos pelo um nó sensor na rede. Neste nó foi definido que o intervalo de transmissão é de 10 segundos e este conjunto deve ser transmitido de uma só vez. A Figura 3.19 demonstra mensagens utilizadas para o teste.

1. Um pacote contendo uma mensagem SP do tipo dados não tratados (RAW_PACKET) contendo um vector de bytes que representa uma string com valor de "Hello World!".

2. Um pacote contendo uma mensagem SP do tipo geolocalização(GPS_PACKET) contendo as seguintes posições -20.451793 para latitude e -54.6698357 para longitude.

3. Um pacote contendo uma mensagem SP do tipo temperatura(TEMPERATURE_PACKET) contendo a seguinte temperatura 25.7C.

4. Um pacote contendo três mensagem SP do tipo RAW_PACKET, GPS_PACKET, TEMPERATURE_PACKET contendo os dados "Hello World Complex!", -20.451793 e 54.6698357, 25.7C respectivamente.

```
void loop()
{
  delay(10000);
  //Pacote RAW_PACKET
  uint8_t teste_1[] = {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,12,
    'H','e','l','l','o',' ', 'w','o','r','l','d','!' };
  //Pacote GPS_PACKET
  uint8_t teste_2[] = {1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,16,
    192,52,115,168,180,191,143,205,192,75,85,189,45,28,152,189};
  //Pacote TEMPERATURE_PACKET
  uint8_t teste_3[] = {1,2,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,8,
    64,57,179,51,51,51,51,51};
  //Conjunto de Pacotes
  uint8_t teste_4[] = {
    3,
    //Primeiro pacote RAW_PACKET
    0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,19,
    'H','e','l','l','o',' ', 'w','o','r','l','d',' ','C','o','p','l','e','x
        ','!',
    //Segundo pacote GPS_PACKET
    1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,16,
    192,52,115,168,180,191,143,205,192,75,85,189,45,28,152,189,
    //Terceiro pacote TEMPERATURE_PACKET
    2,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,8,64,57,179,51,51,51,51,51
  };

  //Construcao do dos pacotes Zigbee
  Tx16Request tx_1 = Tx16Request(0x0001, ACK_OPTION, teste_1, sizeof(
      teste_1),xbee.getNextFrameId());
  Tx16Request tx_2 = Tx16Request(0x0001, ACK_OPTION, teste_2, sizeof(
      teste_2),xbee.getNextFrameId());
  Tx16Request tx_3 = Tx16Request(0x0001, ACK_OPTION, teste_3, sizeof(
      teste_3),xbee.getNextFrameId());
  Tx16Request tx_4 = Tx16Request(0x0001, ACK_OPTION, teste_4, sizeof(
      teste_4),xbee.getNextFrameId());

  //Envio e verificacao de respostas
  xbee.send(tx_1);
  checkResponse();
  xbee.send(tx_2);
  checkResponse();
  xbee.send(tx_3);
  checkResponse();
  xbee.send(tx_4);
  checkResponse();
 }
```

Figura 3.19: Teste desenvolvido no arduino.

Para validação das mensagens que foram transmitidas pelo roteador foi criado um simples cliente que simula o servidor que receberá as informações. Neste cliente toda informação recebida é impressa sem qualquer modificação. A Figura 3.20 demonstra as

mensagens recebidas pelo cliente após o envio dos pacotes detalhados acima. Conforme demonstrado todas as mensagens enviadas pelo nó sensor foram recebidas sem qualquer erro.

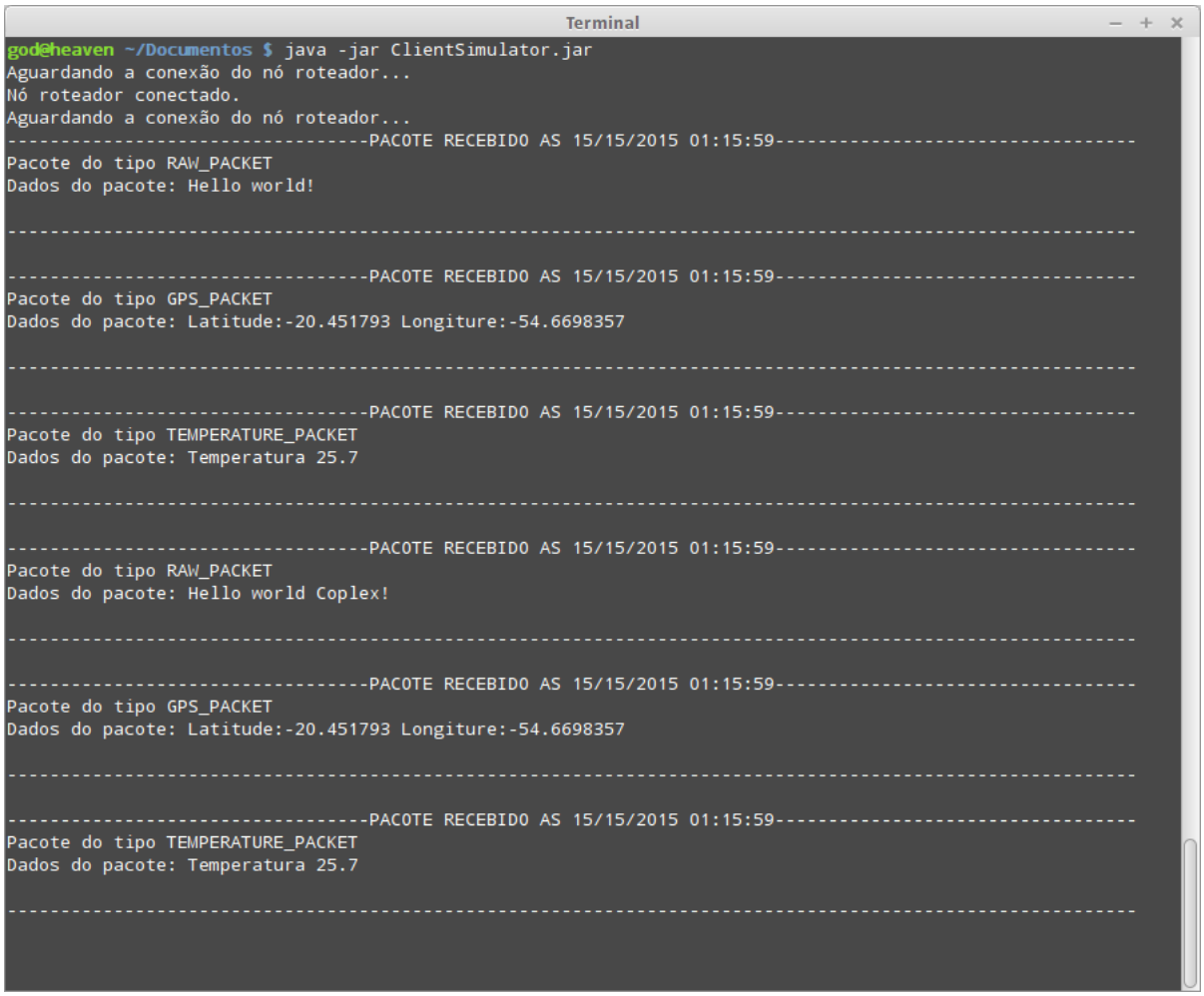


Figura 3.20: Recebimento de pacotes do roteador.

3.4.3 Testes de transmissão com Rádio

Para validação do teste de transmissão de dados com rádio foram realizados testes de transmissão na Embrapa gado de corte em Campo Grande MS. O objetivo deste teste é validar a possibilidade de transmissão de dados a longas distâncias. Para isto foi utilizado dois rádios de transmissão Nanostation M5 que segundo possui alcance de transmissão de 10 KM com visada. Os testes foram realizados a uma distância aproximada de 380 metros. A Figura 3.21 demonstra os pontos onde foram colocados os rádios. No ponto A foi colocado o protótipo construído junto com um nó sensor que simula a rede de sensores sem fio. Além disso foi o protótipo foi conectado ao rádio transmissor Nanostation M5 e no ponto B foi colocado o outro rádio e este conectado ao computador receptor onde estava em funcionamento o software desenvolvido conforme a Seção 3.2.1. Estes rádios

foram configurados para funcionar como uma rede Wi-Fi ponto a ponto. Nos testes realizados foram constatados uma eficiência de aproximadamente 80% na forca do sinal mesmo não possuindo visada devido a diferença de altitude do relevo. Ficou concluído assim a validade do trabalho cujo objetivo é a transmissão de dados.

Figura 3.21: Teste de distância de transmissão realizado.

3.5 Considerações Finais

O desenvolvimento de uma estação base para RSSF usando como plataforma micro-computadores como o Raspberry Pi torna possível o desenvolvimento de softwares robustos que lidam com consideráveis massas de dados possibilitando um pré-processamento e tornando o sistema seguro a falhas através de métodos de contingência. Neste Capítulo foram apresentados o escopo do projeto desenvolvido, as definições de arquitetura de hardware e software utilizadas, o desenvolvimento do software utilizado, a construção de um protótipo de nó sorvedouro e por fim a validação do mesmo com testes de transmissão de dados e transmissão a distância.

Capítulo 4

Considerações Finais

O aumento nas exigências para garantir qualidade e aumento de produtividade exige que cada vez mais que novas tecnologias sejam utilizadas no monitoramento e manejo bovino. Dentre as tecnologias, uma que vem se destacando é a de RSSF. O uso de RSSF com o objetivo de monitoramento e atuação em um meio para o aperfeiçoamento de estudos e também para o melhoramento da produtividade animal.

Um dos desafios da utilização de RSSF é a transmissão de dados do campo para servidores, devido à limitação do alcance de sinal. Para solucionar este problema foi desenvolvido de um nó sorvedouro de uma RSSF com objetivo de transmitir dados coletados em campo para servidores com a finalidade de possibilitar que estes possam ser analisados pelas aplicações que necessitam dessas informações. Este roteador possui algumas características que facilitam e ajudam pesquisadores a coletar dados no ambiente.

Para o desenvolvimento do trabalho, foi utilizado como hardware o Raspberry Pi por ser um micro-computador de baixo custo e possui diversas melhorias no consumo de energia em relação a computadores tradicionais. Além disso, o mesmo apresenta a possibilidade de integrar dois dispositivos de comunicação sem fio, que é essencial para a transmissão de dados do campo para os servidores de aplicação.

Para o desenvolvimento da parte de software, o nó sorvedouro foi modular para permitir que sejam realizadas customizações para que o nó sorvedouro possa ser utilizado em diversas aplicações na área de pecuária.

Neste nó sorvedouro é possível a definição de cada tipo de pacote, também é possível a implementação de pré-processamento para pacotes distintos possibilitando assim uma granularidade final no tratamento de cada tipo de dado coletado. Outra característica importante é que o nó sorvedouro foi implementado um módulo que permite enviar comandos de configuração para a nós sensores na rede. Dessa forma, é possível por exemplo, alterar o tempo de coleta dos dados dos nós sensores.

Este trabalho teve como contribuições principais o desenvolvimento de uma estação base configurável no contexto de possibilitar a configuração e/ou expansão dos tipos de dados trafegados com o objetivo de realizar a transmissão e processamento de pacotes de dados coletados por uma RSSF de monitoramento animal e ambiental em um ambiente

de integração para que se possa realizar análise sobre estes dados e como consequência o melhoramento da produtividade do gado de corte brasileiro e o uso do conhecimento em desenvolvimento de hardwares reconfiguráveis para a construção de uma estação base com custo baixo para a transmissão de dados de um ambiente em integração monitorado.

4.1 Trabalhos Futuros

Nota-se que há espaços para melhora neste nó sorvedouro. Como trabalhos futuros, podem ser citados:

- Uso de painéis solares para o carregamento das baterias utilizadas, com isso será possível a não intervenção no nó para troca de baterias, possibilitando assim o uso contínuo para coleta de dados em intervalos maiores de tempo.
- Construção de uma recipiente a prova d'agua para que o nó não sofra com intemperes do ambiente observado.
- Desenvolvimento de uma DSL (*Description Software Language*) para o roteador, com isso será possível que o pesquisador configure os tipo de pacotes sem que seja necessário a recompilação dos códigos fontes. Isto possibilitará a menor complexidade de configuração do sistema possibilitando a configuração seja realizada por usuários leigos.

Referências Bibliográficas

[1] Approved draft amendment to ieee standard for information technology-telecommunications and information exchange between systems-part 15.4:wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans): Amendment to add alternate phy (amendment of ieee std 802.15.4). *IEEE Approved Std P802.15.4a/D7, Jan 2007*, pages –, 2007. 2.2

[2] Zigbee-based wireless sensor networks for classifying the behaviour of a herd of animals using classification trees. *Biosystems Engineering*, 100(2):167 – 176, 2008. 1

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002. 2.2

[4] Z. Alliance. *ZigBee Specifications*, 2012. 2.1.2, 2.3, 2.3

[5] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks (Elsevier*, 46:605–634, 2004. 1

[6] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Computer Communications*, 30(7):1655 – 1695, 2007. Wired/Wireless Internet Communications. 2.2.2

[7] D. Berckmans. Automatic on-line monitoring of animals by precision livestock farming., 2004.

[8] G. J. Bishop-Hurley, D. L. Swain, D. M. Anderson, P. Sikka, C. Crossman, and P. Corke. Virtual fencing applications: Implementing and testing an automated cattle control system. *Comput. Electron. Agric.*, 56(1):14–22, Mar. 2007. 1

[9] D. J. Bungenstab. *Sistemas de integração lavoura-pecuária-floresta: a produção sustentável, 2ª Edição*. Embrapa, 2012. 1, 2.4.4

[10] A. Camilli, C. E. Cugnasca, A. M. Saraiva, A. R. Hirakawa, and P. L. P. Corrêa. From wireless sensors to field mapping: Anatomy of an application for precision agriculture. *Comput. Electron. Agric.*, 58(1):25–36, Aug. 2007. 1

[11] L. de Jesus. Identificação do comportamento bovino por meio do monitoramento animal. Master's thesis, FACOM, 2014. 2.4.4

[12] S. Farahani. *ZigBee Wireless Networks and Transceivers*. Newnes, 2008. 2.1, 2.1.1, 2.2, 2.3.1

[13] A. Foundation. Apache log4j 2, 2015. 3

[14] S. Gupta, A. Mahalwar, and P. Udaykumar. Analysis of different techniques for locating leaks in pipes in water distribution system using wsn. In *Computational Intelligence on Power, Energy and Controls with their impact on Humanity (CIPECH), 2014 Innovative Applications of*, pages 173–177, Nov 2014. 2.4.2

[15] I. Howitt and J. Gutierrez. Ieee 802.15.4 low rate - wireless personal area network coexistence issues. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 3, pages 1481–1486 vol.3, March 2003. 2.2.2

[16] T. Instrument. *LM2596 SIMPLE SWITCHER® Power Converter 150 kHz 3A Step-Down Voltage Regulator*. 3.3, 3.4.1

[17] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. *SIGARCH Comput. Archit. News*, 30(5):96–107, Oct. 2002. 2.4.4

[18] T. Kalaivani, A. Allirani, and P. Priya. A survey on zigbee based wireless sensor networks in agriculture. In *Trendz in Information Sciences and Computing (TISC), 2011 3rd International Conference on*, pages 85–89, Dec 2011. 2.4.4

[19] M. Khanafer, M. Guennoun, and H. Mouftah. Wsn architectures for intelligent transportation systems. In *New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on*, pages 1–8, Dec 2009. 2.4.3

[20] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi. 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*, MobiSys '04, pages 256–269, New York, NY, USA, 2004. ACM.

[21] L. F. D. Lomba. Identificação do comportamento bovino a partir dos dados de aceleração do animal e monitoramento da luminosidade do ambiente. Master's thesis, FACOM, 2015. 2.4.4

[22] L. F. S. Luiz Carlos Balbino, Alexandre de Oliveira Barcellos. Marco referencial em integração lavoura-pecuária-floresta, 2011. 2.4.4

[23] E. P. Management. *IEC 62591 WirelessHART System Engineering Guide*, 2012. 2.1.2

[24] R. Morais, M. A. Fernandes, S. G. Matos, C. Serôdio, P. Ferreira, and M. Reis. A zigbee multi-powered wireless acquisition device for remote sensing applications in precision viticulture. *Computers and Electronics in Agriculture*, 62(2):94 – 106, 2008. 1

[25] E. Nadimi and H. Søgaard. Observer kalman filter identification and multiple-model adaptive estimation technique for classifying animal behaviour using wireless sensor networks. *Computers and Electronics in Agriculture*, 68(1):9 – 17, 2009. 1

[26] N. Nasser, A. Ali, L. Karim, and S. Belhaouari. An efficient wireless sensor network-based water quality monitoring system. In *Computer Systems and Applications (AICCSA), 2013 ACS International Conference on*, pages 1–4, May 2013. 2.4.2

[27] I. S. of Automation. *ISA 100.11a Specifications*, 2008. 2.1.2

[28] A. Rapp. Xbee-api github repository, 2015. 3

[29] D. V. Richard Connor, Pasquale Steduto and W. van der Hoek. The third united nations world water development report: Water in a changing world, 2009. 2.4.2

[30] J. Sarangapani. *Wireless Ad hoc and Sensor Networks: Protocols, Performance, and Control*. CRC Press, 2007. 2.1

[31] J. B. Schleppe, G. Lachapelle, C. W. Booker, and T. Pittman. Challenges in the design of a gnss ear tag for feedlot cattle. *Comput. Electron. Agric.*, 70(1):84–95, Jan. 2010. 1

[32] G. A. Silva. Iplementação de uma rssf para coletas de dados de ambiência para um sistema de ilpf. Master's thesis, FACOM, 2015. 2.4.4

[33] O. Song and J. Kim. An efficient design of security accelerator for ieee 802.15.4 wireless senor networks. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pages 1–5, Jan 2010.

[34] N. Srivastava. Challenges of next-generation wireless sensor networks and its impact on society. *CoRR*, abs/1002.4680, 2010. 2

[35] D. Tianmin and S. Yao-yao. Design of the intelligent public transportation monitoring system based on wsn. In *Consumer Electronics, Communications and Networks (CECNet), 2011 International Conference on*, pages 4024–4027, April 2011. 2.4.3

[36] C. P. Waltenegus Dargie. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Wiley, 2010. 2.1

[37] Y. Yang. *Microchip MiWi P2P Wireless Protocol*, 2010. 2.1.2

[38] R. Yerra, A. Bharathi, P. Rajalakshmi, and U. Desai. Wsn based power monitoring in smart grids. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2011 Seventh International Conference on*, pages 401–406, Dec 2011. 2.4.1

[39] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Comput. Netw.*, 52(12):2292–2330, Aug. 2008. 1

[40] S. Yinbiao. Internet of things: Wireless sensor networks. 2015. 2.4.3

[41] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware design experiences in zebranet. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 227–238, New York, NY, USA, 2004. ACM. 2.4.4

[42] S. Zhao, H. Lian, C. Huang, and H. Shen. Research of regional power grid operation comprehensive quantitative evalution system and intelligent security early warning system. In *Electricity Distribution (CICED), 2014 China International Conference on*, pages 1749–1754, Sept 2014. 2.4.1