Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Caracterização de uma Rede de Sensores Sem Fio considerando sua Transferência de Dados e Consumo Energético

Breno Felipe Nunes Gomes

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Marcelo Marotta

Brasília
2022

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Caracterização de uma Rede de Sensores Sem Fio considerando sua Transferência de Dados e Consumo Energético

Breno Felipe Nunes Gomes

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Marcelo Marotta (Orientador)
CIC/UnB

Prof. Dr. Marcos Fagundes Caetano
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 07 de Julho de 2022

Dedicatória

Este trabalho é dedicado aos meus pais Aldenice e Elias que sempre me apoiaram e acreditaram em mim. Dedico também aos meus amigos que me acompanharam nessa jornada.

Agradecimentos

Agradeço ao Prof. Dr. Marcelo Marotta pela oportunidade de desenvolver este trabalho, por todas as orientações e pela paciência. Agradeço ao aluno do mestrado Luan B. Santos pelas reuniões para compartilhar conhecimento sobre o simulador Castalia. Agradeço a todos que contribuíram diretamente ou indiretamente para que eu chegasse até aqui.

Resumo

As redes de sensores sem fio(RSSF) no geral tem como propósito a detecção ou monitoramento de determinado fenômeno no mundo físico. Esse tipo de redes possuem os mais diversos tipos de aplicação, algumas áreas que fazem uso de RSSF são: militar, industrial, engenharia e agricultura. A transmissão de dados e consumo energético são um dos grandes desafios dentro de uma RSSF, devido a dependência de baterias, em muitas aplicações reduzindo drasticamente o tempo de vida de operação dessas redes. A escolha de protocolos eficientes e topologia de rede são essenciais para aumentar o tempo de vida dessas redes, muitas vezes uma decisão que é benéfica na transmissão de dados pode aumentar o consumo energético da rede e vice-versa.

Neste trabalho é apresentada uma simulação de uma RSSF para um sistema de irrigação inteligente, usando o simulador de RSSF e Body Area Network(BAN) chamado Castalia. Essa simulação visa a caracterizar o relacionamento entre transferência de dados e consumo energético nas comunicações de redes de sensores sem fio. Considerando que os parâmetros de transmissão de dados de uma RSSF podem influenciar no consumo energético da rede, para extrair um melhor tempo de vida da rede é importante a caracterização do relacionamento de transmissão de dados e consumo energético da rede previamente à sua implementação.

Um sistema de irrigação inteligente foi usado como estudo de caso para simulação apresentada no trabalho. Esse tipo de sistema precisa de uma boa eficiência energética devido a dificuldade de reposição de bateria nos nós da rede, também necessita de uma boa transmissão de dados devido perda de sinal em determinados ambientes por exemplo no solo para sensores subterrâneos [1]. Para caracterização da RSSF do sistema de irrigação foram usadas as métricas de consumo de energia e taxa média de entrega de pacotes.

Palavras-chave: Internet das coisas, Redes de Sensores Sem Fio, Castalia, Pivô Central, simulação

Abstract

The Wireless Sensor Networks (WSN) in general aim to detect or monitor a certain phenomenon in the physical world. This type of network has the most diverse types of applications, some domains that use WSN are: military, industrial, engineering, and agriculture. Data transmission and energy consumption are the significant challenges within a WSN, due to the dependence on batteries, in many applications, drastically reducing the operating lifetime of these networks. The choice of efficient protocols and network topology is essential to increase the lifetime of these networks, often a decision that is beneficial in data transmission can increase the energy consumption of the network and vice versa.

This work presents a simulation of a WSN for an intelligent irrigation system, using the WSN and Body Area Network (BAN) simulator called Castalia. This simulation aims to characterize the relationship between data transfer and energy consumption in wireless sensor network communications. Considering that the data transmission parameters of a WSN can influence the energy consumption of the network, in order to extract a better network lifetime it is important to characterize the relationship between data transmission and energy consumption of the network before the implementation.

An intelligent irrigation system was used as a case study for the simulation presented in the work. This type of system needs good energy efficiency due to the difficulty of replacing the battery in the network nodes, it also needs good data transmission due to signal loss in certain environments, for example on the ground for underground sensors [1]. To characterize the WSN of the irrigation system, the metrics of energy consumption and average package delivery rate were used.

Keywords: Internet of Things, Wireless Sensor Networks, Center Pivot, simulation

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Internet das Coisas (IoT) é um modelo que possibilita a conexão entre objetos pertencentes ao mundo real e a Internet, permitindo que objetos façam a coleta, o processamento e a transmissão de dados sem intervenção humana[2]. Uma rede IoT é composta por dispositivos inteligentes, com a combinação dos dados coletados por esses dispositivos que as aplicações IoT conseguem resolver demandas de diversos setores como: automação residencial, monitoramento de tráfego e automação de irrigação no plantio. Redes de sensores de aplicações IoT enfrentam alguns desafios como consumo de energia e transmissão de dados, desafios são causados pelas limitações de hardware dos dispositivos, em algumas aplicações a dificuldade de reposição de baterias nos dispositivos e perda de sinal em ambientes adversos[7].

O uso de aplicações IoT e outras tecnologias na agricultura é conhecido como Smart Farming, o smart farming pode fazer uso de diversas tecnologias como: veículos autônomos, drones, cloud computing e GPS. Com o uso do Smart Farming os agricultores podem automatizar irrigação, automatizar tratores e ter uma melhor tomada de decisão devido sistemas de monitoramento que fazem uso de sensores [3]. Em função da necessidade de recursos humanos e financeiros para implementação de uma aplicação Smart Farming o uso de simulação para pesquisa e estudo deste tipo de aplicação é muito importante.

As redes de sensores sem fio(RSSF) tem como finalidade a detecção e o monitoramento de algum fenômeno no mundo físico[7]. As RSSF são amplamente utilizadas em aplicações IoT e Smart Farming devido a escalabilidade e a flexibilidade das redes de sensores sem fio. Consumo energético e a transmissão de dados são fatores muito importantes na avaliação de uma RSSF[8]. Neste trabalho é desenvolvida uma simulação de uma RSSF com o objetivo de caracterizar o relacionamento entre transmissão de dados e o consumo energético da rede.

Um sistema de irrigação inteligente foi usado como estudo de caso para simulação apresentada neste trabalho. O sistema de irrigação inteligente é composto por uma RSSF,

a rede é composta por sensores subterrâneos que coletam os dados do solo e por um drone que coleta os dados sensores. Os principais elementos da simulação são os sensores, o drone e o pivô central que é responsável pela irrigação do campo. A simulação é executada em dois cenários diferentes: no primeiro cenário o drone realiza todo caminho de coleta de dados por conta própria e no segundo cenário o drone realiza uma parte do caminho sob pivô central a fim de obter uma economia energética.

1.1 Estrutura do trabalho

No Capítulo 2 apresenta fundamentos básicos, técnicas e tecnologias aplicadas dentro do trabalho. No Capítulo 3 é exposto os cenários das simulações e elucidados mais a fundo. No Capítulo 4 será apresentada toda implementação feita dentro do simulador Castalia. No Capítulo 5 é apontado todos os resultados obtidos das simulações, por fim, no Capítulo 6 concluirá com as considerações finais acerca do estudo.

Capítulo 2

Fundamentos e Trabalhos Relacionados

Neste capítulo serão apresentados conceitos e fundamentos a respeito de: Internet das coisas, Redes de Sensores, Smart Farming e ferramentas usadas no desenvolvimento do trabalho.

2.1 Internet das Coisas(IoT)

A internet das coisas(IoT) possibilita a conexão de objetos do mundo físico com a internet, assim esses objetos conseguem realizar coleta, processamento e a transmissão de dados sem dependência de uma interação constante [2]. As aplicações IoT possibilitaram a automação de serviços que antes eram majoritariamente manuais. Na construção/implementação de uma aplicação IoT é enfrentado diversos desafios e requisitos, alguns dos principais que pode ser listados são:

- A extração de dados em tempo real;
- A baixa latência;
- Filtragem de dispositivos correspondentes através de dados;
- Eficiência energética dos recursos IoT.

Para o pleno funcionamento de uma aplicação IoT é necessário considerar: I) dinamismo: sistema deve levar em consideração as constantes mudanças na topologia da rede; II) escalabilidade: o sistema deve gerenciar um grande número de sensores dentro da rede; III) mobilidade: sistema deve suportar frequente mudança de localização dos dispositivos [2].

As aplicações IoT tem uso nos mais diversos cenários por exemplo : na vigilância dentro cidades usando reconhecimento facial, no monitoramento de estrutura de pontes com uso de sensores, no monitoramento da ambiente para obter dados sobre clima e no monitoramento de saúde para um melhor acompanhamento dos pacientes. As aplicações IoT pode ser divididas em 3 grandes setores:

1. Aplicações IoT para Sociedade(Health Care, Smart City, serviços de vigilância);
2. Aplicações IoT para Indústria(Viação,transporte e logística);
3. Aplicações IoT para meio-ambiente(gerenciamento de desastres naturais, controle de poluição e Smart farming) [2].

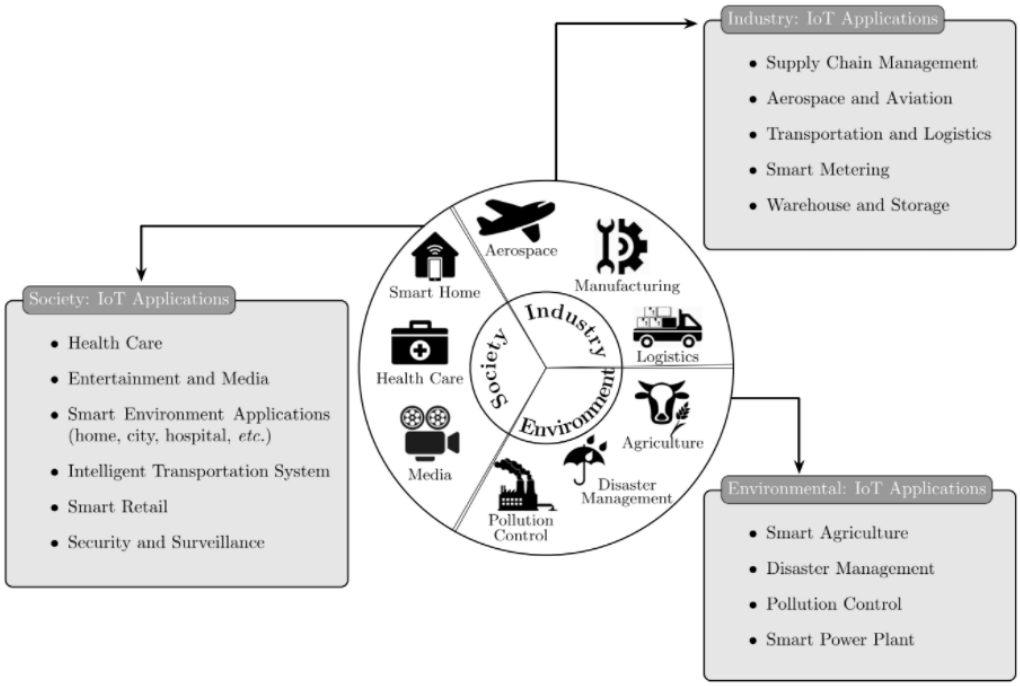


Figura 2.1: Tipos de aplicações IoT [2]

No trabalho de Al-Sarawi, Shadi *et al.* [9] é analisado protocolos de comunicação para Internet das coisas(IoT) é feita comparações com foco nas principais funcionalidades e métricas de taxa de dados, segurança e consumo de energia. Uma rede IoT é composta por um enorme número de dispositivos inteligentes que possuem diversas limitações como armazenamento de dados, carga de bateria e alcance de rádio. Os protocolos de comunicação IoT podem ser categorizadas em duas categorias: Rede de longa distância de baixa potência - *Low Power Wide Area Network(LPWAN)* e redes de curta distância. As comunicações dentro da simulação presente neste trabalho são predominantemente de redes curta distância.

2.2 Redes de Sensores

No trabalho de Akyildiz, Ian F., *et al.* [8] é descrito o conceito de redes de sensores fazendo a conexão de conceitos e tecnologias de sistemas microeletromecânicos, comunicações sem fio e eletrônica digital. O sensores podem ser implementados de duas formas no geral:

- Sensores que são posicionados longe dos fenômenos que estão monitorando neste caso geralmente são sensores maiores e mais complexos;
- Múltiplos sensores com topologia e comunicação minuciosamente projetadas, eles transmitem séries temporais do fenômeno que foi detectado para os nós centrais onde os cálculos serão executados e os dados fundidos.

Uma rede de sensores é composta por um grande número de sensores, que são densamente implantados dentro do fenômeno monitorado ou muito perto do mesmo [8].

No design de Redes de Sensores Sem Fio (RSSF) é necessário amplo conhecimento de variados campos de pesquisa como comunicação sem fio, redes, sistemas embarcados, processamento de sinal e engenharia de software [7]. Na implantação de uma rede de sensores é necessário considerar certas características como a flexibilidade, tolerância a falhas, alta fidelidade de detecção e o baixo custo. As principais restrições que uma redes de sensores são causadas pela tolerância à falhas, escalabilidade, custo, hardware, mudança de topologia e consumo de energia [8]. A rede de sensores sem fio construída na simulação deste trabalho aborda as limitações de topologia em relação ao posicionamento dos sensores e consumo de energia analisando o tempo de vida da rede.

2.2.1 Restrições de hardware

No geral, a arquitetura de RSSF é majoritariamente composta por sensores sem fio que são os nós da rede ilustrado na Figura 2.2. Na Figura 2.2 é mostrado os principais componentes de nó sensor:

- *Unidade de Energia*: é onde a bateria é utilizada, a power unit fornece energia para os outros componentes;
- *Sistema de localização*: a maioria das aplicações RSSF precisam do conhecimento da localização física do nó. Esse sistema pode ser composto por um GPS ou possuir um algoritmo de localização que forneça a localização através de cálculos;
- *Unidade de sensores*: o componente que fornece informações do mundo físico, informações como temperatura, umidade, luz e muitas outras;
- *Unidade de processamento*: é o componente responsável por processar informações e executar algoritmos, também é o principal controlador do sensor;

- *Transceptor*: o componente que realiza a comunicação entre nós, também onde é implementado à conversão de bits para a transmissão em rádio frequência;
- *Unidade de mobilidade*: O componente responsável pela mobilidade do nó;
- *Gerador de energia*: um componente essencial para aplicações em nós precisem longo tempo de vida. O componente pode fazer uso de técnicas de coleta de energia como: energia solar e energia eólica.

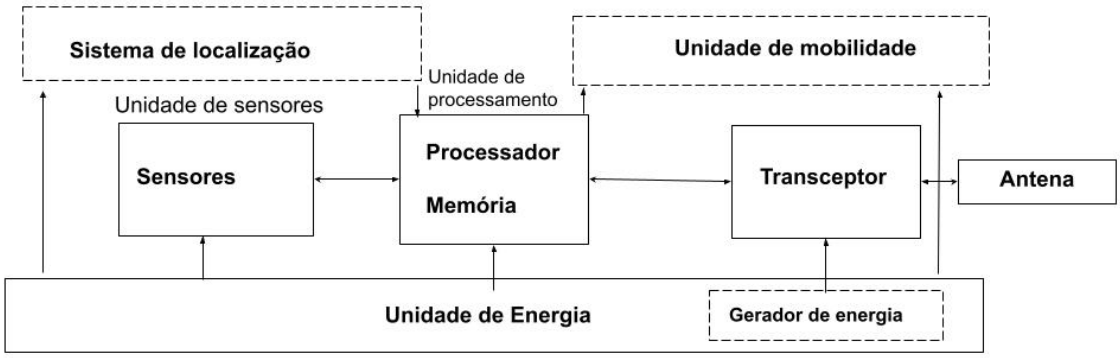


Figura 2.2: Arquitetura de harware de nó sensor

2.2.2 Tolerância à falha

Devido às restrições de hardware nós sensores podem falhar frequentemente ou serem bloqueados por um determinado período de tempo [7]. Essas falhas podem acontecer devido à falta de energia, danos físicos, interferência do ambiente ou problemas com software. A falha de um nó ocasiona sua desconexão da rede. O nível de falha que são permitidas por uma rede para que a mesma continue funcionando adequadamente define sua tolerância à falhas. O nível tolerância à falhas depende do tipo de aplicação, quanto mais crítica uma aplicação mais alto o nível de tolerância à falhas deve ser.

2.2.3 Escalabilidade

A escalabilidade é a capacidade da rede de ampliar seu tamanho sem afetar suas funcionalidades. O número de nós sensores implementados para monitorar/detectar um fenômeno físico pode ser da ordem de centenas ou milhares [7]. Portanto, os protocolos de rede desenvolvidos para essas redes devem ser capazes de lidar com esse grande número de nós de forma eficiente.

2.2.4 Custo de produção

Como as redes de sensores consistem em um grande número de nós sensores, o custo de um único nó é muito importante para justificar o custo total das redes [7] . Se o custo da rede for mais caro do que a implantação de dispositivos de sensor único tradicionais, logo a rede de sensores não é justificada pelo custo. Como consequência, o custo de cada nó deve ser mantido baixo.

2.2.5 Topologia RSSF

O grande número de nós sensores inacessíveis e autônomos que são propensos a falhas frequentes tornam a manutenção de topologia uma tarefa desafiadora. O grande desafio é a implementação desses nós sensores em um campo para que o fenômeno de interesse possa ser monitorado de forma eficiente [7]. A implantação da topologia pode ser dividida em 3 etapas:

- Pré-implantação e implantação: Os nós sensores podem ser implantados em massa ou colocados um à um no campo;
- Pós-implantação: dentro de uma RSSF os movimentos dos sensores afetam a topologia da rede, essas alterações resultam em uma operação diferente da implantação inicial da rede. Portanto, os protocolos de rede devem ser capazes de se adaptar a mudanças na topologia;
- Re-implantação e adição de nós: nesta etapa pode exigir a implantação adicional de nós o que pode afetar a topologia.

2.2.6 Transmissão de média

O funcionamento bem sucedido de uma RSSF depende da comunicação confiável entre os nós da rede [7]. Em uma rede de sensores, os nós podem se comunicar por meio de conexões sem fio. Essas conexões podem ser formadas por rádio, infravermelho, óptico, acústico ou magneto-indutivo. Os requisitos de aplicações incomuns das redes de sensores tornam a escolha do meio de transmissão mais difícil, um exemplo seria uma aplicação marítima que pode exigir uso de uma transmissão por meio aquático. Para isso, será necessário uso de ondas com maior comprimento para que penetre na superfície da água.

No artigo de Roy, Swati Sucharita, *et al.* [10] é discutido uma nova técnica de roteamento de dados para encaminhar dados usando um coletor de dados móvel, os coletores de dados seguem um caminho predefinido para coletar todos os dados da rede. Dentro do artigo [10] é citado três tipos de modelo de entrega de dados:

1. Periódico: os sensores roteiam os dados detectados continuamente dentro de um intervalo.
2. Orientado a eventos: Um sensor somente transfere os dados ao ocorrer um evento.
3. Orientado a consultar: Um sensor envia dados quando um coletor de dados solicita os dados necessários.

Neste trabalho é usado o modelo de entrega de dados orientado a eventos, o evento para envio de dados é proximidade do drone. O drone quando está com uma distância de 20 metros ou menos do sensor, o sensor envia os dados para drone que é o coletor de dados, essa parte é explicada mais detalhadamente no Capítulo 3.

2.2.7 Consumo de energia

O tempo de vida de RSSF é fortemente dependente do tempo de vida da bateria. Dessa forma as fontes que consomem energia durante a operação de cada nó devem ser analisadas. A principal tarefa de um nó sensor em um campo é detectar eventos, realizar processamento local de dados e, em seguida, transmitir os dados. O consumo de energia pode ser dividido em três domínios, sensoriamento, comunicação e processamento de dados, que são realizados pelos sensores, rádio e a CPU respectivamente [7].

2.2.8 Tempo de vida da Rede

No artigo de Bhardwaj, Manish e Garnett, Timothy e Chandrakasan, Anantha P [11] é trabalhado questões sobre o tempo de vida de redes de sensores, uma pergunta fundamental sobre o tempo de vida da rede é "qual o limite de uma rede de sensores que coleta dados de uma região usando um certo número de nó com restrição energética?" [11]. Através da resposta é possível ajustar protocolos de coleta de dados no mundo real e compreender aspectos impedem os protocolos de alcançar esse limite. Também permite analisar a dependência que o tempo de vida da rede tem de fatores como localização, número de nós e configurações de rádio. Na simulação presente neste trabalho com resultados apresentados no Capítulo 5 é analisado o tempo de vida com variação dos parâmetros de envio de pacotes e potência de rádio.

2.3 Smart farming

Smart farming(SF) nada mais é que aplicação de modelos e tecnologias IoT dentro da agricultura de um modo geral, mas também pode ser definida como a integração de informação e tecnologias de comunicação para equipamentos agrícolas e o uso de sensores

e redes de sensores dentro cultivo de plantações e sistema de produção alimentícia. No mundo atual a agricultura é a principal fonte de comida, é de extrema importância alcançar eficiência dentro da produção agrícola dado o aumento populacional esperado de mais 2 bilhões de pessoas no mundo até 2050 [3]. A migração da população rural para cidades acaba ocasionando em um déficit de mão de obra dentro das produções agrícolas. A utilização de automação na agricultura soluciona e auxilia não só no déficit de mão de obra mas também na eficiência da produção.

Existem inúmeras formas de criar aplicações para Smart farming, alguns exemplos de aplicação seriam:

- Tratores autônomos: uso de hardware em tradutores para automação junto ao GPS para mapeamento e navegação;
- Irrigação Automática: pode implementada de diversas formas seja irrigação periódica simples ou com auxílio de sensores;
- Sensores IoT para determinação de umidade: a implementação envolve uso de sensores subterrâneos geralmente para monitoramento de umidade do solo.

Na Figura 2.3 é ilustrado um exemplo de um sistema de Smart farming, onde o fluxo começa no campo com a parte de *Field monitoring and data collection*, que é descrito os componentes de monitoramento que são os sensores e os coletores de dados. Outra parte da imagem é descrito o processo de receber as informações coletadas e processo de tomada de decisão, os componentes dessa parte são a Internet, bancos de dados, dispositivo de controle de sistema e aplicações inteligentes. Os benefícios do uso desse tipo de aplicação são uma produção sustentável e com maior eficiência.

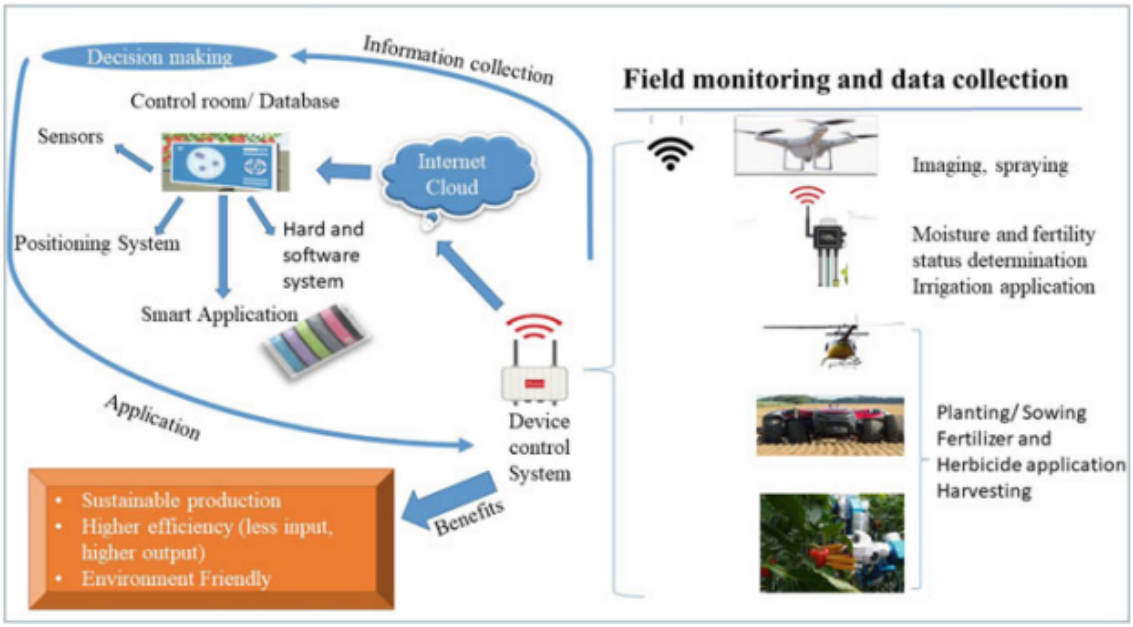


Figura 2.3: Exemplo de esquema de monitoramento para Smart farming [3].

Um sistema de irrigação inteligente tem como principal objetivo realizar o uso da água de forma mais eficiente dentro de uma plantação. Água é o recurso primário para vida humana, estima-se que a agricultura consome 70% da água doce do planeta. Como o receio da perda de produção, agricultores muitas vezes exageram na irrigação ocasionando no desperdício hídrico [4]. Um sistema de irrigação inteligente obtém um uso mais eficiente da água se houver uma integração de informações como qualidade do solo e condições climáticas. Dentro da simulação que é desenvolvida para este trabalho é usado com estudo de caso um sistema de irrigação inteligente para plantações que utilizam pivô central como método de irrigação.

2.4 Swamp

Smart Water Management Platform(Swamp) - Plataforma de Gerenciamento Inteligente de Água é um projeto desenvolvido no trabalho de Kamienski, Carlos, *et al.* [4]. O projeto tem como objetivo principal o desenvolvimento de métodos e abordagens fundamentadas em IoT para a irrigação de precisão na agricultura. A proposta do Swamp é produzir otimização na distribuição e no consumo de água fundamentadas em análise abrangente que reúne informações de todos os cenários do sistema, englobando o ciclo natural da água e a compreensão de informação sobre o crescimento das plantas.

A Figura 2.4 ilustra onde a plataforma Swap atua dentro de sistema de irrigação. A descrição de cada etapa do consumo de água dentro do sistema é representado pelos [4]:

- W1: Reservatório de água. A água originária de diferentes fontes rios, lagos, aquíferos até do mar, seguindo seu ciclo natural;
- W2: Distribuição de água;
- W3: Consumo de água. A agricultura é o maior consumidor mundial de água doce.

A plataforma Swamp irá gerar respostas em tempo real para adequar à irrigação ao passo que as condições da plantação mudem. A arquitetura da plataforma Swap é composta por 5 camadas ilustradas na Figura 2.5:

- Layer 1 - Serviços IoT: tecnologias de sensoriamento de solo, das plantas e clima.
- Layer 2 - Entidades Virtuais e Armazenamento de Dados: componentes na nuvem e na névoa computacional. E armazenar os dados captados pelos sensores.
- Layer 3 - Análise de dados: processar modelos clássicos como necessidade de água das plantas, também possui componentes fundamentados em aprendizado de máquina.

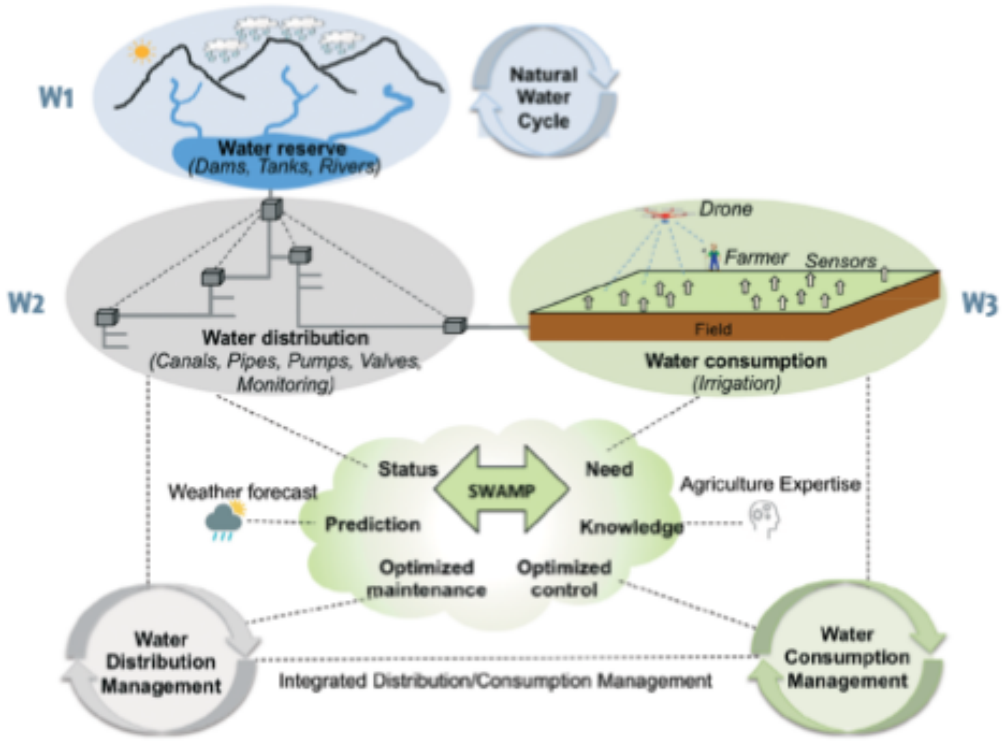


Figura 2.4: Esquema de irrigação de precisão na agricultura baseada em gereciamento inteligente de água [4].

- Layer 4 - Gerenciamento de Dados de Água: essa camada é responsável pela construção de serviços de middleware com alvo na aplicação específica. Essa camada separa as regras de negócio genéricas das aplicações através de uma API para a camada 5.
- Layer 5 – Serviços de Aplicação: Todos os coletados e armazenados são processados e são transformados em informação e serviços para agricultores.

O projeto da plataforma Swamp é uma colaboração entre instituições e empresas do Brasil e da Europa, O projeto possui dois pilotos no Brasil, na Bahia e outros dois na Europa, na Itália e na Europa [4]. Cada um dos pilotos tem um foco que é priorizado:

1. Piloto CBEC (Bolonha/Itália): Tem como foco principal a otimização na distribuição de água para propriedades associadas;
2. Piloto Intercrop(Cartagena/Espanha): Seu maior foco é fazer o uso eficiente da água;
3. Piloto Guaspari (Espirito Santo do Pinhal/Brasil): O objetivo principal dentro do piloto é aumentar a qualidade do vinho produzido;

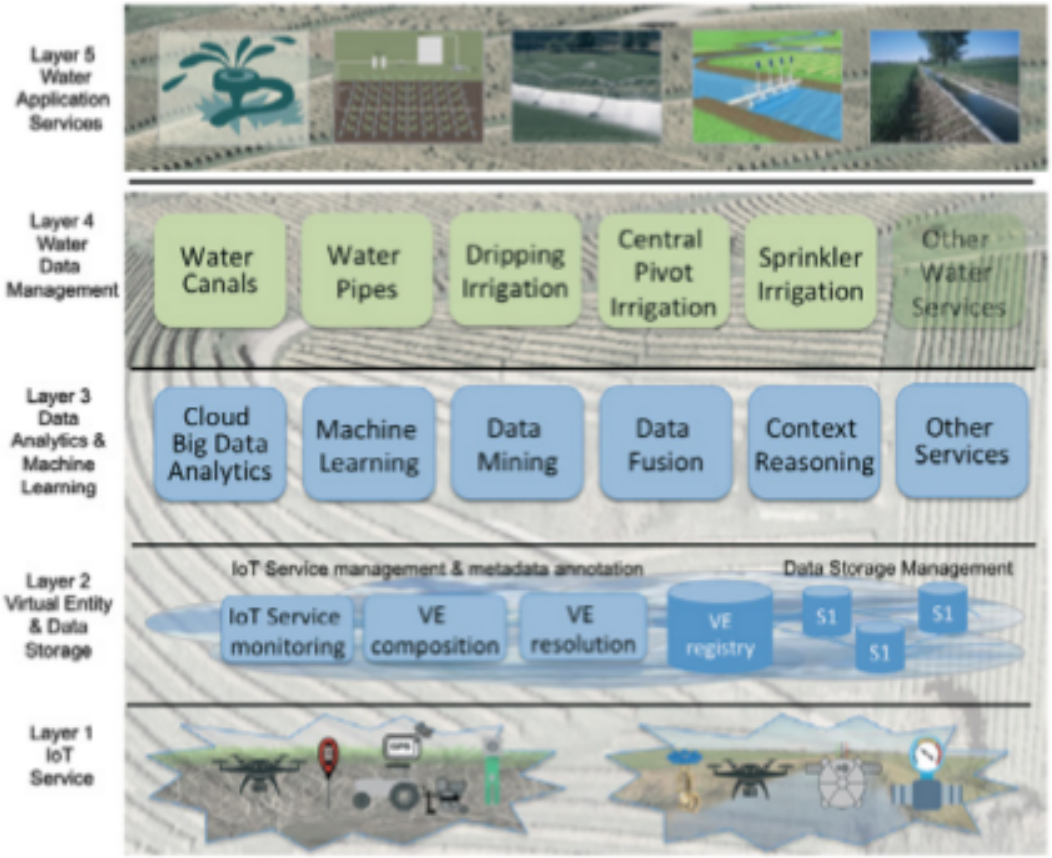


Figura 2.5: Arquitetura Swamp [4].

4. Piloto Matopiba (Luís Eduardo Magalhães/Brasil): Seu principal foco é na redução do consumo de energia.

A simulação desenvolvida neste trabalho, tenho como base para criação dos cenários o piloto desenvolvido em Matopiba (Luís Eduardo Magalhães/Brasil). Dentro do piloto Matopiba um dos grandes desafios é a comunicação, com as distâncias entre o pivô central, áreas de plantio e a sede da fazenda [4]. Na Figura 2.6 é apresentado um cenário do piloto Matopiba, um dos principais elementos dentro do cenário são:

- Sondas de solos, são constituídas por conjuntos de sensores de diferentes tipos em profundidades distintas;
- Drones, que possuem uma atuação como sensores voadores equipados com câmeras térmicas ou multiespectrais e também transporte e coleta de dados dos demais sensores;
- Pivô central, é um pivô elétrico que tem a capacidade de controlar à aspersão de água do pivô e usar taxas variadas para irrigação.

A simulação desenvolvida neste trabalho segue uma arquitetura semelhante a que foi apresentada no projeto SWAMP referente ao piloto de Matopiba. Os principais elementos no campo da simulação são os sensores que coletam os dados do solo, o drone que coleta os dados dos sensores e o pivô central responsável pela irrigação.

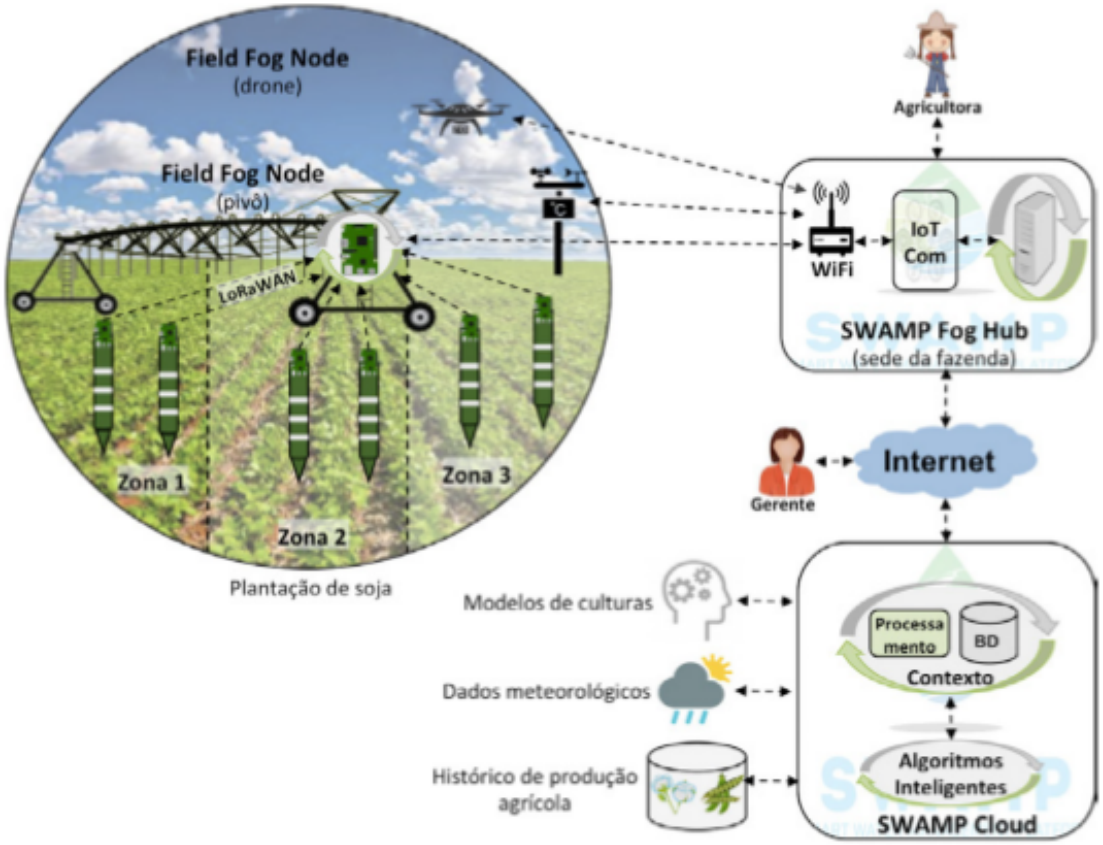


Figura 2.6: Arquitetura Swamp no piloto Matopiba [4].

2.5 Sensores Subterrâneos

No trabalho de Akyildiz, Ian F e Stuntebeck, Erich P [12] é explicado o conceito de *Wireless Underground Sensor Network(WUSN)*, que são as Redes de Sensores Sem Fio Subterrâneas(RSSFS). No artigo é fornecido uma visão geral sobre as aplicações e desafios dentro das RSSFS. Dentro da agricultura os sensores subterrâneos são usados para monitorar condições do solo como água e minerais presentes. Também são usados para monitoramento de estrutura subterrâneas como encanamento, deslizamentos de terra e terremotos esse monitoramento é feito usando sismômetros enterrados[12].

As aplicações que usam sensores subterrâneos podem ser divididas em quatro categorias:

- Monitoramento de ambiente: um exemplo são sensores subterrâneos usados para monitorar condições do solo como água e os minerais presentes;
- Monitoramento de infraestrutura: usados para monitorar estrutura subterrâneas como tubos, fiação elétrica e tanques que armazenam líquidos um exemplo são postos de combustível que armazenam combustíveis em tanques subterrâneos;
- Determinação de localização: uma aplicação desse tipo seria o uso de sensores subterrâneos em uma estrada em que sensores se comunicam com os veículos quando passam pela estrada;
- Monitoramento de segurança: essa aplicação é usada para segurança residencial e comercial, onde os sensores podem ser implantados abaixo do solo em volta de determinado perímetro para detecção de intrusos.

As RSSFS enfrentam desafios particulares devido ao seu ambiente único de implementação, esses desafios são: conservação de energia, o design de topologia, o design de antena e ambientes extremos[12]. Neste trabalho a simulação é do tipo monitoramento de ambiente e principais desafios enfrentados foram conservação de energia e design de topologia.

No artigo de Wang, Dongzhi, e Hui Ming [13] é proposto uma Rede de Sensores Subterrâneos com método de estimativa de distância para sinais sonoros e vibratórios. A qualidade da comunicação da rede dentro de uma RSSFS está diretamente relacionada com a composição do solo e a frequência de operação. Desde que grande parte dos nós estão distribuídos no solo, a implantação real de RSSFS requer mão de obra e recursos financeiros, portanto a simulação tornou-se um método importante para pesquisar redes de sensores subterrâneos sem fio.

Dentro do trabalho Xin Dong e Mehmet C. Vuran e Suat Irmak [14] é desenvolvida uma prova de conceito para um sistema autônomo de irrigação de precisão que é composto da integração de um sistema de irrigação com pivô central com Redes de Sensores Sem Fio Subterrâneos(RSSFS). A proposta é que o sistema de pivô central seja assistido por sensores subterrâneos sem fio que fornecerá capacidades autônomas de gerenciamento de irrigação, monitorando as condições do solo em tempo real usando sensores subterrâneos. Em RSSFS existem três tipos de comunicação[14]:

- Do solo para o solo (*Underground-to-underground*): tanto o nó que envia quanto o nó que recebe estão abaixo do solo;

- Do solo para superfície (*Underground-to-aboveground*): o nó que envia está abaixo do solo e nó que recebe está na superfície acima do solo;
- Da superfície para o solo(*Above Ground-to-underground*): o nó que envia está acima do solo e nó que recebe está abaixo do solo.

Na simulação desenvolvida neste trabalho o principal tipo comunicação foi do solo para superfície (*Underground-to-aboveground*).

2.6 Pivô Central

Dentro do trabalho de Evans, Robert G [15] é apresentado o conceito de pivô central e suas características. Um pivô central consiste basicamente em uma tubulação montada em estruturas motorizadas com rodas para locomoção chamadas de torres. Uma máquina de pivô central gira em torno de um ponto o "pivô" no centro do campo.Um pivô central gira em torno de uma estrutura de tubo no centro do campo, dessa forma cobrindo a região do círculo com menos de 360°, um pivô central pode cobrir de 80% a 90% da área de um quadrado [15]. As principais vantagens são a otimização na aplicação de fertilizantes e água, a redução de custos referentes à mão de obra e a irrigação de plantações de longas distâncias.

Dentro do artigo de Matilla, Diego Mateos, *et al.* [16] é feita uma análise das principais tecnologias de comunicação de sensores utilizadas para o monitoramento de irrigação com pivôs. Os sistema de irrigação de plantações podem ser divididos em três categorias [16]:

- Gravidade: a superfície é inundada com água e a gravidade distribui à água sobre a lavoura ou através de sulcos;
- Micro Irrigação: sistemas de baixa pressão de água que aplicam pequenas quantidades de água perto das plantas de cultivo ou por baixo do solo perto das raízes;
- Sistemas sprinklers(pulverizadores): usam pressão para pulverizar água através de bicos em campos que formam um padrão de pulverização criando precipitação artificial. Dentro desse tipo existem três subtipos: i) pivô central para regar grandes áreas circulares, ii) Aspersores de torre de movimento linear que se movem em linha reta e regam uma área retangular e iii) Aspersores permanentes: ligado a tudo no solo ou enterrado em um padrão em toda a safra;

Neste trabalho a simulação usa o sistema de irrigação o pivô central os cenários estão detalhados no Capítulo 4.

Figura 2.7: Imagem de um pivô central.[5]

2.7 Castalia

Castalia é um simulador de código aberto para Redes de Sensores Sem Fio(RSSF) e *Body Area Network*(BAN) que é amplamente usado na comunidade acadêmica [17]. É um simulador baseada na plataforma OMNeT++ e que é usado por pesquisadores e desenvolvedores que testam e distribuem algoritmos e protocolos realísticos para *wireless channel* e modelos de rádio [6]. No Castalia também é possível avaliar diferentes características da plataforma para aplicações específicas e pode realizar simulações de uma ampla variedade de plataformas.

As principais características do Castalia são [18]:

- Ser baseado em medições de dados realísticos para melhorar o canal: i) Os nós não simplesmente se conectam uns aos outros, o caminho de perda de pacote trabalha durante a transferência de pacotes. ii) Modelos complexos de variação temporal para caminho de perda. iii) Suporta mobilidade dos nós. iv) intensidade do sinal é calculada com interferência;
- Os modelos de rádio são baseados em rádios reais de baixa potência: i) Multiníveis de potência TX. ii) estados com diferentes consumos de energia e atrasos alternados entre eles. iii) modelagem realística de RSSI;
- Clock drift é implementado para maior realismo nas comunicações sem fio;
- Protocolos MAC e roteamento disponíveis;
- O Castalia é projetado para o desenvolvimento de novos módulos e melhoramento;
- Alto desempenho herdado do OMNet++ event-driven simulator.

A arquitetura do castalia é composta por nós que são conectados ao *Wireless Channel* é por onde um nó envia um pacote para outro nó. O *Wireless Channel* é o canal onde ocorrem as trocas de mensagem. Os nós também são conectados ao processos físicos que os mesmos monitoram através do *Physical Process*. Um nó pode monitorar vários processos físicos, A arquitetura do Castalia é ilustrada na Figura 2.8.

Na Figura 2.9 é ilustrado a estrutura interna de nó do Castalia. O nó é composto pelos módulos:

- *Sensors Manager*: é o módulo responsável pelos sensores, o sensor manager é conectado diretamente com os processos físicos;
- *Application*: é módulo responsável pela parte de aplicação da rede, dentro da aplicação os pacotes são enviados para outras camadas das redes;

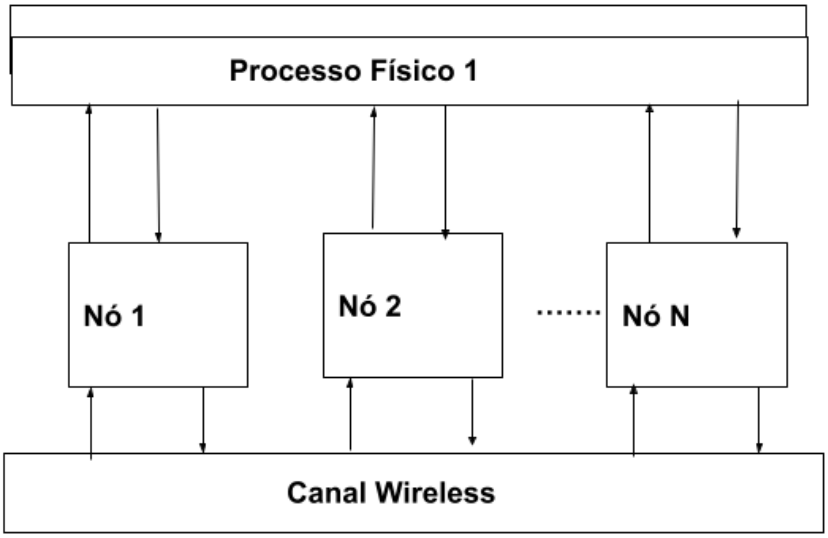


Figura 2.8: Arquitetura Castalia Simulator.

- *Routing*: o módulo de *Routing* é onde será implementado os algoritmos de roteamento;
- *MAC*: o módulo onde estão implantados os protocolos *MAC(media access control)*;
- *Radio*: o módulo onde é implementado a comunicação por rádio e todas suas configurações;
- *Resource Manager*: o módulo que gerencia todos os recursos de hardware do nó como bateria, CPU e memória, sua principal função é controle de energia;
- *Mobility Manager*: módulo responsável pela mobilidade do nó ou seja sua movimentação pelo campo de simulação;

Cada nó possui todos os componentes descritos acima e na Figura 2.9. Para implementação da simulação presente neste trabalho foi feita uma pequena alteração em relação aos módulos de mobilidade. Com a alteração agora os módulos de mobilidade possuem uma conexão com gerenciador de recursos, assim permitindo que um módulo de mobilidade envie mensagens para um módulo de gerenciamento de recursos para efetuar um desconto de energia no nó. Na Figura 2.10 é ilustrado essa modificação dentro da estrutura de um nó no Castalia e na Figura 2.9 é o nó em sua composição original.

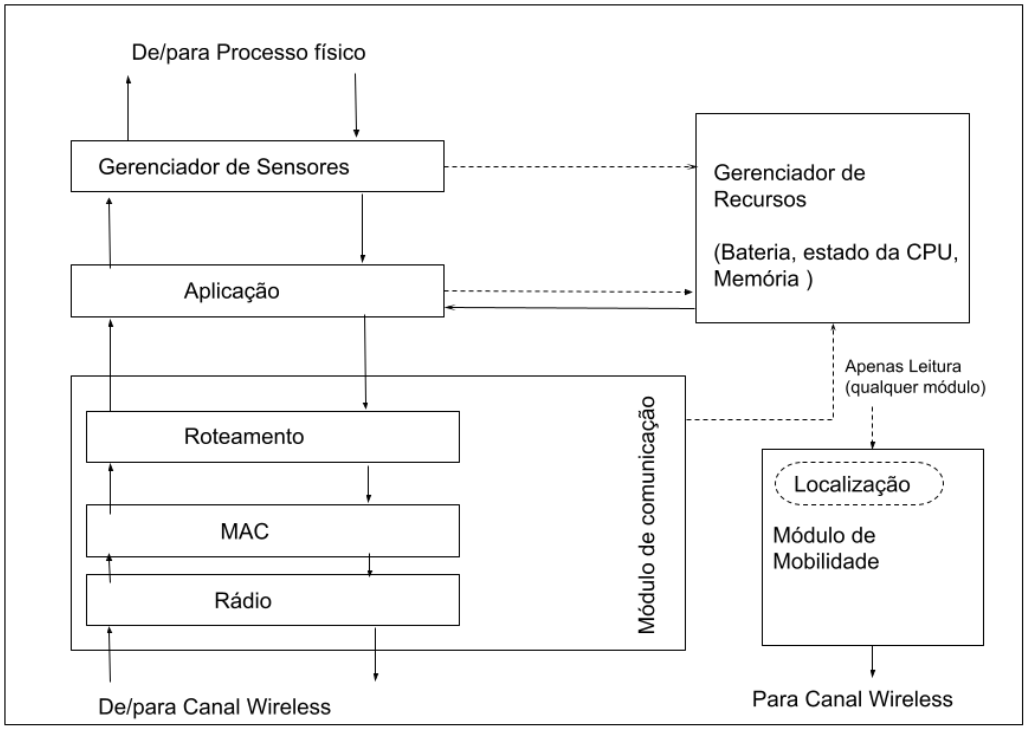


Figura 2.9: Submódulos de um nó do simulador Castalia

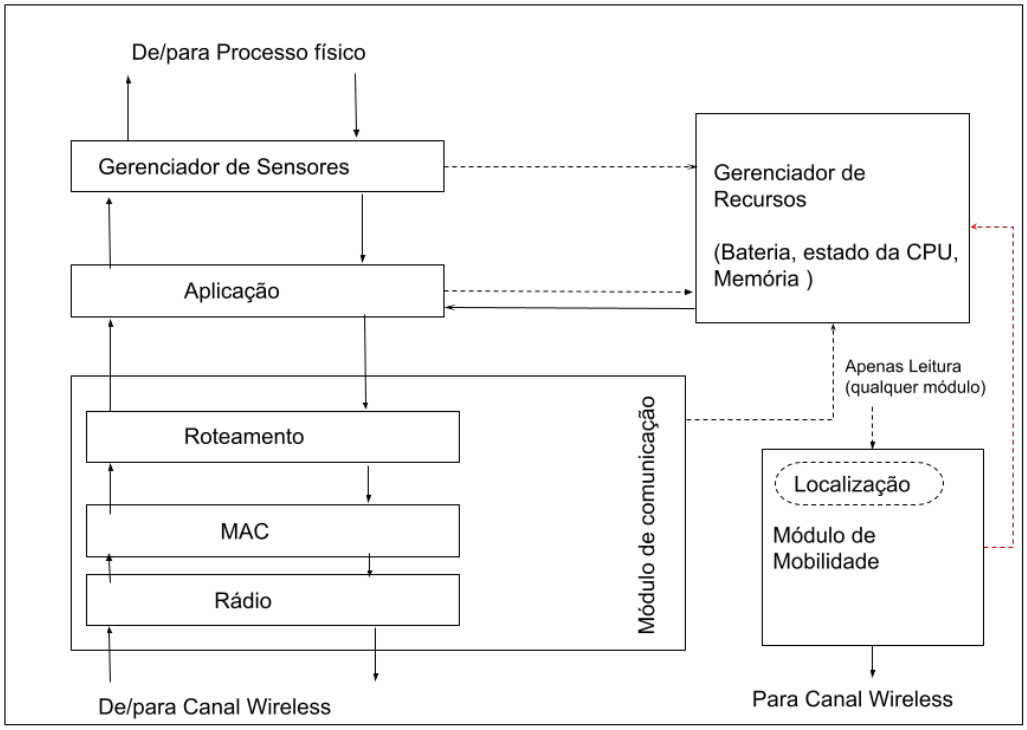


Figura 2.10: Submódulos de um nó do simulador Castalia com modificação

2.8 Omnet++

OMNeT++ é um framework para construir simuladores de redes que foi desenvolvido na linguagem de programação C++. O OMNeT++ ganhou muita popularidade na comunidade científica como uma plataforma de simulação de redes. Os componentes dentro do são programados em C++ são chamados de módulos, as estruturas e componentes físicos são especificados na linguagem NED. Os principais partes que constituem o OMNeT++ [19]:

- Biblioteca do kernel de simulação (C++);
- A linguagem de descrição NED;
- IDE de simulação baseada na plataforma Eclipse;
- Simulação iterativa GUI (Qtenv);
- Interface de linha de comando para execução de simulação (Cmdenv);
- Utilitários (makefile crition tool, etc).

A linguagem de descrição NED descreve a topologia do modelo, NED é a abreviação de *Neworking Descriptor*. Dentro do arquivo NED é definido os parâmetros do módulo da rede, os submódulos, e conexões do módulo. Os arquivos NED são responsáveis por definir a estrutura de um nó e a topologia da rede dentro de um projeto Omnet++.

2.9 Considerações Finais

Neste capítulo foram apresentados conceitos fundamentais para entendimento do trabalho desenvolvido nos Capítulo 3 e o Capítulo 4. Na simulação apresentada neste trabalho é abordada algumas limitações das RSSF como topologia e consumo energético. É utilizado o modelo de entrega de dados orientado a eventos na simulação, para que os sensores envie dados para o drone o mesmo precisa estar dentro de um raio de transferência de dados. O conceito de coletor de dados de uma RSSF [10] é desempenhado pelo drone na simulação que será apresentada posteriormente. Para caracterização da RSSF apresentada na simulação os cenários serão executados com diferentes valores para os parâmetros de envio de pacotes e potência de rádio.

Capítulo 3

Cenários

Neste capítulo apresenta uma contextualização da simulação e seus cenários. A seção de contextualização tem como objetivo esclarecer o contexto em que a simulação está inserida. A seção de parâmetros busca elucidar os elementos da simulação e os parâmetros aplicados.

3.1 Contextualização

Atualmente práticas agrícolas como semear, irrigar, manejar fertilizantes e controlar pragas, são tarefas possíveis de serem realizadas de forma autônoma de acordo com a avaliação do campo e infraestrutura disponível. Para um sistema de irrigação inteligente é fundamental o conhecimento de quando deve ser feita a irrigação e quanto de água deve ser usada para otimizar a produção da plantação [14]. O gerenciamento de irrigação apropriado terá um efeito de minimizar as perdas de produção devido falta de água para plantas e otimizar o uso racional da água diminuindo o desperdício hídrico.

Um sistema de irrigação inteligente é usado como estudo de caso para caracterização da RSSF aqui apresentada. A simulação desenvolvida neste trabalho segue uma infraestrutura semelhante ao piloto de Matopiba do projeto SWAMP [4], em que campo é irrigado por pivô central, e a rede constituída por sensores subterrâneos que monitoram o solo e o drone fazendo o papel de coletor dados coletando os dados dos sensores. O propósito da simulação é caracterizar uma rede de sensores sem fio do sistema de irrigação inteligente a partir dos dados de consumo energético e transferência de dados. Um dos principais objetivos é encontrar um cenário que otimize a coleta de dados e o tempo de vida da rede.

A simulação é composta por três elementos básicos: o pivô central, sensores e o drone. O pivô central tem como propósito realizar a irrigação da plantação, os sensores no solo são responsáveis por coletar informações de qualidade e umidade do solo e o drone é

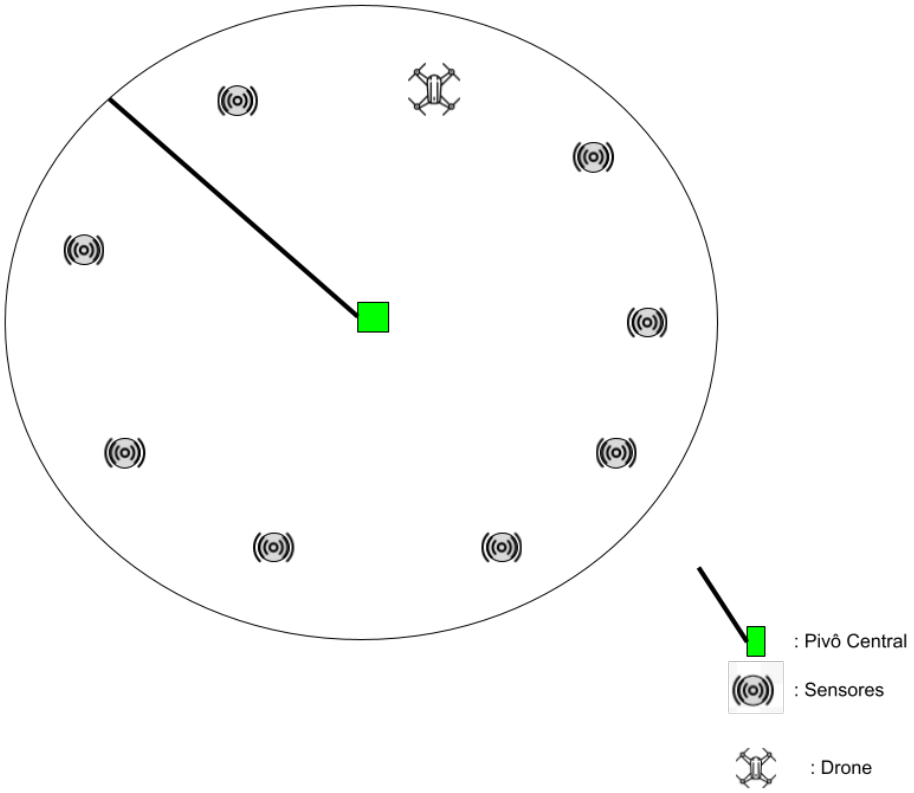


Figura 3.1: Componentes básicos da simulação.

responsável por coletar as informações de todos os sensores e transmitir para nó externo que tratará as informações. Na Figura 3.1 é ilustrado os componentes que estão presentes na simulação.

O drone coletor de dados é o único nó da rede que se movimenta, o pivô central não é um nó da rede. O drone faz uma rota circular no campo coletando os dados dos sensores. A simulação será executada com dois cenários: no primeiro o drone realiza todo o percurso do campo por conta própria e no segundo durante um quarto do campo o drone fica sob o pivô central e usa o movimento do pivô, com isso o drone se locomove mais lentamente mas obtém uma economia de energia. Nos dois cenários, é analisado se houve ganho tanto no tempo de vida da rede quanto no tempo de vida do coletor de dados, os dados são apresentados no Capítulo 5.

3.2 Parâmetros

3.2.1 Campo

O Campo da simulação foi definido com as dimensões de 500 metros de largura e 500 metros de comprimento. Devido ao custo do pivô central, geralmente seu uso é feito para irrigar áreas de 3.5 hectares para 65 hectares [14], às dimensões do campo de simulação foi definido de acordo com a capacidade de irrigação de pivô central e as dimensões do experimento realizado no trabalho de Dong, Xin, et al [14]. A área do campo de simulação possui aproximadamente 25 hectares.

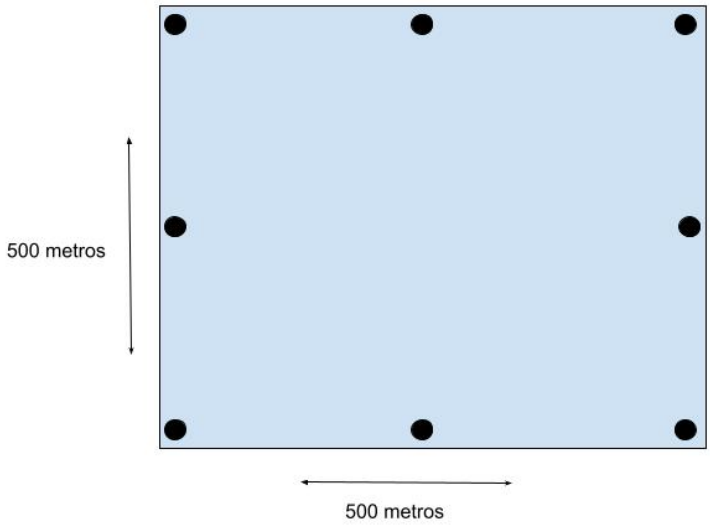


Figura 3.2: Mapeamento dos sensores no campo de simulação.

3.2.2 Topologia

No campo terão um total 8 nós de sensores distribuídos como ilustrado na Figura 3.2, na Tabela 3.1 é mostrado a posição de cada nó segundo eixo x e eixo y do plano cartesiano. O posicionamento dos sensores teve como base o posicionamento de sensores realizado no experimento do trabalho de Dong, Xin, et al [14]. No experimento usado 8 nós sensores nas extremidades de uma circunferência, o primeiro sensor é posicionado no grau 0° e os sensores seguintes cada 45° a frente do anterior como é ilustrado na Figura 3.3. Na

simulação deste trabalho o campo possui as dimensões de 500x500 e os 8 nós sensores estão posicionados como ilustrado na Figura 3.2.

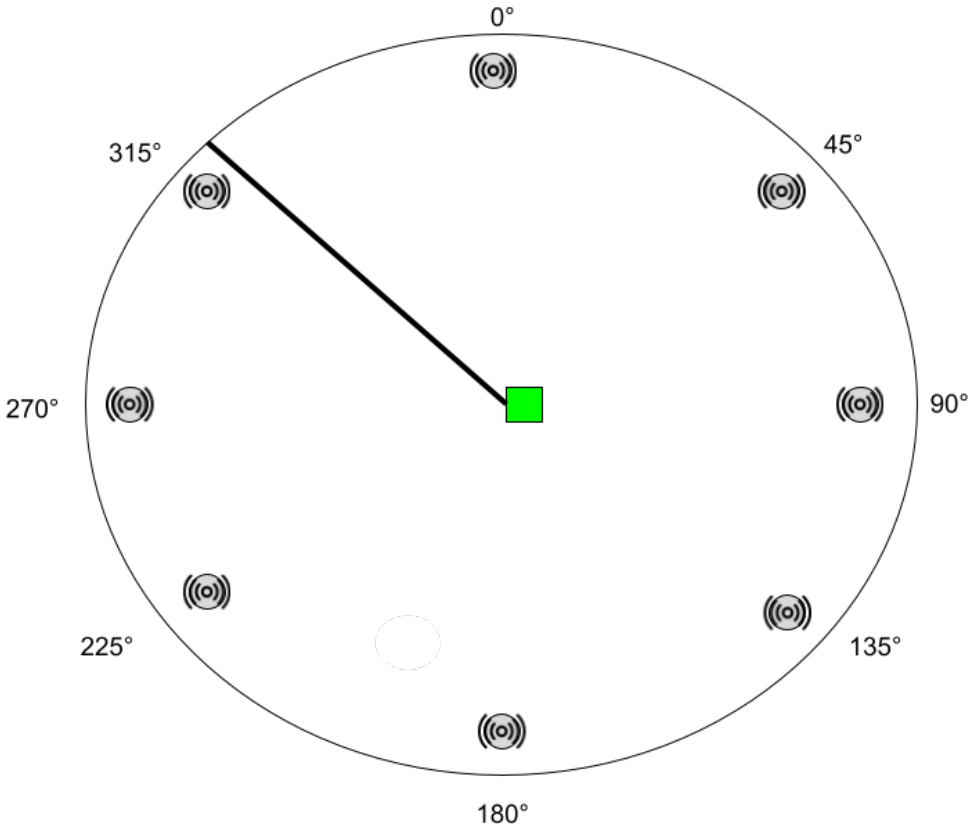


Figura 3.3: Posicionamento dos sensores em uma circunferência.

3.2.3 Drone

O drone é o coletor de dados da rede é o único nó móvel da rede, o mesmo faz uma rota circular no campo para realizar a coleta de dados dos sensores. A posição inicial do drone considerando o campo em coordenadas no plano cartesiano é x igual à 0 e y igual à 0, ao iniciar a simulação o drone segue o rota circular pelas extremidades do campo passando pelos sensores até o fim da simulação voando a uma altura de 0,5 metros, a rota do drone é ilustrada na Figura 3.5. O gasto de energia do drone por metro foi definido como 1500 mW para simulação tendo como base o gasto de energia do drone *Parrot Ar* para o movimento horizontal apresentado no trabalho de Ahmed, Shaimaa, et al. [20], a velocidade média do drone assumida é de 10 metros por segundo, foi definida dentro da faixa de velocidade do modelo *Parrot Ar*.

Os nós de sensores do solo possuem uma distância de transferência de dados para drone, que é ilustrada na Figura 3.2. Quando o drone está a uma distância de 20 metros

Tabela 3.1: Posição dos nós de sensores de solo.

Nó	**X**	**Y**
Nó[1]	0	0
Nó[2]	250	0
Nó[3]	500	0
Nó[4]	500	250
Nó[5]	500	500
Nó[6]	250	500
Nó[7]	0	500
Nó[8]	0	250

considerando o eixo x e o eixo y, ocorre a transmissão de dados dos sensores para o drone. Assim oferece uma possibilidade de uma transmissão mais confiável e diminuição do gasto de energia da rede.

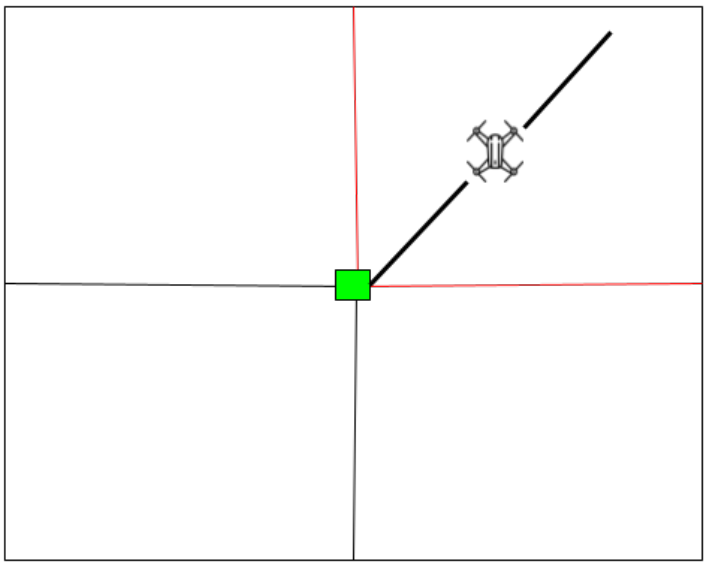


Figura 3.4: Área do campo que drone usa o movimento do pivô central.

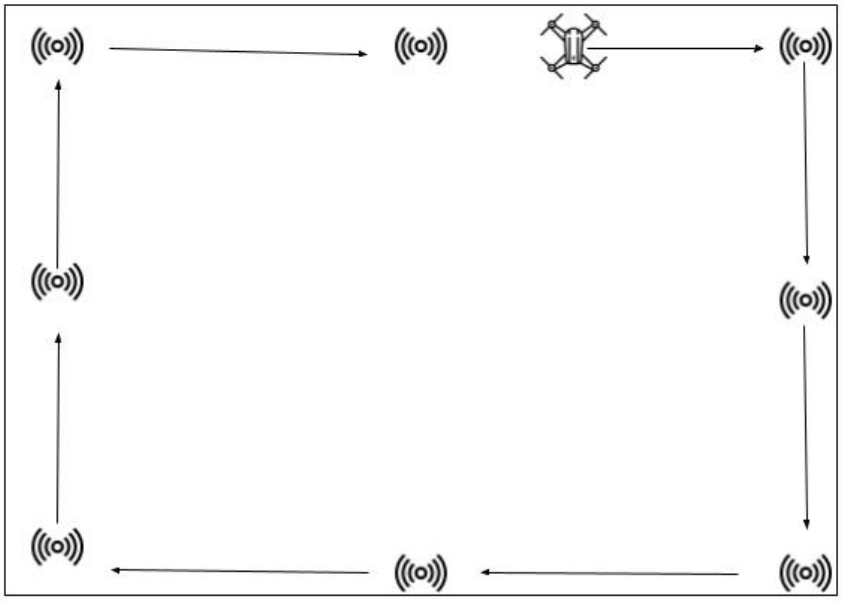


Figura 3.5: Movimentação do drone pelo campo.

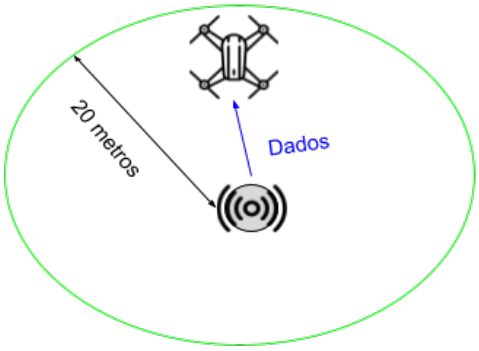


Figura 3.6: Zona de transferência de dados sensores para o drone.

3.2.4 Pivô Central

O pivô central é responsável pela irrigação dentro do sistema irrigação. Dentro da simulação o pivô central não é um nó da rede, o mesmo é abstraído dentro módulo de mobilidade do simulador Castalia. Na simulação o pivô central é utilizado em um dos cenários para economia energética do drone, neste cenário em um quarto do campo o drone fica sob o pivô usando o movimento do mesmo para se locomover como é ilustrado na Figura 3.4. Um pivô central pode ser mover em uma velocidade de 250 metros/hora [21] convertendo para metros por segundo resulta em aproximadamente 0,07 m/s, que a velocidade usada para o pivô na simulação.

3.3 Síntese do Capítulo

O Capítulo 3 apresentou o contexto do estudo de caso usado para a simulação, um sistema de irrigação inteligente que foi utilizado como estudo de caso. O capítulo ilustrou e apresentou elementos presentes na simulação e seus papéis dentro do sistema. Neste capítulo também descreveu e elucidou parâmetros aplicados na simulação.No Capítulo 4 será aprofundado que foi apresentado no Capítulo 3 à nível de implementação no simulador Castalia.

Capítulo 4

Protótipo

4.1 Arquitetura Castalia

O simulador usado nas simulações foi o Castalia que um simulador de RSSF e Body Area Network. O castalia foi desenvolvido usado como base o OMNet++ é um framework para construir simuladores de redes, O OMNeT++ foi desenvolvido com a linguagem de programação C++. Os módulos do castalia estão estruturados como na Figura 4.1, módulo SN(*Sensor Network*) representa a rede de sensores, todos os nós da rede pertencem ao módulo SN. O módulo de phycalProcess é responsável por todos os processos físicos, que são os fenômenos que uma rede de sensores monitoram. O módulo wirellesChanell é o canal pelo qual os nós da rede se comunicam. Na Figura 4.2 é ilustrado como um diagrama como funciona a interação desses módulos.

Dentro do módulo SN possui um vetor de *Node* que são os nós da rede, cada *Node* possui os submódulos de aplicação, comunicação, gerenciamento de mobilidade, gerenciamento de sensores e gerenciamento de recursos. Os módulos de aplicação são responsáveis pela camada de aplicação. Os módulos de comunicação são responsáveis pela transmissão, protocolos MAC e protocolos de roteamento. Os módulos de gerenciamento de mobilidade possuem a responsabilidade de movimentar o nó pelo campo de simulação. Os módulos de gerenciamento de recursos tem como principal função gerir energia de um nó. Os módulos de gerenciamento de sensores são responsáveis por fazer a conexão com os processos físicos e gerir a configuração dos sensores. Na Figura 4.3 ilustrada a estrutura interna de node do Castalia e marca os módulos que receberam adições para realizar a simulação.

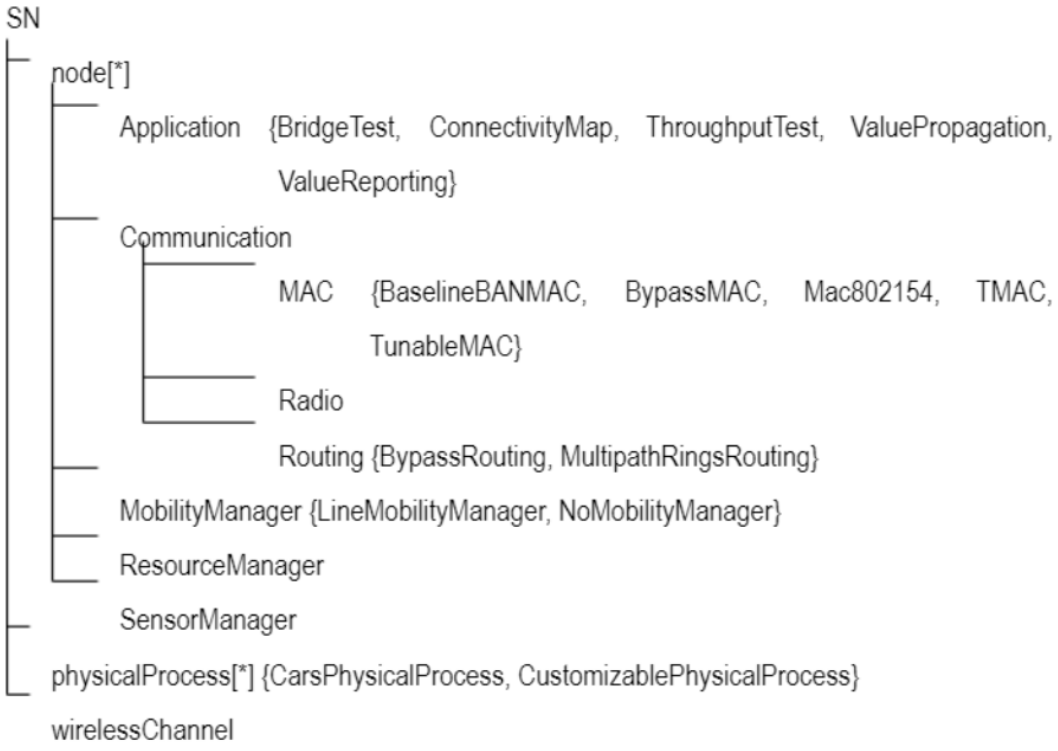


Figura 4.1: Estrutura módulos Castalia [6]

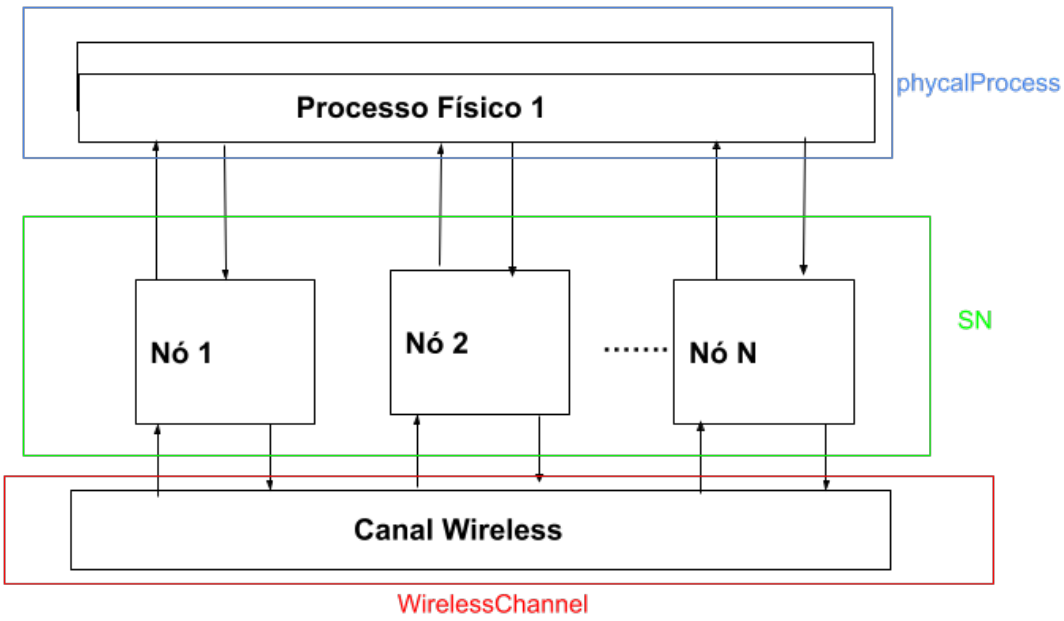


Figura 4.2: Arquitetura Castalia

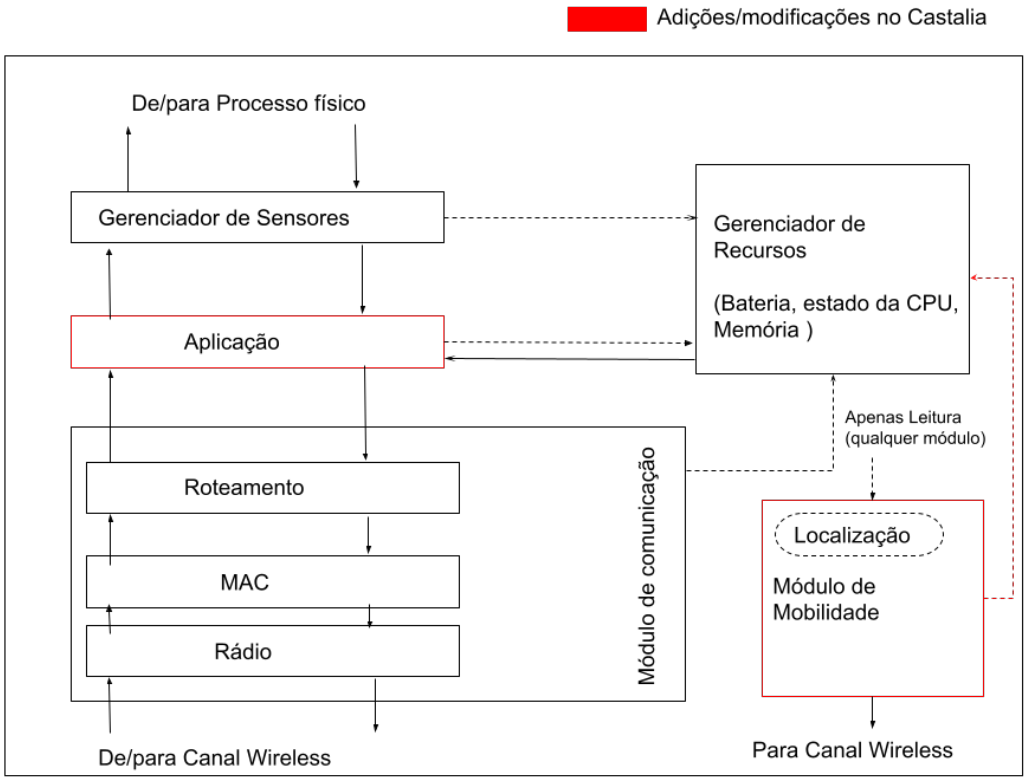


Figura 4.3: Estrutura interna de um node do Castalia

4.2 Desenvolvimento

4.2.1 Módulo de mobilidade

Para criação dos cenários apresentados e simulados dentro deste trabalho foi necessário à criação de novas implementações dentro do simulador Castalia. Foi criado um módulo de mobilidade para a movimentação do drone durante a simulação. A criação de um módulo de segue e herda a implementação do módulo genérico de movimentação chamado *VirtualMobilityManager*.

Algoritmo 4.1: VirtualMobilityManager

```
#ifndef _VIRTUALBILITYMANAGER_H_
#define _VIRTUALBILITYMANAGER_H_

#include "WirelessChannelMessages_m.h"
#include "CastaliaModule.h"

using namespace std;

struct NodeLocation_type {
        double x;
        double y;
        double z;
        double phi;
        double theta;
        int cell;
};

class VirtualMobilityManager: public CastaliaModule {
 protected:
        NodeLocation_type nodeLocation;
        int index;

        cModule *node, *wchannel, *network;
        WirelessChannelNodeMoveMessage *positionUpdateMsg;

        virtual void initialize();
        virtual void notifyWirelessChannel();
        virtual void setLocation(double x, double y, double z, double phi =
            0, double theta = 0);
        virtual void setLocation(NodeLocation_type);
        void parseDeployment();

 public:
```

```
        virtual NodeLocation_type getLocation();
};
#endif
```

Dentro da classe *VirtualMobilityManager* possui uma estrutura *NodeLocation_type* que contém todos os dados sobre a localização do nó dentro do campo. A classe possui os métodos *initialize*, *notifyWirelessChannel* e *setLocation*. O método *Initialize* é responsável por declarar a posição inicial do nó. O método *notifyWirelessChannel* faz o envio de mensagem para o *Wireless Channel*. O método *setLocation* define uma nova localização para o nó.

Para realizar a mobilidade do drone foi criado o módulo *CircleMobilityManager*, o módulo realiza uma movimentação circular com nó no sentido horário pelas extremidades do campo. Variações de posição ocorrem no eixo x e eixo y, ou seja, a altura do nó é pré-definida. A movimentação do pivô central é artificialmente simulada dentro desse módulo, contendo dois parâmetros referentes ao pivô: um parâmetro booleano que definirá se o drone usará à movimentação do pivô para economia de energia e o outro parâmetro definirá a velocidade em metros por segundo do pivô.

Algoritmo 4.2: CircleMobilityManager

```
#ifndef _MOBILITYMODULE_H_
#define _MOBILITYMODULE_H_

#include "MobilityManagerMessage_m.h"
#include "VirtualMobilityManager.h"

using namespace std;

class CircleMobilityManager: public VirtualMobilityManager {
 private:
        /*--- The .ned file's parameters ---*/
        double updateInterval;
        double speed;
        double drawEnergy;
        bool droneOnPivo;
        double pivo_speed;

        /*--- Custom class parameters ---*/
    int x_field;
    int y_field;
    int z_field;
    int totalSNnodes;


 protected:
        void initialize();
        void handleMessage(cMessage * msg);
};

#endif
```

Parâmetros da classe CircleMobilityManager:

- *updateInterval*: responsável pelo tem de movimentação em milisegundos;
- *speed*: é a velocidade do drone em metros por segundo;
- *drawEnergy*: o gasto de energia por metro para o movimento drone em miliWatts;
- *droneOnPivo*: um booleano que determina se o drone irá ou não usar o movimento pivô;
- *pivo_speed*: a velocidade do pivô em metros por segundo;
- *x_field*: a largura do campo de simulação em metros;
- *y_field*: o comprimento do campo de simulação em metros;
- *z_field*: a altura do campo de simulação em metros;
- *totalSNnodes*: o total de nós na rede.

4.2.2 Módulo de aplicação

Para criação de um módulo de aplicação é necessário que a classe do novo módulo tenha herança da classe *VirtualAplication*. Para realização da simulação foi produzido um novo módulo de aplicação o *NewApp* baseado no módulo *ThroughputTest*. Por padrão o módulo módulo faz com que todos os nós da rede envie pacotes para Node[0] que no caso é o drone e também dentro do módulo é implementado a zona de transferência de dados do sensores para o drone.

Algoritmo 4.3: VirtualApplication

```
#ifndef VIRTUALAPPLICATIONMODULE
#define VIRTUALAPPLICATIONMODULE

#include <sstream>
#include <string>
#include <omnetpp.h>
#include "ApplicationPacket_m.h"
#include "SensorManagerMessage_m.h"
#include "ResourceManager.h"
#include "VirtualEnergyManager.h"
#include "Radio.h"
#include "VirtualMobilityManager.h"
#include "CastaliaModule.h"
#include "TimerService.h"
```

```
#define SELF_NETWORK_ADDRESS selfAddress.c_str()

using namespace std;

class VirtualApplication: public CastaliaModule, public TimerService {
 protected:
        /*--- The .ned file's parameters ---*/
        string applicationID;
        int priority;
        int packetHeaderOverhead;
        int constantDataPayload;
        bool isSink;
        double latencyMax;
        int latencyBuckets;

        /*--- Custom class parameters ---*/
        int self;

        string selfAddress;
        ResourceManager *resMgrModule;
        VirtualEnergyManager *enMgrModule;
        VirtualMobilityManager *mobilityModule;
        Radio *radioModule;
        bool disabled;
        double cpuClockDrift;

        virtual void initialize();
        virtual void startup() {}
        virtual void handleMessage(cMessage * msg);
        virtual void finish();
        virtual void finishSpecific() {}

        void requestSensorReading(int index = 0);
        void toNetworkLayer(cMessage *);
        void toNetworkLayer(cPacket *, const char *);

        ApplicationPacket *createGenericDataPacket(double, unsigned int,
            int = -1);
        virtual void fromNetworkLayer(ApplicationPacket *, const char *,
            double, double) = 0;
        virtual void handleSensorReading(SensorReadingMessage *) {}
        virtual void handleNetworkControlMessage(cMessage *) {}
        virtual void handleMacControlMessage(cMessage *) {}
        virtual void handleRadioControlMessage(RadioControlMessage *) {}
```

```
};

#endif
```

Parâmetros da classe VirtualApplication:

- *self*:Id do nó atual;
- *selfAddress*: endereço do nó atual;
- *resMgrModule*: ponteiro para o módulo de gerencimento dos recursos;
- *enMgrModule*: ponteiro para o módulo de gerencimento de energia;
- *mobilityModule*: ponteiro para o módulo de mobilidade;
- *radioModule*: ponteiro para o módulo de rádio;
- *disabled*: determina se o módulo de aplicação está habilitado para envio de pacotes;
- *cpuClockDrift*: variável do clock dentro do módulo de aplicação.

Na classe *VirtualAplication* é a classe genérica para módulos de aplicação, todo módulo de aplicação deve herdar da *VirtualAplication*. A classe possui os métodos *initialize*, *startup*, *handleMessage*, *finish*, *finishSpecific*, *requestSensorReading* e *toNetworkLayer*. O método Initialize é iniciar todas as propriedades e define o qual a aplicação pelo *applicationID*. O método *handleMessage* é responsável pelo tratamento de mensagens que chegam no nó. O método *finishSpecific* determina o final para uma aplicação que herda da classe *VirtualAplication*. O método *toNetworkLayer* encaminha os pacotes para os módulos de comunicação.

Dentro do módulo *NewApp* no momento do envio do pacote, os sensores somente conseguem enviar pacotes para o drone quando estão a uma determinada distância do mesmo, que é a zona de transferência de dados ilustrado na Figura 3.6. A implementação tem como objetivo a diminuição da taxa de perda de pacotes e o aumento do tempo de vida da rede. Como os nós não estarão enviando pacotes a todo momento isso reduzirá a taxa de perda de pacote e também o gasto de energia do envio desses pacotes.

Algoritmo 4.4: NewApp

```
#ifndef _NewApp_H_
#define _NewApp_H_
#include "VirtualApplication.h"

#include <map>

enum NewAppTimers {
        SEND_PACKET = 1
};

using namespace std;
class NewApp: public VirtualApplication {
 private:
        double packet_rate;
        double startupDelay;
        double delayLimit;
        float packet_spacing;
        int dataSN;
        int recipientId;
        string recipientAddress;

        double raioPacket;

        //variables below are used to determine the packet delivery rates.

        int numNodes;
        map<long,int> packetsReceived;
        map<long,int> bytesReceived;
        map<long,int> packetsSent;
        VirtualMobilityManager *mobility;

protected:
        void startup();
        void fromNetworkLayer(ApplicationPacket *, const char *, double,
            double);
        void handleRadioControlMessage(RadioControlMessage *);
```

```
        void timerFiredCallback(int);
        void finishSpecific();

 public:
        int getPacketsSent(int addr) { return packetsSent[addr]; }
        int getPacketsReceived(int addr) { return packetsReceived[addr]; }
        int getBytesReceived(int addr) { return bytesReceived[addr]; }

};

#endif
```

Parâmetros da classe NewApp:

- *packet_rate*: define a taxa de pacotes por segundo;
- *startupDelay*: tempo de *delay* para início de envio de pacotes;
- *delayLimit*: limite *delay* para recebimento do pacote no nó destino;
- *packet_spacing*: tempo de intervalo entre o envio dos pacotes;
- *recipientId*: Id do nó destino do pacote;
- *dataSN*: contador de pacotes enviados na rede;
- *recipientAddress*: endereço do nó destino do pacote.

4.2.3 Função *PowerDrawn*

Dentro da pasta */src/helpStructure* do simulador Castalia foi adicionada uma alteração para que possibilitasse que módulos de mobilidades tivesse permissão para usar a função de gasto de energia. A função *powerDrawn* é a função responsável pelo gasto de energia. Originalmente somente módulos do *SensorManager* e *Radio* possuíam permissão para executar a função *powerDrawn*, como a alteração feita no código possibilitou que os módulos de mobilidade também tivesse a permissão para uso da função. O arquivo editado para essa modificação foi o *CastaliaModule.cc*.

Algoritmo da função powerDrawn:

function POWERDRAWN($power$)
  if $classPointer$ is not $resourceManager$ **then**
    $name \leftarrow getName()$
    if $name ==' Radio'$ **or** $name$ == 'SensorManager' **or** $name$ == 'MobilityManager' **then**
      $classPointer \leftarrow getSubmodule('ResourceManager')$
    end if
  else
    throw Error("Modulo não tem permissão para chamar a função powerDraw()")
  end if
  $drawPowerMsg \leftarrow ResourceManagerMessage("PowerconsumptionMessage")$
  $drawPowerMsg.setPowerConsumed(power)$
end function

No algoritmo acima é descrito a função *PowerDrawn(power)* que é responsável por realizar o desconto de energia na bateria de nó dentro do simulador Castalia. Originalmente o simulador Castalia somente permite que os módulos de *Radio* e *SensorManager* usem a função *PowerDrawn* mas com a modificação feita para este trabalho os módulos de mobilidade também possuem a permissão para uso da função. Primeiro passo na função é verificar se a variável *classPointer* está apontando para o *resourceManager*, caso não esteja é verificado se o módulo é um módulo *Radio*, *SensorManger* ou *MobilityManager* se sim o *classPointer* recebe o módulo de *resourceManager* senão é disparado um erro informando que módulo não possui permissão para uso da função. Caso o módulo tenha permissão é criado uma mensagem de consumo de energia e é efetuado o desconto de energia com valor passado na variável *power*.

4.3 Especificações

O sistema operacional no qual a simulação é executada, é o Ubuntu versão 20.04 que usa versão 5.13 do kernel Linux. O OMNeT++ foi utilizado em sua versão 4.6, à instalação do mesmo é um pré-requisito para instalação do Castalia. A versão do Castalia utilizada foi a 3.3, os dados a respeito de ambiente também estão presentes na Tabela 4.1.

Todos os parâmetros da simulação estão descritos na Tabela 4.1, a simulação possui um tempo de execução total de 1000 segundos. A rede possui um total de 9 nós, sendo que o primeiro nó é o drone e os demais nós são sensores. O campo possui uma largura de 500 metros que no Castalia é definido com field_x = 500, possui o comprimento de 500 metros que no Castalia é definido com field_y = 500 e possui uma altura de 5 metros que no Castalia é definido com field_z = 5. As potências rádio usadas são 5dBm, -10dBm e -15dBm. As taxas de pacotes são de 5 pacotes por segundo, 20 pacotes por segundo e 80 pacotes por segundo. Para cada cenário foram executadas todas as combinações entre as potências de rádio e as taxas de pacotes por segundo.

Os dois grandes cenários da simulação se diferenciam pela movimentação do drone, no primeiro cenário o drone realiza todo o percurso por conta própria e no segundo cenário em um quarto do percurso se move sob o pivô economizando energia. No primeiro cenário todo o percurso o drone a velocidade é de 10 metros por segundo, e no segundo cenário durante um quarto do percurso quando o drone está sob pivô assumirá a mesma velocidade do pivô de 0,07 metros por segundo. A variável *"droneOnPivo"* determina qual cenário será executado, quando estiver definida como *false* o drone percorrerá todo percurso por conta própria e quando estiver definida como *true* o drone durante um quarto do percurso fica sob o pivô central.

Tabela 4.1: Ambiente e Softwares.

Softwares/SO	**Versões**
Castalia Simulator	3.3
OMNet++	4.6
Ubuntu	20.04.4 LTS

Tabela 4.2: Parâmetros de simulação.

Nomes das variáveis	**Parâmetros de simulação**	**Valores**
sim-time-limit	Tempo de simulação	1000s
SN.numNodes	Número de nós	9
SN.field_y e SN.field_x	Área do campo	500x500 m^2
SN.field_z	Altura do campo	5 metros
MobilityManager.speedDrone	Velocidade drone	10 m/s
MobilityManager.speedPivo	Velocidade do Pivô	0.07 m/s
Radio.maxPhyFrameSize	Tamanho máximo pacote	2 kb
Radio.TxOutputPower	Potência rádio	{5dBm,-10dBm,-15dBm}
Application.packet$_rate$	Taxa de pacotes/s	{5,20,80}
Application.raioPacket	Zona de transferência de dados	20 metros
MobilityManager.droneOnPivo	Drone usando movimento do Pivô	{true, false}

4.4 Síntese do Capítulo

Neste capítulo foi apresentada a arquitetura do Castalia, a hierarquia de módulos e as modificações no Castalia. Na seção de desenvolvimento foram apresentados os módulos criados para simulação e as modificações feitas em funções do Castalia. Na seção de especificações foi apresentado as configurações do ambiente de simulação e os parâmetros usados.

Capítulo 5

Resultados

Neste capítulo serão apresentados todos os resultados obtidos dos cenários simulados no Castalia. A simulação foi dividida em dois grandes cenários, um em que drone faz todo percurso do campo por conta própria coletando os dados dos sensores e no outro cenário durante um quarto do campo o drone fica sob o pivô central realizando a coleta de dados, assim usando o movimento do pivô para se locomover no campo, com isso o drone perde velocidade mas economiza energia. Dentro de cada cenário possuem sub-cenários devido a variação de dois parâmetros a taxa de pacotes e a potência de rádio. Na taxa de pacotes são aplicados três valores: 5 p/s, 20 p/s e 80 p/s. Na potência de rádio também possuem três valores para simulação: 5dBm, -10dBm e -15dBm. Nos dois cenários são executadas todas as combinações entre os valores de taxas de pacotes e potência de rádio.

5.1 Cenário: sem uso da movimentação do Pivô

Para o gráfico de pacotes enviados por segundo na Figura 5.1 cada barra representa uma taxa de pacotes por segundo(5 p/s, 20 p/s, 80p/s), as legendas no eixo x indicam os valores de potência de rádio e os valores no eixo y indicam a quantidade de pacotes enviados por nó. O intervalo de confiança de 95% para cada potência de rádio é ilustrada na Tabela 5.1. A potência de 5dBm é a que possui maior taxa de pacotes enviados por nó e também é a que possui a com maior variação.

Tabela 5.1: Tabela de intervalo de confiança de 95% para pacotes enviados por nó(Cenário sem uso da movimentação do pivô).

Pontência de Rádio	**Limite inferior**	**Limite superior**
-15dBM	17.314	53.2357
-10dBm	22.7062	67.6438
5dBm	7.268	128.023

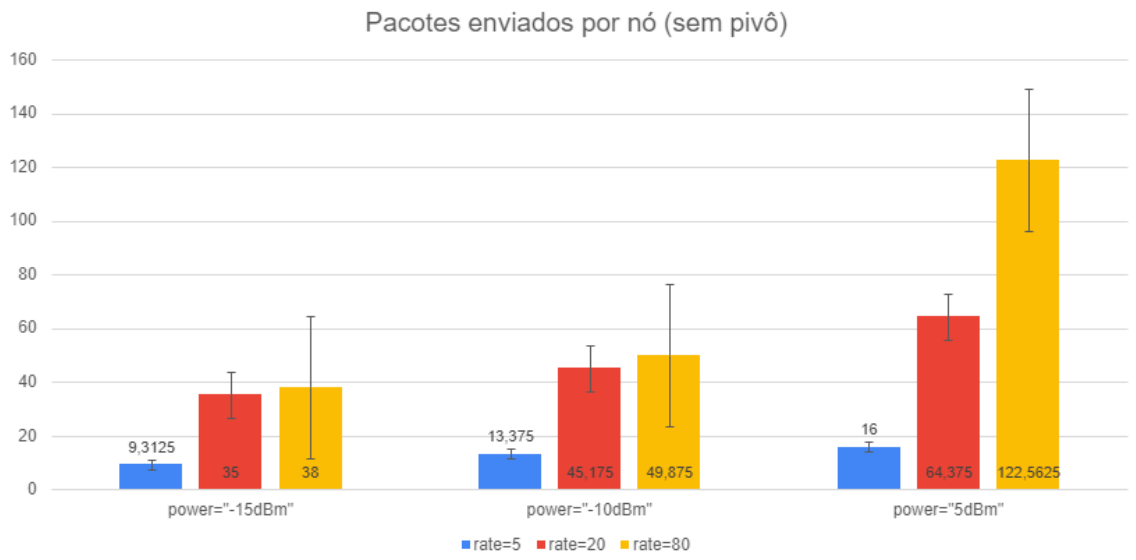


Figura 5.1: Gráfico de pacotes enviados por nó(Cenário sem uso da movimentação do pivô).

Tabela 5.2: Tabela de intervalo de confiança de 95% para taxa de perda de pacote(Cenário sem uso da movimentação do pivô).

Pontência de Rádio	Limite inferior	Limite superior
-15dBm	0.366	0.874
-10bBm	0.157	0.836
5dBm	-0.003	0.585

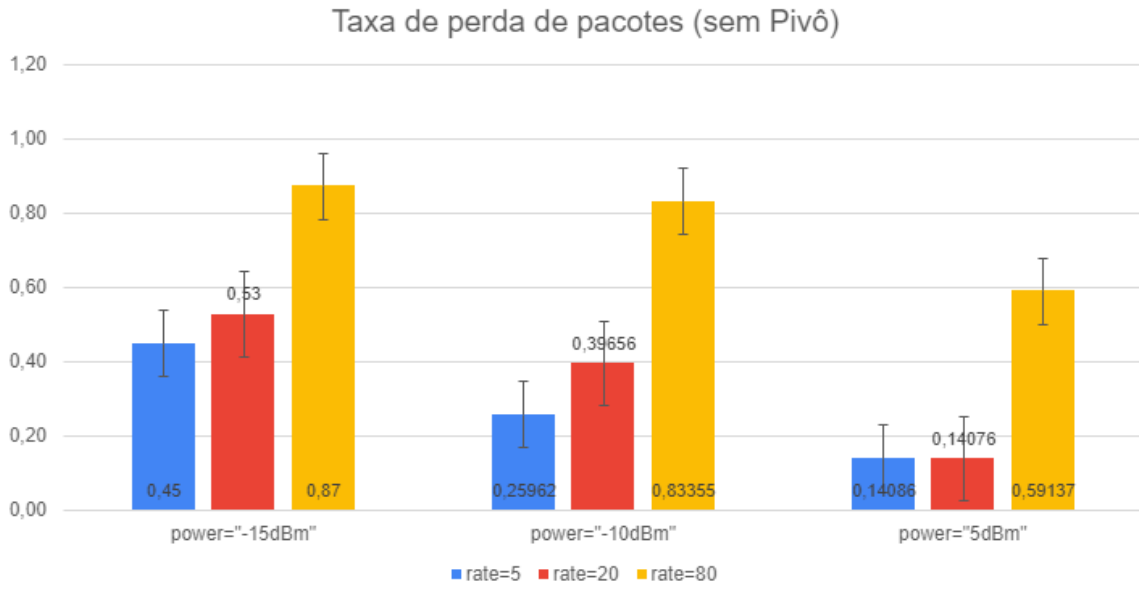


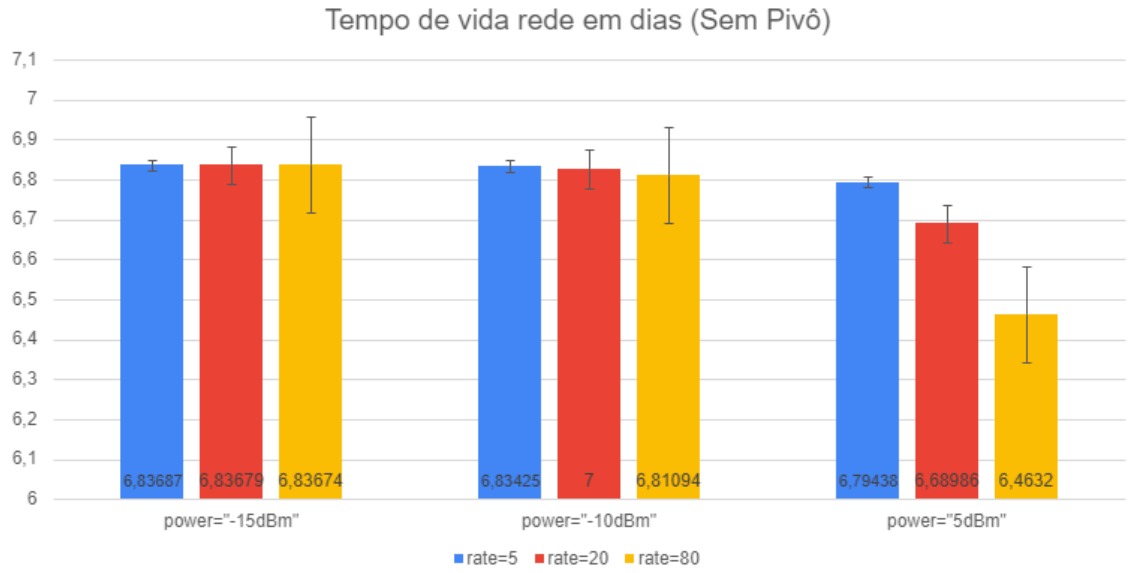
Figura 5.2: Gráfico de taxa de perda de pacote.(Cenário sem uso da movimentação do pivô).

No gráfico de taxa de perda de pacotes ilustrado na Figura 5.2 os números no eixo y representam as taxas de perda de pacote e as legendas no eixo x representam a potência de rádio em dBm. O "rate" é a taxa de pacotes por segundo que são as barras, 5 pacotes por segundo é a barra azul, 20 pacotes por segundo é a barra vermelha e 80 pacotes por segundo é a barra amarela. O intervalo de confiança de 95 % referente a taxa de perda de pacotes para cada potência de rádio é ilustrada na Tabela 5.2. Foi possível observar que quanto menor número de pacotes enviados por segundos menor a taxa de perda, logo a taxa de 5 pacotes por segundo obteve taxas de perda de pacotes menores, e a potência de rádio com menor taxa de perda de pacotes foi de 5dBm.

Tabela 5.3: Tabela de intervalo de confiança de 95% para o tempo de vida da rede(Cenário sem uso da movimentação do pivô).

Pontência de Rádio	**Limite inferior**	**Limite superior**
-15dBm	6,837	6,839
-10bBm	6.81	6,838
5dBm	6.46	6.84

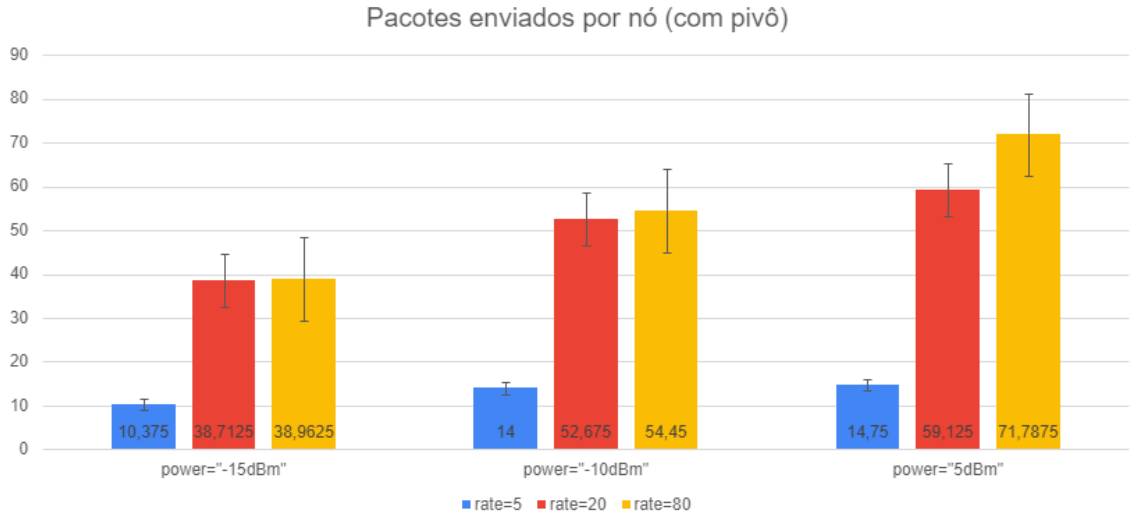
Figura 5.3: Gráfico de tempo de vida da rede(Cenário sem uso da movimentação do pivô).



Na Figura 5.3 é ilustrado o gráfico de tempo de vida da rede, no eixo y estão os tempos de vida da rede em dias e no eixo x estão as potências de rádio em dBm. O "rate" é a taxa de pacotes por segundo que são as barras, 5 pacotes por segundo é a barra azul, 20 pacotes por segundo é a barra vermelha e 80 pacotes por segundo é a barra amarela. O intervalo de confiança de 95 % do tempo de vida da rede em dias para cada potência de rádio está ilustrado na Tabela 5.3. Foi possível concluir que a potência de -10dBm obteve maior tempo de vida e a potência de 5dBm obteve menor tempo de vida da rede. A variação do tempo de vida da rede foi muito baixo tanto para diferentes potências de rádio quanto para diferentes taxas de pacotes por segundo.

5.2 Cenário: com uso da movimentação do Pivô

Figura 5.4: Gráfico de pacotes enviados por nó(Cenário com uso da movimentação do pivô).



No gráfico de pacotes enviados por segundo na Figura 5.4 cada barra representa uma taxa de pacotes por segundo(5 p/s, 20 p/s, 80 p/s), as legendas no eixo x indicam os valores de potência de rádio e as legendas no eixo y indicam a quantidade de pacotes enviados por nó. O intervalo de confiança de 95% de pacotes enviados por nó para cada potência de rádio é ilustrada na Tabela 5.4. A potência de 5dBm é a que possui maior taxa de pacotes enviados por nós e também a com maior variação, assim como no cenário sem o uso da movimentação do pivô.

Tabela 5.4: Tabela de intervalo de confiança de 95% para pacotes enviados por nó(Cenário com uso da movimentação do pivô).

Pontência de Rádio	**Limite inferior**	**Limite superior**
-15dBM	10.754	47.946
-10dBm	14.471	66.254
5dBm	14.661	82.447

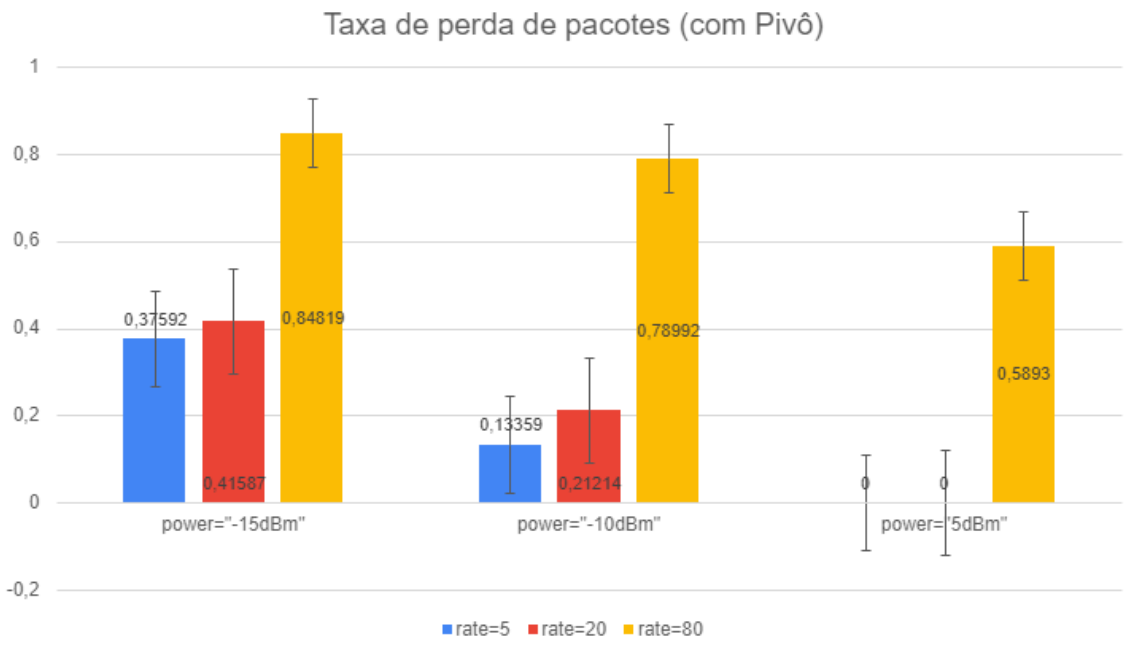
Tabela 5.5: Tabela de intervalo de confiança de 95% para taxa de perda de pacote(Cenário com uso da movimentação do pivô).

Pontência de Rádio	**Limite inferior**	**Limite superior**
-15dBm	0.250	0.843
-10bBm	-0.027	0.784
5dBm	-0.189	0.581

Tabela 5.6: Tabela de intervalo de confiança de 95% para o tempo de vida da rede(Cenário com uso da movimentação do pivô).

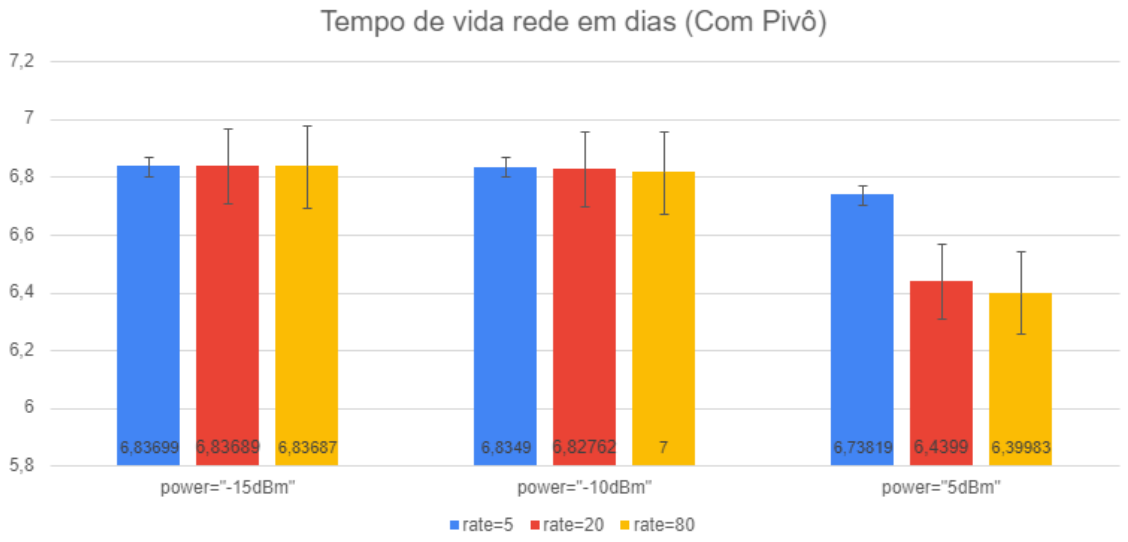
Pontência de Rádio	**Limite inferior**	**Limite superior**
-15dBm	6,837	6,839
-10bBm	6.81	6,838
5dBm	6.46	6.84

Figura 5.5: Gráfico de taxa de perda de pacote(Cenário com uso da movimentação do pivô).



No gráfico de taxa de perda de pacotes ilustrado na Figura 5.5 os números no eixo y representam as taxas de perda de pacote e as legendas no eixo x representam as potências de rádio em dBm. O "rate" é a taxa de pacotes por segundo que são as barras, 5 pacotes por segundo é a barra azul, 20 pacotes por segundo é a barra vermelha e 80 pacotes por segundo é a barra amarela. O intervalo de confiança de 95 % referente a taxa de perda de pacotes para cada potência de rádio é ilustrada na Tabela 5.5. Foi possível observar que quanto menores taxas de pacotes por segundos enviados menor é a taxa de perda, e a potência de rádio com menor taxa de perda de pacotes foi a de 5dBm assim como no cenário sem uso do pivô mas com taxa de perdas ainda menores neste cenário em que o drone usa a movimentação do pivô.

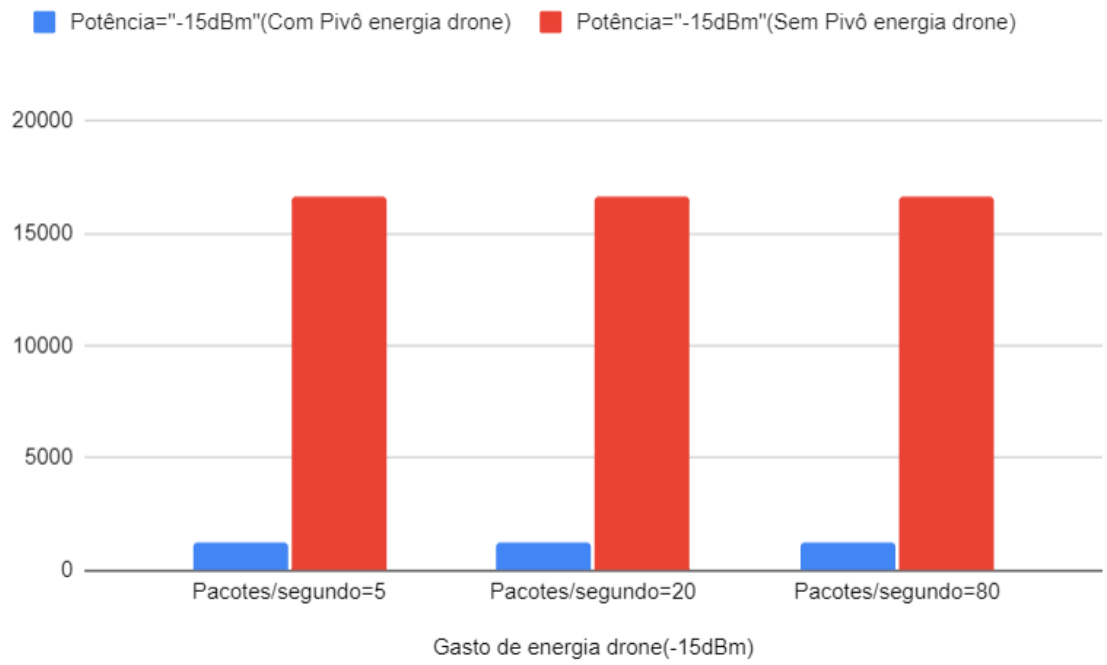
Figura 5.6: Gráfico de tempo de vida da rede(Cenário com uso da movimentação do pivô).



Na Figura 5.6 é ilustrado o gráfico de tempo de vida da rede, no eixo y estão os tempos de vida da rede em dias e no eixo x estão as potências de rádio em dBm. O "rate" é a taxa de pacotes por segundo que são as barras, 5 pacotes por segundo é a barra azul, 20 pacotes por segundo é a barra vermelha e 80 pacotes por segundo é a barra amarela. O intervalo de confiança de 95 % do tempo de vida da rede em dias para cada potência de rádio é ilustrado na Tabela 5.6. Foi possível concluir que a potência -10dBm obteve maior tempo de vida e a potência de 5dBm obteve os menores tempos de vida da rede. Comparando o tempo de vida da rede no cenário em o drone(coletor de dados) usa o movimento do pivô e com o cenário que o coletor de dados não usa o movimento do pivô, conclui se que não houve um ganho em tempo de vida rede pois a redução de gasto de energia é somente no coletor da rede e nos nós de sensores acaba ocorrendo um gasto maior de energia com transmissão no cenário usando o movimento do pivô, devido o drone se mover mais lentamente em cima do pivô e o drone fica mais tempo na zona de transferência de dados.

5.3 Gasto de energia do coletor de dados

Figura 5.7: Gráfico de gasto de energia do drone(Potência -15dBm).



Nesta seção é mostrado o gasto de energia do drone o coletor de dados da rede. Joules é a unidade medida de energia extraída do simulador e que é usada nos gráficos e a quantidade inicial de energia do drone de 18738 Joules. Os gráficos nas Figura 5.7, Figura 5.8 e Figura 5.9 ilustra o gasto de energia do drone comparando o gasto de energia do drone nos dois cenários em que o drone usa o movimento do pivô para economia de energia e no cenário que não é usado. No eixo x possui as legendas para taxa de pacotes por segundo e no eixo y os valores do gasto de energia em Joule. O intervalo de confiança de 95 % refente o gasto de energia do drone no cenário sem uso da movimentação do pivô para as potências de rádio estão ilustrados na Tabela 5.7. O intervalo de confiança de 95 % refente o gasto de energia do drone no cenário com uso da movimentação do pivô para as potências de rádio estão ilustrados na Tabela 5.8.

Figura 5.8: Gráfico de gasto de energia do drone(Potência -10dBm).

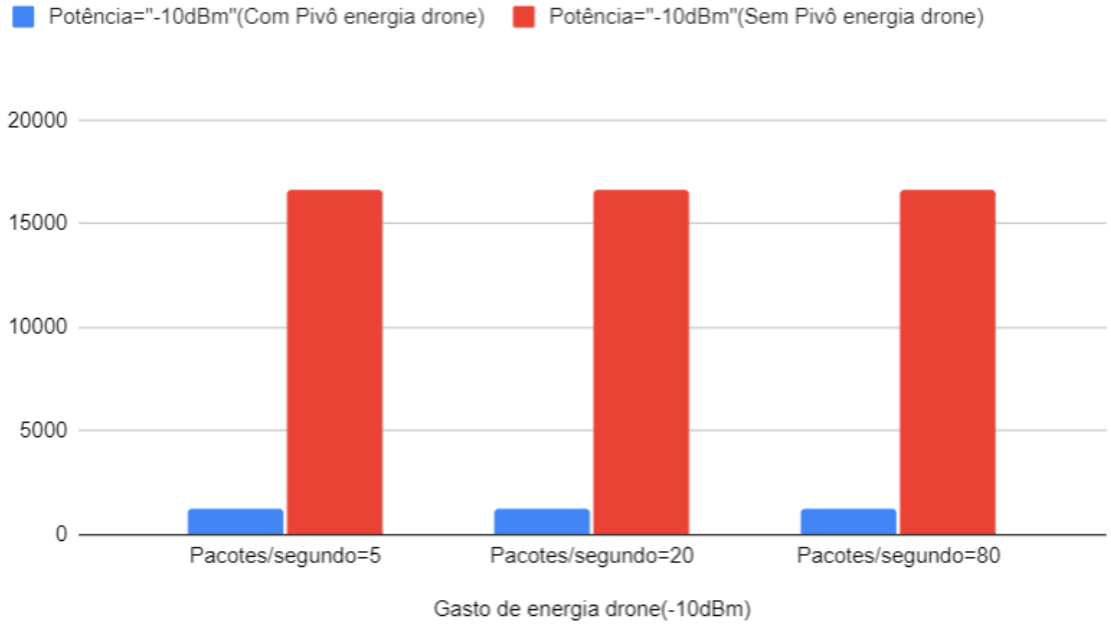


Figura 5.9: Gráfico de gasto de energia do drone(Potência -5dBm).

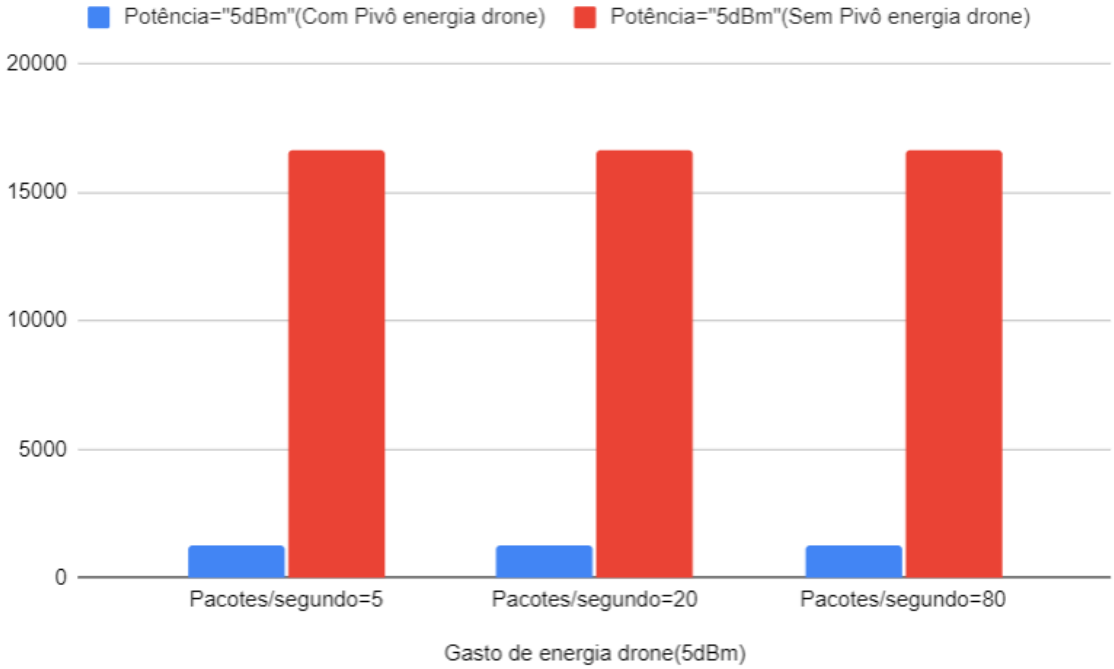


Tabela 5.7: Tabela de intervalo de confiança de 95% para o gasto de energia do drone(Cenário sem uso da movimentação do pivô).

Pontência de Rádio	**Limite inferior**	**Limite superior**
-15dBm	16668.86 J	16668.86 J
-10bBm	16668.86 J	16668.86 J
5dBm	16668.64 J	16668.87 J

Tabela 5.8: Tabela de intervalo de confiança de 95% para o gasto de energia do drone(Cenário com uso da movimentação do pivô).

Pontência de Rádio	**Limite inferior**	**Limite superior**
-15dBm	1260.88 J	1260.88 J
-10bBm	1260.88 J	1260.88 J
5dBm	1260.89 J	1260.97 J

O gasto de energia do drone quase não tem alteração seja na variação de taxa de pacotes seja na variação de potência de rádio dentro de um mesmo cenário. Calculando a média do gasto de energia para o cenário em que drone não usa movimento do pivô e faz todo percurso por conta própria a média resultou em 16668.83 já no cenário em que drone usa movimento do pivô por um ¼ do campo a média resultou em 1260.90. Portanto, o cenário em que o drone usou o movimento do pivô resultou em uma economia de 92% para nó coletor de dados(drone) em relação ao cenário que drone faz todo o percurso por conta própria.

Capítulo 6

Conclusão

Neste trabalho o principal objetivo foi caracterizar uma rede de sensores sem fio, com foco em analisar sua transferência de dados e consumo energético. Dois fatores que são grandes desafios dentro na implementação de redes de sensores devido a limitação de hardware usados dentro desse tipo de rede. Para caracterizar a rede foi usado o simulador Castalia que é amplamente usado para simulação de redes de sensores sem fio e *Body Area Network(BAN)* no meio acadêmico. Como estudo de caso para simulação foi usado um sistema de irrigação inteligente que foi implementado dentro do simulador.

As métricas usadas para avaliação dentro foram a quantidade de pacotes enviados, taxa de perda de pacotes, tempo de vida da rede e carga de bateria. Foram analisados dois grandes cenários dentro das simulações, um cenário em o coletor de dados(drone) realizava todo o percurso no campo por conta própria sempre gastando energia com sua movimentação e outro cenário em que durante ¼ do campo o drone usaria o movimento do pivô central poupando energia mas perdendo velocidade neste trecho.

Foi possível analisar que na transmissão de dados dentro das potências de rádio analisadas, com aumento da potência ocorreu uma redução na taxa de perda de pacote, a potência com menor perda de pacote foi a de 5dBm. Também ocorre aumento da taxa de perda de pacote com aumento da taxa de pacotes por segundos. Em relação a comparação do cenário com uso do movimento do pivô central pelo drone e o cenário sem uso, houve um comportamento semelhante quando se trata de transferência de dados, mas no cenário com uso do movimento do pivô obteve taxas de perda de pacotes menores.

Quanto ao consumo energético dentro das potências analisadas -15dBm, -10dBm, 5dBm, no geral a de -10dBm obteve melhores tempo de vida da rede(lifetime network) com rede tendo mais tempo de funcionamento sem reabastecimento energético. Taxa de pacotes que forneceu um maior de tempo de vida da rede dentro das taxas de 5 pacotes/segundo, 20 pacotes/segundo, 80 pacotes/segundo foi de 5 p/s. Comparando os dois cenários em que o coletor de dados(drone) usa o movimento do pivô central e o cenário

sem o uso do movimento do pivô, em relação ao tempo de vida da rede não se obteve ganho devido o maior gasto de energia do sensores para transmissão de dados para o coletor de dados, com drone se movimentando mais lento usando pivô ficava mais tempo na zona de transmissão de dados exigindo mais dos sensores. Em relação a energia do nó coletor de dados(drone) no cenário com uso da movimentação do pivô central, obteve um ganho de aproximadamente 92% de energia comparando com cenário sem uso do pivô central.

6.1 Trabalhos Futuros

Para trabalhos futuros, é possível realizar simulações com uso de técnicas de coleta de energia(Energy Harvesting) [22, 23, 24] tanto para nó coletor de dados quanto para os demais nós da rede. Com a coleta de energia aumentaria o tempo de vida da rede no geral e também a possibilidade de criar cenários mais complexos. O GreenCastalia é um framework para implementação de coleta de energia dentro simulador Castalia [25], seria um grande auxílio na implementação dos cenários com coleta de energia.

Para futuras implementações seria útil a criação de limitador de envio de dados do sensores para coletor de dados em casos que o coletor fique muito tempo dentro da zona de transferência de dados. Esse tipo de implementação teria possivelmente um impacto positivo tanto para o tempo de vida da rede quanto na taxa de perda de pacotes. A implementação de um módulo próprio de processos físicos dentro do Castalia para sensores subterrâneos também seria um bom incremento a simulação.

Referências

[1] Wohwe Sambo, Damien, Anna Forster, Blaise Omer Yenke, Idrissa Sarr, Bamba Gueye e Paul Dayang: *Wireless underground sensor networks path loss model for precision agriculture (wusn-plm).* IEEE Sensors Journal, 20(10):5298–5313, 2020. v, vi

[2] Pattar, Santosh e Buyya, Rajkumar e Venugopal K. R. e Iyengar S. S. e Patnaik L. M.: *Searching for the iot resources: Fundamentals, requirements, comprehensive review, and future directions.* IEEE Communications Surveys Tutorials, 20(3):2101–2132, 2018. ix, 1, 3, 4

[3] Virk, Ahmad Latif e Noor, Mehmood Ali e Fiaz Sajid e Hussain Saddam e Hussain Hafiz Athar e Rehman Muzammal e Ahsan Muhammad e Ma Wei: *Smart Farming: An Overview.* Springer International Publishing, Cham, 2020, ISBN 978-3-030-37794-6. `https://doi.org/10.1007/978-3-030-37794-6_10`. ix, 1, 9, 10

[4] Kamienski, Carlos e Soininen, Juha Pekka e Taumberger Markus e Fernandes Stenio e Toscano Attilio e Cinotti Tullio Salmon e Maia Rodrigo Filev e Neto Andre Torre: *Swamp: an iot-based smart water management platform for precision irrigation in agriculture.* Em *2018 Global Internet of Things Summit (GIoTS)*, páginas 1–6, 2018. ix, 11, 12, 13, 14, 23

[5] Embrapa: *Área irrigada por pivôs centrais no Brasil atinge 1,6 milhão de hectares.* `https://www.embrapa.br/en/busca-de-noticias/-/noticia/59843654/area-irrigada-por-pivos-centrais-no-brasil-atinge-16-milhao-de-hectares`, 2021. [Online; último acesso 16/04/2022]. ix, 17

[6] Boulis, Thanassis: *Castalia - User Manual.* `https://github.com/boulis/Castalia`, 2013. [Online; último acesso 13/04/2022]. ix, 18, 31

[7] Akyildiz, I.F. e Vuran, M.C.: *Wireless Sensor Networks.* Advanced Texts in Communications and Networking. Wiley, 2010, ISBN 9780470515198. `https://books.google.com.br/books?id=7YBHYJsSmS8C`. 1, 5, 6, 7, 8

[8] E. Cayirci, I.F. Akyildiz e W. Su e Y. Sankarasubramaniam e: *Wireless sensor networks: a survey.* Computer Networks, 38(4):393–422, 2002, ISSN 1389-1286. `https://www.sciencedirect.com/science/article/pii/S1389128601003024`. 1, 5

[9] Al-Sarawi, Shadi e Anbar, Mohammed e Alieyan Kamal e Alzubaidi Mahmood: *Internet of things (iot) communication protocols.* Em *2017 8th International conference on information technology (ICIT)*, páginas 685–690. IEEE, 2017. 4

[10] Roy, Swati Sucharita, Deepak Puthal, Suraj Sharma, Saraju P Mohanty e Albert Y Zomaya: *Building a sustainable internet of things: Energy-efficient routing using low-power sensors will meet the need.* IEEE Consumer Electronics Magazine, 7(2):42–49, 2018. 7, 22

[11] Bhardwaj, Manish e Garnett, Timothy e Chandrakasan Anantha P: *Upper bounds on the lifetime of sensor networks.* Em *ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No. 01CH37240)*, volume 3, páginas 785–790. IEEE, 2001. 8

[12] Akyildiz, Ian F e Stuntebeck, Erich P: *Wireless underground sensor networks: Research challenges.* Ad Hoc Networks, 4(6):669–686, 2006. 14, 15

[13] Wang, Dongzhi e Hui Ming: *Simulation test of wireless underground sensor network in stadiums.* Journal of Sensors, 2021, 2021. 15

[14] Suat Irmak, Xin Dong e Mehmet C. Vuran e: *Autonomous precision agriculture through integration of wireless underground sensor networks with center pivot irrigation systems.* Ad Hoc Networks, 11(7):1975–1987, 2013, ISSN 1570-8705. `https://www.sciencedirect.com/science/article/pii/S1570870512001291`, Theory, Algorithms and Applications of Wireless Networked Robotics Recent Advances in Vehicular Communications and Networking. 15, 23, 25

[15] Evans, Robert G: *Center pivot irrigation.* Agricultural Systems Research Unit, North Plain Agricultural Research laboratory. USDN-Agricultural Research Service, 1500, 2001. 16

[16] Matilla, Diego Mateos, Álvaro Lozano Murciego, Diego Manuel Jiménez Bravo, André Sales Mendes e Valderi Reis Quietinho Leithardt: *Low cost center pivot irrigation monitoring systems based on iot and lorawan technologies.* Em *2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, páginas 262–267. IEEE, 2020. 16

[17] Pediaditakis, Dimosthenis e Tselishchev, Yuri e Boulis Athanassios: *Performance and scalability evaluation of the castalia wireless sensor network simulator.* SIMUTools '10, Brussels, BEL, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ISBN 9789639799875. `https://doi.org/10.4108/ICST.SIMUTOOLS2010.8727`. 18

[18] Ngo, Khoa Anh, Trong Thua Huynh e De Thu Huynh: *Simulation wireless sensor networks in castalia.* Em *Proceedings of the 2018 International Conference on Intelligent Information Technology*, páginas 39–44, 2018. 18

[19] Omnetpp: *What is OMNeT++?* `https://omnetpp.org/intro/`, 2021. [Online; último acesso 19/04/2022]. 22

[20] Ahmed, Shaimaa e Mohamed, Amr e Harras Khaled e Kholief Mohamed e Mesbah Saleh: *Energy efficient path planning techniques for uav-based systems with space discretization*. Em *2016 IEEE Wireless Communications and Networking Conference*, páginas 1–6. IEEE, 2016. 26

[21] Hernandez, Fernando Braz Tangerino: *Pivô central: história e características*. `https://irrigacao.blogspot.com/2010/02/pivo-central-historia-e-caracteristicas.html`, 2008. [Online; último acesso 12/04/2022]. 29

[22] Jayakumar, Hrishikesh, Kangwoo Lee, Woo Suk Lee, Arnab Raha, Younghyun Kim e Vijay Raghunathan: *Powering the internet of things*. Em *Proceedings of the 2014 international symposium on Low power electronics and design*, páginas 375–380, 2014. 56

[23] Shaikh, Faisal Karim e Sherali Zeadally: *Energy harvesting in wireless sensor networks: A comprehensive review*. Renewable and Sustainable Energy Reviews, 55:1041–1054, 2016. 56

[24] Kausar, ASM Zahid, Ahmed Wasif Reza, Mashad Uddin Saleh e Harikrishnan Ramiah: *Energizing wireless sensor networks by energy harvesting systems: Scopes, challenges and approaches*. Renewable and Sustainable Energy Reviews, 38:973–989, 2014. 56

[25] Benedetti, David, Chiara Petrioli e Dora Spenza: *Greencastalia: An energy-harvesting-enabled framework for the castalia simulator*. Em *Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems*, ENSSys '13, New York, NY, USA, 2013. Association for Computing Machinery, ISBN 9781450324328. `https://doi.org/10.1145/2534208.2534215`. 56

Apêndice A

Implementação módulo de mobilidade para o Castalia

Algoritmo A.1: CircleMobilityManager.cc

```
    #include "CircleMobilityManager.h"

Define_Module(CircleMobilityManager);


void CircleMobilityManager::initialize()
{
        VirtualMobilityManager::initialize();

        updateInterval = par("updateInterval");
        updateInterval = updateInterval / 1000;
    x_field = network->par("field_x");
        y_field = network->par("field_y");
        z_field = network->par("field_z");
        drawEnergy = par("drawPowerMove");
        droneOnPivo= par("droneOnPivo");
        step = par("speedDrone");
        pivo_speed = par("speedPivo");

        loc1_x = nodeLocation.x;
        loc1_y = nodeLocation.y;
        loc1_z = nodeLocation.z;

        speed = par("speed");

        //direction = 1;
```

```
        if(loc1_x == 0 || loc1_x == x_field){

                setLocation(loc1_x, loc1_y, loc1_z);
                scheduleAt(simTime() + updateInterval,
                        new MobilityManagerMessage("Periodic location
                            update message", MOBILITY_PERIODIC));

        }


}


void CircleMobilityManager::handleMessage(cMessage * msg){
    int msgKind = msg->getKind();
        switch (msgKind) {

                case MOBILITY_PERIODIC:{
                        if (droneOnPivo && (nodeLocation.x <= x_field/2 &&
                            nodeLocation.y >= y_field/2) ) {
                                if(nodeLocation.x < x_field &&
                                    nodeLocation.y == y_field){
                                        if(nodeLocation.x + pivo_speed <
                                            x_field){
                                                nodeLocation.x+=pivo_speed;
                                        }
                                        else{
                                                nodeLocation.x = x_field;
                                        }
                                }
                                else if(nodeLocation.x > 0  && nodeLocation
                                    .y == 0){
                                        if(nodeLocation.x - pivo_speed > 0)
                                            {
                                                nodeLocation.x-= pivo_speed
                                                    ;
                                        }
                                        else{
                                                nodeLocation.x = 0;
                                        }

                                }
                                else if(nodeLocation.x == x_field &&
                                    nodeLocation.y > 0){
```

```
                        if(nodeLocation.y - pivo_speed > 0)
                            {
                                nodeLocation.y -=
                                    pivo_speed;
                        }else {
                                nodeLocation.y =0;
                        }

                }
                else if(nodeLocation.x == 0 && nodeLocation
                    .y < y_field){
                        if(nodeLocation.y + pivo_speed <
                            y_field){
                                nodeLocation.y +=
                                    pivo_speed;
                        }
                        else{
                                nodeLocation.y = y_field;
                        }
                }

            }
            else {
                if(nodeLocation.x < x_field &&
                    nodeLocation.y == y_field){

                        powerDrawn(drawEnergy * step);

                        if(nodeLocation.x + step < x_field)
                            {
                                nodeLocation.x+=step;
                        }
                        else{
                                nodeLocation.x = x_field;
                        }
                }
                else if(nodeLocation.x > 0  && nodeLocation
                    .y == 0){
                        if(nodeLocation.x - step > 0){
                                nodeLocation.x-= step;
                        }
                        else{
                                nodeLocation.x = 0;
                        }
```

```
                                }
                                else if(nodeLocation.x == x_field &&
                                    nodeLocation.y > 0){

                                        if(nodeLocation.y - step > 0){
                                                nodeLocation.y -= step;
                                        }else {
                                                nodeLocation.y =0;
                                        }

                                }
                                else if(nodeLocation.x == 0 && nodeLocation
                                    .y < y_field){ // aumentando o y
                                        if(nodeLocation.y + step < y_field)
                                            {
                                                nodeLocation.y += step;
                                        }
                                        else{
                                                nodeLocation.y = y_field;
                                        }
                                }


                        }

                        notifyWirelessChannel();
                        scheduleAt(simTime() + updateInterval,
                        new MobilityManagerMessage("Periodic location
                            update message", MOBILITY_PERIODIC));

                        trace() << "changed location(x:y:z) to " <<
                            nodeLocation.x <<
                                ":" << nodeLocation.y << ":" <<
                                    nodeLocation.z;
                        break;
                }
                default:{
                        trace() << "WARNING: Unexpected message " <<
                            msgKind;
                }
        }

        delete msg;
        msg = NULL;
}
```

Algoritmo A.2: CircleMobilityManager.ned

```
    package node.mobilityManager.circleMobilityManager;

simple CircleMobilityManager like node.mobilityManager.iMobilityManager {
    parameters:
        bool collectTraceInfo = default (false);
        double updateInterval = default (1000);
        double xCoorDestination = default (0);
        double yCoorDestination = default (0);
        double zCoorDestination = default (0);
        double speed = default (1);
        int speedDrone = default(1);
        double drawPowerMove = default(15000); // mW
        bool droneOnPivo = default(false);
        double speedPivo = default(0.07);
   }
```

Algoritmo A.3: CircleMobilityManager.h

```
    #ifndef _MOBILITYMODULE_H_
#define _MOBILITYMODULE_H_

#include "MobilityManagerMessage_m.h"
#include "VirtualMobilityManager.h"

using namespace std;

class CircleMobilityManager: public VirtualMobilityManager {
 private:
        /*--- The .ned file's parameters ---*/
        double updateInterval;
        double loc1_x;
        double loc1_y;
        double loc1_z;
        double loc2_x;
        double loc2_y;
        double loc2_z;
        double speed;
        int    step;

        /*--- Custom class parameters ---*/
    int x_field;
    int y_field;
    int z_field;
        int totalSNnodes;
```

```
        double drawEnergy;
        bool droneOnPivo;
        double pivo_speed;
 protected:
        void initialize();
        void handleMessage(cMessage * msg);
};

#endif
```

Apêndice B

Implementação módulo de aplicação para o Castalia

Algoritmo B.1: NewApp.cc

```
#include "NewApp.h"

Define_Module(NewApp);

void NewApp::startup(){
        trace() << "␣inicio";
        packet_rate = par("packet_rate");
        recipientAddress = par("nextRecipient").stringValue();
    recipientId = atoi(recipientAddress.c_str());
        startupDelay = par("startupDelay");
    delayLimit = par("delayLimit");
        packet_spacing =  packet_rate > 0 ? 1 / float (packet_rate) : -1;
        dataSN = 0;
    numNodes = getParentModule()->getParentModule()->par("numNodes");
        raioPacket = par("raioPacket");
        mobility = check_and_cast<VirtualMobilityManager*>(getParentModule
            ()->getSubmodule("MobilityManager"));

    packetsSent.clear();
    packetsReceived.clear();
        bytesReceived.clear();

    if (packet_spacing > 0 && recipientAddress.compare(SELF_NETWORK_ADDRESS
        ) != 0)
            setTimer(SEND_PACKET, packet_spacing + startupDelay);
    else
        trace() << "Not␣sending␣packets";
```

```
    declareOutput("Pacotes␣enviados␣por␣no");
}


void NewApp::fromNetworkLayer(ApplicationPacket * rcvPacket, const char *
    source, double rssi, double lqi){
        int sequenceNumber = rcvPacket->getSequenceNumber();
        int sourceId = atoi(source);

        if (recipientAddress.compare(SELF_NETWORK_ADDRESS) == 0) {
                if (delayLimit == 0 || (simTime() - rcvPacket->
                    getCreationTime()) <= delayLimit) {
                        trace() << "Pacote␣recebido␣#" << sequenceNumber <<
                            "␣do␣no␣" << source;
                        collectOutput("Pacotes␣enviados␣por␣no", sourceId);
                        packetsReceived[sourceId]++;
                        bytesReceived[sourceId] += rcvPacket->getByteLength
                            ();
                } else {
                        trace() << "Pacote␣#" << sequenceNumber << "␣do␣no␣
                            " << source <<
                                "␣excedeu␣o␣tempo␣limite␣de␣delay␣" <<
                                    delayLimit << "s";
        }
        }else {
                ApplicationPacket* fwdPacket = rcvPacket->dup();

                fwdPacket->setByteLength(0);
                toNetworkLayer(fwdPacket, recipientAddress.c_str());
        }


}


void NewApp::handleRadioControlMessage(RadioControlMessage *radioMsg)
{
        switch (radioMsg->getRadioControlMessageKind()) {
                case CARRIER_SENSE_INTERRUPT:
                        trace() << "CS␣Interrupt␣received!␣current␣RSSI␣
                            value␣is:␣" << radioModule->readRSSI();
                        break;
        }
}
```

```
void NewApp::timerFiredCallback(int index){
        switch (index) {
                case SEND_PACKET:{
                        cTopology *topo;
                        topo = new cTopology("topo");
                        topo->extractByNedTypeName(cStringTokenizer("node.
                            Node").asVector());
                        VirtualMobilityManager *nextRecipientMobilityModule
                              = dynamic_cast<VirtualMobilityManager*>
                        (topo->getNode(recipientId)->getModule()->
                            getSubmodule("MobilityManager"));

                        double distance = sqrt(
                                pow(mobility->getLocation().x -
                                    nextRecipientMobilityModule->getLocation
                                    ().x, 2)
                                + pow(mobility->getLocation().y -
                                    nextRecipientMobilityModule->getLocation
                                    ().y, 2));

                        if(abs(distance) <= raioPacket){
                                trace() << "Mandando pacote #" << dataSN;
                                toNetworkLayer(createGenericDataPacket(0,
                                    dataSN),par("nextRecipient"));
                                packetsSent[recipientId]++;
                                dataSN++;
                        }
                        setTimer(SEND_PACKET, packet_spacing);
                        break;
                }
        }

}




void NewApp::finishSpecific(){
        declareOutput("Taxa de pacotes recebidos");
        declareOutput("Taxa de perda de pacote");

    cTopology *topo;
        topo = new cTopology("topo");
```

```
        topo->extractByNedTypeName(cStringTokenizer("node.Node").asVector()
            );

    long bytesDelivered = 0;
    for(int i=0; i < numNodes; i++){
        NewApp *appModule  = dynamic_cast<NewApp*>
                        (topo->getNode(i)->getModule()->getSubmodule("
                            Application"));
        if(appModule){
            int packetsSent = appModule->getPacketsSent(self);
            if (packetsSent > 0) {
                                float rate = (float)packetsReceived[i]/
                                    packetsSent;
                                collectOutput("Taxa de pacotes recebidos",
                                    i, "total", rate);
                                collectOutput("Taxa de perda de pacote", i,
                                     "total", 1-rate);
                        }

            bytesDelivered += appModule->getBytesReceived(self);
        }
    }
    delete(topo);

    if (packet_rate > 0 && bytesDelivered > 0) {
                double energy = (enMgrModule->getTotEnergySupplied() *
                    1000000000)/(bytesDelivered * 8);        //in nanojoules
                    /bit
                declareOutput("Energy nJ/bit");
                collectOutput("Energy nJ/bit","",energy);
        }
}
```

Algoritmo B.2: NewApp.ned

```
    package node.application.newApp;

simple NewApp like node.application.iApplication {
    parameters:
        string applicationID = default("newApp");
        bool collectTraceInfo = default (true);
        int priority = default (1);
        int packetHeaderOverhead = default (5);          // in bytes
        int constantDataPayload = default (200);         // in bytes
        double delayLimit = default (0);                          //
            application delay limit (0 - no limit)
```

```
        // definir meus parametros
        string nextRecipient = default ("0");    // Destination for packets
            received in this node.

        double packet_rate = default (0);        // packets per second, by
            default we transmit no packets
            double startupDelay = default (0);   // delay in seconds before
                the app stars producing packets
        double raioPacket = default(35);


    gates:
         output toCommunicationModule;
        output toSensorDeviceManager;
        input fromCommunicationModule;
        input fromSensorDeviceManager;
        input fromResourceManager;

}
```

Algoritmo B.3: NewApp.h

```
    #ifndef _NewApp_H_
#define _NewApp_H_
#include "VirtualApplication.h"

#include <map>

enum NewAppTimers {
        SEND_PACKET = 1
};

using namespace std;
class NewApp: public VirtualApplication {
 private:
        double packet_rate;
        double startupDelay;
        double delayLimit;
        float packet_spacing;
        int dataSN;
        int recipientId;
        string recipientAddress;

        double raioPacket;
```

```
        //variables below are used to determine the packet delivery rates.

        int numNodes;
        map<long,int> packetsReceived;
        map<long,int> bytesReceived;
        map<long,int> packetsSent;
        VirtualMobilityManager *mobility;

protected:
        void startup();
        void fromNetworkLayer(ApplicationPacket *, const char *, double,
            double);
        void handleRadioControlMessage(RadioControlMessage *);
        void timerFiredCallback(int);
        void finishSpecific();

 public:
        int getPacketsSent(int addr) { return packetsSent[addr]; }
        int getPacketsReceived(int addr) { return packetsReceived[addr]; }
        int getBytesReceived(int addr) { return bytesReceived[addr]; }

};

#endif
```