

Daniel Hideaki Makita

**Desenvolvimento de um Protótipo Utilizando  
uma Rede de Sensores Sem Fio de Baixo Custo  
para Monitoramento Ambiental**

São José dos Campos  
Brasil

2019



Daniel Hideaki Makita

**Desenvolvimento de um Protótipo Utilizando uma Rede  
de Sensores Sem Fio de Baixo Custo para  
Monitoramento Ambiental**

Trabalho de graduação apresentado ao Instituto de Ciência e Tecnologia da Universidade Federal de São Paulo, como parte das atividades para obtenção do título de bacharel em Engenharia de Computação.

Universidade Federal de São Paulo - UNIFESP  
Instituto de Ciência e Tecnologia - ICT  
Programa de Graduação

Orientador: Prof. Dr. Tiago de Oliveira

São José dos Campos  
Brasil

2019

Na qualidade de titular dos direitos autorais, em consonância com a Lei de direitos autorais nº 9610/98, autorizo a publicação livre e gratuita desse trabalho no Repositório Institucional da UNIFESP ou em outro meio eletrônico da instituição, sem qualquer resarcimento dos direitos autorais para leitura, impressão e/ou download em meio eletrônico para fins de divulgação intelectual, desde que citada a fonte.

Ficha catalográfica gerada automaticamente com dados fornecidos pelo(a) autor(a)

Makita, Daniel Hideaki

Desenvolvimento de um Protótipo Utilizando uma Rede de Sensores Sem Fio de Baixo Custo para Monitoramento Ambiental/ Daniel Hideaki Makita

Orientação: Tiago de Oliveira-São José dos Campos, 2019.

139 p.

Development of a Prototype Using a Low Cost Wireless Sensors Network for Environmental Monitoring

Trabalho de Conclusão de Curso-Engenharia da Computação-Universidade Federal de São Paulo-Instituto de Ciência e Tecnologia, 2019.

1. Rede de Sensores Sem Fio. 2. Monitoramento Ambiental. 3. Protótipo. 4. Baixo Custo. 5. Arduino, Raspberry Pi, nRF24L01. I. Oliveira, Tiago de

Daniel Hideaki Makita

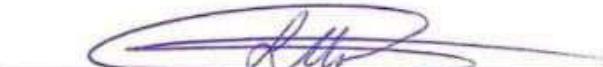
**Desenvolvimento de um Protótipo Utilizando uma Rede  
de Sensores Sem Fio de Baixo Custo para  
Monitoramento Ambiental**

Trabalho de graduação apresentado ao Instituto de Ciência e Tecnologia da Universidade Federal de São Paulo, como parte das atividades para obtenção do título de bacharel em Engenharia de Computação.

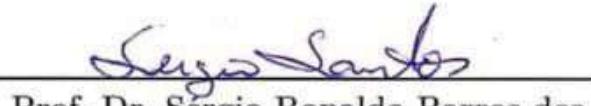


---

Prof. Dr. Tiago de Oliveira  
Orientador



Prof. Dr. Lauro Paulo da Silva Neto  
Convidado 1



Prof. Dr. Sérgio Ronaldo Barros dos  
Santos  
Convidado 2

São José dos Campos  
Brasil  
2019



*“The problems that exist in the world today cannot be solved by the level of thinking that created them”. Albert Einstein*



# Agradecimentos

Aos membros da banca, que dedicaram o seu tempo para ler e avaliar o meu trabalho e em especial ao meu orientador professor Dr. Tiago de Oliveira, pela dedicação do seu tempo à orientação, pela atenção, paciência e a pronta disposição em me ajudar quando foi necessário.

Aos meus colegas de turma que fizeram parte da minha trajetória durante a graduação, pelo apoio, companheirismo e pelas histórias e aprendizados de faculdade que serão lembradas para sempre. Em especial aos colegas Edward Hsiao e Jader Froza, que me emprestaram alguns componentes de *hardware* e ao Victor Luiz pelo suporte prestado durante algumas etapas do desenvolvimento deste trabalho.

Aos meus pais, Rumi e Yoiti, ao meu irmão Fabio, a minha avó Mitsuko e a minha namorada Taína pelo suporte, amor incondicional, por estarem sempre ao meu lado e por não terem me deixado desistir quando tive vontade.

Vocês são a minha base, fica registrado aqui a minha eterna gratidão!



*“Que os vossos esforços desafiem as impossibilidades,  
lembrai-vos de que as grandes coisas do homem  
foram conquistadas do que parecia impossível”.*

*Charles Chaplin*



# Resumo

Nos últimos anos, o aumento na ocorrência de desastres naturais causados por inundações, deslizamento de terra, secas e tempestades tem chamado atenção de autoridades brasileiras. Com o intuito de monitorar e alertar tais ameaças a fim de evitar ou minimizar os impactos sociais, econômicos e ambientais, foi criado em 2011 o Centro nacional de Monitoramento e Alerta de Desastres Naturais (Cemaden). Com essa proposta, o Cemaden é responsável por pesquisar e implementar soluções tecnológicas para esta demanda, porém o alto custo dos componentes utilizados torna financeiramente inviável a maior abrangência em áreas de risco no Brasil.

Dentro deste contexto foi criado o projeto CIGARRA que possui o objetivo de desenvolver uma solução de baixo custo, flexível, confiável e escalável para realização do monitoramento e alerta de desastres naturais.

Neste trabalho, foi desenvolvido um protótipo de uma plataforma de Rede de Sensores Sem Fio para realização de monitoramento ambiental atendendo os requisitos do projeto CIGARRA. Foi utilizado um computador *Raspberry Pi 3+* como nó coordenador e nos nós sensores foram utilizados um *Arduino Uno*, *Nano* e *Due*. Além dos componentes de *hardware*, foi desenvolvida uma interface gráfica simples que permite e o gerenciamento da rede, um *Bot* no *Telegram* e também uma interface com a plataforma *ThingSpeak* para possibilitar a visualização dos dados em tempo real pelos usuários.

Foram realizados dois diferentes tipos de testes em bancada para validação da plataforma. O primeiro teste contempla a execução de todos os comandos implementados para todos os nós da rede. O segundo teste consiste em simulações de falhas entre os nós sensores da rede. Ambos os testes demonstraram o correto funcionamento da rede, tanto na execução de comandos quanto na recuperação à falhas dos nós.

**Palavras-chave:** Rede de Sensores Sem Fio, RSSF, *Internet* das Coisas, IoT, CIGARRA, *Arduino*, *Raspberry Pi*, Baixo Custo, Escalável, Flexível, Confiável, UNIFESP.



# Abstract

In the last years, the occurrence of natural disasters caused by floods, landslides, drought and storms has drawn the attention of the Brazilian authorities. In order to monitor and alert the threats to avoid or reduce the social, economical and environmental impacts, in 2011 the “Centro nacional de Monitoramento e Alerta de Desastres Naturais” was created. Cemaden has the responsibility to research and implement innovative technology solutions, however the high cost of the components makes it financially unfeasable to expand it into more areas of risk in Brazil.

In this context the CIGARRA was created with the objective of building a solution that is low cost, flexible, reliable and scalable to monitor and alert those environmental threats.

The objective of this project is to build a low cost, flexible, reliable and scalable prototype platform of a Wireless Sensors Network to monitor the environment. The Raspberry Pi 3+ was used as the coordinator node and Arduinos Uno, Nano and Due as the sensors node. Besides the hardware, it was built a simple graphical interface to manage the network, also it was developed a Telegram bot and an interface with the ThingSpeak platform to visualize the data in real time.

It was executed two different kind of tests to validate the platform. The first test consists of the execution of all implemented commands for all the nodes in the network. The second test consists of failure simulations between the sensor nodes of the network. Both tests were successful and proved the correct operation of the network, both with command executions and network failure recovery.

**Keywords:** *Wireless Sensors Network, WSN, Internet of Things, IoT, CIGARRA, Arduino, Raspberry Pi, Low cost, Scalable, Flexible, Reliable, UNIFESP.*



# Listas de ilustrações

Figura 1 – População por número de mortes causadas por desastres naturais no Brasil . . . . .	26
Figura 2 – Exemplo de configuração do projeto CIGARRA . . . . .	28
Figura 3 – Arquitetura de um nó sensor . . . . .	34
Figura 4 – Taxonomia das aplicações . . . . .	36
Figura 5 – Gráfico percentual das diferentes áreas de aplicação . . . . .	37
Figura 6 – Nó sensor com a plataforma Panstamps . . . . .	37
Figura 7 – Configuração dos testes com NRF24L01P e XBee . . . . .	38
Figura 8 – Arquitetura do sistema . . . . .	39
Figura 9 – Arduino Uno Rev3 . . . . .	44
Figura 10 – Arduino Nano . . . . .	45
Figura 11 – Arduino Due . . . . .	46
Figura 12 – <i>Raspberry Pi</i> 3 modelo B . . . . .	47
Figura 13 – Sensor de Umidade e Temperatura - DHT11 . . . . .	49
Figura 14 – Pinagem do sensor DHT11 . . . . .	49
Figura 15 – Diferentes canais da banda 2.4GHz . . . . .	51
Figura 16 – Módulo nRF24L01 . . . . .	52
Figura 17 – Módulo nRF24L01+ com antena externa . . . . .	52
Figura 18 – Amplificador de ruído e potência . . . . .	53
Figura 19 – Exemplo de um <i>dashboard</i> . . . . .	54
Figura 20 – <i>Bot</i> criado para o projeto . . . . .	55
Figura 21 – Interface Gráfica para gerenciamento . . . . .	56
Figura 22 – Topologia exemplo maniacbug . . . . .	57
Figura 23 – Topologia exemplo trmh20 . . . . .	58
Figura 24 – Topologia do projeto . . . . .	60
Figura 25 – Exemplo de topologia possível . . . . .	61
Figura 26 – Esquemático <i>Raspberry Pi</i> . . . . .	64
Figura 27 – Nó coordenador <i>Raspberry Pi</i> . . . . .	64
Figura 28 – Esquemático Arduino Uno . . . . .	65
Figura 29 – Nó sensor Arduino Uno . . . . .	65
Figura 30 – Esquemático Arduino Nano . . . . .	66
Figura 31 – Nó sensor Arduino Nano . . . . .	66
Figura 32 – Esquemático Arduino Due . . . . .	67
Figura 33 – Nó sensor Arduino Due . . . . .	67
Figura 34 – Modos de Operação . . . . .	69
Figura 35 – Solicitando ao nó 01 o envio de 1 <i>frame</i> . . . . .	84

Figura 36 – N�o 01 enviando o dado 1 vez . . . . .	84
Figura 37 – Solicitando ao n�o 011 o envio de 2 <i>frames</i> . . . . .	85
Figura 38 – N�o 011 enviando 2 vezes os dados . . . . .	85
Figura 39 – Solicitando ao n�o 021 o envio de 3 <i>frames</i> . . . . .	86
Figura 40 – N�o 021 enviando 3 vezes os dados . . . . .	86
Figura 41 – Configurando n�o 01 como mestre (ativo) e frequ�ncia de envio de 1s . . . . .	87
Figura 42 – N�o 01 enviando dados a cada 1 segundo . . . . .	87
Figura 43 – Configurando n�o 011 como mestre (ativo) e frequ�ncia de envio de 2s . . . . .	88
Figura 44 – N�o 011 enviando dados a cada 2 segundos . . . . .	88
Figura 45 – Configurando n�o 021 como mestre (ativo) e frequ�ncia de envio de 3s . . . . .	89
Figura 46 – N�o 021 enviando dados a cada 3 segundos . . . . .	89
Figura 47 – Configurando n�o 01 como escravo (passivo) . . . . .	90
Figura 48 – N�o 01 entrando em modo escravo . . . . .	90
Figura 49 – Configurando n�o 011 como escravo (passivo) . . . . .	91
Figura 50 – N�o 011 entrando em modo escravo (passivo) . . . . .	91
Figura 51 – Configurando n�o 021 como escravo (passivo) . . . . .	92
Figura 52 – N�o 021 entrando em modo escravo (passivo) . . . . .	92
Figura 53 – <i>Log</i> da ltima solicita�o via <i>Telegram</i> . . . . .	93
Figura 54 – Tela do computador no navegador <i>web</i> do <i>Telegram</i> . . . . .	93
Figura 55 – Tela do celular no aplicativo <i>Telegram</i> . . . . .	94
Figura 56 – Gr�fico dos dados do n�o 01 . . . . .	94
Figura 57 – Gr�fico dos dados do n�o 011 . . . . .	95
Figura 58 – Gr�fico dos dados do n�o 021 . . . . .	95
Figura 59 – Dados enviados ao <i>ThingSpeak</i> . . . . .	95
Figura 60 – Falha no n�o 011 . . . . .	96
Figura 61 – Falha no n�o 021 . . . . .	97
Figura 62 – Falha no n�o 01 . . . . .	98
Figura 63 – Acessando as configura�es do <i>Raspberry</i> . . . . .	108
Figura 64 – Clique em <i>Enabled</i> para ativar o SPI . . . . .	108
Figura 65 – Clique em <i>Yes</i> para que a altera�o seja aplicada . . . . .	109
Figura 66 – Instale a biblioteca no ambiente de desenvolvimento . . . . .	109
Figura 67 – Bibliotecas instaladas com sucesso . . . . .	110
Figura 68 – Gerenciador de bibliotecas . . . . .	111
Figura 69 – Instala�o da biblioteca . . . . .	112
Figura 70 – C�digos exemplos da biblioteca DHT11 . . . . .	112

# Lista de tabelas

Tabela 1 – Distância e perda de pacotes . . . . .	40
Tabela 2 – Especificações técnicas do Arduino UNO Rev3 . . . . .	44
Tabela 3 – Especificações técnicas do Arduino Nano . . . . .	45
Tabela 4 – Especificações técnicas do Arduino Due . . . . .	46
Tabela 5 – Especificações técnicas do <i>Raspberry Pi</i> 3 modelo B . . . . .	48
Tabela 6 – Ligações físicas do sensor . . . . .	50
Tabela 7 – Potência de saída vs consumo energético . . . . .	51
Tabela 8 – Endereçamento dos Nós . . . . .	59
Tabela 9 – Ligação <i>Raspberry Pi</i> e módulo nRF24L01+ . . . . .	63
Tabela 10 – Ligação Arduino Uno e módulos . . . . .	64
Tabela 11 – Ligação Arduino Nano e módulos . . . . .	66
Tabela 12 – Ligação Arduino DUE e módulos . . . . .	67
Tabela 13 – Custos Plataforma - Mercado Livre . . . . .	81
Tabela 14 – Custos Plataforma - Dx.com . . . . .	81
Tabela 15 – Custos nó coordenador . . . . .	82
Tabela 16 – Custos nó sensor . . . . .	82



# Lista de abreviaturas e siglas

API	<i>Application Programming Interface</i>
CE	<i>Chip Enable</i>
CLK	<i>Clock</i>
CSN	<i>Chip Select Not</i>
dBm	<i>Decibel Miliwatt</i>
EEPROM	<i>Electrically-Erasable Programmable Read-Only Memory</i>
GHz	<i>Giga Hertz</i>
GND	<i>Ground</i>
GPIO	<i>General Purpose Input Output</i>
GUI	<i>Graphic User Interface</i>
HDMI	<i>High-Definition Multimedia Interface</i>
IDE	<i>Integrated Development Environment</i>
IoT	<i>Internet of Things</i>
LAN	<i>Local Area Network</i>
LPA	<i>Low-Noise Power Amplifier</i>
Mbps	<i>Megabits per second</i>
MCTI	Ministério da Ciência, Tecnologia, Inovações e Comunicações
MEMS	<i>Micro Electro-Mechanical Systems</i>
MHz	<i>Mega Hertz</i>
MI	<i>Master In</i>
MO	<i>Master out</i>
mV	<i>Milivolt</i>
PA	<i>Power Amplifier</i>

PCB	Placa de Circuito Impresso
PWM	<i>Pulse Width Modulation</i>
PWR	<i>Power</i>
RSSF	Rede de Sensores Sem Fio
SD	<i>Secure Digital</i>
SMA	<i>SubMiniature version A</i>
SPI	<i>Serial Peripheral Interface</i>
USB	<i>Universal Serial Bus</i>
UNIFESP	Universidade Federal de São Paulo
V	<i>Volt</i>
WSN	<i>Wireless Sensors Network</i>

# Sumário

<b>1</b>	<b>INTRODUÇÃO . . . . .</b>	<b>25</b>
<b>1.1</b>	<b>Contexto do Trabalho . . . . .</b>	<b>25</b>
<b>1.2</b>	<b>Definição do Problema . . . . .</b>	<b>25</b>
<b>1.3</b>	<b>Motivação . . . . .</b>	<b>26</b>
<b>1.4</b>	<b>Objetivos . . . . .</b>	<b>28</b>
1.4.1	Gerais . . . . .	28
1.4.2	Específicos . . . . .	28
<b>1.5</b>	<b>Metodologia . . . . .</b>	<b>29</b>
<b>1.6</b>	<b>Estrutura do Trabalho . . . . .</b>	<b>29</b>
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA . . . . .</b>	<b>31</b>
<b>2.1</b>	<b>Revisão Bibliográfica . . . . .</b>	<b>31</b>
2.1.1	Base de Dados . . . . .	31
2.1.2	Questões de Pesquisa . . . . .	31
2.1.3	<i>String</i> de Busca . . . . .	32
2.1.4	Critérios de Inclusão . . . . .	32
2.1.5	Critérios de Exclusão . . . . .	32
<b>2.2</b>	<b>Desastres Naturais . . . . .</b>	<b>32</b>
<b>2.3</b>	<b>Redes de Sensores Sem Fio . . . . .</b>	<b>33</b>
2.3.1	Nó Sensor . . . . .	33
2.3.2	Nó Coordenador . . . . .	33
2.3.3	Unidade de Sensoriamento . . . . .	33
2.3.4	Unidade de Processamento . . . . .	34
2.3.5	Unidade de Comunicação . . . . .	34
<b>2.4</b>	<b><i>Internet of Things - IoT</i> . . . . .</b>	<b>35</b>
<b>2.5</b>	<b>Trabalhos Relacionados . . . . .</b>	<b>35</b>
<b>3</b>	<b>COMPONENTES DE HARDWARE/SOFTWARE UTILIZADOS E TOPOLOGIA DO SISTEMA DE REDE IMPLEMENTADA . . . . .</b>	<b>43</b>
<b>3.1</b>	<b>Arduino . . . . .</b>	<b>43</b>
3.1.1	Arduino Uno Rev3 . . . . .	44
3.1.2	Arduino Nano . . . . .	45
3.1.3	Arduino Due . . . . .	46
<b>3.2</b>	<b>Raspberry Pi 3 modelo B . . . . .</b>	<b>47</b>
<b>3.3</b>	<b>Sensor DHT11 . . . . .</b>	<b>48</b>
<b>3.4</b>	<b>Comunicação Sem Fio . . . . .</b>	<b>50</b>

3.5	<i>ThingSpeak</i>	54
3.6	<i>Telegram Chatbot</i>	55
3.7	Interface Gráfica desenvolvida com <i>Qt Designer</i>	56
3.8	Topologia da Rede	57
4	<b>DESENVOLVIMENTO DA PLATAFORMA</b>	63
4.1	Ligaçāo Física dos Componentes	63
4.2	Configuração do Ambiente de Desenvolvimento	68
4.3	Desenvolvimento do <i>Software</i>	68
4.3.1	Modos de Operação	68
4.3.2	<i>Raspberry Pi</i>	69
4.3.3	Arduino	76
4.4	Custos dos Componentes	81
5	<b>RESULTADOS E DISCUSSĀO</b>	83
5.1	Testes de Funcionamento da Plataforma Desenvolvida	83
5.2	Testes de Simulações de Falhas	96
6	<b>CONSIDERAÇĀES FINAIS</b>	99
6.1	Dificuldades Encontradas	100
6.2	Trabalhos Futuros	100
	<b>REFERÊNCIAS</b>	101
	<b>APÊNDICES</b>	105
	<b>APÊNDICE A – CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO</b>	107
A.0.1	<i>Raspberry Pi</i>	107
A.0.1.1	Biblioteca <i>RF24Network</i>	107
A.0.2	Arduino	109
A.0.2.1	Biblioteca <i>RF24Network</i>	109
A.0.2.2	Biblioteca Sensor - DHT11	111
	<b>APÊNDICE B – CÓDIGO RASPBERRY PI</b>	113
	<b>APÊNDICE C – CÓDIGO ARDUINO UNO</b>	119
	<b>APÊNDICE D – CÓDIGO ARDUINO DUE</b>	123
	<b>APÊNDICE E – CÓDIGO ARDUINO NANO</b>	127

**APÊNDICE F – CÓDIGO INTERFACE GRÁFICA . . . . . 131**



# 1 Introdução

## 1.1 Contexto do Trabalho

No Brasil, o número e a intensidade dos desastres naturais causados por inundações, deslizamentos de terra, secas e tempestades tem chamado cada vez mais a atenção das autoridades brasileiras e do Governo Federal. Em Julho de 2011, foi criado pelo Decreto Presidencial no 7.513, o Centro nacional de Monitoramento e Alerta de Desastres Naturais (Cemaden), órgão vinculado ao Ministério da Ciência, Tecnologia, Inovações e Comunicações (MCTI). Fenômenos naturais severos, estão diretamente relacionados com características de uma região, tais como solo, topografia, vegetação e condições meteorológicas. Desastres naturais ocorrem quando estes fenômenos atingem os locais onde os seres humanos vivem, causando danos e prejuízos.

O Cemaden é responsável por monitorar ameaças naturais em áreas suscetíveis à ocorrência de desastres naturais. Além de realizar pesquisas, um dos principais objetivos do órgão é também a contribuição através de inovações tecnológicas a melhorias de seus sistemas na detecção e alerta, de forma a notificar as autoridades para que medidas necessárias sejam tomadas de forma eficiente, resultando assim a diminuição dos prejuízos, sejam nas áreas sociais, econômicas ou ambientais.

## 1.2 Definição do Problema

Atualmente os sistemas utilizados pelo Cemaden para monitoramento de desastres naturais possuem custos elevados, o que dificulta uma abrangência maior em áreas de interesse. O principal meio de comunicação é feito utilizando redes móveis de celular, 3G ou 4G, porém muitas vezes é comum presenciar falhas em condições que antecedem desastres naturais. Vale ressaltar que tecnologias de comunicação com redes móveis também são de alto custo, baixa abrangência em algumas áreas e possui pouca confiabilidade em condições ambientais desfavoráveis.

A plataforma e o conjunto de sensores são adquiridos comercialmente e podem possuir valores que se aproximam de dezena de milhares de reais. Portanto, constata-se uma limitação de recursos financeiros para que o Cemaden possa monitorar mais áreas de risco em território nacional.

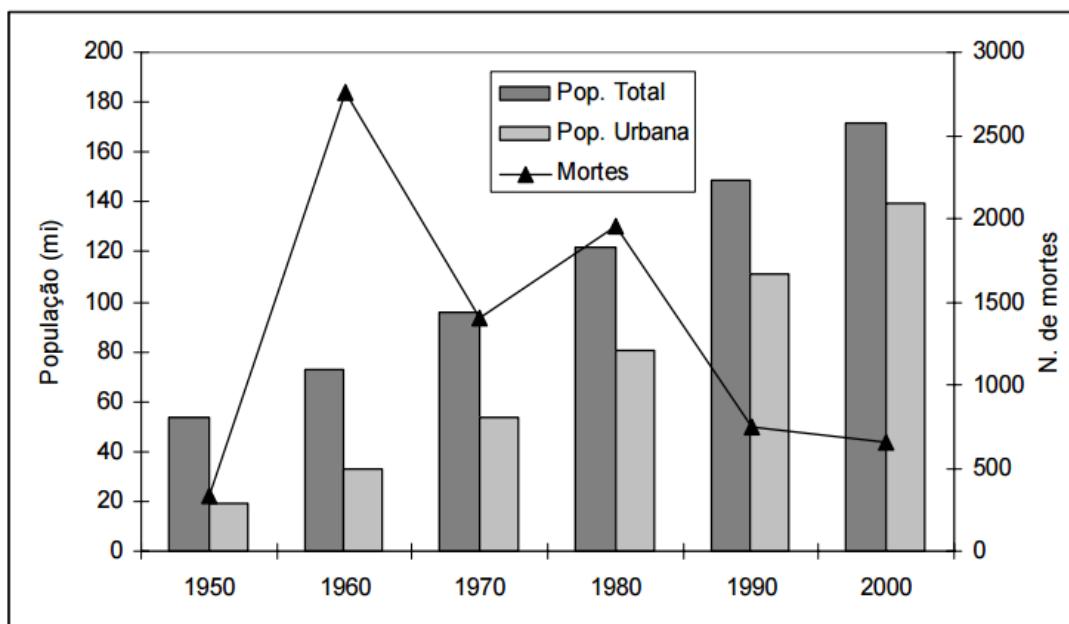
### 1.3 Motivação

Segundo ([ALCÁNTARA-AYALA, 2002](#)), a ocorrência de desastres naturais não está ligada somente a vulnerabilidade causada por condições geoambientais, mas também pela fragilidade causada pelo sistema sócio, econômico, político e cultural.

Muitas vezes não é possível impedir que tais fenômenos ocorram, mas é possível minimizar seus efeitos com prevenção e principalmente monitoramento e alerta destas ameaças. A cada R\$ 1,00 investidos em prevenção equivale em média de R\$ 25,00 a R\$ 30,00 em obras de reconstrução após desastres ([CASTRO et al., 1996](#)).

As causas de mortes devido a desastres naturais têm reduzido no Brasil e isto se deve a ações preventivas, principalmente associadas a defesa civil ([KOBINYAMA et al., 2004](#)). Mesmo com o aumento da população é possível observar uma queda na quantidade de mortes causadas por desastres naturais, como pode ser observado na [Figura 1](#).

Figura 1 – População por número de mortes causadas por desastres naturais no Brasil



Fonte: ([KOBINYAMA et al., 2004](#))

([Licco; Mac Dowell , 2015](#)) cita que, de acordo com um levantamento feito pelo Ministério da Integração Nacional, o Brasil sofreu mais de 30 mil desastres naturais entre 1990 e 2012, uma média de 1363 desastres ao ano.

O Atlas Brasileiro de Desastres Naturais ([UFSC, 2013](#)) mostrou que, entre 1991 e 2012 foram registradas 31909 catástrofes no Brasil, com 73% das ocorrências na última década. E as tipologias dos desastres com maior recorrência no Brasil são as estiagens, secas e inundações bruscas e alagamentos. A diferença entre a estiagem e a seca é que o primeiro fenômeno ocorre em um intervalo de tempo, enquanto a seca ocorre de forma

permanentemente.

De acordo com um relatório feito pelo Escritório das Nações Unidas para Redução do Risco de Desastres ([NATIONS, 2014](#)), existe uma necessidade fundamental em minimizar os impactos de riscos futuros de desastres naturais em comunidades em áreas de risco. Uma das soluções encontradas para que as cidades consigam viver com esses fenômenos naturais se encontra na redução dos níveis existentes de riscos que favorecem os desastres, fortalecendo a resiliência social, ambiental e econômica ([Ribeiro; Vieira; Tômio, 2017](#)).

Dentro deste contexto, o objetivo geral do projeto é no desenvolvimento de uma solução de baixo custo, flexível, confiável e escalável para realização do monitoramento e alerta de desastres naturais.

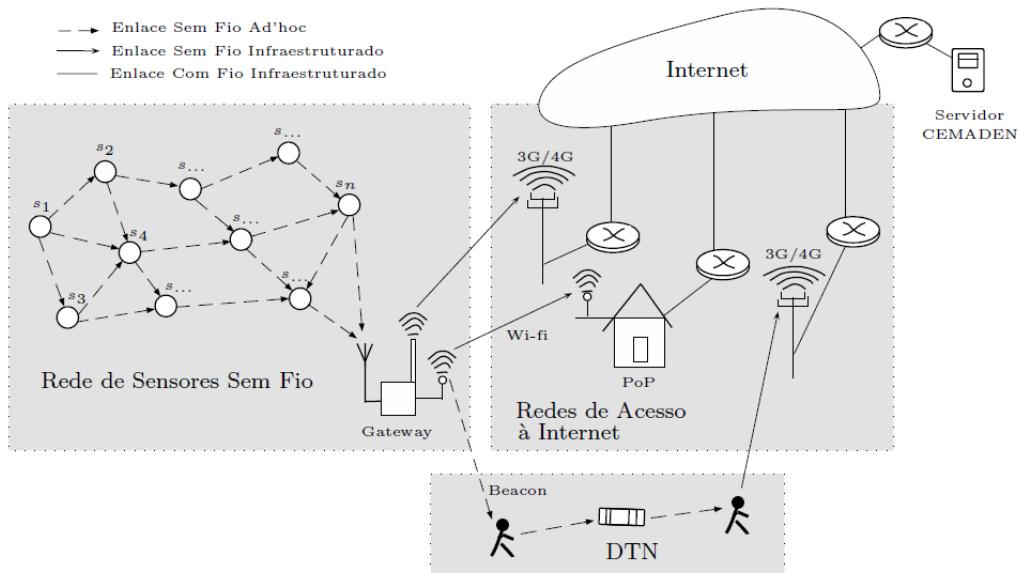
No projeto CIGARRA planeja-se o desenvolvimento de uma plataforma que utiliza a tecnologia de Rede de Sensores Sem Fio (RSSF) para realização de monitoramentos e alertas a desastres naturais. Este trabalho levou em conta o desenvolvimento de uma solução de baixo custo, flexível, confiável e escalável, descritas com maiores detalhes abaixo:

- **Baixo custo:** A redução no custo da plataforma e dos sensores permitirá uma maior abrangência de monitoramentos, tanto na quantidade de nós sensores quanto de redes de sensores. Além disso, serão utilizados componentes de baixo custo e de código aberto para o desenvolvimento do projeto de *hardware* e *software*.
- **Flexível:** O desenvolvimento de uma plataforma flexível possibilita o uso do mesmo *hardware* para realização de monitoramentos com sensores diferentes. A fácil troca e instalação de sensores na plataforma permite rápida manutenção e a realização de diferentes tipos de monitoramentos. Existem por exemplo, regiões no país que são afetadas pela chuva ou pela seca portanto, a flexibilidade da plataforma permite a adaptação do *hardware* com sensores para a realização de monitoramento em diversas regiões do país.
- **Confiável:** Uma vez que estamos lidando com vidas, é de extrema importância que a plataforma seja capaz de monitorar e alertar e que mesmo em casos de falhas em alguns nós sensores a rede permaneça funcional.
- **Escalável:** O Cemaden é responsável por monitorar aproximadamente 1000 municípios no país. O sistema deve ser capaz de tratar grandes volumes de dados tanto na transmissão, recepção e processamento.

A arquitetura do projeto CIGARRA pode ser observada na [Figura 2](#). A Rede de Sensores Sem Fio, composta por inúmeros nós sensores, é responsável por fazer a coleta de dados desejados em uma área específica onde foi instalada. Também é de responsabilidade da RSSF trafegar os dados, por uma comunicação sem fio, até o nó coordenador (*gateway*).

Os dados são recebidos pelo *gateway*, que tem o papel de enviá-los através de redes 3G/4G ou *Wi-Fi* para o servidor do Cemaden. Por fim, os dados enviados deverão ser coletados e analisados em tempo real para a realização do monitoramento e alerta para as possíveis ocorrências de desastres naturais. A rede deve ser tolerante a atrasos e desconexões (DTN).

Figura 2 – Exemplo de configuração do projeto CIGARRA



Fonte: (CONCEIÇÃO, 2015)

## 1.4 Objetivos

Os objetivos deste trabalho de graduação foram subdivididos entre gerais e específicos e serão apresentados nas seguintes seções.

### 1.4.1 Gerais

O objetivo geral deste trabalho é de desenvolver uma plataforma com uma Rede de Sensores Sem Fio compostas por nós sensores e nó coordenador que atendam os principais requisitos do projeto CIGARRA: baixo custo, flexíveis, confiáveis, escaláveis e de código aberto, focando em seu sistema de comunicação.

### 1.4.2 Específicos

Os objetivos específicos referem-se a:

- Realizar um levantamento bibliográfico para compreender as principais tecnologias e abordagens utilizadas para implantação de uma RSSF para monitoramentos em geral.
- Verificar as tecnologias e abordagens estudadas para definir a viabilidade da plataforma levando em conta os requisitos principais do projeto.
- Selecionar os componentes de *hardware* que atendem os requisitos e que serão utilizados no projeto.
- Desenvolver a plataforma de *hardware* e *software* para o sistema de comunicação e realizar testes para fins de melhorias e validação.
- Documentar todos os materiais e métodos para que seja possível a continuidade do projeto.

## 1.5 Metodologia

Para a elaboração deste trabalho, primeiramente foram levantados requisitos e dificuldades enfrentadas ao se criar um sistema de monitoramento ambiental de baixo custo no Brasil.

Após entender os principais requisitos e as dificuldades enfrentadas, foi feita uma revisão bibliográfica para entender quais tecnologias e abordagens estão sendo utilizadas no momento para realização de monitoramento.

Após o levantamento bibliográfico, a escolha da tecnologia e abordagem foi feita atendendo os principais requisitos do projeto CIGARRA, que são: baixo custo, flexibilidade, confiabilidade e escalabilidade.

A aplicação prática da solução proposta resultou no desenvolvimento e validação de uma plataforma de *hardware* e *software* para a criação de uma Rede de Sensores Sem Fio para monitoramento ambiental.

## 1.6 Estrutura do Trabalho

O presente trabalho está dividido em 5 capítulos. O [Capítulo 1](#) apresenta a introdução, contexto, definição do problema e a motivação para a realização deste trabalho.

O [Capítulo 2](#) apresenta o levantamento bibliográfico sobre Redes de Sensores Sem Fio, as tecnologias que estão sendo utilizadas atualmente e os trabalhos relacionados.

O [Capítulo 3](#) apresenta os materiais, custos, plataformas, ferramentas utilizadas e configuração do ambiente de desenvolvimento deste trabalho.

O [Capítulo 4](#) apresenta aspectos do desenvolvimento da paltaforma de *software* e *hardware*.

O [Capítulo 5](#) apresenta os resultados obtidos através de testes e simulações de falha na rede de sensores.

O [Capítulo 6](#) apresenta as considerações finais, as principais dificuldades encontradas durante o desenvolvimento do trabalho e tópicos interessantes para o desenvolvimento de trabalhos e estudos futuros.

## 2 Fundamentação Teórica

Neste capítulo, serão apresentados conceitos adquiridos através de estudos e do levantamento bibliográfico. Além dos principais conceitos teóricos necessários para o entendimento do projeto, será apresentado o método de pesquisa realizado para o levantamento bibliográfico.

### 2.1 Revisão Bibliográfica

O presente capítulo descreve como foi feito o levantamento bibliográfico para estudo das tecnologias utilizadas na realização de monitoramentos utilizando RSSF com componentes de baixo custo e de código aberto.

#### 2.1.1 Base de Dados

A base de dados utilizada para este levantamento bibliográfico foi o Scopus. Além do levantamento bibliográfico feito no *Scopus*, também foram utilizados trabalhos de graduação, teses de mestrado e doutorado, artigos, *datasheets*, *sites* e fóruns sobre tecnologia embarcada.

Um protocolo de revisão foi criado para definir as regras e os procedimentos que foram levados em conta para a realização da revisão. O protocolo é de extrema importância, pois evita os possíveis vieses que seriam causados pelo pesquisador ([KITCHENHAM, 2004](#)).

#### 2.1.2 Questões de Pesquisa

O levantamento bibliográfico deve ser capaz de responder as questões de pesquisa previamente definidas pelo pesquisador. As questões de pesquisa foram definidas como:

1. Quais são os tipos de aplicações que utilizam redes de sensores sem fio para monitoramentos?
2. Quais tecnologias de baixo custo estão sendo utilizadas para monitoramento e alerta de desastres naturais?
3. Quais são os resultados positivos observados pelo uso de tecnologias de baixo custo para monitoramentos?
4. Quais os problemas observados pelo uso de tecnologias de baixo custo para monitoramentos?

### 2.1.3 *String* de Busca

Antes de realizar as buscas, foi necessário criar uma *string* de busca que representasse o conteúdo de interesse a ser pesquisado.

Parte do levantamento bibliográfico foi feito utilizando a *string* de busca definida abaixo:

*TITLE-ABS-KEY(("wireless sensors network"OR "wireless mesh network") AND ("monitoring"OR "surveillance") AND "low cost"AND "open source")*

### 2.1.4 Critérios de Inclusão

Os estudos incluídos no levantamento bibliográfico satisfizeram as seguintes condições:

1. Estudos que abordam o uso de rede de sensores sem fio;
2. Estudos que abordam o uso de tecnologias de baixo custo;
3. Estudos *open-source*;
4. Estudos que realizam algum tipo de monitoramento.

### 2.1.5 Critérios de Exclusão

Os estudos que violam os critérios de exclusão, descritos abaixo, foram excluídos do levantamento:

1. Estudos que não abordam o uso de rede de sensores sem fio;
2. Estudos que abordam tecnologias de alto custo;
3. Estudos incompletos ou duplicados;
4. Estudos sem o acesso na íntegra;
5. Estudos que não estejam em inglês ou português.

## 2.2 Desastres Naturais

Podem-se considerar desastres naturais quando fenômenos como: inundações, escorregamentos, secas e furacões atingem e causam danos sociais e econômicos para o ser humano (KOBIYAMA et al., 2004). Segundo (CASTRO, 2009), um desastre natural pode ser definido como o resultado de algum evento hostil, seja ele provocado pelo homem ou pelo meio ambiente, sobre um ecossistema e que cause danos econômicos e sociais.

## 2.3 Redes de Sensores Sem Fio

Uma Rede de Sensores Sem Fio (RSSF) consiste em um grande número de nós sensores e coordenador que utilizam comunicação sem fio que podem ser instalados em uma área de interesse. Tais redes podem ser utilizadas em inúmeras aplicações, como monitoramento ambiental, cuidados de saúde, monitoramento de infraestruturas, segurança pública, segurança privada, médicas, transportes e militares (RUIZ, 2003).

Com o avanço na área de sistemas eletromecânicos (MEMS - *Micro Electro-Mechanical Systems*), comunicações sem fio tem exercido um papel fundamental para o uso e desenvolvimento de sensores inteligentes. A tendência é que a produção, destes sensores, seja feita em larga escala para baratear seu custo e adquirir maiores investimentos nesta área de pesquisa e desenvolvimento (LOUREIRO et al., 2003).

Segundo (DELAMO et al., 2015), uma rede de sensores sem fio é uma rede distribuída com dispositivos autônomos chamados de motes. Dentro dessa rede, pelo menos um mote deve funcionar como nó coordenador (*Gateway*). Cada mote contém uma unidade de energia, unidade de processamento, unidade sensora e de unidade de comunicação.

### 2.3.1 Nô Sensor

Também conhecida como nodo, cada nó sensor da RSSF pode ser equipado com sensores para a realização de detecção de eventos de acordo com o interesse especificado.

A [Figura 3](#) representa a arquitetura de um nó sensor. Uma RSSF é composta por vários nós sensores, podendo cada um possuir uma configuração diferente de sensores.

Para uma aplicação de monitoramento ambiental seria interessante que o nodo contenha sensores de temperatura, umidade do solo, chuva, vento, deslizamento de terra, etc. Já para uma aplicação de segurança privada seria interessante utilizar sensores de presença, câmeras ou som. Isto justifica a grande diversidade de aplicações com o uso de RSSF para monitoramento.

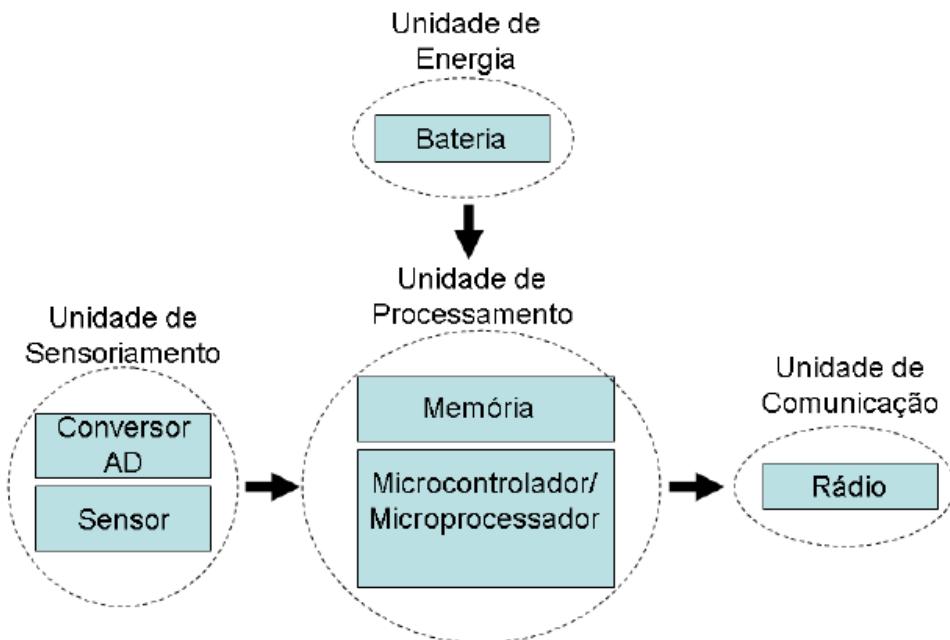
### 2.3.2 Nô Coordenador

Além de exercer o papel do gerenciamento de toda a RSSF, o nó coordenador ou *gateway*, é responsável por coletar, processar e disponibilizar os dados para o usuário. Os dados podem ser salvos em uma memória não volátil ou enviados por meio de Wi-Fi ou 3G/4G para acesso remoto do usuário, como pode ser visto na [Figura 2](#).

### 2.3.3 Unidade de Sensoriamento

A unidade de sensoriamento é responsável por realizar medições de sinais físicos e transformá-los em sinais analógicos ou digitais para o armazenamento da informação

Figura 3 – Arquitetura de um nó sensor



Fonte: ([CONCEIÇÃO, 2015](#))

pela unidade de processamento. A unidade de sensoriamento deve conter os sensores para coleta dos dados e interfaces de conversão entre sinais analógicos e digitais.

Para este projeto, foi previsto o uso de dois sensores, um sensor de temperatura e de umidade relativa do ar.

#### 2.3.4 Unidade de Processamento

A unidade de processamento geralmente é composta por uma memória e um microcontrolador. Em alguns casos a memória interna do microcontrolador pode ser suficiente para o armazenamento de dados, porém quando não for suficiente, uma memória externa poderá ser utilizada para armazenar os dados.

A unidade de processamento é responsável por gerenciar a coleta dos dados adquiridos pelos sensores e repassá-los para a unidade de comunicação.

#### 2.3.5 Unidade de Comunicação

A unidade de comunicação é composta por um rádio. Os rádios podem ser classificados em receptores, transmissores ou transceptores. Rádios receptores somente são capazes de receber dados. Os transmissores são capazes de somente enviar dados. Já os transceptores são capazes de receber e enviar dados.

Neste trabalho, o rádio a ser utilizado deverá ser um rádio transceptor, pois o nodo deverá ser capaz de enviar os dados dos sensores e receber comandos do *gateway* para realização do gerenciamento da coleta de dados.

Segundo ([DEMENTYEV et al., 2013](#)), o maior gasto energético em aplicações de RSSF se deve a transmissão de dados através das comunicações sem fio. Portanto, é de extrema importância a seleção, otimização e controle dos dados a serem transmitidos para a minimização do tempo de transmissão.

## 2.4 *Internet of Things - IoT*

O termo Internet das Coisas, mais conhecido em inglês como *Internet of Things*, não é um termo relativamente novo. Ele foi citado pela primeira vez por ([ASHTON, 2010](#)) em 1999 em uma apresentação para a empresa Procter & Gamble (P&G) como tendo um potencial e impacto tão grande quanto o surgimento da *Internet*.

De uma maneira geral, o conceito de IoT pode ser descrito como um ambiente de objetos físicos que estão interconectados através da *Internet* por meio de sensores ou atuadores, criando um ecossistema de computação onipresente, voltado para a facilitação do cotidiano das pessoas ([MAGRINI, 2018](#)).

Ainda segundo ([MAGRINI, 2018](#)), existem algumas divergências em relação ao conceito de IoT, porém o que todas definições têm em comum é que elas se concentram em como computadores, sensores, atuadores e objetos interagem entre si.

O grande desafio existente atualmente é na construção de uma arquitetura geral e padronizada para IoT devido a sua complexidade e grande variabilidade de dispositivos e tecnologias que estão emergindo ([Zanella et al., 2014](#)).

De acordo com ([OLIVEIRA, 2017](#)), o conceito de *Internet* das Coisas tem mudado a percepção das pessoas com as “coisas” que estão ao nosso redor no dia a dia. Observamos diversas aplicações que utilizam o conceito de IoT, tais como: cidades inteligentes, segurança, monitoramento ambiental, trânsito, mobilidade, etc. A rápida adoção de IoT se deve principalmente ao desenvolvimento tecnológico e barateamento dos componentes de *hardware* que possuem essa característica e capacidade de se conectar com a *Internet*.

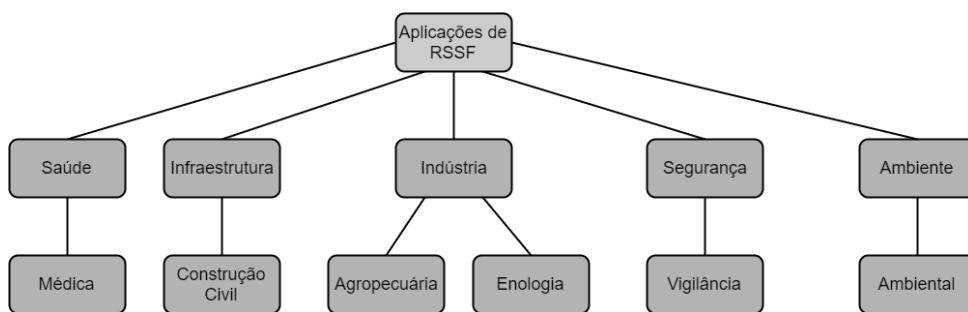
## 2.5 Trabalhos Relacionados

Nesta seção, serão apresentados os trabalhos relevantes adquiridos pelo levantamento bibliográfico seguindo o protocolo definido anteriormente na [seção 2.1](#). O levantamento foi feito para se ter o conhecimento das tecnologias que estão sendo utilizadas atualmente para a realização de monitoramentos em geral.

A busca inicial utilizando a base de dados Scopus retornou 21 resultados. Dentre os 21 resultados, dois estudos foram excluídos, um por não fornecer acesso completo e outro por ser um estudo incompleto, portanto resultando em 19 estudos a serem analisados.

Inicialmente os trabalhos foram classificados levando em conta suas diferentes áreas de aplicação. Dentre os estudos obtidos, as áreas de aplicações de RSSF para realização de monitoramentos observadas foram: saúde, infraestrutura, indústria, segurança e ambiente. A [Figura 4](#) representa uma taxonomia das aplicações de RSSF feita através deste levantamento bibliográfico.

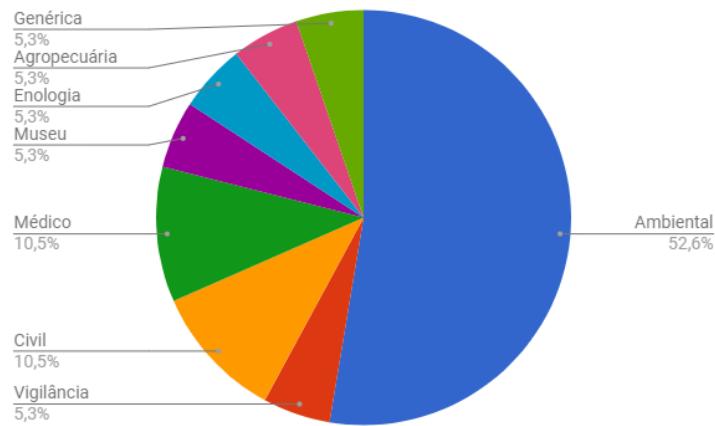
Figura 4 – Taxonomia das aplicações



Fonte: O Autor

Dos 19 estudos resultantes do levantamento, 2 estudos utilizam tecnologia de RSSF para aplicações médicas ([SÁMANO-ROBLES; GAMEIRO, 2011](#); [ADAMS et al., 2007](#)), 2 para aplicações de construção civil ([JANG et al., 2011](#); [MOORE et al., 2013](#)), 1 para aplicações agropecuárias ([CHELLI; CHAVHAN et al., 2013](#)), 1 para aplicação de enologia ([GENNARO et al., 2014](#)), 1 para aplicações de vigilância ([LUPU et al., 2014](#)), 1 para aplicação de monitoramento de níveis de luminosidades em um museu ([LONDERO; FAIRBANKS-HARRIS; WHITMORE, 2016](#)), 1 para monitoramentos em geral ([GONZALEZ et al., 2007](#)) e 10 para aplicações ambientais ([LEE; LIN, 2016](#); [NIKHADE, 2015](#); [DELAMO et al., 2015](#); [LIN et al., 2015](#); [FERDOUSH; LI, 2014](#); [COOK; MYERS; TREVATHAN, 2013](#); [CAMA et al., 2013](#); [CIANCETTA et al., 2010a](#); [CIANCETTA et al., 2010b](#); [WHELAN; FUCHS; JANOVAN, 2008](#)). Todas as 10 aplicações ambientais que foram estudadas faziam ou a medição de temperatura, umidade relativa do ar, umidade do solo ou detecção de incêndios. Tecnologias de RSSF para realização de monitoramentos tem se tornado uma tendência, pois pode ser aplicada a diversas áreas de interesse, como mostrado na [Figura 5](#). Nesta figura, encontra-se um gráfico percentual de cada tipo de aplicação que foi gerado pelo levantamento bibliográfico utilizando a base de dados Scopus.

Figura 5 – Gráfico percentual das diferentes áreas de aplicação

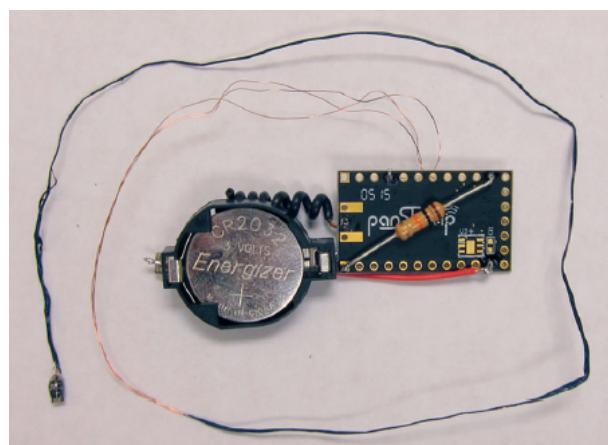


Fonte: O Autor

O trabalho apresentado por (YANG; SOH; YAP, 2017) apresenta uma aplicação de RSSF para monitoramento remoto em uma galeria de um museu da Universidade de Yale. Os autores citam que a plataforma é flexível à alterações, de baixo custo em comparação a dispositivos comerciais e permite o *logging* e a visualização remota dos dados dos sensores.

O *hardware*, visto na Figura 6, é composto por uma placa da empresa Panstamps, vendida comercialmente com microcontrolador MSP430 da *Texas Instruments* e especialmente projetada para ter um baixo consumo energético para aplicações que utilizem comunicação sem fio. O rádio de comunicação já é integrado na plataforma e pode operar nas bandas de: 433, 868, 905, 915 e 918MHz. Segundo (DALY; MELIA; BALDWIN, 2010; MING et al., 2008) a banda de 433MHz é particularmente interessante em casos de florestas densas ou locais que contenham paredes de concreto, pois possui maior poder de penetração em objetos em relação as bandas de frequências maiores.

Figura 6 – Nô sensor com a plataforma Panstamps

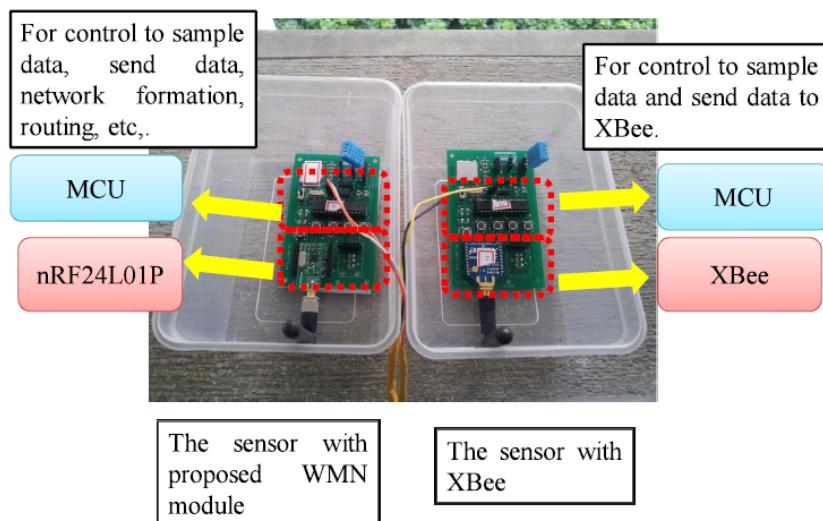


Fonte: (YANG; SOH; YAP , 2017)

O trabalho de (LEE; LIN, 2016) apresenta uma comparação entre as taxas de envio e recebimento de pacotes utilizando os módulos de comunicação nRF24L01P e XBee, ambos são componentes de prateleira e de fácil acesso.

Os módulos utilizados para realização dos testes podem ser observados na Figura 7. Segundo (LEE; LIN, 2016), os resultados dos testes se mostraram satisfatórios. As taxas foram de: 94,09% e 91,19% para entrega de pacotes, 5,14% e 10,25% para o desvio padrão, em relação aos módulos nRF24L01P e XBee, respectivamente.

Figura 7 – Configuração dos testes com NRF24L01P e XBee



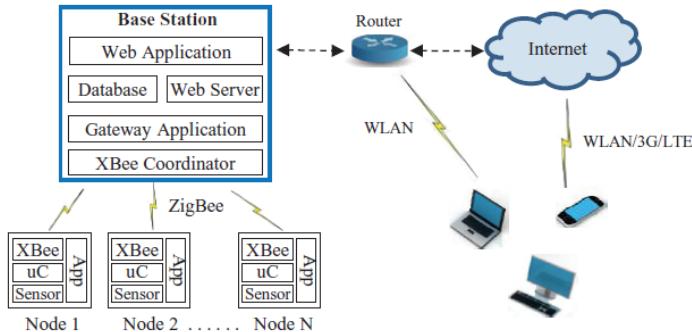
Fonte: (LEE; LIN, 2016)

O trabalho apresentado por (NIKHADE, 2015) faz uso das tecnologias do *Raspberry Pi* e XBee para a realização de monitoramento ambiental. O autor cita que a plataforma possui módulos de baixo custo, baixo consumo energético e altamente escalável em relação a quantidade de sensores e nós sensores. O *Raspberry Pi* foi utilizado como nó coordenador e o módulo XBee para transmissão dos dados. Para o nó sensor, o microcontrolador utilizado foi o ATMEGA324PA de 8 bits.

Esta plataforma possui baixo custo, baixo consumo energético, é compacta, de fácil manuseio para instalação e manutenção. A grande vantagem do uso do *Raspberry Pi* como nó coordenador se dá pela facilidade na integração com a RSSF e diversas aplicações como servidor de banco de dados e servidor web. O *Raspberry Pi* permite o uso de redes de sensores com diferentes tipos de plataformas, sendo assim muito útil para a realização de monitoramentos ambientais (NIKHADE, 2015).

(FERDOUSH; LI, 2014) faz uso do computador *Raspberry Pi* como nó coordenador, Arduino como controlador do nó sensor e o módulo XBee como comunicador sem fio, como pode ser observado na Figura 8.

Figura 8 – Arquitetura do sistema



Fonte: ([FERDOUSH; LI, 2014](#))

O autor destaca que o sistema possui inúmeros pontos positivos, dentre eles se destacam o baixo custo, tamanho compacto, escalabilidade, fácil customização, instalação e manutenção. Porém, o melhor benefício se encontra no uso do computador *Raspberry Pi*, pois o uso do mesmo facilita na integração dos servidores de banco de dados e servidores hospedados na web.

Os autores ([MAIA; OLIVEIRA, 2016](#)) desenvolveram um trabalho que consistia na confecção de um sistema embarcado *dualband* capaz de se comportar como um nó de uma rede de sensores sem fio.

O artigo descreve o desenvolvimento de um sistema embarcado com um microcontrolador ARM Cortex-M0+ que contém dois módulos transceptores de rádio frequência. O primeiro módulo trata-se do nRF24L01+, que transmite a uma frequência de 2.4GHz, enquanto o segundo módulo MX-FS-03V transmite a uma frequência de 433MHz.

O sistema é capaz de escolher, através de algoritmos, qual módulo utilizar para a propagação dos dados, a fim de garantir a entrega dos pacotes e de consumir a menor quantidade de energia possível.

Segundo ([DORMER, 2008](#)), a banda de frequência de 2.4GHz possui grande perda de qualidade por metro, principalmente na presença de chuva, tem um consumo maior e dificuldade em transpor obstáculos. A banda de frequência de 433MHz possui maior sensibilidade de recepção, menor perda de qualidade por metro e menor consumo.

O trabalho de ([HSIAO et al., 2017](#)) apresentou uma plataforma de RSSF de baixo custo, flexível, confiável e escalável para aplicações de monitoramento e alerta de desastres naturais. O autor fez o uso do Arduino UNO R3 para os nós sensores, o módulo Intel Edison como nó coordenador, o módulo nRF2401+ para comunicação sem fio e sensores de chama, gás, chuva, temperatura, pressão, umidade do solo e luminosidade. Porém, o módulo Intel Edison que foi utilizado como nó coordenador foi descontinuado pela Intel no ano de 2017.

O autor descreveu o cenário de testes, onde incluía a distância entre os nós e a porcentagem de perda de pacotes durante as transmissões, o qual pode ser observado na [Tabela 1](#) abaixo:

Tabela 1 – Distância e perda de pacotes

Distância	Perda de Pacotes
25 metros	3%
50 metros	12%
75 metros	31%
80 metros	74%

Fonte: ([HSIAO et al., 2017](#))

O estudo conduzido por ([SAHA et al., 2017](#)) fez um comparativo entre o módulo de comunicação sem fio nRF24L01+ com o módulo XBee. As métricas utilizadas para comparação foram: medição do *throughput* da rede, eficiência no roteamento e consumo de energia.

O comparativo do *throughput*, levou em conta a taxa de transferência da rede, tanto em comunicações ponto a ponto quanto *multihop*, o módulo nRF24L01+ mostrou taxas consideravelmente maiores que o módulo XBee.

O segundo teste mediu o tempo de recuperação do módulo nRF24L01+ à falhas. Foram realizadas 60 amostragens e o tempo mínimo, médio e máximo de recuperação medidos em milissegundos foram de: 91, 3667 e 31590, respectivamente. O que demonstra a rápida recuperação da biblioteca RFNetwork, desenvolvida por trmh20, em caso de falhas nos nós sensores.

Já no comparativo energético, foi observado que o módulo nRF24L01+ possui oportunidades de melhoria, uma vez que foi observado que o módulo nRF24L01+ consumia energia mesmo em modo *sleep* devido a uma constante corrente de fuga. Outro ponto negativo se da pelo fato do módulo nRF24L01+ não ser programável, sendo assim necessário o uso de um microcontrolador externo, o que leva a um aumento no consumo energético do nó.

Os autores chegaram a conclusão de que em questões de velocidade de comunicação e custos de *hardware*, seguramente o módulo nRF24L01+ é a melhor escolha. O alto custo do módulo XBee pode muitas vezes inviabilizar projetos de rede de sensores sem fio com grandes quantidades de nós.

Com o levantamento bibliográfico realizado foi possível observar os componentes mais utilizados nas diversas aplicações, seus resultados experimentais e por fim definir para este trabalho quais os componentes a serem utilizados. Também foi possível observar a integração e teste das unidades de processamento, comunicação, sensoriamento e energia.

Sendo assim, para este projeto, foi escolhido o uso do módulo de comunicação sem fio nRF24L01+. Para o coordenador da rede, foi escolhido o *Raspberry Pi* 3 e para os nós sensores foi considerado o uso do Arduino. Todas os componentes descritos anteriormente foram selecionados para o uso pois possuem baixo custo, se mostraram flexíveis, de fácil instalação e manuseio e viabilidade para o escopo deste projeto.



# 3 Componentes de *Hardware/Software* Utilizados e Topologia do Sistema de Rede Implementada

## 3.1 Arduino

De acordo com ([ARDUINO, 2019](#)), o Arduino é uma plataforma *open-source* de prototipagem eletrônica e programável mais utilizada no mundo para a construção de projetos eletrônicos. Originalmente criada em um projeto de pesquisa por Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis no Instituto de *Design* de Interação de Ivrea em meados de 2000 se baseou no projeto *Processing*, uma linguagem para aprendizado de programação no contexto de artes visuais desenvolvidos por Casey Reas e Ben Fry ([ARDUINO, 2019](#)).

A primeira versão do Arduino surgiu em 2005, com o intuito de possibilitar que estudantes de *Design*, que não tinham conhecimentos prévios em programação, pudessem desenvolver protótipos que ligassem o mundo digital ao real. E desde então, a plataforma se tornou a mais popular e utilizada para prototipação eletrônica ao redor do mundo.

E por ser de código e *hardware* livres, contribuições são feitas através de milhares de usuários ao redor do mundo. Usuários contribuem de forma ativa na depuração de códigos, criações de bibliotecas, criação de melhorias, projetos, tutoriais, etc. Devido ao amplo suporte da comunidade e principalmente ao seu baixo custo, esta plataforma foi escolhida para ser utilizada para realizar as leituras dos sensores.

Desde o surgimento do primeiro Arduino, muitas placas de desenvolvimento foram lançadas no mercado. Para este trabalho, os seguintes modelos foram selecionados para o desenvolvimento e podem ser observadas nas [Figura 9](#), [Figura 11](#) e [Figura 10](#):

- Arduino Uno Rev3
- Arduino Due
- Arduino Nano

Famílias de placas diferentes foram propositalmente considerados para demonstrar a modularidade do código desenvolvido para os nós sensores. Independentemente do Arduino, as bibliotecas utilizadas e o código desenvolvido são totalmente compatíveis

entre as famílias. O desenvolvimento de um código reutilizável e modular exerce um papel fundamental para a manutenção e operação de uma RSSF.

### 3.1.1 Arduino Uno Rev3

Figura 9 – Arduino Uno Rev3



Fonte: [Website oficial Arduino](#)

As demais configurações do *Hardware* do Arduino Uno utilizado podem ser observadas na tabela abaixo:

Tabela 2 – Especificações técnicas do Arduino UNO Rev3

<b>Microcontrolador</b>	ATmega328P
<b>Tensão de operação</b>	5V
<b>Portas digitais E/S</b>	14 (6 portas com saída PWM)
<b>Portas PWM E/S</b>	6
<b>Portas analógicas E/S</b>	6
<b>Memória flash</b>	32KB (0.5KB utilizado pelo bootloader)
<b>Memória EEPROM</b>	1KB
<b>Velocidade de clock</b>	16MHz
<b>Consumo energético</b>	45mA

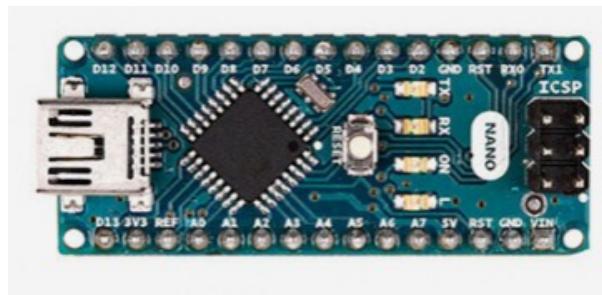
Os seguintes materiais foram utilizados para configurar, executar o desenvolvimento e testes do nó sensor da rede:

1. Arduino Uno Rev3;
2. 1x *Protoboard*;
3. *Jumpers* macho-fêmea e macho-macho;
4. Cabo USB A/B para carregamento do código;

5. Módulo nRF24L01+ para comunicação com Arduino Due;
6. Módulo sensor DHT11

### 3.1.2 Arduino Nano

Figura 10 – Arduino Nano



Fonte: [Website oficial Arduino](#)

As demais configurações do *Hardware* Arduino Nano utilizado podem ser observadas na tabela abaixo:

Tabela 3 – Especificações técnicas do Arduino Nano

<b>Microcontrolador</b>	ATmega328
<b>Tensão de operação</b>	5V
<b>Portas digitais E/S</b>	22 (6 portas com saída PWM)
<b>Portas PWM entrada</b>	6
<b>Portas analógicas E/S</b>	8
<b>Memória flash</b>	32KB (2KB utilizado pelo bootloader)
<b>Memória EEPROM</b>	1KB
<b>Velocidade de clock</b>	16MHz
<b>Consumo energético</b>	19mA

1. Arduino Nano;
2. 1x *Protoboard*;
3. *Jumpers* macho-fêmea e macho-macho;
4. Cabo micro USB para carregamento do código;
5. Módulo nRF24L01+ para comunicação com Arduino Due;
6. Módulo sensor DHT11

### 3.1.3 Arduino Due

A placa Arduino Due possui uma configuração mais robusta em relação as placas apresentadas anteriormente. Ela possui um microcontrolador AT91SAM3X8E de 32 bits, opera com tensão de 3.3V ao contrário de 5V apresentados pelas placas anteriores, possui 54 portas digitais de entrada e saída, memória EEPROM de 1KB, *clock* superior com 84MHz, porém possui a desvantagem de possuir um consumo energético muito maior em relação aos outros modelos.

Por possuir maior poder de processamento devido a sua velocidade superior de *clock* e arquitetura de 32bits, este modelo foi definido como o responsável por receber os dados dos nós sensores vindos do Arduino Uno Rev3 e Nano e fazer a interface com o computador *Raspberry Pi 3*.

Figura 11 – Arduino Due



Fonte: [Website oficial Arduino](#)

As demais configurações do *Hardware* Arduino Due utilizado podem ser observadas na tabela abaixo:

Tabela 4 – Especificações técnicas do Arduino Due

<b>Microcontrolador</b>	AT91SAM3X8E
<b>Tensão de operação</b>	3.3V
<b>Portas digitais E/S</b>	54 (12 portas com saída PWM)
<b>Portas PWM entrada</b>	6
<b>Portas analógicas entrada</b>	12
<b>Portas analógicas saída</b>	2
<b>Memória flash</b>	512KB (2KB utilizado pelo bootloader)
<b>Memória EEPROM</b>	1KB
<b>Velocidade de clock</b>	84MHz
<b>Consumo energético</b>	80mA

### 3.2 Raspberry Pi 3 modelo B

O computador *Raspberry Pi*, nome dado a série de computadores feitos pela *Raspberry Pi Foundation*, fundação com sede no Reino Unido tem como um dos principais objetivos educar e fornecer acessibilidade da computação para a população ([FOUNDATION, 2019](#)).

O primeiro *Raspberry Pi* foi lançado em 2012 e possuía um processador *single core* de 700MHz e 256MB de memória RAM. O modelo mais atual no mercado é o *Raspberry Pi 4*, que na configuração mais robusta, possui 4GB de memória RAM e um processador *Quad core Cortex-A72 (ARM v8)* de *64bits*.

Por conta do seu baixo custo, encontrado por cerca de 35 dólares, este computador tem sido utilizado cada vez mais em escolas, casas, indústrias, comércios, etc. Seja para o ensino à programação, automação residencial, monitoramentos, uso pessoal, o *Raspberry Pi* se mostrou uma forma eficiente e barata para atender necessidades gerais.

O *Raspberry Pi 3* modelo B, representado na [Figura 12](#), foi escolhido como o Nô Coordenador da rede por conta de ser uma plataforma *open source*, possuir alto poder de processamento, baixo custo, presença de interfaces GPIOs, comunicação com a rede de *Internet* e compatibilidade com diversas aplicações por conta de ser um computador.

Figura 12 – *Raspberry Pi 3* modelo B



Fonte: [Website oficial Raspberry](#)

Demais configurações do *Hardware Raspberry Pi 3* utilizado podem ser observadas na tabela abaixo:

Tabela 5 – Especificações técnicas do *Raspberry Pi* 3 modelo B

<b>Processador</b>	<i>Quad Core Broadcom BCM2837 64bit</i>
<b>Tensão de operação</b>	5V @ 2A
<b>GPIOs</b>	40
<b>Memória RAM</b>	1GB
<b>Wireless LAN and Bluetooth</b>	BCM43438
<b>Portas USB 2.0</b>	4
<b>Saída de Vídeo</b>	HDMI
<b>Velocidade de clock</b>	1GHz
<b>Consumo energético</b>	350mA

Os seguintes itens foram utilizados para configurar, executar o desenvolvimento e testes do nó Coordenador da rede:

1. *Raspberry Pi* 3 modelo B;
2. Cartão micro SD com o Sistema Operacional [NOOBS](#) instalado;
3. Fonte micro USB de 2.1A;
4. Cabo HDMI para ligar ao monitor;
5. *Mouse* e teclado;
6. Conexão com a *Internet*;
7. Módulo nRF24L01+ para comunicação com Arduino Due;
8. Módulo sensor DHT11

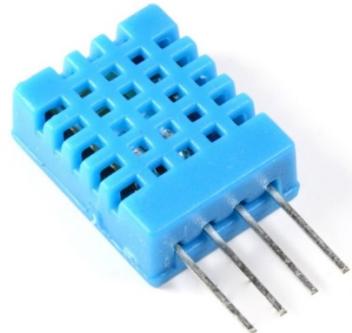
### 3.3 Sensor DHT11

Todos os nós da rede, com exceção do nó coordenador, utilizam o sensor DHT11 fabricado pela empresa Aosong ([AOSONG, 2019](#)). Este sensor fornece, através de sinais digitais, medições da temperatura e umidade relativa do ambiente.

De acordo com o fabricante, o sensor de umidade possui resolução de *16 bits* e precisão de  $\pm 5\%$  a  $25^{\circ}\text{C}$ . O sensor de temperatura também possui resolução de *16 bits*, porém uma taxa de precisão de  $\pm 2\%$  a  $25^{\circ}\text{C}$ .

De acordo com o fabricante, as principais características do sensor são: baixo custo, alta durabilidade, estabilidade, baixa suscetibilidade a ruídos e possibilidade de calibração do sensor através do *software*.

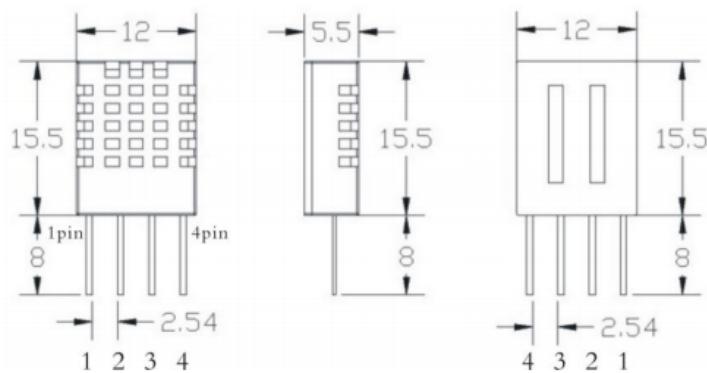
Figura 13 – Sensor de Umidade e Temperatura - DHT11

Fonte: [Website Filipeflop](#)

As leituras de temperatura são realizadas através de um termistor NTC e as leituras de umidade através de uma unidade capacitiva. Dentro do módulo DHT11, existe um microcontrolador responsável por realizar as medições e transmissão dos dados através da saída serial emulada por *software*.

A configuração da pinagem do sensor pode ser observada na [Figura 14](#).

Figura 14 – Pinagem do sensor DHT11

Fonte: [Datasheet](#)

O sensor pode ser alimentado com tensões de 3.5 a 5.5V. A troca de dados ocorre através de um único barramento bidirecional de sinal, ou seja, as informações são lidas entre o sensor e o microcontrolador e depois trafegadas através da porta serial.

As configurações detalhadas das ligações do sensor podem ser observadas na tabela abaixo:

Tabela 6 – Ligações físicas do sensor

Pino	Descrição
1	Alimentação 3.5 ~5.5V
2	Dados
3	Não conetado
4	Terra

É sugerido pelo fabricante três diferentes configurações para a garantia de um bom funcionamento nas leituras do sensor:

1. Uso de cabo de até 20cm e um resistor *pull-up* de 5.1k *ohms* na saída digital do sensor;
2. Ao utilizar alimentação de 3.5V, não é recomendado utilizar cabos maiores que 20cm por conta de problemas com a fonte de alimentação;
3. Não é recomendado que a frequência de leitura ocorra em um intervalo menor que 5 segundos.

### 3.4 Comunicação Sem Fio

Para a comunicação via rádio sem fio, o módulo transceptor nRF24L01+ foi escolhido por ser uma opção de baixo custo, confiável e baixo consumo energético, atendendo assim os requisitos deste projeto. Este módulo é capaz de receber e transmitir informações por ondas de rádio na frequência homologada de 2.4GHz.

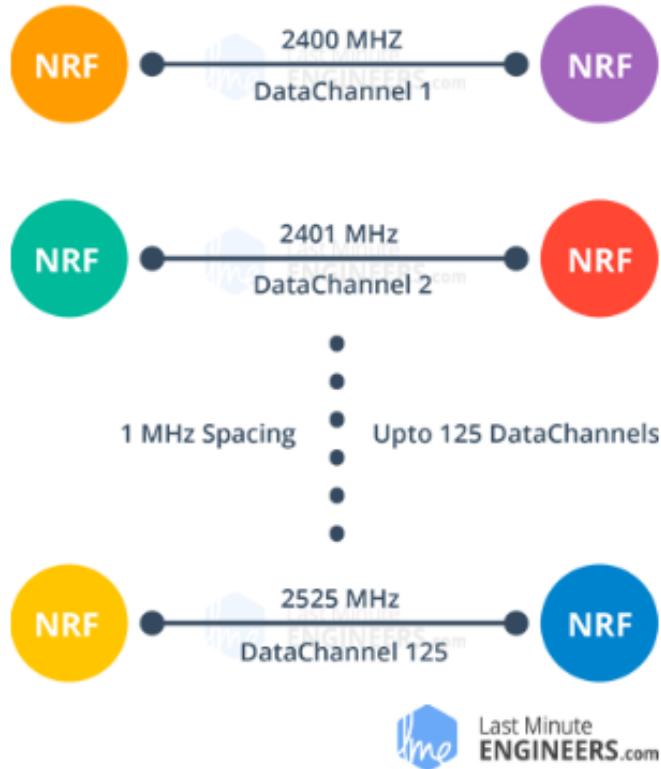
A confiabilidade na comunicação é aumentada através de mecanismos de *software* que utilizam mecanismos de validação de pacotes e mensagens de *acknowledgement* de recepção.

A comunicação entre o controlador e o módulo ocorre através da interface de 4 pinos SPI (*Serial Peripheral Interface*) com uma taxa máxima de comunicação de 10Mbps. Através desta interface, é possível realizar configurações para selecionar a frequência do canal (125 canais disponíveis), a potência de saída (0dBm, -6dBm, -12dBm e -18dBm) e a taxa de transmissão (250kbps, 1Mbps, e 2Mbps).

O parâmetro de frequência pode ser útil para selecionar um canal com menor uso, evitando assim interferências por outros dispositivos dentro do alcance que utilizam a mesma frequência de banda de 2.4GHz. Esta frequência de 2.4GHz nos permite utilizar 125 diferentes canais, que variam entre 2.400 a 2.525GHz com intervalos de 1MHz, como pode ser visto na [15](#).

O parâmetro de potência de saída, como o nome diz, configura a potência de saída do rádio. De acordo com o fabricante, com a potência de saída configurada como 0 dBm, o consumo para uma transmissão a 2000Kbps é de 11.3mA. Além disso, quanto maior a

Figura 15 – Diferentes canais da banda 2.4GHz



Fonte: <<https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/>>

potência de saída fornecida, maior o alcance da transmissão, porém maior será o consumo energético. Portanto, deve-se utilizar este parâmetro com cautela e equilíbrio dependendo de cada cenário. A Tabela 7 apresenta as possíveis configurações de potência de saída e o seu dado consumo energético.

Tabela 7 – Potência de saída vs consumo energético

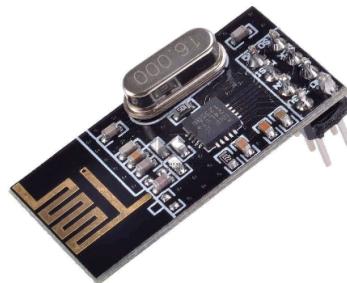
Potência de saída	Consumo energético
0dBm	11,3mA
-6dBm	9.0mA
-12dBm	7,5mA
-18dBm	7.0mA

Por último, a taxa de transmissão é configurável entre os valores de 250Kbps, 1Mbps, e 2Mbps. Para este projeto, a taxa de transmissão selecionada foi de 250Kbps, o que é mais que suficiente para transmissões pequenas e curtas de pacotes com informações a respeito da temperatura e umidade.

Para este projeto, foram selecionados dois diferentes módulos de comunicação nRF24L01+ comumente encontrados no mercado, sendo detalhados nos próximos parágrafos.

O primeiro módulo, representado na [Figura 16](#), utiliza uma antena integrada na própria PCB (Placa de Circuito Impresso). Porém, devido ao tipo da antena ser interna, o alcance do rádio é menor que o módulo que utiliza uma antena externa. Segundo o *datasheet* do fabricante, o alcance para essa antena em um ambiente aberto, livre de interferências é de aproximadamente 100 metros. Em ambientes internos com obstruções como paredes e interferência de sinais, o alcance da antena será reduzido. O custo atual desse módulo no Brasil é de aproximadamente R\$ 8,00.

Figura 16 – Módulo nRF24L01



Fonte: <<https://www.curtocircuito.com.br/modulo-wireless-2-4g-nrf24l01.html?>>

Já o segundo módulo, representado pela [Figura 17](#), utiliza uma antena externa que é ligada através de um conector RF coaxial padrão SMA (*SubMiniature version A*).

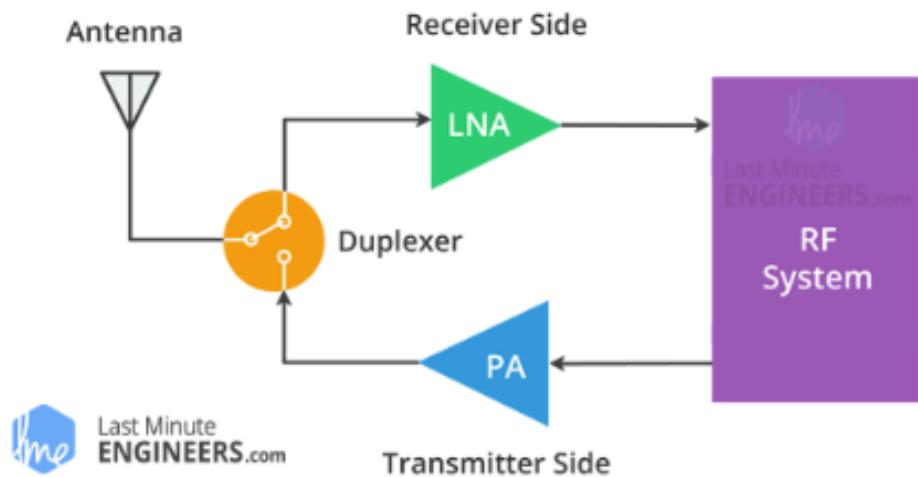
Figura 17 – Módulo nRF24L01+ com antena externa



Fonte: <<https://www.curtocircuito.com.br/modulo-wireless-2-4g-nrf24l01-antena.html?>>

Este circuito, representado pela [Figura 18](#), possui integrado um Amplificador de Potência (PA - *Power Amplifier*) e um Amplificador de Baixo Ruído (LPA - *Low-Noise Power Amplifier*). O Amplificador de Potência é responsável por amplificar a potência do sinal transmitido pelo rádio (1mV para 0.5 a 1V). Já o Amplificador de Baixo Ruído é responsável por amplificar um sinal baixo recebido pela antena.

Figura 18 – Amplificador de ruído e potência



Fonte: <<https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/>>

Segundo o *datasheet* do fabricante, o alcance para essa antena em um ambiente aberto e livre de interferências é de aproximadamente 1000 metros, o que é aproximadamente 10x maior que o módulo com antena integrada. O custo atual desse módulo com a antena externa no Brasil é de aproximadamente R\$ 22,00.

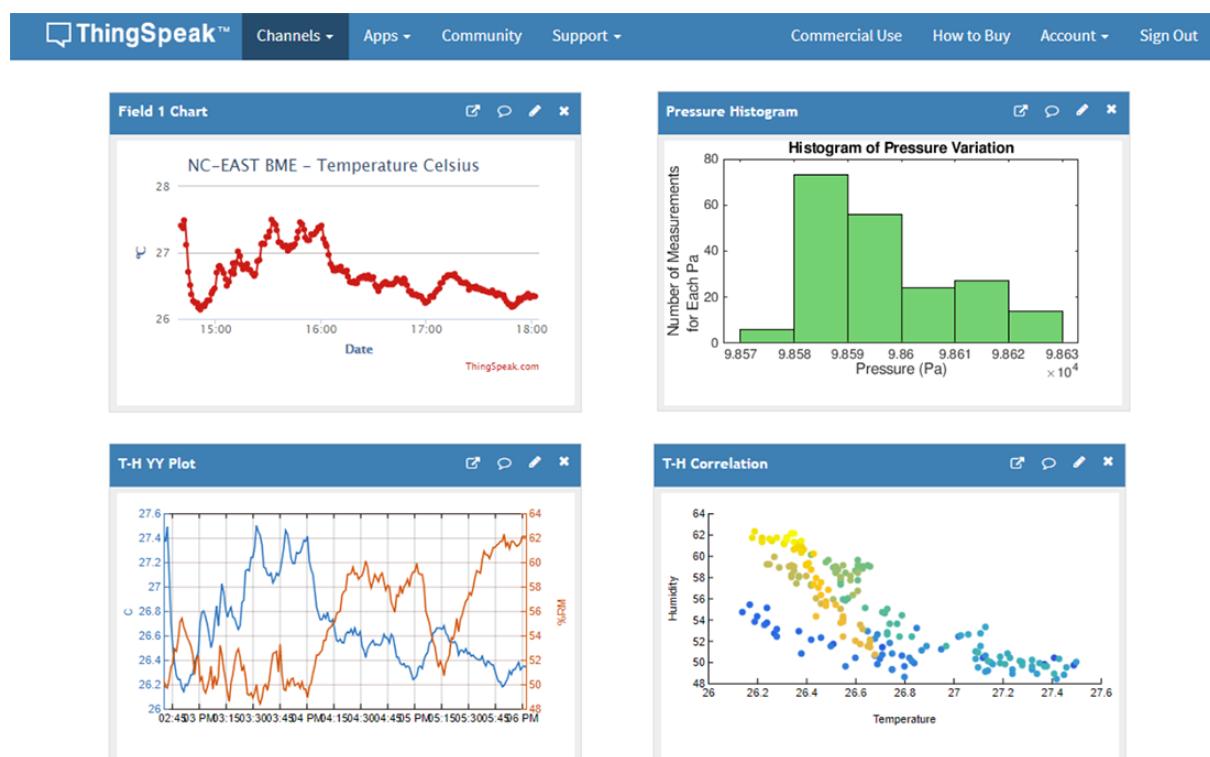
### 3.5 ThingSpeak

O *ThingSpeak* é uma plataforma *cloud* de IoT (*Internet of Things*) que permite que o usuário agregue, visualize e analise seus dados em tempo real, como pode ser visto na [Figura 19](#). Para este trabalho esta plataforma foi selecionada por ser uma plataforma gratuita, de fácil integração com diversos sistemas e por ter documentação e suporte ativo da comunidade.

Apesar da plataforma fornecer suporte para análise de dados em tempo real utilizando o *software MATLAB®*, neste projeto, a plataforma será utilizada somente para a visualização e agregação dos dados em tempo real.

O cadastro gratuito para uso da plataforma pode ser realizado através deste endereço: <<https://thingspeak.com/>>.

Figura 19 – Exemplo de um *dashboard*



Fonte: <<http://www.thingspeak.com>>

### 3.6 Telegram Chatbot

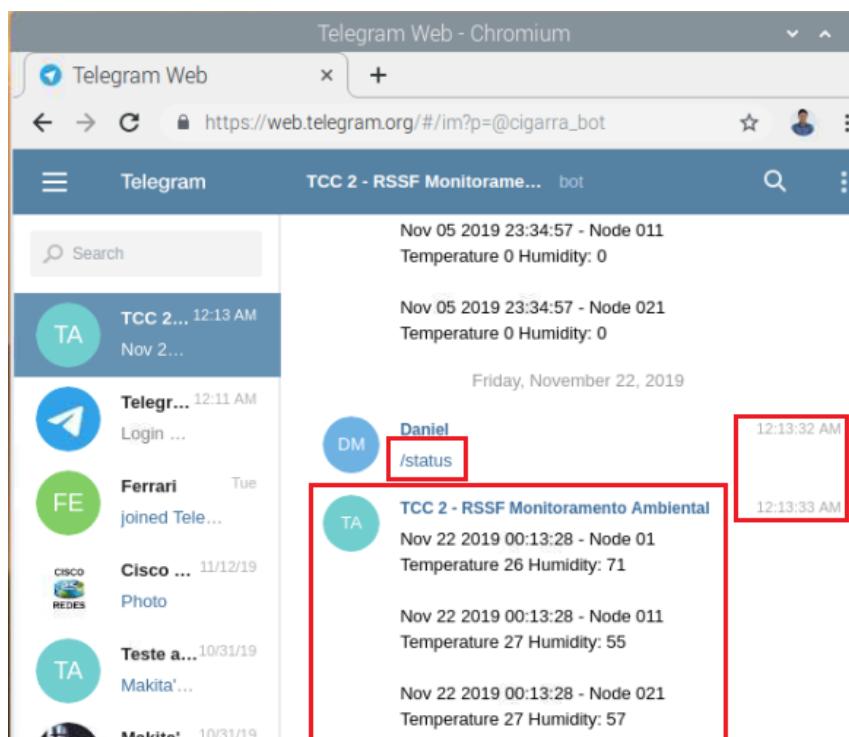
Com o intuito de auxiliar na rotina de testes e instalações de novos nós sensores na rede, foi criado um *bot* no aplicativo de mensagens *Telegram* para que o técnico responsável pelas instalações em campo consiga verificar e validar suas atividades.

Qualquer usuário que tenha o aplicativo instalado no celular também poderá interagir com o *bot* a fim de se obter as leituras mais recentes de todos os nós sensores.

A responsabilidade de receber os comandos e encaminhar os dados aos usuários será do nó coordenador da rede através do auxílio da API do *Telegram*. Esta API é gratuita e a documentação está disponibilizada no *site* oficial do aplicativo em: <<https://core.telegram.org/>>.

Para este projeto, foi criado um *bot* dentro de um grupo de conversa. Todos os usuários que estiverem dentro do grupo poderão interagir com o *bot* enviando o comando `/status`. Este comando irá solicitar ao nó coordenador que ele retorne a última medida de temperatura e umidade de todos os nós sensores da rede, como pode ser observado na Figura 20.

Figura 20 – *Bot* criado para o projeto



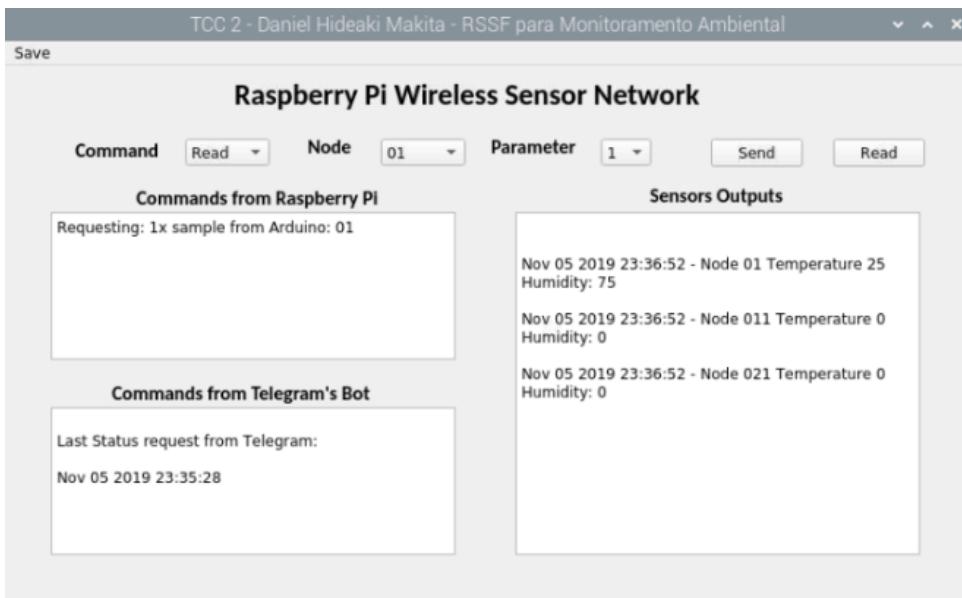
Fonte: O Autor

### 3.7 Interface Gráfica desenvolvida com *Qt Designer*

Com o intuito de tornar o gerenciamento dos sensores mais prático e intuitivo, foi criado, através do auxílio do *software Qt Designer*, uma interface gráfica para o usuário.

Essa interface gráfica, representada na [Figura 21](#), permite o envio de comandos para os nós sensores e a visualização dos dados em cada nó da rede.

Figura 21 – Interface Gráfica para gerenciamento



Fonte: O Autor

Logo abaixo do título *Raspberry Pi Wireless Sensor Network* estão as opções de seleção para enviar comandos para todos os nós disponíveis da rede de sensores.

O primeiro campo, nomeado como *Command* permite selecionar um dentre três comandos possíveis: *Slave*, *Master* e *Read*.

O comando *Slave* configura o nó especificado em *Node* para entrar no modo passivo. Neste modo, as leituras são retornadas somente através de requisições vindas do mestre (nó coordenador). Para o comando *Slave*, o argumento *Parameter* não é utilizado.

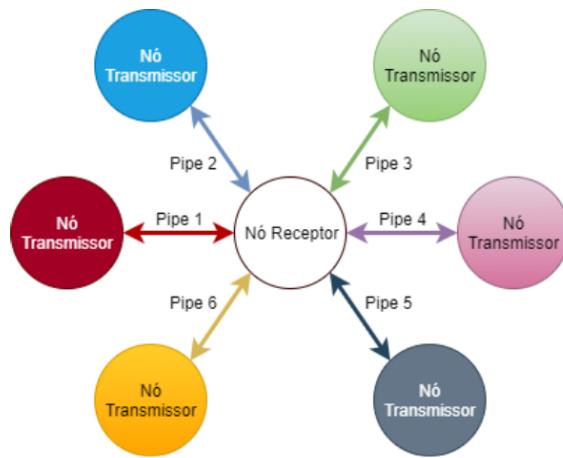
Já o comando *Master*, configura o nó especificado em *Node* para entrar no modo ativo. Neste modo, as leituras são enviadas periodicamente para o nó coordenador. Para o comando *Master*, o argumento definido em *Parameter* não é utilizado.

Por fim, o comando *Read* retorna 'n' leituras, sendo 'n' a quantidade de leituras a serem realizadas, especificadas no campo *Parameter*, do nó sensor especificado em *Node*.

## 3.8 Topologia da Rede

Durante o desenvolvimento do projeto, foi notada uma limitação existente na biblioteca original desenvolvida pelo usuário “*maniacbug*”, disponível em <<https://github.com/maniacbug/RF24>>. Tal limitação se refere ao número restrito de 6 registradores que armazenam os endereços para comunicação, descritos como *pipes*. A Figura 22 representa a topologia com o maior número de nós possíveis em uma rede utilizando esta biblioteca.

Figura 22 – Topologia exemplo maniacbug

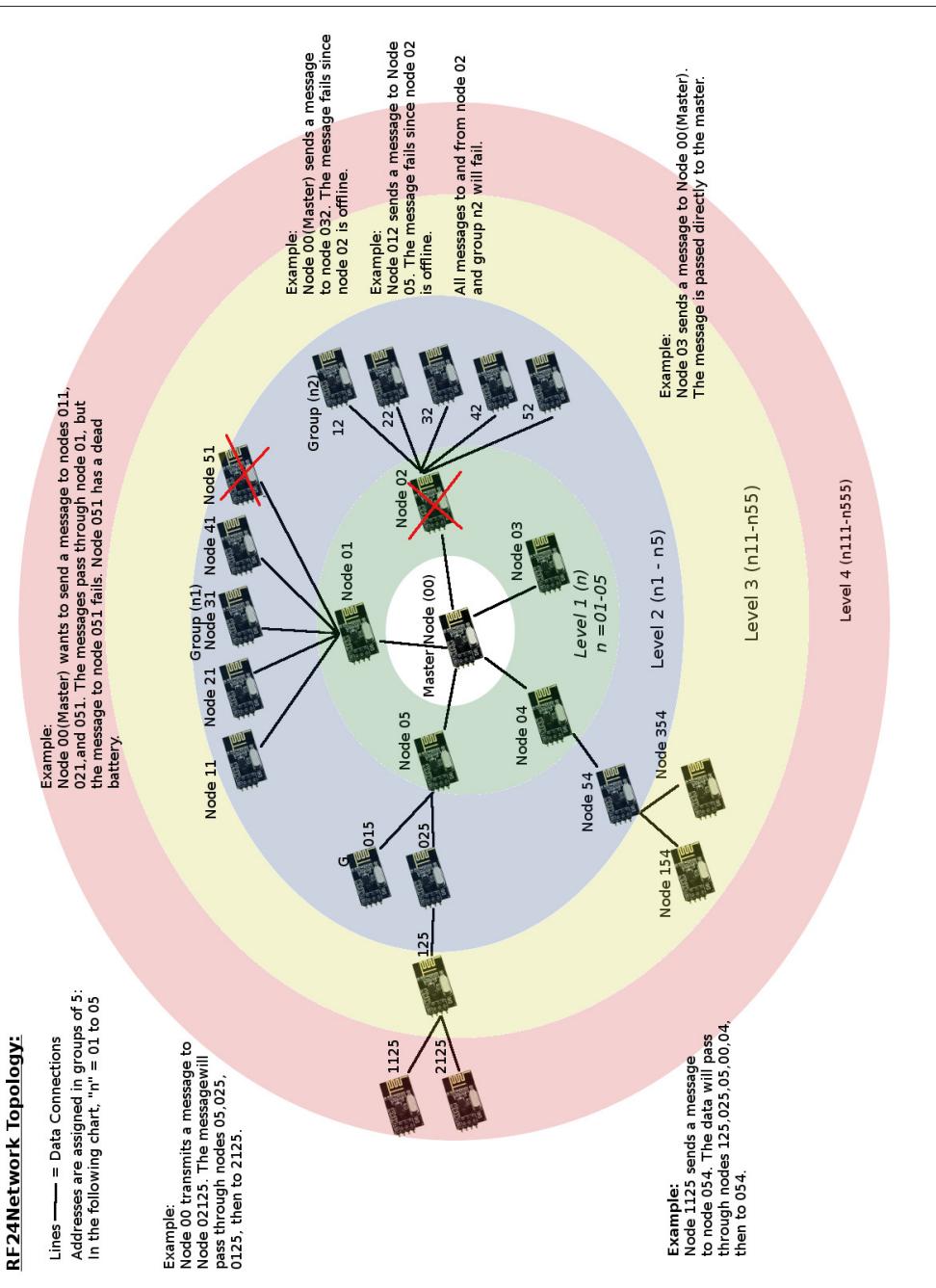


Fonte: O Autor

Uma solução para contornar essa limitação foi criada pelo autor nomeado como "trmh20". A solução apresentada por este usuário é um *fork* da biblioteca original desenvolvida pelo autor nomeado "maniacbug", porém existe a implementação de uma funcionalidade para roteamento de pacotes entre os nós, o que possibilita que mais nós sejam utilizados na rede. A biblioteca está disponível e pode ser consultada em: <<https://trmh20.github.io/RF24Network/>>.

Um exemplo de uma topologia apresentada pelo criador da biblioteca pode ser vista na Figura 23. Nela é possível verificar que é formada uma estrutura em árvore entre os nós da rede de sensores. Devido a limitação de até 6 conexões simultâneas por nó, cada nó, com exceção do coordenador, deverá ter um pai e de 0 a 5 filhos. Já o nó coordenador não possui pai e tem de 0 a 5 filhos. A profundidade máxima permitida é de 5, ou seja, existe a limitação da quantidade máxima de 781 nós na rede.

Figura 23 – Topologia exemplo trmh20



Fonte: <<https://tmrh20.github.io/RF24Network/Tuning.html>>

O endereço de cada nó deverá ser único e definido pelo administrador da rede dentro do código a ser embarcado nos microcontroladores. Este endereço especifica exatamente em que nível o nó está na árvore. O endereço possui um tamanho de 15 bits e possui formato octal, onde cada dígito representa um nível do nó coordenador. Um exemplo de endereçamento proposto pelo desenvolvedor pode ser observado na Tabela 8.

Tabela 8 – Endereçamento dos Nós

Nó Coordenador	Endereçamento							
			00			00		
Nível 1			01			04		
Nível 2		011		021			014	
Nível 3	111			121	221			114
Nível 4					12121		1114	2114
								3114

O roteamento de pacotes através de nós da rede é feito de forma invisível ao usuário, ou seja, a biblioteca realiza toda a parte de cálculo e roteamento entre os nós para que a informação chegue ao nó desejado, tudo isso é feito de forma implícita através do comando *update*.

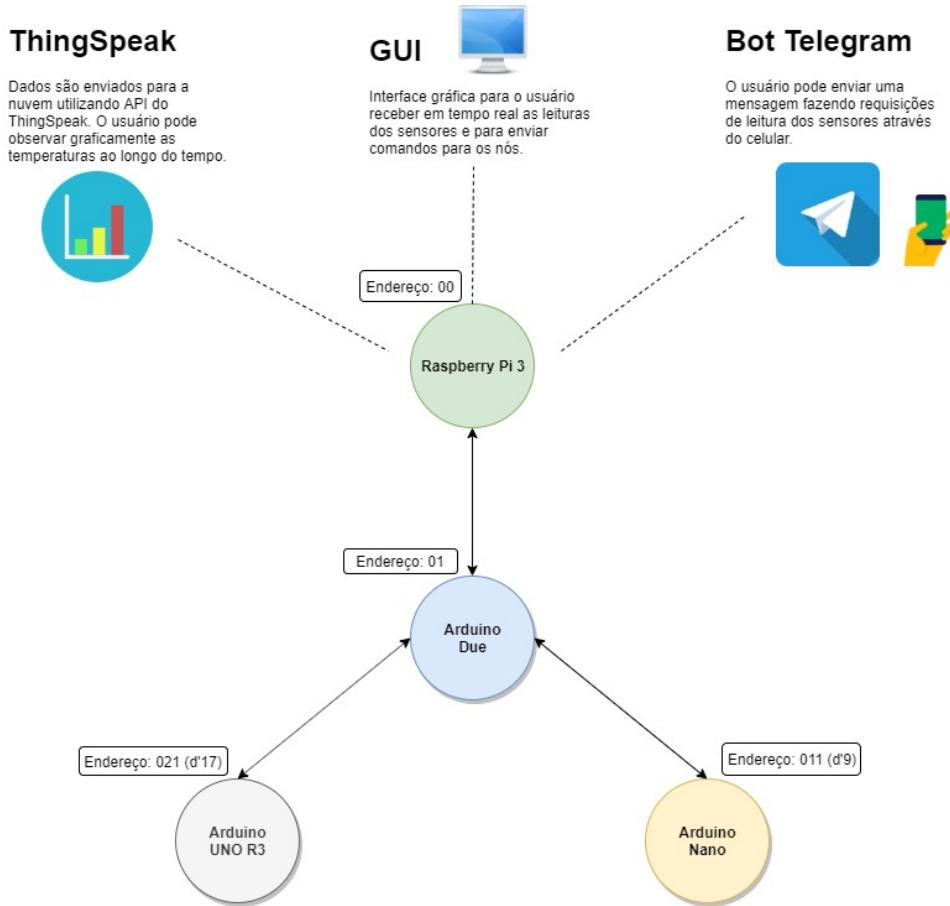
Caso o roteamento do pacote seja executado com sucesso, um sinal de *ACK* é enviado do nó destino ao nó emissor. Caso contrário, um sinal de *timeout* é emitido ao nó emissor.

A grande vantagem em se utilizar roteamento entre nós é no aumento da abrangência de área de sensoriamento. Sem o roteamento implementado, estariamos limitados ao alcance do rádio entre o nó coordenador e sensor por conta da limitação de *hardware* do rádio, que possui 6 registradores para armazenar endereços para a comunicação com rádios diferentes.

A topologia desenvolvida para este projeto pode ser observada na [Figura 24](#). Nela é possível conferir os nós sensores, o nó coordenador, seus respectivos *hardwares* e endereços definidos durante o desenvolvimento da plataforma.

Além da topologia, o esquemático da [Figura 24](#) conta com funcionalidades adicionais criadas para o gerenciamento e validação da rede. Como podemos observar, além da responsabilidade de enviar comandos e receber dados dos sensores, o nó coordenador é responsável por enviar os dados dos sensores para a plataforma *ThingSpeak*, para que os dados possam ser visualizados de forma gráfica e em tempo real. Também é de responsabilidade do nó coordenador executar uma interface gráfica que facilite o gerenciamento da rede de sensores pelo usuário. E por fim, também é de responsabilidade do nó coordenador a interface entre a API do *Telegram* para que o usuário consiga receber dados dos sensores através do seu próprio *smartphone*.

Figura 24 – Topologia do projeto

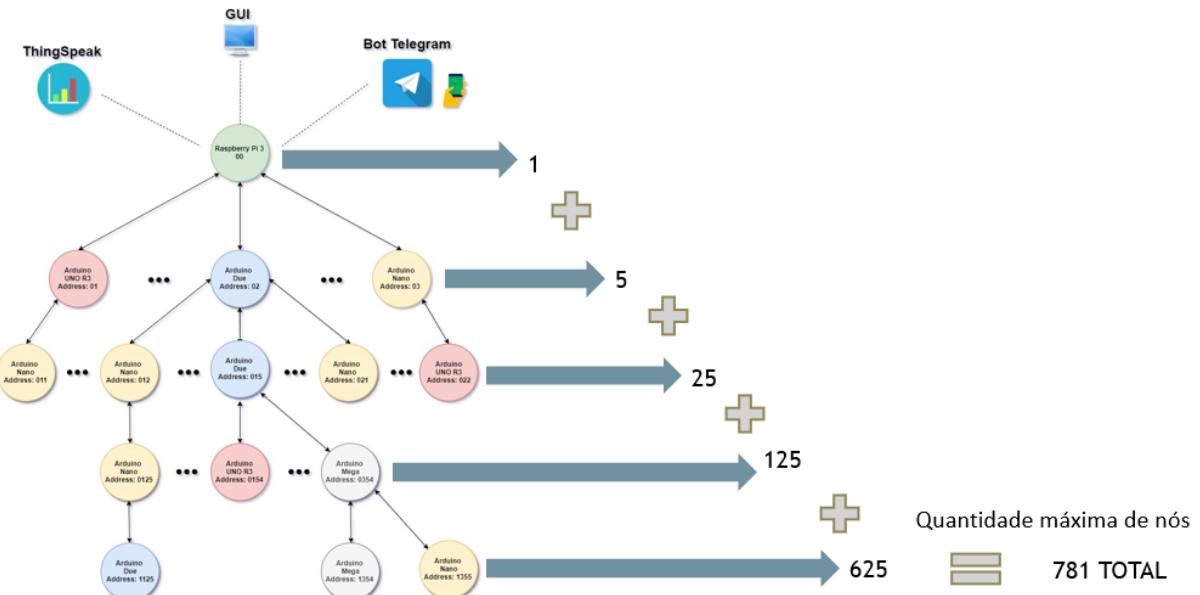


Fonte: O Autor

A Figura 25 foi criada com o intuito de representar uma possível expansão da topologia da rede com a inclusão de nós sensores adicionais. Isto nos mostra a escalabilidade da plataforma através do uso de roteamento de pacotes entre os nós, podendo chegar a 781 nós no total devido ao limite no endereçamento que é definido entre 3 a 5 bytes na biblioteca do *software*.

O primeiro nível da árvore sempre possuirá 1 nó por ser o nível do nó coordenador da rede. O segundo nível pode possuir entre 0 a 5 nós sensores. O terceiro nível pode possuir entre 0 a 25 nós sensores. O quarto nível pode possuir entre 0 a 125 nós sensores. Já o quinto nível, pode possuir entre 0 a 525 nós sensores.

Figura 25 – Exemplo de topologia possível



Fonte: O Autor



# 4 Desenvolvimento da Plataforma

## 4.1 Ligação Física dos Componentes

Nesta seção, serão apresentadas as ligações físicas entre todos os componentes dos nós sensores e nó coordenador. As ligações físicas estão representadas pelas tabelas abaixo.

Os esquemáticos eletrônicos foram criados através do *software* Fritzing, disponível para *download* em: <<https://fritzing.org/>>.

A [Tabela 9](#) representa a configuração em que as ligações físicas entre o *Raspberry Pi* e módulo nRF24L01+ foram feitas. A [Figura 26](#) representa o esquemático criado através do *software* Fritzing. Já a [Figura 27](#) representa uma imagem real do nó coordenador na bancada de testes.

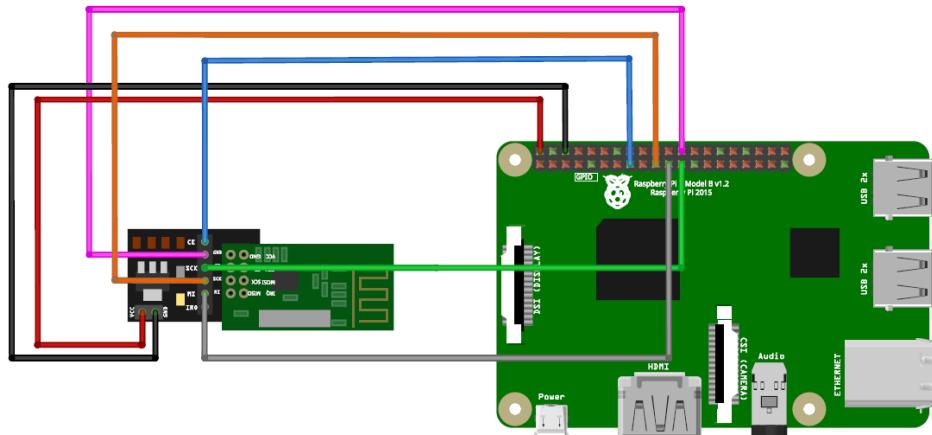
O módulo adaptador possui seu próprio regulador de tensão de 3,3 volts e um conjunto de capacitores de filtro, para que você possa alimentá-lo com uma fonte de alimentação de 5 volts.

Devido a sensibilidade do rádio por conta de variações de energia, é interessante que se utilize este adaptador para evitar perdas no desempenho do rádio.

Tabela 9 – Ligação *Raspberry Pi* e módulo nRF24L01+

<b>Raspberry Pi</b>		<b>Rádio</b>	
<b>GPIO</b>	<b>#Pino</b>	<b>Adaptador</b>	<b>nRF24L01+</b>
GPIO 11 (SPI0 CLK)	23	CLK	CLK
GPIO 9 (SPI0 MISO)	21	MI	MI
GPIO 10 (SPI0 MOSI)	19	MO	MO
GPIO 8 (SPI0 CS0)	24	CSN	CSN
GPIO 22	15	CE	CE
5V PWR	2	5V in	3V3 out
GND	9	GND	GND

A [Tabela 10](#) representa a configuração em que as ligações físicas entre o Arduino Uno Rev3, DHT11 e módulo nRF24L01+ foram feitas. A [Figura 28](#) representa o esquemático criado através do *software* Fritzing. Já a [Figura 29](#) representa uma imagem real do nó coordenador na bancada de testes.

Figura 26 – Esquemático *Raspberry Pi*

Fonte: O Autor

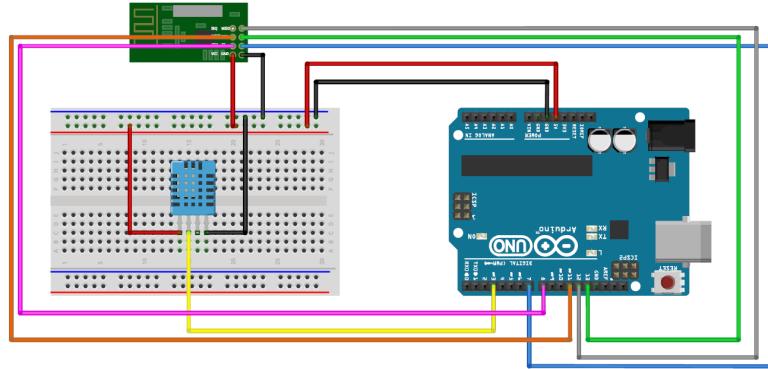
Figura 27 – Nó coordenador *Raspberry Pi*

Fonte: O Autor

Tabela 10 – Ligação Arduino Uno e módulos

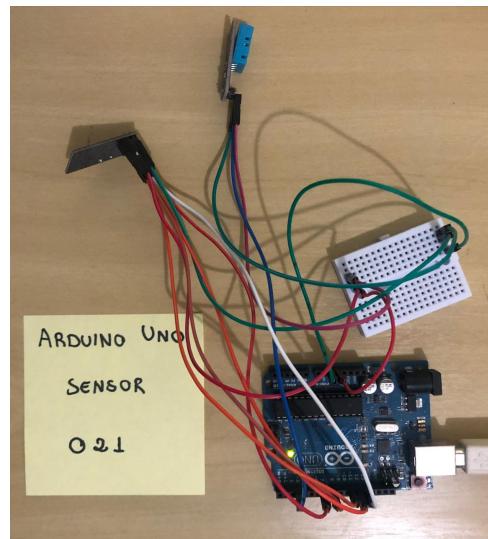
Arduino UNO	Rádio	Sensor
Portas	nRF24L01+	DHT11
D13 (SCK)	SCK	-
D12 (MISO)	MI	-
D11 (MOSI)	MO	-
D8	CSN	-
D7	CE	-
D3	-	Data
5V	5V	5V
GND	GND	GND

Figura 28 – Esquemático Arduino Uno



Fonte: O Autor

Figura 29 – Nô sensor Arduino Uno



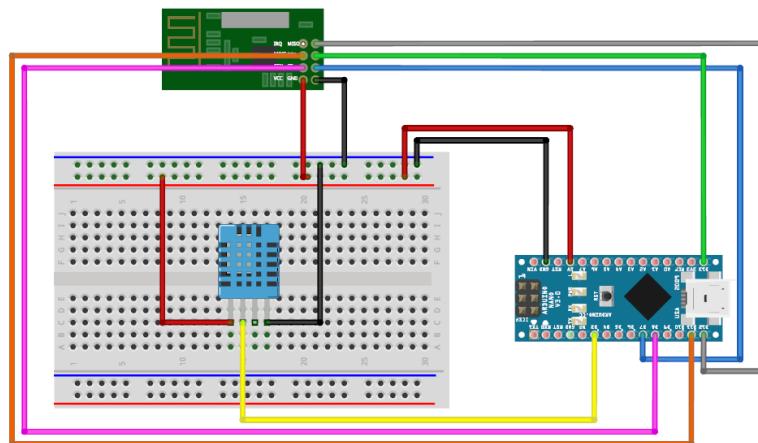
Fonte: O Autor

A [Tabela 11](#) representa a configuração em que as ligações físicas entre o Arduino Nano, DHT11 e módulo nRF24L01+ foram feitas. A [Figura 30](#) representa o esquemático criado através do *software* Fritzing. Já a [Figura 31](#) representa uma imagem real do nó coordenador na bancada de testes.

Tabela 11 – Ligação Arduino Nano e módulos

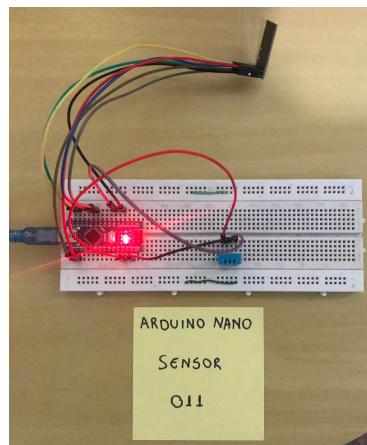
Arduino Nano Portas	Rádio nRF24L01+	Sensor DHT11
D13 (SCK)	SCK	-
D12 (MISO)	MI	-
D11 (MOSI)	MO	-
D8	CSN	-
D7	CE	-
3V3	VCC	-
D3	-	Data
5V	-	5V
GND	GND	GND

Figura 30 – Esquemático Arduino Nano



Fonte: O Autor

Figura 31 – Nó sensor Arduino Nano



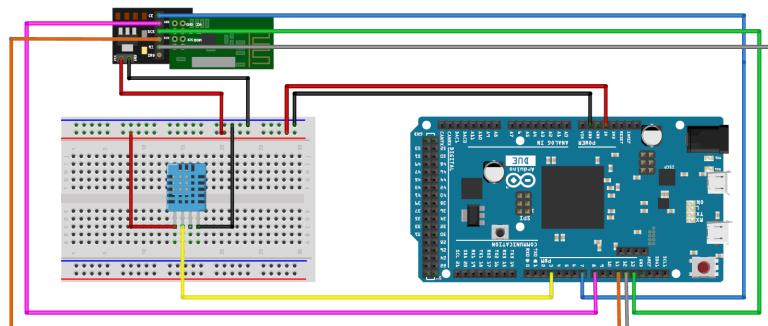
Fonte: O Autor

A Tabela 12 representa a configuração em que as ligações físicas entre o Arduino Due, DHT11 e módulo nRF24L01+ foram feitas. A Figura 32 representa o esquemático criado através do software Fritzing. Já a Figura 33 representa uma imagem real do nó coordenador na bancada de testes.

Tabela 12 – Ligação Arduino DUE e módulos

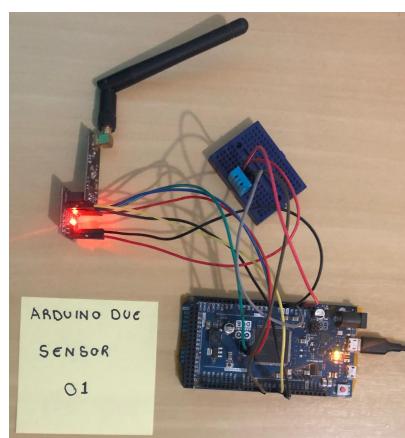
Arduino DUE Portas	Rádio		Sensor
	Adaptador	nRF24L01+	DHT11
D13 (SCK)	SCK	SCK	-
D12 (MISO)	MI	MI	-
D11 (MOSI)	MO	MO	-
D8	CSN	CSN	-
D7	CE	CE	-
D3	-	-	Data
5V	5V in	3V3 out	5V
GND	GND	GND	GND

Figura 32 – Esquemático Arduino Due



Fonte: O Autor

Figura 33 – Nó sensor Arduino Due



Fonte: O Autor

## 4.2 Configuração do Ambiente de Desenvolvimento

O passo a passo para a realização da configuração do ambiente de desenvolvimento para o correto funcionamento deste projeto está descrito no [Apêndice A](#).

## 4.3 Desenvolvimento do *Software*

Por conta da alta complexidade na integração dos nós na rede, foi necessário adotar uma estratégia de desenvolvimento incremental de componentes. Os testes eram realizados a medida que o desenvolvimento parcial dos componentes era finalizado. Ao validar todos os testes, os componentes eram então integrados, testados e validados.

Um aspecto importante que foi levado em conta para o desenvolvimento do *software* foi na reusabilidade do código desenvolvido. O código que foi desenvolvido para o Arduino Uno, Nano e Due é exatamente o mesmo, com uma exceção na linha de código em que se define o endereço do nó, que por sua vez é um endereço único em toda a rede. Graças a isso, este código pode ser utilizado em demais versões de Arduino para trabalhos futuros.

Na sequência, a subseção [4.3.1](#) contém os modos de operação que foram definidos para os nós sensores. Já as duas próximas subseções representam trechos importantes do *software* desenvolvido para o *Raspberry Pi* e Arduino, respectivamente.

Todos os códigos desenvolvidos, tanto do nó coordenador quanto dos nós sensores, são abertos e estão publicamente disponibilizados no meu *Github* em:  
[<https://github.com/makitadaniel/TCC2>](https://github.com/makitadaniel/TCC2).

### 4.3.1 Modos de Operação

Basicamente, existem dois modos de operações para os nós sensores. O primeiro modo se trata do modo escravo (passivo), onde o nó sensor deverá somente responder com os dados dos sensores através de requisições vindas do coordenador. Já o segundo modo se trata do modo mestre (ativo), onde o nó sensor deverá enviar os dados dos sensores de forma periódica, definida pelo usuário, ao coordenador.

Caso um nó seja reiniciado por qualquer motivo, ele deve retornar ao modo inicial escravo (passivo).

Os nós sensores somente devem mudar para modo mestre (ativo) através de comandos vindos do nó coordenador. Ao enviar o comando para alterar o nó para modo mestre (ativo), o nó coordenador deverá fornecer um parâmetro que representa o intervalo em segundos com que os dados serão transmitidos ao coordenador.

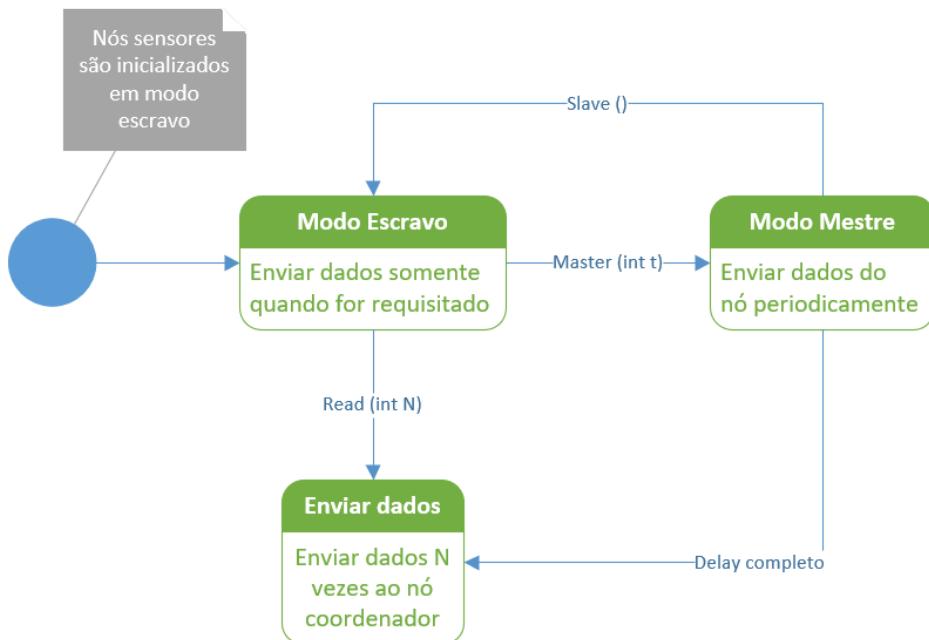
Os nós sensores somente deverão responder ao comando de leitura se estiverem no modo escravo (passivo), caso eles estejam em modo mestre (ativo), o comando não

considerado. Ao enviar o comando para realizar uma leitura do nó, o coordenador deverá fornecer a quantidade de vezes em que o dado deverá ser enviado.

Por fim, os nós deverão retornar ao modo escravo (passivo) somente quando o nó reiniciar ou quando o nó estiver em modo mestre (ativo) e receber um comando vindo do coordenador para retornar ao modo escravo (passivo). Demais combinações de comandos serão ignoradas pelo nó sensor.

As possíveis transições e os comandos válidos para cada modo podem ser observadas na [Figura 34](#).

Figura 34 – Modos de Operação



Fonte: O Autor

### 4.3.2 Raspberry Pi

Esta subseção contém explicações e os trechos mais relevantes do código desenvolvido para o nó coordenador. O código na íntegra pode ser consultado no [Apêndice B](#).

O trecho a seguir, descrito em [4.1](#), representa as bibliotecas necessárias para a execução das rotinas do nó coordenador:

```

1 # Bibliotecas auxiliares
2 import sys
3 import threading #Biblioteca que utiliza threads para executar a
                  #interface grafica juntamente com ThingSpeak e o Telegram
4 import time #Incluida para obter o log do horario

```

```

5 from struct import *
6 from datetime import datetime #Incluida para obter o log do horario
7
8
9 # Bibliotecas para NRF24
10 from RF24 import *
11 from RF24Network import *
12
13 # Biblioteca para integracao com Telegram
14 from telegram.ext import Updater, InlineQueryHandler, CommandHandler
15
16 # Biblioteca para integracao com PyQt5
17 from PyQt5 import QtCore, QtGui, QtWidgets, uic
18 from PyQt5.QtCore import QTimer
19
20 # Biblioteca pra integracao com ThingSpeak
21 from urllib.request import urlopen

```

Listing 4.1 – Bibliotecas utilizadas

O trecho a seguir, descrito em 4.2, é responsável por indicar a ligação física do rádio com o *Raspberry Pi*, a configuração da velocidade de comunicação SPI em 8MHz e a inicialização do rádio. É possível observar os endereços do nó coordenador e do nó filho serem configurados como 00 e 01, respectivamente.

```

1 # CE, CSN, Velocidade SPI
2 radio = RF24(RPI_V2_GPIO_P1_15, RPI_V2_GPIO_P1_24,
   BCM2835_SPI_SPEED_8MHZ)
3 network = RF24Network(radio)
4
5 # Endereco do no coordenador 00
6 this_node = octlit("00")
7 # Endereco do filho do coordenador 01
8 other_node = octlit("01")
9
10 # Inicializando o radio
11 radio.begin()
12 # Delay para garantir que o radio foi inicializado
13 time.sleep(0.1)
14 # Configurando canal 110
15 network.begin(110, this_node)
16 # Imprimindo detalhes do radio
17 radio.printDetails()
18 # Enderecos em decimal dos nos da rede
19 nodeAddrs = [1, 9, 17]

```

Listing 4.2 – Configuração e inicialização do rádio

Para a integração do *Raspberry Pi* com a plataforma *ThingSpeak*, utilizado para visualizar os dados de forma gráfica em tempo real, se faz necessário o trecho descrito em 4.3. Nele é possível observar a chave API utilizada e o método responsável por enviar as informações para a nuvem.

```

1 #ThingSpeak API write key 02TR9UYCP211DHN2
2 thingSpeakAPI = '02TR9UYCP211DHN2'
3 baseURL = 'https://api.thingspeak.com/update?api_key=%s' % thingSpeakAPI
4
5 # Metodo que envia os dados para o ThingSpeak
6 def thingSpeak():
7     global g_temp01, g_humid01, g_temp011, g_humid011, g_temp021,
    g_humid021
8     while True:
9         conn = urlopen(baseURL + '&field1=%s&field2=%s&field3=%s&field4
    =%s&field5=%s&field6=%s' % (g_temp01, g_humid01, g_temp011,
    g_humid011, g_temp021, g_humid021))
10        conn.close()
11        time.sleep(20)

```

Listing 4.3 – Configuração e envio de dados ao ThingSpeak

O método *receiveDecodeCommand()*, descrito em 4.4, é responsável por interpretar o comando vindo da interface gráfica e chamar os métodos responsáveis por cada função.

```

1 # Metodo interpretar comando da interface grafica
2 def receiveDecodeCommand():
3     while True:
4         command, node , parameter = map(int, input("Command and
    parameter: ").split())
5         #print('command: ', command, 'node: ', node, 'parameter: ',
    parameter)
6         node = octlit(str(node))
7
8         if command == 1 and (node in nodeAddrs): # Configurando Arduino
    como Slave
9             print('Comando para configurar Arduino: ', node, ' como
    SLAVE')
10            sendCommand(command, node, 0)
11            elif command == 2 and (node in nodeAddrs): # Configurando
    Arduino como mestre (ativo)
12                print('Comando para configurar Arduino: ', node, ' como
    mestre (ativo)')
13                sendCommand(command, node, 0)

```

```

14     elif command == 10 and (node in nodeAddrs): # Solicitando x
15         leituras do Arduino
16             print('Solicita: ', parameter, ' leituras dos sensores do
17             Arduino: ', node)
18             sendCommand(command, node, parameter)
19     elif command == 20 and (node in nodeAddrs): # Alterando a
20         frequencia de leitura do Arduino
21             print('Alterar frequencia de leitura do Arduino: ', node, ',
22             para: ', parameter, 'seg')
23             sendCommand(command, node, parameter)
24     else:
25         print("Digite um Comando ou No valido")

```

Listing 4.4 – Método para traduzir comando da interface gráfica

O trecho descrito em 4.5 representa os dois principais métodos de comunicação entre os nós. O comando *sendCommand* é responsável por enviar ao nó destino o comando vindo do usuário juntamente com um parâmetro, caso seja utilizado. Já o comando *receivePayload()* é responsável por receber os dados de temperatura e umidade vindas dos nós sensores da rede.

```

1 # Metodo criado para enviar comandos
2 def sendCommand(command, node, parameter):
3     payload = pack('<LL', command, parameter)
4     ok = network.write(RF24NetworkHeader(node), payload)
5     if ok == False :
6         print('Nao foi possivel enviar comando ao Arduino: ', node)
7
8 # Metodo para receber dados
9 def receivePayload():
10    global g_temp01, g_humid01, g_temp011, g_humid011, g_temp021,
11        g_humid021, g_timeStamp
12    while True:
13        network.update()
14        while network.available():
15            header, payload = network.read(8) # frame de 8 bytes
16            temperature, humidity = unpack('<LL', bytes(payload)) # 1 byte de
17            temperatura e 1 de umidade
18            if header.from_node != 0:
19                dateDateTimeObj = datetime.now()
20                dateObj = dateDateTimeObj.date()
21                timeObj = dateDateTimeObj.time()
22                g_timeStamp = dateObj.strftime('%b %d %Y ') + timeObj.strftime('
23                %H:%M:%S - ')
24                print( g_timeStamp + 'Received Temperature:\t', temperature, '\
25                tHumidity: ', humidity, '\tfrom Node: ', oct(header.from_node))

```

```

22
23     if header.from_node == 1:
24         g_temp01 = temperature
25         g_humid01 = humidity
26     elif header.from_node == 9:
27         g_temp011 = temperature
28         g_humid011 = humidity
29     elif header.from_node == 17:
30         g_temp021 = temperature
31         g_humid021 = humidity
32
33     g_output = str(g_timeStamp) + 'Node 01 Temperature ' + str(
34         g_temp01) + ' Humidity: ' + str(g_humid01) + '\n\n' + str(g_timeStamp)
35     + 'Node 011 Temperature ' + str(g_temp011) + ' Humidity: ' + str(
36         g_humid011) + '\n\n' + str(g_timeStamp) + 'Node 021 Temperature ' +
37     str(g_temp021) + ' Humidity: ' + str(g_humid021) + '\n\n'

```

Listing 4.5 – Métodos de envio de comando e recebimento de dados

O trecho descrito em 4.6 representa a classe criada da interface gráfica utilizada pelo usuário. Além disso, temos os *handlers* para os botões de leitura de dados e envio de comandos.

```

1 # Interface Grafica
2 class Ui(QtWidgets.QMainWindow):
3     def __init__(self):
4         # Chamando o construtor da classe
5         super(Ui, self).__init__()
6         # Carregando o arquivo fonte .ui
7         uic.loadUi('TCC.ui', self)
8
9         self.sendBtn = self.findChild(QtWidgets.QPushButton, 'sendButton')
10        self.sendBtn.clicked.connect(self.sendButtonPressed)
11
12         self.readBtn = self.findChild(QtWidgets.QPushButton, 'readButton')
13        self.readBtn.clicked.connect(self.readButtonPressed)
14
15         self.commandRaspberryTxt = self.findChild(QtWidgets.QTextBrowser,
16             'commandRaspberryText')
17         self.commandTelegramTxt = self.findChild(QtWidgets.QTextBrowser,
18             'commandTelegramText')
19         self.outputTxt = self.findChild(QtWidgets.QTextBrowser,
20             'outputText')
21
22         self.cmdBox = self.findChild(QtWidgets.QComboBox, 'commandBox')

```



```

53     app = QtWidgets.QApplication(sys.argv)
54     window = Ui()
55     app.exec_()

```

Listing 4.6 – Interface Gráfica e Handlers

Por fim, no laço principal do programa, descrito em 4.7, é possível observar a criação de *Threads* para a execução dos métodos de leitura, interface gráfica, integração com *Telegram* e *ThingSpeak*. O uso de *Threads* foi necessário por conta de performance e execução de tarefas de forma paralela e imperceptíveis ao usuário.

```

1 def main():
2     #Thread para receber comandos do usuario por linha de comando
3     #input_thread = threading.Thread(target = receiveDecodeCommand)
4     #input_thread.daemon = True
5     #input_thread.start()
6
7     #ThingSpeak Thread
8     thingSpeak_thread = threading.Thread(target = thingSpeak)
9     thingSpeak_thread.daemon = True
10    thingSpeak_thread.start()
11
12    #Thread para receber dados
13    receive_thread = threading.Thread(target = receivePayload)
14    receive_thread.daemon = True
15    receive_thread.start()
16
17    #GUI Thread
18    gui_thread = threading.Thread(target = GUI)
19    gui_thread.daemon = True
20    gui_thread.start()
21
22    #Thread do Telegram
23    updater = Updater('939667550:AAG1IpdlLnWjyjvcGQEULCCG_6AUNFfNZlg',
24    use_context = True)
25    updater.dispatcher.add_handler(CommandHandler('status', status))
26    updater.start_polling()
27
28    while True:
29        pass
30
31 if __name__ == "__main__":
32     main()

```

Listing 4.7 – Código main

### 4.3.3 Arduino

O desenvolvimento do código dos nós sensores foi feito de forma a possibilitar a reusabilidade do código para diferentes plataformas da família Arduino. Para este projeto, foram considerados Arduinos de diferentes famílias para assim validar a reusabilidade do código, evitando assim retrabalhos e facilitando a manutenção da rede de sensores, como pode ser observado de forma integral no [Apêndice C](#), [Apêndice D](#) e [Apêndice E](#).

O trecho descrito em [4.8](#) conta com todas as bibliotecas necessárias para o correto funcionamento do nó sensor.

```

1 /* Bibliotecas para o radio */
2 #include <RF24Network.h>
3 #include <RF24.h>
4 #include <SPI.h>
5
6 /* Bibliotecas para sensor DHT11 */
7 #include <Adafruit_Sensor.h>
8 #include <DHT.h>
9 #include <DHT_U.h>
10
11 /* Entrada digital onde o sensor DHT11 esta conectado */
12 #define DHTPIN 3
13 /* DHT11 */
14 #define DHTTYPE     DHT11

```

Listing 4.8 – Bibliotecas utilizadas

O trecho descrito em [4.9](#) corresponde a configuração do módulo nRF24L01+. Na linha 2 é possível observar a referência em *software* onde o módulo foi fisicamente ligado ao Arduino (Pino CE e CSN). Neste trecho também é possível observar os endereços sendo configurados de 021 para o nó sensor e 00 para o coordenador.

```

1 /* Configuracao ligacao fisica */
2 RF24 radio(7,8);
3
4 RF24Network network(radio);
5 /* ALTERAR!!! - Endereco deste no sensor */
6 const uint16_t this_node = 021;
7 /* Endereco do no coordenador */
8 const uint16_t other_node = 00;
9
10 uint32_t readingDelay = 4000; /* Delay de leitura do sensor */

```

Listing 4.9 – Configuração do rádio

A seguir é possível observar as estruturas descritas em 4.10 para armazenar o comando e os dados dos sensores. A primeira estrutura de dados abaixo possui um elemento para armazenar o código do comando e outro elemento para salvar o parâmetro passado através da função.

A segunda estrutura é responsável por armazenar as informações de temperatura e umidade que são lidos através do sensor.

```

1 /* Estrutura do comando */
2 struct payload_t {
3     unsigned long command;
4     unsigned long parameter;
5 };
6
7 /* Estrutura dos dados*/
8 struct payload_to_send{
9     unsigned long temperature;
10    unsigned long humidity;
11};

```

Listing 4.10 – Estrutura de dados dos comandos e dados

O trecho a seguir, descrito em 4.11, é responsável por configurar a ligação física do sensor, o tipo do sensor e a declaração da estrutura de dados que armazenam os valores de temperatura e umidade, respectivamente.

```

1 /* DHT11 */
2 DHT_Unified dht(DHTPIN, DHTTYPE);
3 uint32_t delayMS;
4 /* Estrutura do payload com temperatura e umidade */
5 payload_to_send payload_sensor = {30, 90};
6 /* Por padrão o Arduino é inicializado sempre como mestre (ativo) */
7 boolean slave = false;

```

Listing 4.11 – Configuração sensor DHT11

O método *initDHT()*, descrito em 4.12, é responsável pela inicialização do sensor. Já o método *readDHT()* é responsável por realizar as leituras do sensor e armazenar as informações dentro da estrutura de dados.

```

1 /* Inicializando o sensor */
2 void initDHT(){
3     dht.begin();
4     sensor_t sensor;

```

```

5   dht.temperature().getSensor(&sensor);
6   dht.humidity().getSensor(&sensor);
7   delayMS = sensor.min_delay / 1000;
8 }
9
10 /* Metodo para realizar leituras do sensor */
11 void readDHT(){
12   sensors_event_t event;
13   dht.temperature().getEvent(&event);
14
15   /* Lendo a temperatura do sensor */
16   payload_sensor.temperature = event.temperature;
17   if (isnan(event.temperature)) {
18     Serial.println(F("Erro ao ler temperatura!"));
19   }
20
21   /* Lendo a umidade do sensor */
22   dht.humidity().getEvent(&event);
23   payload_sensor.humidity = event.relative_humidity;
24   if (isnan(event.relative_humidity)) {
25     Serial.println(F("Erro ao ler umidade!"));
26   }
27 }
```

Listing 4.12 – Inicialização do sensor

O método *sendData*, descrito em 4.13, envia ao nó coordenador o último valor de leitura obtido pelos sensores. O parâmetro *samples* identifica a quantidade de vezes em que o mesmo dado deverá ser enviado ao coordenador.

```

1 /* Metodo criado para enviar os dados ao coordenador */
2 void sendData(uint8_t samples){
3   int i=0;
4   RF24NetworkHeader header2(other_node);
5   Serial.print("Enviando dado do no 01 para 00 ..");
6
7   for(i = 0; i < samples; i++){
8     bool ok = network.write(header2,&payload_sensor,sizeof(
9       payload_sensor));
10    if (ok){
11      Serial.print(" Temp: ");
12      Serial.print(payload_sensor.temperature);
13      Serial.print(" Umidade: ");
14      Serial.println(payload_sensor.humidity);
15    }
16    else
17      Serial.println(" falhou..");
```

```

17     delay(100);
18 }
19 }
```

Listing 4.13 – Envio das informações do sensor

O método *setup*, descrito em 4.14, é responsável pela inicialização da porta serial, barramento SPI, rádio e sensor DHT11. Este método é essencial para o correto funcionamento do nó sensor, pois inicializa os barramentos e os módulos que estão conectados ao Arduino.

```

1 void setup(void){
2     /* Inicializando a porta serial para debug */
3     Serial.begin(115200);
4     /* Inicializando SPI para comunicação com sensor */
5     SPI.begin();
6     /* Inicializando radio */
7     radio.begin();
8     /* Selecionando o canal a ser utilizado e o endereço do no */
9     network.begin(110, this_node);
10    /* Inicializando o sensor DHT11 */
11    initDHT();
12 }
```

Listing 4.14 – Setup

O último trecho, descrito em 4.15, representa o laço principal do código. Este laço é executado de forma infinita e ele é responsável por detectar pacotes a serem lidos, interpretação e execução de comandos e leituras do sensor.

```

1
2 /* Main loop */
3 void loop(void){
4     /* Atualização da rede */
5     network.update();
6
7     /* Verificando se existe pacote a ser lido */
8     while (network.available()){
9         RF24NetworkHeader headerR;
10        payload_t payload;
11        network.read(headerR,&payload,sizeof(payload));
12
13        Serial.print("Comando recebido: ");
14        Serial.print(payload.command);
15        Serial.print(" parametro: ");
```

```

16     Serial.print(payload.parameter);
17     Serial.print(" de: ");
18     Serial.println(headerR.from_node);
19
20     /* Comando para entrar em modo escravo (passivo) */
21     if (payload.command == 1){
22         slave = true;
23         Serial.println("Arduino mudando para modo escravo (passivo) -"
24             " responde por requisicio o RPI");
25     }
26     /* Comando para entrar em modo mestre (ativo) */
27     else if (payload.command == 2) {
28         slave = false;
29         /* Convertendo segundos para milisegundos */
30         readingDelay = payload.parameter * 1000;
31         Serial.print("Arduino mudando para modo mestre (ativo) - Envia"
32             " periodicamente a cada ");
33         Serial.print(readingDelay);
34         Serial.println("x segundos");
35     }
36     /* Respondendo requisicao de leitura caso esteja no modo escravo ("
37     " passivo) */
38     else if (payload.command == 10 && slave) {
39         Serial.println("Arduino recebeu requisicao de leitura do RPI");
40         Serial.println(payload.parameter);
41         sendData(payload.parameter);
42     }
43
44
45     /* Se o arduino for mestre (ativo), enviar periodicamente os dados */
46     if(!slave){
47         readDHT();
48         sendData(1);
49     }
50     delay(readingDelay);
51 }
```

Listing 4.15 – Código main

## 4.4 Custos dos Componentes

Esta seção contém a lista dos materiais necessários para a execução do projeto junto com seu custo pesquisado em loja nacional e internacional. Além da listagem dos materiais utilizados e seus preços, também foi levantada uma estimativa de custos por cada nó coordenador e nó sensor, com opção de utilizar a antena interna ou externa.

A lista dos materiais que foram utilizados no projeto são: 1 *Raspberry Pi 3+*, 1 Arduino Uno Rev3, 1 Arduino Due, 1 Arduino Nano, 2 módulos nRF24L01+ com antena externa, 2 módulos nRF24L01+ com antena interna, 3 sensores DHT11, 3 *protoboards* e 40 cabos do tipo macho-macho. A relação entre os materiais e seus respectivos custos pode ser observado na [Tabela 13](#) e [Tabela 14](#). O valor do dólar considerado para conversão foi de R\$ 4,40.

Tabela 13 – Custos Plataforma - Mercado Livre

Componente	Quantidade	Preço (R\$)
<i>Raspberry Pi 3</i>	1	228,00
Arduino Uno	1	27,00
Arduino Due	1	89,00
Arduino Nano	1	19,00
nRF24L01+ antena interna	2	7,00
nRF24L01+ antena externa	2	27,00
Sensor DHT11	3	6,00
<i>Protoboard</i>	3	6,00
Kit 40 cabos macho-macho	1	14,00
<b>Custo total da plataforma:</b>		487,00

Tabela 14 – Custos Plataforma - Dx.com

Componente	Quantidade	Preço (\$)
<i>Raspberry Pi 3+</i>	1	46,00
Arduino Uno	1	9,00
Arduino Due	1	17,00
Arduino Nano	1	3,50
nRF24L01+ antena interna	2	3,00
nRF24L01+ antena externa	2	4,00
Sensor DHT11	3	1,70
<i>Protoboard</i>	3	2,00
Kit 40 cabos macho-macho	1	2,00
<b>Custo total da plataforma:</b>		\$ 96,6 (R\$ 385,434)

Os materiais e os custos dos nós coordenadores e sensores estão descritos na Tabela 15 e Tabela 16. Apesar do custo ser maior, os módulos com antenas externas foram selecionados para comparativo de preços devido ao seu desempenho. Os preços dos componentes estão todos em moeda real e foram levantados através do Mercado Livre, disponível para acesso em: <[www.mercadolivre.com.br](http://www.mercadolivre.com.br)>.

Tabela 15 – Custos nó coordenador

Nó coordenador	Custo (R\$)
Raspberry Pi 3	228,00
nRF24L01+ antena externa	27,00
Protoboard	6,00
Cabos macho-macho	7,00
<b>Total</b>	<b>R\$ 268,00</b>

Tabela 16 – Custos nó sensor

Nó sensor	Custo (R\$)
Arduino Nano	19,00
nRF24L01+ antena externa	27,00
Sensor DHT11	6,00
Protoboard	6,00
Cabos macho-macho	7,00
<b>Total</b>	<b>65,00</b>

# 5 Resultados e Discussão

Este capítulo contém a descrição e resultados de todos os testes que foram realizados a fim de validar a plataforma com os componentes integrados. Além da validação dos componentes da plataforma, descritos na [seção 5.1](#), foram realizados testes de simulações de falhas entre os nós, visto na [seção 5.2](#).

Um vídeo foi gravado com a execução de todos os testes descritos para validação de todos os comandos e também as simulações de falhas entre os nós da rede. O *link* com o vídeo está disponível para acesso em <<https://www.youtube.com/watch?v=ZApJ0AZtLC4>>.

## 5.1 Testes de Funcionamento da Plataforma Desenvolvida

Para um melhor entendimento, os testes foram numerados de 1 a 11. As numerações 1, 2 e 3 representam testes feitos para a validação do comando de leitura vindo do nó coordenador para os nós sensores. O nó coordenador solicita 1 leitura do nó 01, 2 leituras do nó 011 e 3 leituras do nó 021.

Já os testes 3, 4 e 5 são feitos a fim de validar o comando de configuração dos nós sensores como mestre (ativo) da comunicação e configurando a frequência com que os dados são enviados. O nó 01 é configurado para atuar como mestre e com uma frequência de envio de 1 pacote de dados (*frame*) por segundo. O nó 011 é configurado para atuar como mestre (ativo) com frequência de envio de 1 pacote de dados (*frame*) a cada 2 segundos. Já o nó 021 é configurado como mestre (ativo) e com uma frequência de envio de 1 *frame* a cada 3 segundos.

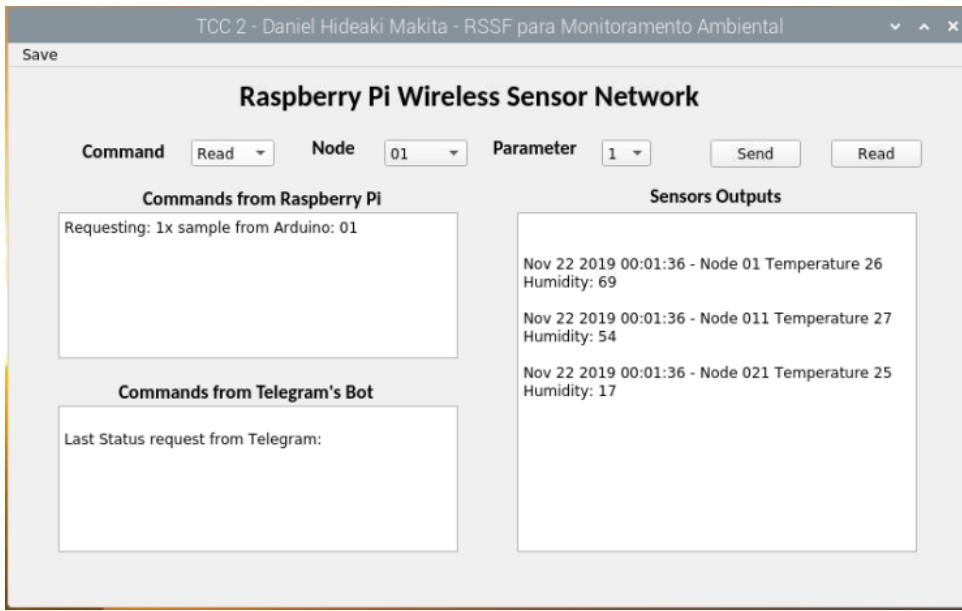
Os testes 7, 8 e 9 foram realizados para validar o funcionamento do comando de configuração do nó sensor como modo escravo (passivo). Os testes são realizados para configurar os nós 01, 011 e 021 em modo escravo (passivo), respectivamente.

Por fim, os testes 10 e 11 validam o funcionamento da integração do nó coordenador com as plataformas *ThingSpeak* e *Telegram*.

### Teste 1. Solicitar 1 leitura do nó 01

Na [Figura 35](#) é possível observar uma solicitação vinda do usuário para o nó 01 enviar por 1 vez os dados do sensor. Na [Figura 36](#) é possível observar o recebimento por parte do nó coordenador os dados vindos do nó 01.

Figura 35 – Solicitando ao nó 01 o envio de 1 frame



Fonte: O Autor

Figura 36 – Nô 01 enviando o dado 1 vez

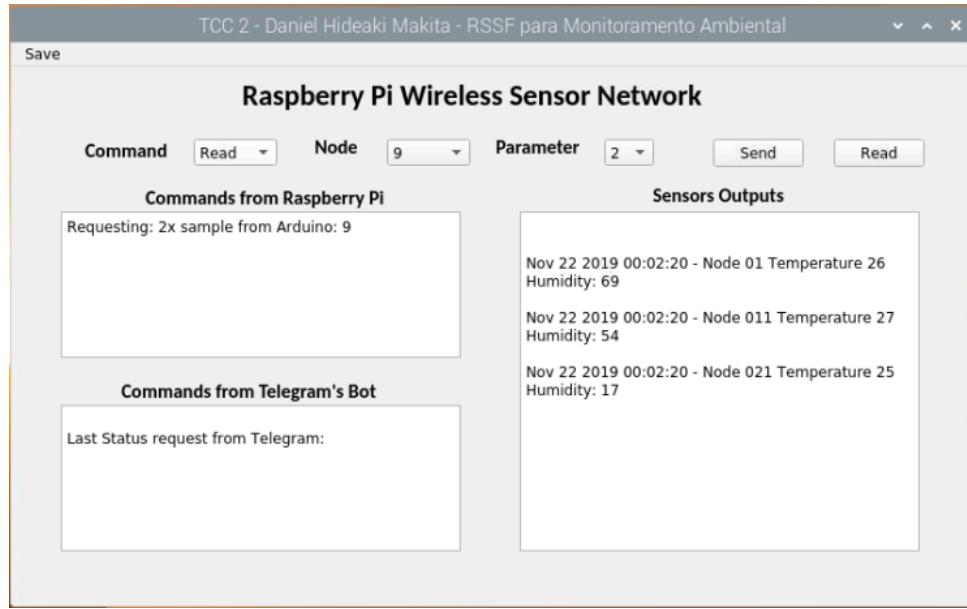
```
pi@raspberrypi: ~/Desktop/TCC_2
File Edit Tabs Help
Nov 22 2019 00:00:32 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:00:35 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:35 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:00:37 - Received Temperature: 25 Humidity: 17 from Node: 0o21
Nov 22 2019 00:00:37 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:37 - Received Temperature: 27 Humidity: 69 from Node: 0o1
Nov 22 2019 00:00:39 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:39 - Received Temperature: 25 Humidity: 17 from Node: 0o21
Nov 22 2019 00:00:39 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:00:41 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:41 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:00:43 - Received Temperature: 25 Humidity: 17 from Node: 0o21
Nov 22 2019 00:00:43 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:43 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:00:46 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:48 - Received Temperature: 25 Humidity: 17 from Node: 0o21
Nov 22 2019 00:00:48 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:50 - Received Temperature: 26 Humidity: 17 from Node: 0o21
Nov 22 2019 00:00:50 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:52 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:52 - Received Temperature: 25 Humidity: 17 from Node: 0o21
Nov 22 2019 00:01:28 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:01:36 - Received Temperature: 26 Humidity: 69 from Node: 0o1
```

Fonte: O Autor

### Teste 2. Solicitar 2 leituras do nó 011

Na [Figura 37](#) é possível observar uma solicitação vinda do usuário para o nó 011 enviar por 2 vezes os dados do sensor. Na [Figura 38](#) é possível observar o recebimento por parte do nó coordenador os dados vindos do nó 011.

Figura 37 – Solicitando ao nó 011 o envio de 2 frames



Fonte: O Autor

Figura 38 – Nó 011 enviando 2 vezes os dados

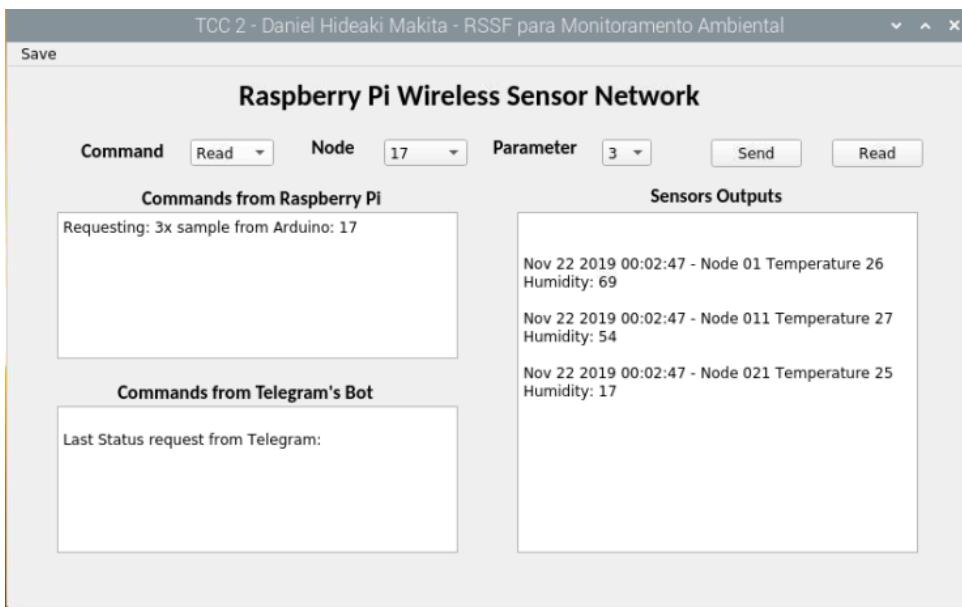
```
pi@raspberrypi: ~/Desktop/TCC_2
File Edit Tabs Help
Nov 22 2019 00:00:39 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:00:39 - Received Temperature: 25 Humidity: 17 from Node: 0021
Nov 22 2019 00:00:39 - Received Temperature: 26 Humidity: 69 from Node: 001
Nov 22 2019 00:00:41 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:00:41 - Received Temperature: 26 Humidity: 69 from Node: 001
Nov 22 2019 00:00:43 - Received Temperature: 25 Humidity: 17 from Node: 0021
Nov 22 2019 00:00:43 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:00:43 - Received Temperature: 26 Humidity: 69 from Node: 001
Nov 22 2019 00:00:46 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:00:46 - Received Temperature: 25 Humidity: 17 from Node: 0021
Nov 22 2019 00:00:48 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:00:50 - Received Temperature: 26 Humidity: 17 from Node: 0021
Nov 22 2019 00:00:50 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:00:52 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:00:52 - Received Temperature: 25 Humidity: 17 from Node: 0021
Nov 22 2019 00:01:28 - Received Temperature: 26 Humidity: 69 from Node: 001
Nov 22 2019 00:01:36 - Received Temperature: 26 Humidity: 69 from Node: 001
Nov 22 2019 00:02:04 - Received Temperature: 26 Humidity: 69 from Node: 001
Nov 22 2019 00:02:04 - Received Temperature: 26 Humidity: 69 from Node: 001
Nov 22 2019 00:02:10 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:02:12 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:02:20 - Received Temperature: 27 Humidity: 54 from Node: 0011
Nov 22 2019 00:02:20 - Received Temperature: 27 Humidity: 54 from Node: 0011
```

Fonte: O Autor

### Teste 3. Solicitar 3 leituras do nó 021

Na [Figura 39](#) é possível observar uma solicitação vinda do usuário para o nó 021 enviar por 3 vezes os dados do sensor. Na [Figura 40](#) é possível observar o recebimento por parte do nó coordenador os dados vindos do nó 021.

Figura 39 – Solicitando ao nó 021 o envio de 3 frames



Fonte: O Autor

Figura 40 – Nô 021 enviando 3 vezes os dados

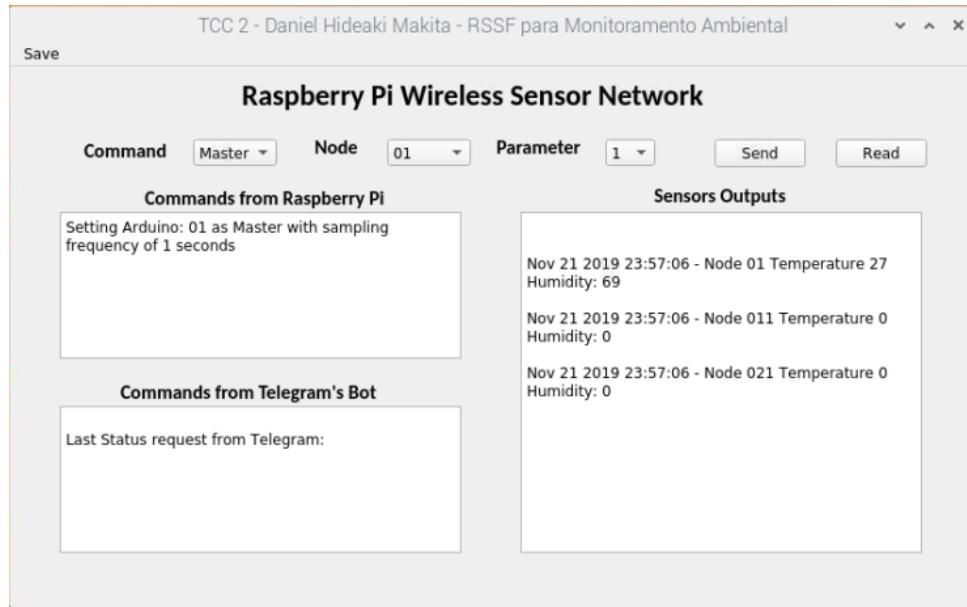
```
pi@raspberrypi: ~/Desktop/TCC_2
File Edit Tabs Help
Nov 22 2019 00:00:43 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:43 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:00:46 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:46 - Received Temperature: 25 Humidity: 17 from Node: 0o21
Nov 22 2019 00:00:48 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:50 - Received Temperature: 26 Humidity: 17 from Node: 0o21
Nov 22 2019 00:00:50 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:52 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:00:52 - Received Temperature: 25 Humidity: 17 from Node: 0o21
Nov 22 2019 00:01:28 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:01:36 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:02:04 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:02:04 - Received Temperature: 26 Humidity: 69 from Node: 0o1
Nov 22 2019 00:02:10 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:02:12 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:02:20 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:02:20 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:02:39 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:02:39 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:02:39 - Received Temperature: 27 Humidity: 54 from Node: 0o11
Nov 22 2019 00:02:47 - Received Temperature: 25 Humidity: 17 from Node: 0o21
Nov 22 2019 00:02:47 - Received Temperature: 25 Humidity: 17 from Node: 0o21
Nov 22 2019 00:02:47 - Received Temperature: 25 Humidity: 17 from Node: 0o21
```

Fonte: O Autor

**Teste 4.** Configurar o nó 01 como mestre (ativo) e frequência de envio a cada 1 segundo

Na [Figura 41](#) é possível observar uma solicitação vinda do usuário para configurar o nó 01 como mestre (ativo) e enviar os dados a uma frequência de 1 segundo. Na [Figura 42](#) é possível observar o recebimento por parte do nó coordenador os dados vindos do nó 01 na frequência de 1 segundo.

Figura 41 – Configurando nó 01 como mestre (ativo) e frequência de envio de 1s



Fonte: O Autor

Figura 42 – Nô 01 enviando dados a cada 1 segundo

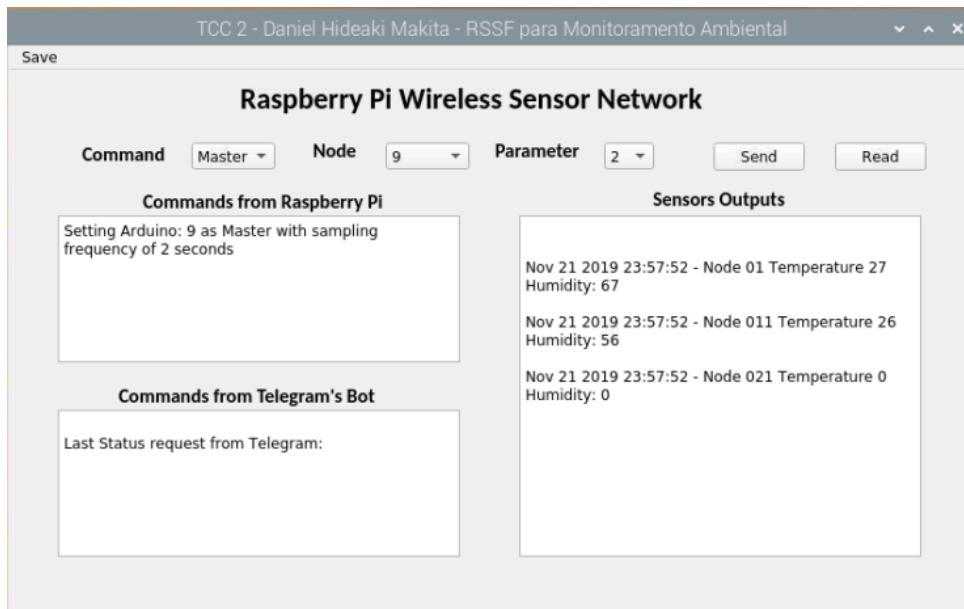
```
pi@raspberrypi: ~/Desktop/TCC_2
File Edit Tabs Help
Nov 21 2019 23:56:42 - Received Temperature: 27 Humidity: 76 from Node: 001
Nov 21 2019 23:56:43 - Received Temperature: 27 Humidity: 76 from Node: 001
Nov 21 2019 23:56:44 - Received Temperature: 27 Humidity: 75 from Node: 001
Nov 21 2019 23:56:45 - Received Temperature: 27 Humidity: 75 from Node: 001
Nov 21 2019 23:56:47 - Received Temperature: 27 Humidity: 73 from Node: 001
Nov 21 2019 23:56:48 - Received Temperature: 27 Humidity: 73 from Node: 001
Nov 21 2019 23:56:49 - Received Temperature: 27 Humidity: 73 from Node: 001
Nov 21 2019 23:56:50 - Received Temperature: 27 Humidity: 73 from Node: 001
Nov 21 2019 23:56:51 - Received Temperature: 27 Humidity: 72 from Node: 001
Nov 21 2019 23:56:52 - Received Temperature: 27 Humidity: 72 from Node: 001
Nov 21 2019 23:56:53 - Received Temperature: 27 Humidity: 71 from Node: 001
Nov 21 2019 23:56:55 - Received Temperature: 27 Humidity: 71 from Node: 001
Nov 21 2019 23:56:56 - Received Temperature: 27 Humidity: 71 from Node: 001
Nov 21 2019 23:56:57 - Received Temperature: 27 Humidity: 71 from Node: 001
Nov 21 2019 23:56:58 - Received Temperature: 27 Humidity: 70 from Node: 001
Nov 21 2019 23:56:59 - Received Temperature: 27 Humidity: 70 from Node: 001
Nov 21 2019 23:57:00 - Received Temperature: 27 Humidity: 70 from Node: 001
Nov 21 2019 23:57:01 - Received Temperature: 27 Humidity: 70 from Node: 001
Nov 21 2019 23:57:03 - Received Temperature: 27 Humidity: 69 from Node: 001
Nov 21 2019 23:57:04 - Received Temperature: 27 Humidity: 69 from Node: 001
Nov 21 2019 23:57:05 - Received Temperature: 27 Humidity: 69 from Node: 001
Nov 21 2019 23:57:06 - Received Temperature: 27 Humidity: 69 from Node: 001
Nov 21 2019 23:57:07 - Received Temperature: 27 Humidity: 69 from Node: 001
```

Fonte: O Autor

**Teste 5.** Configurar o nó 011 como mestre (ativo) e frequencia de envio a cada 2 segundos

Na [Figura 43](#) é possível observar uma solicitação vinda do usuário para configurar o nó 011 como mestre (ativo) e enviar os dados a uma frequência de 2 segundos. Na [Figura 44](#) é possível observar o recebimento por parte do nó coordenador os dados vindos do nó 01 na frequência de 2 segundos.

Figura 43 – Configurando nó 011 como mestre (ativo) e frequênciade envio de 2s



Fonte: O Autor

Figura 44 – Nó 011 enviando dados a cada 2 segundos

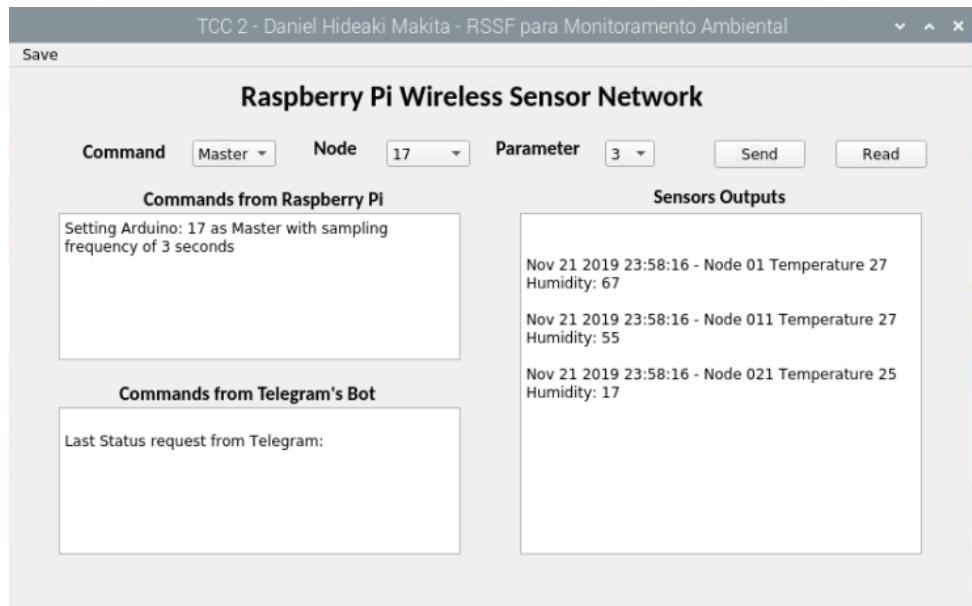
```
pi@raspberrypi: ~/Desktop/TCC_2
File Edit Tabs Help
Nov 21 2019 23:57:32 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:33 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:35 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:36 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:37 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:38 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:39 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:40 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:41 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:43 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:44 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:45 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:46 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:47 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:48 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:49 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 21 2019 23:57:49 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:51 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:52 - Received Temperature: 27 Humidity: 55 from Node: 0011
Nov 21 2019 23:57:52 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:53 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:57:54 - Received Temperature: 27 Humidity: 56 from Node: 0011
Nov 21 2019 23:57:54 - Received Temperature: 27 Humidity: 67 from Node: 001
```

Fonte: O Autor

**Teste 6.** Configurar o nó 021 como mestre (ativo) e frequencia de envio a cada 3 segundos

Na [Figura 45](#) é possível observar uma solicitação vinda do usuário para configurar o nó 021 como mestre (ativo) e enviar os dados a uma frequência de 3 segundos. Na [Figura 46](#) é possível observar o recebimento por parte do nó coordenador os dados vindos do nó 021 na frequência de 3 segundos.

Figura 45 – Configurando nó 021 como mestre (ativo) e frequênciade envio de 3s



Fonte: O Autor

Figura 46 – Nô 021 enviando dados a cada 3 segundos

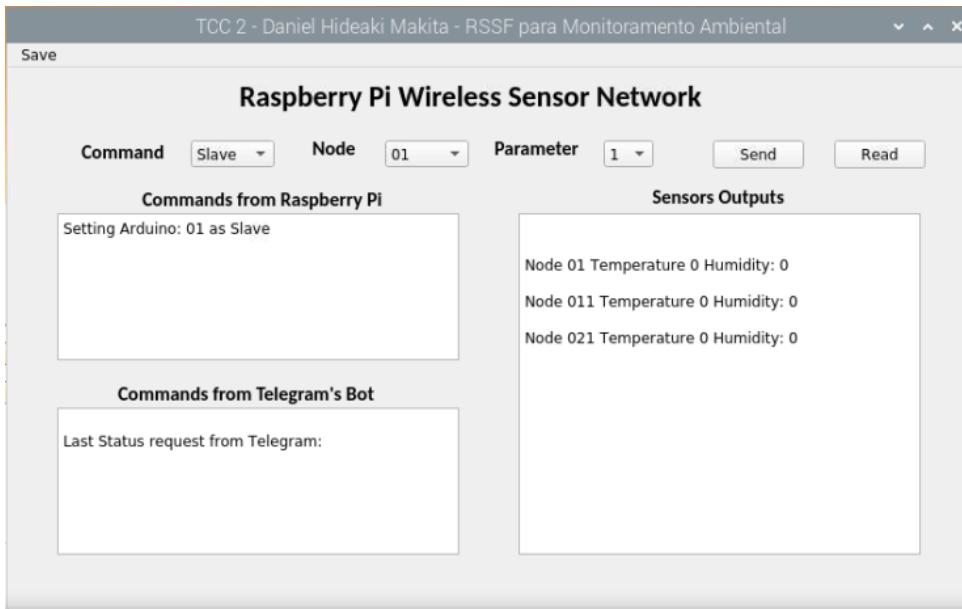
```
pi@raspberrypi: ~/Desktop/TCC_2
File Edit Tabs Help
Nov 21 2019 23:58:04 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:06 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:07 - Received Temperature: 27 Humidity: 55 from Node: 0011
Nov 21 2019 23:58:07 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:08 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:09 - Received Temperature: 27 Humidity: 55 from Node: 0011
Nov 21 2019 23:58:09 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:10 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:11 - Received Temperature: 27 Humidity: 55 from Node: 0011
Nov 21 2019 23:58:11 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:13 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:14 - Received Temperature: 27 Humidity: 55 from Node: 0011
Nov 21 2019 23:58:14 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:15 - Received Temperature: 25 Humidity: 17 from Node: 0021
Nov 21 2019 23:58:15 - Received Temperature: 27 Humidity: 55 from Node: 0011
Nov 21 2019 23:58:16 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:17 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:17 - Received Temperature: 27 Humidity: 55 from Node: 0011
Nov 21 2019 23:58:17 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:18 - Received Temperature: 25 Humidity: 18 from Node: 0021
Nov 21 2019 23:58:18 - Received Temperature: 27 Humidity: 67 from Node: 001
Nov 21 2019 23:58:20 - Received Temperature: 27 Humidity: 55 from Node: 0011
Nov 21 2019 23:58:20 - Received Temperature: 27 Humidity: 67 from Node: 001
```

Fonte: O Autor

**Teste 7.** Configurar nó 01 como escravo (passivo)

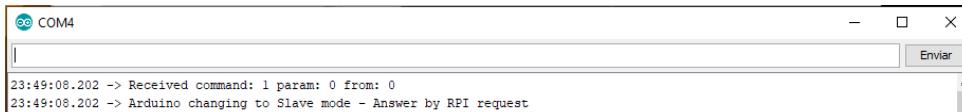
Na Figura 47 é possível observar que o usuário solicitou que o nó 01 fosse configurado em modo escravo. Já na Figura 48 é possível confirmar o recebimento e execução do comando pelo nó 01.

Figura 47 – Configurando nó 01 como escravo (passivo)



Fonte: O Autor

Figura 48 – Nô 01 entrando em modo escravo

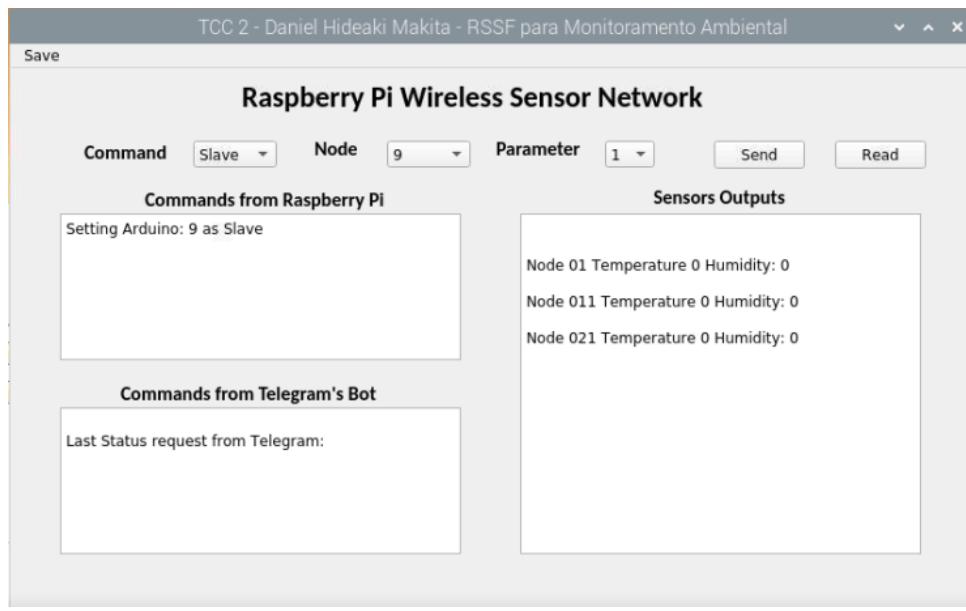


Fonte: O Autor

**Teste 8.** Configurar nó 011 como escravo (passivo)

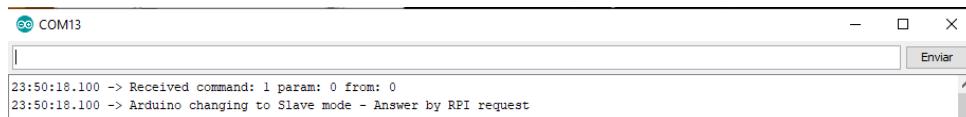
Na [Figura 49](#) é possível observar que o usuário solicitou que o nó 9 (octal 011) fosse configurado em modo escravo. Já na [Figura 50](#) é possível confirmar o recebimento e execução do comando pelo nó 011.

Figura 49 – Configurando nó 011 como escravo (passivo)



Fonte: O Autor

Figura 50 – Nô 011 entrando em modo escravo (passivo)

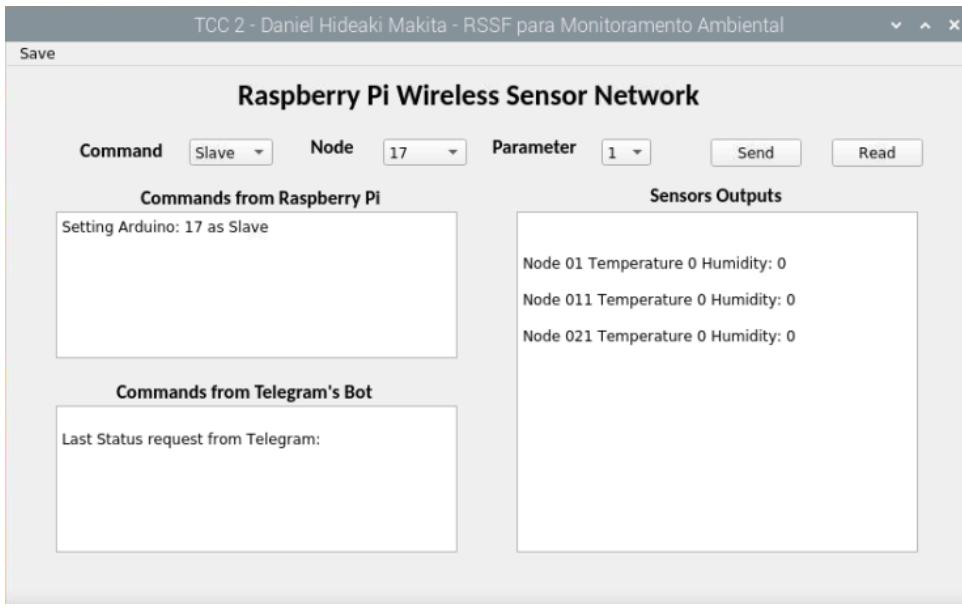


Fonte: O Autor

**Teste 9.** Configurar nó 021 como escravo (passivo)

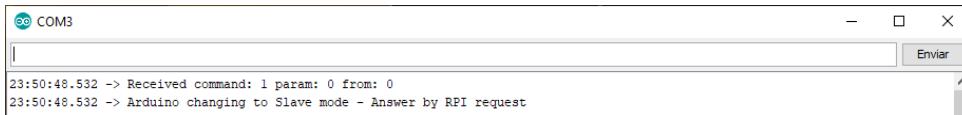
Na [Figura 51](#) é possível observar que o usuário solicitou que o nó 17 (octal 021) fosse configurado em modo escravo (passivo). Já na [Figura 52](#) é possível confirmar o recebimento e execução do comando pelo nó 021.

Figura 51 – Configurando nó 021 como escravo (passivo)



Fonte: O Autor

Figura 52 – Nô 021 entrando em modo escravo (passivo)



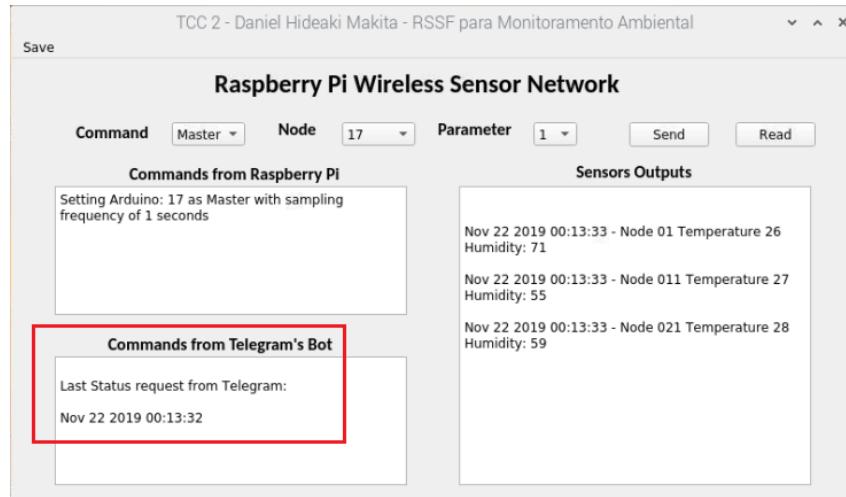
Fonte: O Autor

### Teste 10. Solicitar leitura através do Telegram

A Figura 53 apresenta o *log* de quando a solicitação de leitura foi feita pelo usuário utilizando o *Telegram*.

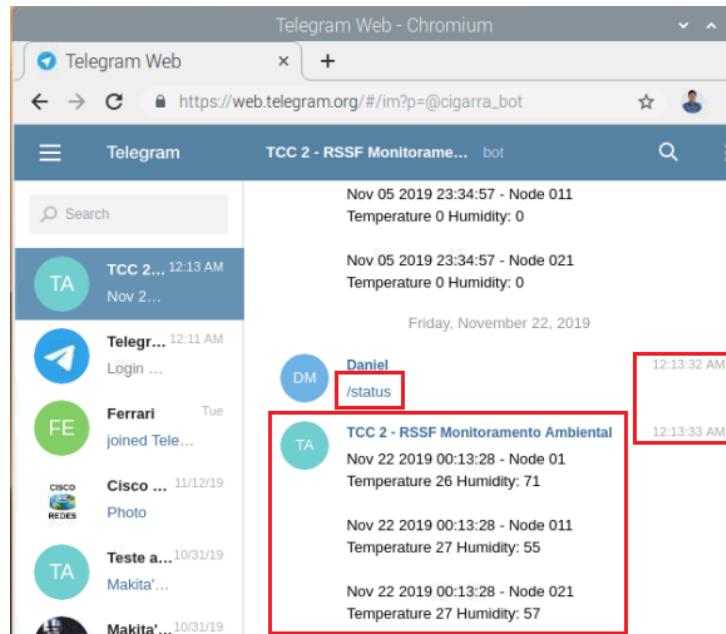
Na Figura 20 é possível observar o usuário solicitando os valores dos sensores através do comando *status* no navegador.

Figura 53 – Log da última solicitação via *Telegram*



Fonte: O Autor

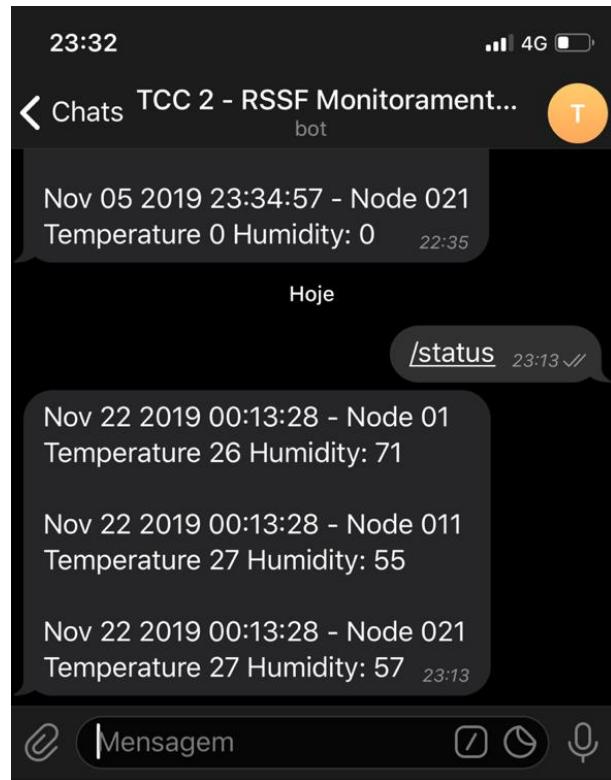
Figura 54 – Tela do computador no navegador web do *Telegram*



Fonte: O Autor

Já na Figura 55, a solicitação é feita através do aplicativo no *smartphone*.

Figura 55 – Tela do celular no aplicativo *Telegram*

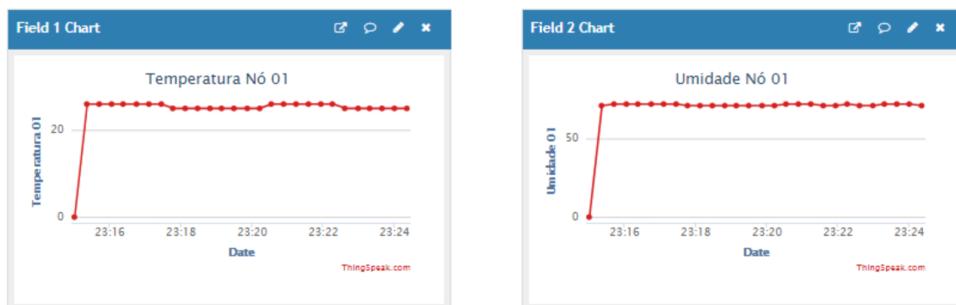


Fonte: O Autor

#### Teste 11. Validar integração com a plataforma *ThingSpeak*

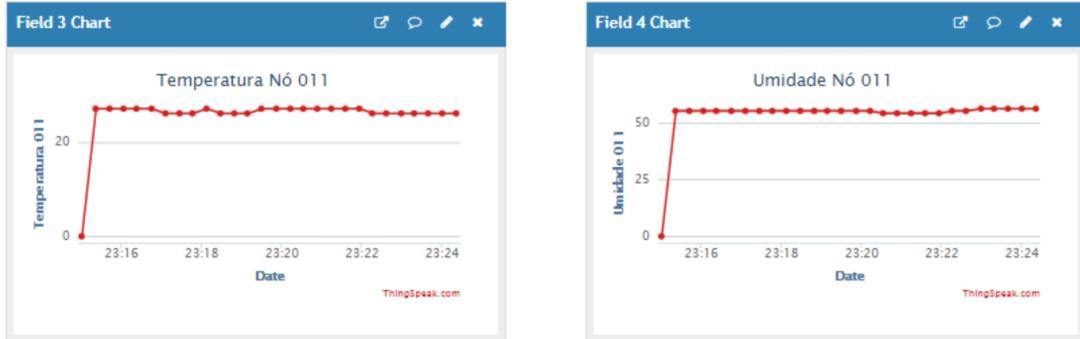
Na Figura 56, Figura 57 e Figura 58, é possível observar a validação da integração do nó coordenador com o *ThingSpeak* através da visualização gráfica de todos os valores obtidos nos nós sensores 01, 011 e 021 em relação a escala de tempo em horas e minutos.

Figura 56 – Gráfico dos dados do nó 01



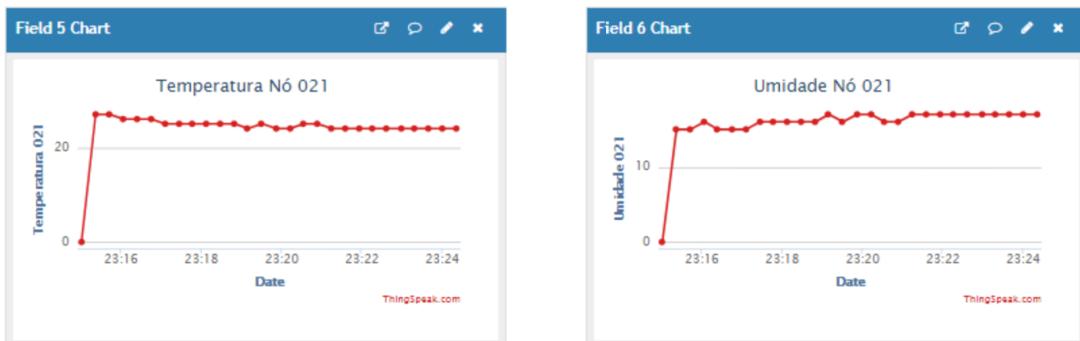
Fonte: O Autor

Figura 57 – Gráfico dos dados do nó 011



Fonte: O Autor

Figura 58 – Gráfico dos dados do nó 021



Fonte: O Autor

A Figura 59 representa os dados enviados do nó coordenador a plataforma *ThingSpeak* para a geração dos gráficos da Figura 56, Figura 57 e Figura 58.

Figura 59 – Dados enviados ao *ThingSpeak*

```
pi@raspberrypi: ~/Desktop/TCC_2
File Edit Tabs Help
Nov 22 2019 00:24:43 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:24:43 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:24:44 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:24:44 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:24:44 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:24:45 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:24:45 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:24:45 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:24:46 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:24:46 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:24:46 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:24:48 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:24:48 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:24:48 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:24:49 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:24:49 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:24:49 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:24:50 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:24:50 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:24:50 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:24:51 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:24:51 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:24:51 - Received Temperature: 25 Humidity: 72 from Node: 001
```

Fonte: O Autor

## 5.2 Testes de Simulações de Falhas

Três testes de simulação foram realizados a fim de verificar o comportamento da plataforma diante à falhas do sistema. As simulações foram realizadas removendo a alimentação dos nós 011, 021 e 01, respectivamente.

Após a falha, os nós foram conectados novamente à alimentação para verificar o reestabelecimento completo da topologia inicial da rede.

### Teste 1. Simular falha do nó sensor 011

Pelo terminal, observado na [Figura 60](#), é possível observar que com a falha do nó 011, o nó 01 continuou a operar normalmente. Este comportamento era esperado pois na topologia deste trabalho, o nó 011 não faz o roteamento de pacotes.

Figura 60 – Falha no nó 011

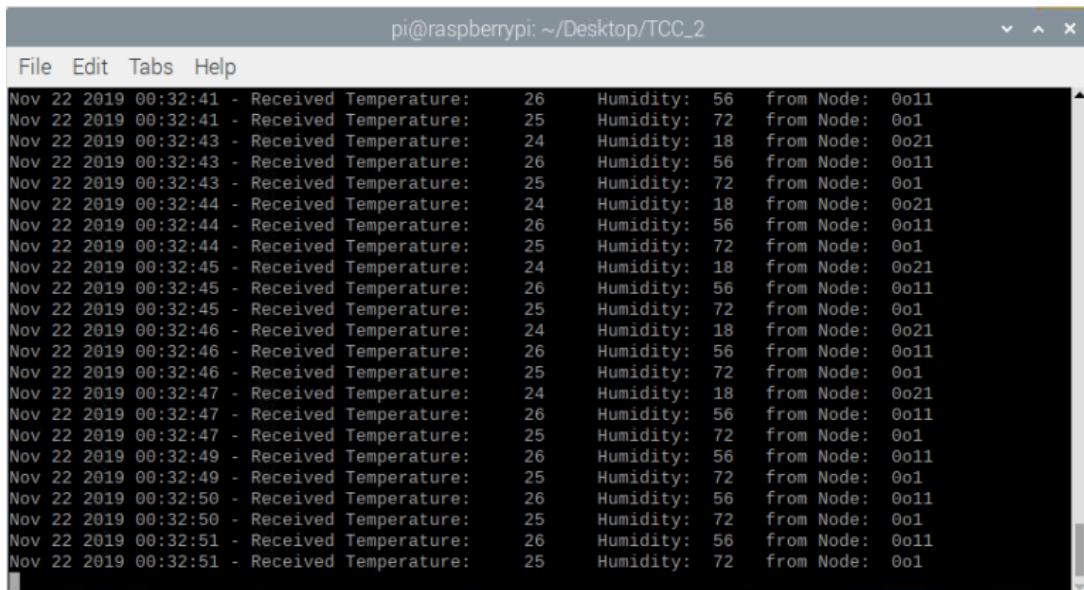
```
pi@raspberrypi: ~/Desktop/TCC_2
File Edit Tabs Help
Nov 22 2019 00:32:56 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:56 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:57 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:57 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:58 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:58 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:58 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:59 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:59 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:00 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:33:00 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:01 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:33:01 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:03 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:33:03 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:04 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:33:04 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:05 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:33:05 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:06 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:33:06 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:07 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:09 - Received Temperature: 25 Humidity: 72 from Node: 001
```

Fonte: O Autor

**Teste 2.** Simular falha do nó sensor 021

Pelo terminal, obeservado na [Figura 61](#), é possível observar que apesar da falha do nó 021, os nós 01 e 011 continuaram a operar normalmente. Este comportamento era esperado pois na topologia deste trabalho, o nó 021 não faz o roteamento de pacotes.

Figura 61 – Falha no nó 021



```
pi@raspberrypi: ~/Desktop/TCC_2
File Edit Tabs Help
Nov 22 2019 00:32:41 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:41 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:43 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:32:43 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:43 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:44 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:32:44 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:44 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:45 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:32:45 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:45 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:46 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:32:46 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:46 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:47 - Received Temperature: 24 Humidity: 18 from Node: 0021
Nov 22 2019 00:32:47 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:47 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:49 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:49 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:50 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:50 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:32:51 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:32:51 - Received Temperature: 25 Humidity: 72 from Node: 001
```

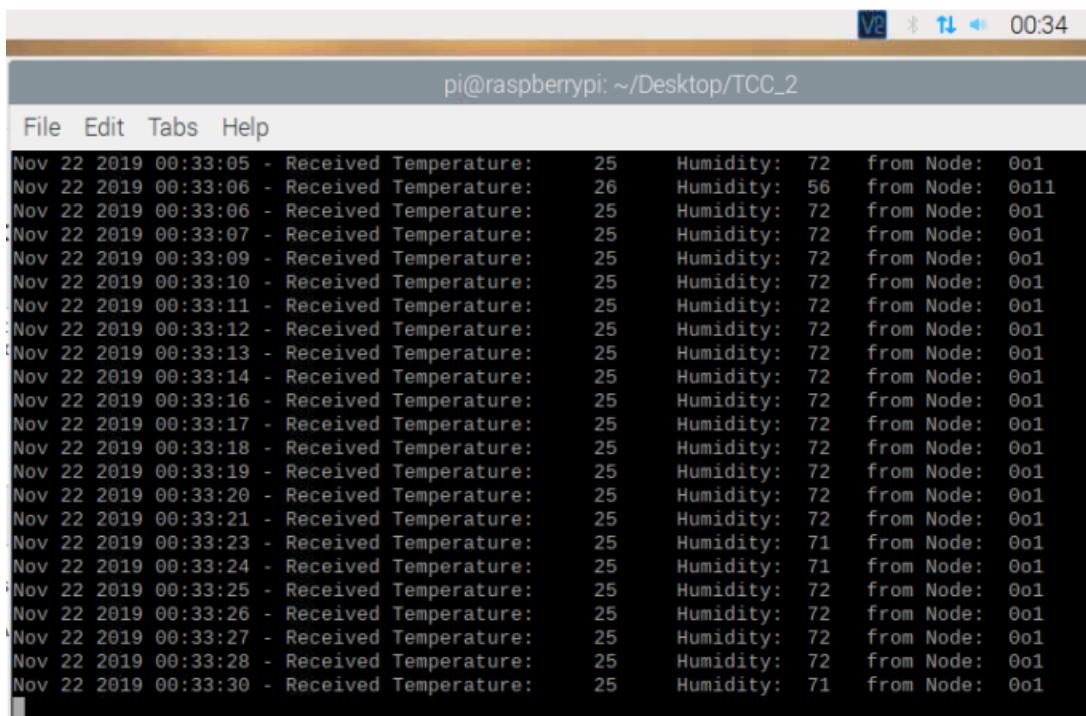
Fonte: O Autor

**Teste 3.** Simular falha do nó sensor 01

Neste teste, onde o nó 01 é desativado, é possível constatar através da [Figura 62](#), que os pacotes não estão sendo mais enviados ao nó coordenador. É possível observar no canto superior direito o horário de 00:34 em que o teste foi realizado e verificar que a última transmissão ocorreu em 00:33:30.

Este comportamento já era esperado pois o nó 01 faz o roteamento de todos os nós abaixo dele para o nó coordenador.

Figura 62 – Falha no nó 01



```
Nov 22 2019 00:33:05 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:06 - Received Temperature: 26 Humidity: 56 from Node: 0011
Nov 22 2019 00:33:06 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:07 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:09 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:10 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:11 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:12 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:13 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:14 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:16 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:17 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:18 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:19 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:20 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:21 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:23 - Received Temperature: 25 Humidity: 71 from Node: 001
Nov 22 2019 00:33:24 - Received Temperature: 25 Humidity: 71 from Node: 001
Nov 22 2019 00:33:25 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:26 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:27 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:28 - Received Temperature: 25 Humidity: 72 from Node: 001
Nov 22 2019 00:33:30 - Received Temperature: 25 Humidity: 71 from Node: 001
```

Fonte: O Autor

## 6 Considerações Finais

No decorrer deste trabalho, foram apresentados os conceitos e as consequências de desastres naturais para a sociedade, sendo um tema que tem levantado cada vez mais interesses e preocupações do Governo para o estudo e aplicações de tecnologias que sejam acessíveis e que possibilitem a redução dos impactos causados por estes eventos.

Por ser uma área abrangente e relativamente recente, foi feito um levantamento bibliográfico a fim de entender quais as tecnologias e em que aplicações estão sendo utilizadas no contexto de redes de sensores. Foi necessário analisar os pontos positivos e negativos de cada tecnologia levando em conta os principais requisitos em se utilizar uma plataforma de baixo custo, flexível, escalável até 781 nós totais e confiável na transmissão de pacotes entre os nós.

Entendendo o atual cenário no país e definindo as tecnologias que poderiam ser utilizadas, o objetivo deste trabalho se concentrou no desenvolvimento de um protótipo de uma plataforma composta por uma Rede de Sensores Sem Fio com aplicação no monitoramento de desastres naturais.

A plataforma de *hardware* foi desenvolvida com um computador *Raspberry Pi 3+* para o nó coordenador e para os nós sensores, Arduinos da família Uno, Due e Nano. O sensor utilizado nos nós sensores foi o modelo DHT11. Para o módulo de comunicação sem fio, foram utilizados rádios do modelo nRF24L01.

A plataforma de *software* desenvolvida conta com uma interface gráfica para gerenciamento da rede, uma interface com *ThingSpeak* para visualização dos dados de forma gráfica em tempo real e uma interface com um *bot* do *Telegram*, permitindo que usuários interajam com a rede através do *smartphone*.

A topologia criada para o projeto, definida na [Figura 24](#), possui uma estrutura em árvore com um nó coordenador e até 781 nós sensores. O nó coordenador é responsável por enviar comandos e solicitar leituras aos nós sensores da rede. Através da interface gráfica, o usuário é capaz de interagir com a rede solicitando informações dos sensores, desativando o envio de dados dos nós sensores ou configurar a frequência em que os nós sensores enviarão os dados.

Após o desenvolvimento, foram realizados dois diferentes tipos de testes em bancada para validação da plataforma. O primeiro teste contempla a execução de todos os comandos implementados para todos os nós da rede. O segundo teste consiste em simulações de falhas de energia entre os nós sensores da rede. Ambos os testes demonstraram o correto funcionamento da rede, tanto na execução de comandos quanto na recuperação à falhas de

energia dos nós.

Dada a diversidade em áreas em que a RSSF pode ser aplicada e pela constante evolução tecnológica, é extremamente importante o investimento em pesquisas por parte do Governo, indústrias e academias, para que a aplicação dessas tecnologias possa de fato agregar valor para a Sociedade.

O presente trabalho demonstrou a possibilidade no uso de dispositivos de baixo custo em uma RSSF para aplicação de monitoramento ambiental, sem perder a sua confiabilidade, flexibilidade e escalabilidade. Como o trabalho foi feito com componentes acessíveis, ferramentas gratuitas e de código aberto, trabalhos futuros podem ser realizados a fim de otimizar tanto a parte de *software* quanto a de *hardware*.

## 6.1 Dificuldades Encontradas

A principal dificuldade encontrada durante este trabalho se encontrou na definição da arquitetura e componentes que seriam utilizados para a RSSF. Esta dificuldade foi superada após realizar o levantamento bibliográfico, pois permitiu o entendimento de quais tecnologias estão sendo utilizadas para monitoramento com RSSF.

A segunda maior dificuldade foi encontrada com problemas técnicos durante a integração dos componentes e testes em bancada. Por algumas vezes, a ligação física foi feita de forma equivocada, sendo solucionada somente após conferir as tabelas criadas das ligações físicas, como visto na [Tabela 9](#). Houveram casos de mal contato entre fios, *protoboard* e também um módulo nRF24L01 com defeito, nestes casos, o problema foi solucionado isolando o erro até chegar ao componente danificado.

## 6.2 Trabalhos Futuros

É interessante que trabalhos futuros explorem por exemplo: o projeto de unidades de energia a fim de melhorar a autonomia dos nós sensores (Painéis solares e baterias), a exploração para uso de plataformas gratuitas para armazenamento e tratamento das informações dos sensores (Por exemplo: *Amazon Web Services*), confecção de placas de circuito para os nós da rede e para sensores próprios, construção de *cases* para os nós e entre outros.

## Referências

- ADAMS, J. et al. Consideration of security in telehealth wireless sensor network monitoring systems. In: ACTA PRESS. *The Third IASTED International Conference on Telehealth*. [S.l.], 2007. p. 57–60. Citado na página 36.
- ALCÁNTARA-AYALA, I. Geomorphology, natural hazards, vulnerability and prevention of natural disasters in developing countries. *Geomorphology*, Elsevier, v. 47, n. 2, p. 107–124, 2002. Citado na página 26.
- AOSONG. *Datasheet*. 2019. Disponível em: <<https://akizukidenshi.com/download/ds/aosong/DHT11.pdf>>. Citado na página 48.
- ARDUINO. *About Us*. 2019. Disponível em: <<https://www.arduino.cc/en/Main/AboutUs>>. Citado na página 43.
- ARDUINO. *Arduino*. 2019. Disponível em: <<https://www.arduino.cc/en/Main/AboutUs>>. Citado na página 43.
- ASHTON, K. That 'internet of things' thing. *RFID Journal*, v. 1, n. 1, 2010. Citado na página 35.
- CAMA, A. et al. Integration of communication technologies in sensor networks to monitor the amazon environment. *Journal of Cleaner Production*, Elsevier, v. 59, p. 32–42, 2013. Citado na página 36.
- CASTRO, A. L. C. *Glossário de defesa civil estudos de riscos e medicina de desastres*. [S.l.]: Brasília: Ministério da Integração Nacional, 2009. Citado na página 32.
- CASTRO, A. L. C. de et al. *Manual de desastres: Desastres naturais*. [S.l.]: Ministério do Planejamento e Orçamento, Secretaria Especial de Políticas Regionais, Departamento de Defesa Civil, 1996. Citado na página 26.
- CHELLI, K.; CHAVHAN, S. et al. Development of wireless sensor node to monitor poultry farm. In: *Mobile Communication and Power Engineering*. [S.l.]: Springer, 2013. p. 27–32. Citado na página 36.
- CIANCETTA, F. et al. A plug-n-play wireless sensor network based on web service for monitoring climatic parameters. In: IEEE. *Virtual Environments Human-Computer Interfaces and Measurement Systems (VECIMS), 2010 IEEE International Conference on*. [S.l.], 2010. p. 72–76. Citado na página 36.
- CIANCETTA, F. et al. An rfid plug-n-play smart sensors for monitoring forest fires. In: *Proc. IMEKO TC-4 and TC-19 Symposium and IWADC Instrumentation for the ICT Area, Kosice*. [S.l.: s.n.], 2010. Citado na página 36.
- CONCEIÇÃO, A. F. d. Plataforma de sensoriamento aberta, escalável de baixo custo para monitoramento e alerta de desastres naturais. 2015. Citado 2 vezes nas páginas 28 e 34.

- COOK, M.; MYERS, T.; TREVATHAN, J. A prototype home-based environmental monitoring system. *International Journal of Smart Home, Science and Engineering Research Support Society*, v. 7, p. 393–407, 2013. Citado na página 36.
- DALY, D.; MELIA, T.; BALDWIN, G. Concrete embedded rfid for way-point positioning. In: IEEE. *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*. [S.l.], 2010. p. 1–10. Citado na página 37.
- DELAMO, M. et al. Designing an open source maintenance-free environmental monitoring application for wireless sensor networks. *Journal of Systems and Software*, Elsevier, v. 103, p. 238–247, 2015. Citado 2 vezes nas páginas 33 e 36.
- DEMENTYEV, A. et al. Power consumption analysis of bluetooth low energy, zigbee and ant sensor nodes in a cyclic sleep scenario. In: IEEE. *Wireless Symposium (IWS), 2013 IEEE International*. [S.l.], 2013. p. 1–4. Citado na página 35.
- DORMER, M. *Choice of frequency band can really make a difference*. 2008. Disponível em: <<http://www.radiometrix.com/files/additional/choice-of-frequency-band-can-really-make-a-difference.pdf>>. Citado na página 39.
- FERDOUSH, S.; LI, X. Wireless sensor network system design using raspberry pi and arduino for environmental monitoring applications. *Procedia Computer Science*, Elsevier, v. 34, p. 103–110, 2014. Citado 3 vezes nas páginas 36, 38 e 39.
- FOUNDATION, R. P. *About us*. 2019. Disponível em: <<https://www.raspberrypi.org/about/>>. Citado na página 47.
- GENNARO, S. F. D. et al. An open-source and low-cost monitoring system for precision enology. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 14, n. 12, p. 23388–23397, 2014. Citado na página 36.
- GONZALEZ, A. et al. A networked embedded system for monitoring and control. *IFAC Proceedings Volumes*, Elsevier, v. 40, n. 1, p. 204–210, 2007. Citado na página 36.
- HSIAO, E. et al. Implementação em hardware de uma rede de sensores para monitoramento e alerta de desastres naturais. XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2017. Citado 2 vezes nas páginas 39 e 40.
- JANG, S. et al. Full-scale experimental validation of high-fidelity wireless measurement on a historic truss bridge. *Advances in Structural Engineering*, SAGE Publications Sage UK: London, England, v. 14, n. 1, p. 93–101, 2011. Citado na página 36.
- KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004. Citado na página 31.
- KOBIYAMA, M. et al. Papel da comunidade e da universidade no gerenciamento de desastres naturais. *Simpósio Brasileiro de Desastres Naturais*, v. 1, p. 834–846, 2004. Citado 2 vezes nas páginas 26 e 32.
- LEE, H.-C.; LIN, H.-H. Design and evaluation of an open-source wireless mesh networking module for environmental monitoring. *IEEE Sensors Journal*, IEEE, v. 16, n. 7, p. 2162–2171, 2016. Citado 2 vezes nas páginas 36 e 38.

- Licco, E. A.; Mac Dowell , S. F. Alagamentos, enchentes enxurradas e inundações: Digressões sobre seus impactos sócio econômicos e governança. v. 1, n. 1, 2015. Citado na página 26.
- LIN, H.-H. et al. An open-source wireless mesh networking module for environmental monitoring. In: IEEE. *Instrumentation and Measurement Technology Conference (I2MTC), 2015 IEEE International*. [S.l.], 2015. p. 1002–1007. Citado na página 36.
- LONDERO, P.; FAIRBANKS-HARRIS, T.; WHITMORE, P. M. An open-source, internet-of-things approach for remote sensing in museums. *Journal of the American Institute for Conservation*, Taylor & Francis, v. 55, n. 3, p. 166–175, 2016. Citado na página 36.
- LOUREIRO, A. A. et al. Redes de sensores sem fio. In: *Simpósio Brasileiro de Redes de Computadores*. [S.l.: s.n.], 2003. v. 21, p. 19–23. Citado na página 33.
- LUPU, E. et al. Evaluation of zigbee technology for a low-cost video surveillance system. In: SPRINGER. *International Conference on Advancements of Medicine and Health Care through Technology; 5th–7th June 2014, Cluj-Napoca, Romania*. [S.l.], 2014. p. 233–236. Citado na página 36.
- MAGRINI, E. *A Internet das Coisas*. [S.l.]: FGV EDITORA, 2018. v. 1. Citado na página 35.
- MAIA, M. C.; OLIVEIRA, S. C. Confecção de sistema embarcado dual-banda capaz de se comportar como um nó de uma rede de sensores sem fio. *Revista de Engenharia e Pesquisa Aplicada*, v. 1, n. 1, 2016. Citado na página 39.
- MING, X. et al. A wireless sensor system for long-term microclimate monitoring in wildland cultural heritage sites. In: IEEE. *Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on*. [S.l.], 2008. p. 207–214. Citado na página 37.
- MOORE, L. et al. Wireless network deployment as low cost building management system solution. In: *SMARTGREENS*. [S.l.: s.n.], 2013. p. 64–70. Citado na página 36.
- NATIONS, G. A. U. *Implementation of the international strategy for disaster reduction*. 2014. Disponível em: <<http://www.preventionweb.net/files/resolutions/N1452549.pdf>>. Citado na página 27.
- NIKHADE, S. Wireless sensor network system using raspberry pi and zigbee for environmental monitoring applications. In: . [s.n.], 2015. p. 376–381. Cited By 9. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-84993660502&doi=10.1109%2fICSTM.2015.7225445&partnerID=40&md5=5a065e324fdf9d4811383373cf44e6e8>>. Citado 2 vezes nas páginas 36 e 38.
- OLIVEIRA, S. de. *Internet das Coisas com ESP8266, ARDUINO e Raspberry Pi*. [S.l.]: Novatec, 2017. Citado na página 35.
- Ribeiro, J.; Vieira, R.; Tômio, D. Análise da percepção do risco de desastres naturais por meio da expressão gráfica de estudantes do projeto defesa civil na escola. *LDV-Forum*, v. 1, n. 1, 2017. Citado na página 27.

RUIZ, L. B. *Maná: uma arquitetura para gerenciamento de redes de sensores sem fio*. Tese (Doutorado) — Universidade Federal de Minas Gerais, 2003. Citado na página 33.

SAHA, H. et al. Comparative performance analysis between nrf24l01+ and xbee zb module based wireless ad-hoc networks. *International Journal of Computer Network and Information Security*, v. 9, p. 36–44, 07 2017. Citado na página 40.

SÁMANO-ROBLES, R.; GAMEIRO, A. Design of a low cost rfid platform with added value sensing capabilities for humanitarian relief applications. In: ACM. *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief*. [S.l.], 2011. p. 291–297. Citado na página 36.

UFSC, C. *Atlas Brasileiro de Desastres Naturais*. [S.l.: s.n.], 2013. v. 2. Citado na página 26.

WHELAN, M. J.; FUCHS, M. P.; JANOVAN, K. D. Large scale remote sensing for environmental monitoring of infrastructure. *Journal of Environmental Monitoring*, Royal Society of Chemistry, v. 10, n. 7, p. 812–816, 2008. Citado na página 36.

YANG, C.; SOH, C.; YAP, V. A non-intrusive appliance load monitoring for efficient energy consumption based on naive bayes classifier. *Sustainable Computing: Informatics and Systems*, v. 14, p. 34–42, 2017. Cited By 0. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85015994503&doi=10.1016%2fj.suscom.2017.03.001&partnerID=40&md5=556a585ae1ced5c2564f804c33257c53>>. Citado na página 37.

Zanella, A. et al. Internet of things for smart cities. *IEEE Internet of Things Journal*, v. 1, n. 1, p. 22–32, Feb 2014. ISSN 2372-2541. Citado na página 35.

# Apêndices



# APÊNDICE A – Configuração do Ambiente de Desenvolvimento

Este apêndice contém todas as instruções para realizar a configuração do ambiente de desenvolvimento para o correto funcionamento deste projeto. As instruções englobam as configurações necessárias no ambiente do *Raspberry Pi* e do computador que irá desenvolver e programar os códigos do Arduino.

Todas as bibliotecas que estão sendo utilizadas no projeto foram devidamente selecionadas levando em conta critérios como: código aberto, presença de documentação do código, compatibilidade entre as plataformas e suporte da comunidade.

Nesta seção será apresentado o passo a passo para instalação e configuração das bibliotecas do módulo do rádio nRF24L01+, sensor DHT11, API *ThingSpeak*, API *Telegram* e *Qt Designer*. Também será apresentada a funcionalidade e o principal papel de cada módulo presente neste projeto.

## A.0.1 *Raspberry Pi*

### A.0.1.1 Biblioteca RF24Network

Antes de instalar a biblioteca no *Raspberry Pi*, certifique-se de que a interface SPI está ativa. O protocolo SPI é utilizado na comunicação entre o *Raspberry Pi* e o módulo nRF24L01+. Para verificar se a interface está ativa, siga os passos representados na [Figura 63](#), [Figura 64](#) e [Figura 65](#).

Para a instalação da biblioteca RF24 e RF24Network execute os seguintes comandos dentro do terminal. As seguintes instruções foram retiradas e traduzidas do [Github](#) do desenvolvedor da biblioteca <<https://github.com/nRF24/RF24Network>>.

Crie um repositório chamado rf24libs para salvar os arquivos e entre dentro dele:

```
1 $ mkdir ~/rf24libs
2 $ cd ~/rf24libs
```

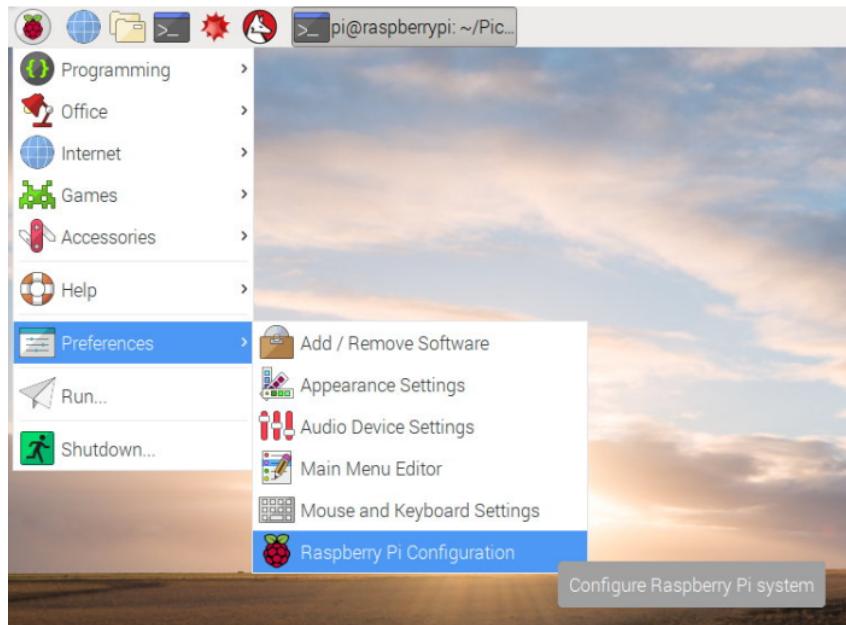
Clone o repositório RF24:

```
1 $ git clone https://github.com/tmrh20/RF24.git RF24
```

Navegue até o repositório e construa a biblioteca:

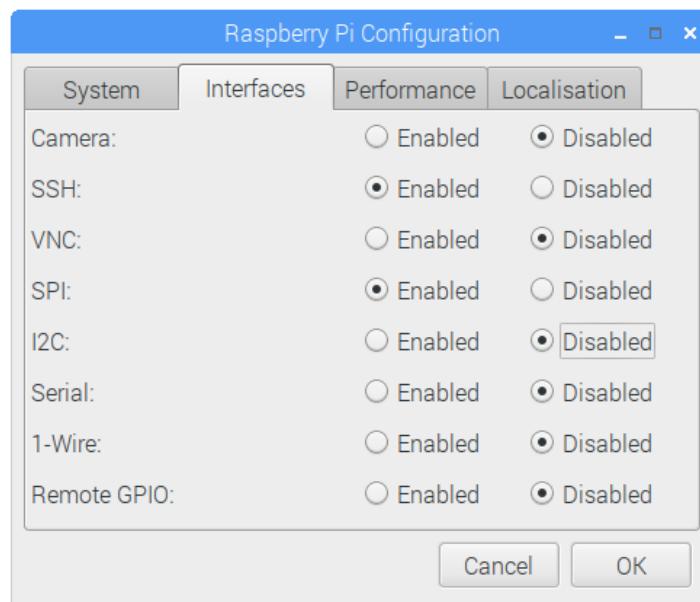
```
1 $ cd RF24
2 $ sudo make install
```

Figura 63 – Acessando as configurações do *Raspberry*



Fonte: O Autor

Figura 64 – Clique em *Enabled* para ativar o SPI



Fonte: O Autor

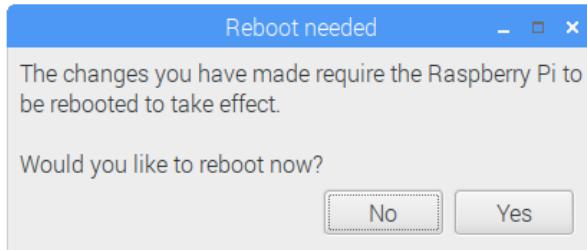
Navegue até o repositório rf24libs e clone RF24Network:

```
1 $ cd ~/rf24libs
2 $ git clone https://github.com/tmrh20/RF24Network.git RF24Network
```

Instale a biblioteca RF24Network:

```
1 $ sudo make install
```

Figura 65 – Clique em Yes para que a alteração seja aplicada



Fonte: O Autor

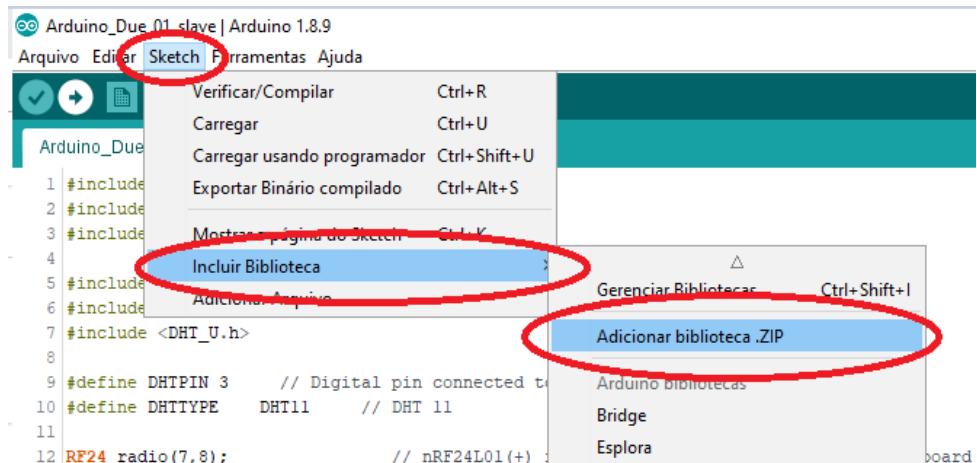
Feito isso, o módulo nRF24L01+ estará apto a ser utilizado através da biblioteca instalada no *Raspberry Pi*.

## A.0.2 Arduino

### A.0.2.1 Biblioteca *RF24Network*

Considerando que a IDE do Arduino já esteja instalada no computador, seguiremos com a instalação da biblioteca *RF24Network*, como pode ser consultado na Figura 66. Esta biblioteca pode ser obtida através do *Github* do desenvolvedor disponível em: <<https://github.com/nRF24/RF24Network/archive/master.zip>>.

Figura 66 – Instale a biblioteca no ambiente de desenvolvimento

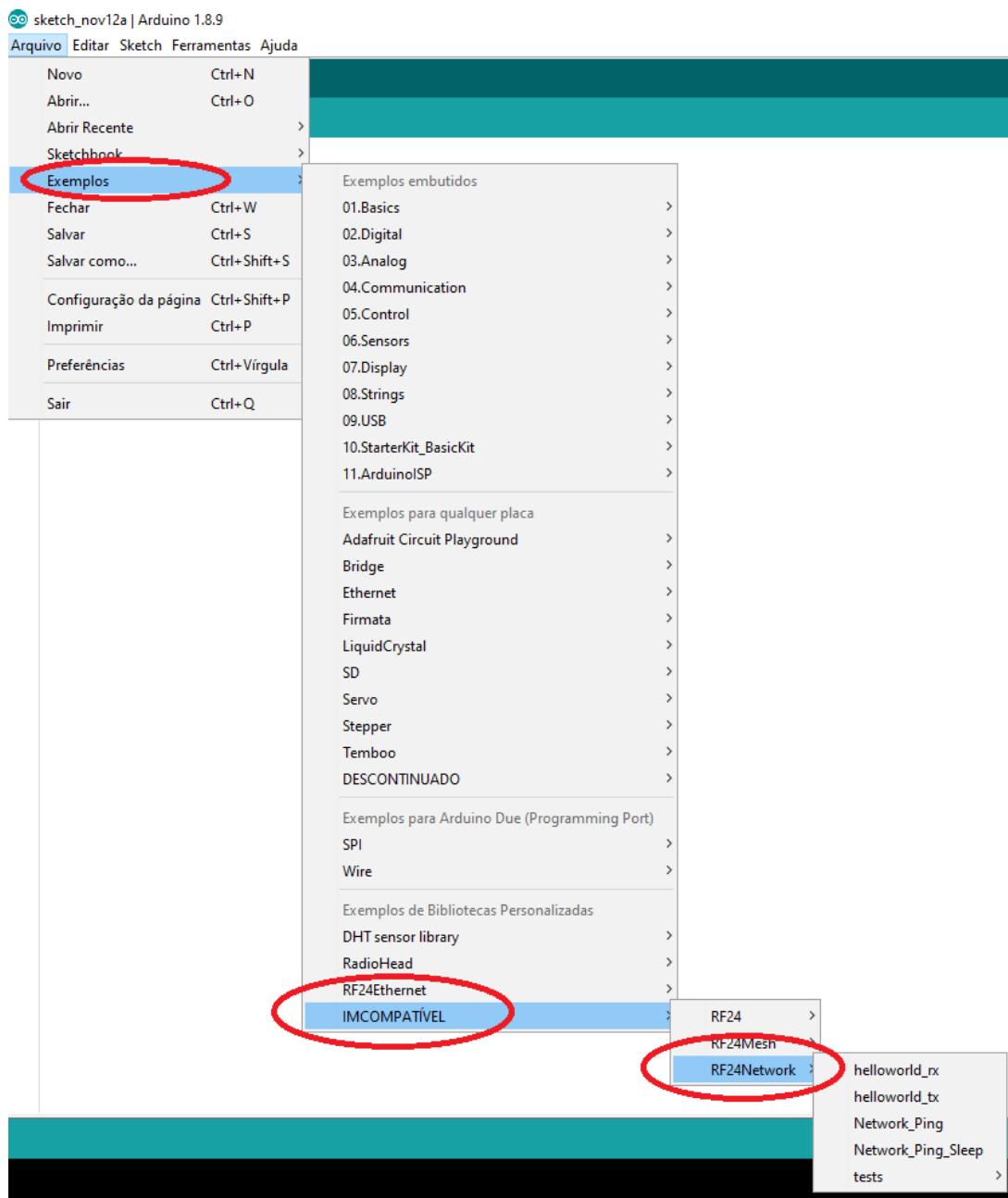


Fonte: O Autor

Feita a instalação, a biblioteca já estará apta para ser utilizada em todos os Arduinos (Uno, Nano e Due) por conta da compatibilidade entre os modelos da placa.

A mensagem de incompatibilidade apresentada na [Figura 67](#) aparentemente é um erro na IDE do Arduino. Os exemplos disponíveis podem ser testados a fim de validar a instalação e correto funcionamento da biblioteca para uso do módulo de rádio.

Figura 67 – Bibliotecas instaladas com sucesso



Fonte: O Autor

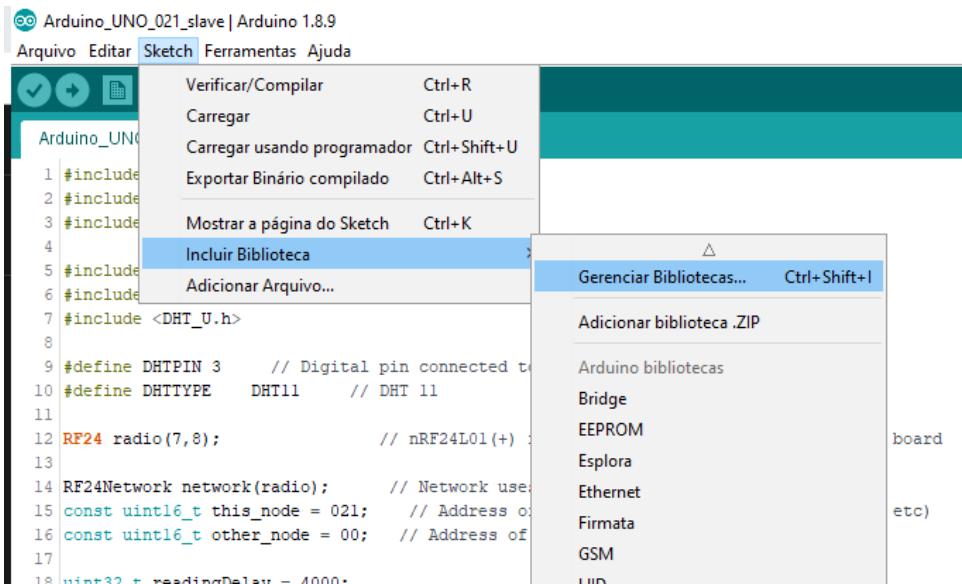
### A.0.2.2 Biblioteca Sensor - DHT11

Para ser possível realizar leituras do módulo sensor de temperatura e umidade DHT11, é necessário instalar a biblioteca chamada *DHT-sensor-library* disponível para *download* em <<https://github.com/adafruit/DHT-sensor-library>>.

Para a instalação da biblioteca do sensor no ambiente de desenvolvimento do Arduino é necessário realizar o *download* e salvar em formato comprimido .zip. Feito isso, dentro do ambiente de desenvolvimento, são necessários dois simples passos.

O primeiro passo é abrir o gerenciador de bibliotecas, mostrado na [Figura 68](#).

Figura 68 – Gerenciador de bibliotecas

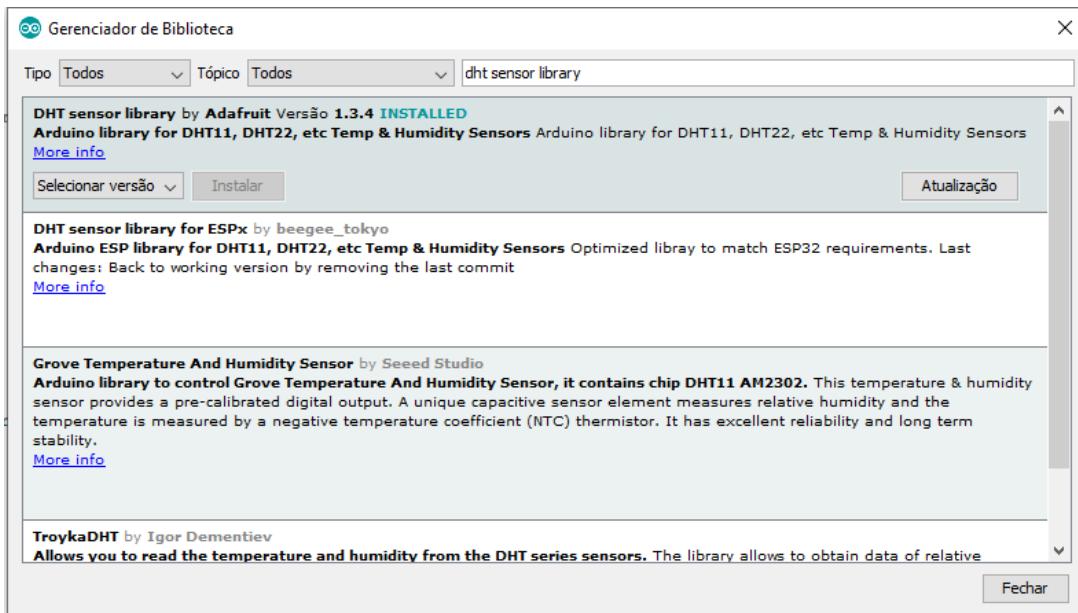


Fonte: O Autor

Já no segundo passo, é necessário primeiramente buscar pela biblioteca chamada *DHT sensor library* desenvolvido pela empresa *Adafruit*. Selecione a versão mais atual e por fim instale a biblioteca.

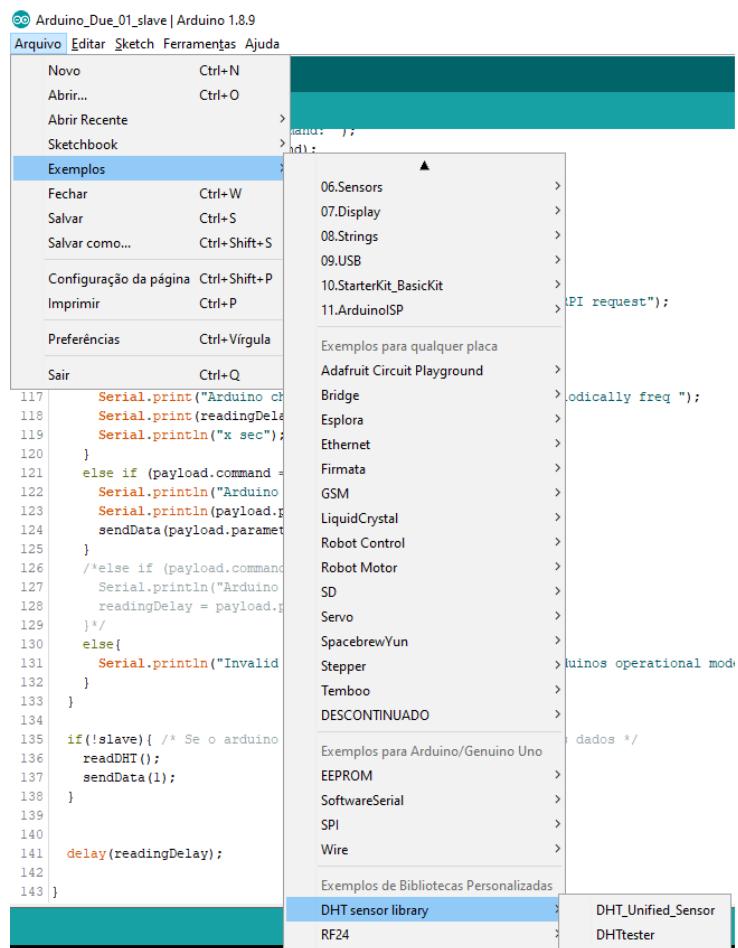
Para validar a instalação da biblioteca, é possível executar exemplos fornecidos pelo próprio desenvolvedor, como visto na [Figura 70](#).

Figura 69 – Instalação da biblioteca



Fonte: O Autor

Figura 70 – Códigos exemplos da biblioteca DHT11



Fonte: O Autor

# APÊNDICE B – Código *Raspberry Pi*

```

1 import sys
2 import threading
3 import time
4
5 from struct import *
6 from RF24 import *
7 from RF24Network import *
8
9 from datetime import datetime
10
11 from telegram.ext import Updater, InlineQueryHandler, CommandHandler
12 from PyQt5 import QtCore, QtGui, QtWidgets, uic
13 from PyQt5.QtCore import QTimer
14
15 import sys
16 from urllib.request import urlopen
17
18 # CE Pin, CSN Pin, SPI Speed
19 radio = RF24(RPI_V2_GPIO_P1_15, RPI_V2_GPIO_P1_24,
               BCM2835_SPI_SPEED_8MHZ)
20 network = RF24Network(radio)
21
22 octlit = lambda n:int(n, 8)
23
24 # Endereco do no coordenador 00
25 this_node = octlit("00")
26 # Endereco do filho do coordenador 01
27 other_node = octlit("01")
28
29 radio.begin()
30 time.sleep(0.1) # Delay para garantir que o radio foi inicializado
31 network.begin(110, this_node) # Configurando canal 110
32 radio.printDetails() # Imprimindo detalhes do radio
33
34 nodeAddrs = [1, 9, 17] # Enderecos em decimal dos nos da rede
35
36 #ThingSpeak API write key 02TR9UYCP211DHN2
37 thingSpeakAPI = '02TR9UYCP211DHN2'
38 baseURL = 'https://api.thingspeak.com/update?api_key=%s' % thingSpeakAPI
39
40 g_timeStamp = ''

```

```
41 g_statusTime = ''
42 g_output= ''
43 g_temp01 = 0
44 g_humid01 = 0
45 g_temp011 = 0
46 g_humid011 = 0
47 g_temp021 = 0
48 g_humid021 = 0
49
50 # Metodo para receber e decodificar o comando
51 def receiveDecodeCommand():
52     while True:
53         command, node , parameter = map(int, input("Command and
54                                         parameter: ").split())
55         #print('command: ', command, 'node: ', node, 'parameter: ',
56         parameter)
57         node = octlit(str(node))
58
59         if command == 1 and (node in nodeAddrs): # Configurando Arduino
60             como Slave
61             print('Comando para configurar Arduino: ', node, ' como
62             SLAVE')
63             sendCommand(command, node, 0)
64         elif command == 2 and (node in nodeAddrs): # Configurando
65             Arduino como mestre (ativo)
66             print('Comando para configurar Arduino: ', node, ' como
67             mestre (ativo)')
68             sendCommand(command, node, 0)
69         elif command == 10 and (node in nodeAddrs): # Solicitando x
70             leituras do Arduino
71             print('Solicita: ', parameter, ' leituras dos sensores do
72             Arduino: ', node)
73             sendCommand(command, node, parameter)
74         elif command == 20 and (node in nodeAddrs): # Alterando a
75             frequencia de leitura do Arduino
76             print('Alterar frequencia de leitura do Arduino: ', node,
77             ' para: ', parameter, 'seg')
78             sendCommand(command, node, parameter)
79         else:
80             print("Digite um Comando ou No valido")
81
82 # Metodo criado para enviar comandos
83 def sendCommand(command, node, parameter):
84     payload = pack('<LL', command, parameter)
85     ok = network.write(RF24NetworkHeader(node), payload)
86     if ok == False :
87         print('Nao foi possivel enviar comando ao Arduino: ', node)
```

```

78
79 # Metodo para receber dados
80 def receivePayload():
81     global g_temp01, g_humid01, g_temp011, g_humid011, g_temp021,
82         g_humid021, g_timeStamp
83     while True:
84         network.update()
85         while network.available():
86             header, payload = network.read(8) # frame de 8 bytes
87             temperature, humidity = unpack('<LL', bytes(payload)) # 1 byte de
88             temperatura e 1 de umidade
89             if header.from_node != 0:
90                 dateDateTimeObj = datetime.now()
91                 dateObj = dateDateTimeObj.date()
92                 timeObj = dateDateTimeObj.time()
93                 g_timeStamp = dateObj.strftime('%b %d %Y') + timeObj.strftime('
94                 %H:%M:%S - ')
95                 print(g_timeStamp + 'Received Temperature:\t', temperature, '\
96                 tHumidity: ', humidity, '\tfrom Node: ', oct(header.from_node))
97
98             if header.from_node == 1:
99                 g_temp01 = temperature
100                g_humid01 = humidity
101            elif header.from_node == 9:
102                g_temp011 = temperature
103                g_humid011 = humidity
104            elif header.from_node == 17:
105                g_temp021 = temperature
106                g_humid021 = humidity
107
108                g_output = str(g_timeStamp) + 'Node 01 Temperature ' + str(
109                g_temp01) + ' Humidity: ' + str(g_humid01) + '\n\n' + str(g_timeStamp)
110                + 'Node 011 Temperature ' + str(g_temp011) + ' Humidity: ' + str(
111                g_humid011) + '\n\n' + str(g_timeStamp) + 'Node 021 Temperature ' +
112                str(g_temp021) + ' Humidity: ' + str(g_humid021) + '\n\n'
113
114 # Interface Grafica
115 class Ui(QtWidgets.QMainWindow):
116     def __init__(self):
117         super(Ui, self).__init__()# Call the inherited classes __init__
118         method
119         uic.loadUi('TCC.ui', self) #Load the .ui file
120
121         self.sendBtn = self.findChild(QtWidgets.QPushButton, 'sendButton
122         ')
123         self.sendBtn.clicked.connect(self.sendButtonPressed)

```

```
115     self.readBtn = self.findChild(QtWidgets.QPushButton, 'readButton')
116
117     self.readBtn.clicked.connect(self.readButtonPressed)
118
118     self.commandRaspberryTxt = self.findChild(QtWidgets.QTextBrowser,
119 , 'commandRaspberryText')
119     self.commandTelegramTxt = self.findChild(QtWidgets.QTextBrowser,
120 , 'commandTelegramText')
120     self.outputTxt = self.findChild(QtWidgets.QTextBrowser, 'outputText')
121
122     self.cmdBox = self.findChild(QtWidgets.QComboBox, 'cmdBox')
123     self.nodeBox = self.findChild(QtWidgets.QComboBox, 'nodeBox')
124     self.paramBox = self.findChild(QtWidgets.QComboBox, 'parameterBox')
125
126     self.qTimer = QTimer()
127     self.qTimer.setInterval(5000)
128     self.qTimer.timeout.connect(self.readButtonPressed)
129     self.qTimer.start()
130
131     self.show() # Mostrar a interface
132
133 # Metodo chamado quando botao send e pressionado
134 def sendButtonPressed(self):
135     if self.cmdBox.currentText() == 'Read':
136         cmdOutput = 'Requesting: ' + self.paramBox.currentText() + 'x sample from Arduino: ' + self.nodeBox.currentText()
137         sendCommand(10, int(self.nodeBox.currentText()), int(self.paramBox.currentText()))
138     elif self.cmdBox.currentText() == 'Slave':
139         cmdOutput = 'Setting Arduino: ' + self.nodeBox.currentText() + ' as Slave'
140         sendCommand(1, int(self.nodeBox.currentText()), 0)
141     elif self.cmdBox.currentText() == 'Master':
142         cmdOutput = 'Setting Arduino: ' + self.nodeBox.currentText() + ' as Master' + ' with sampling frequency of ' + self.paramBox.currentText() + ' seconds'
143         sendCommand(2, int(self.nodeBox.currentText()), int(self.paramBox.currentText()))
144         self.commandRaspberryTxt.setText(cmdOutput)
145
146 # Metodo chamado quando o botao read e pressionado
147 def readButtonPressed(self):
148     global g_temp01, g_humid01, g_temp011, g_humid011, g_temp021,
g_humid021, g_timeStamp, g_output, g_statusTime
```

```

149     g_output = '\n\n' + str(g_timeStamp) + 'Node 01 Temperature ' +
150     str(g_temp01) + ' Humidity: ' + str(g_humid01) + '\n\n' + str(
151     g_timeStamp) + 'Node 011 Temperature ' + str(g_temp011) + ' Humidity:
152     ' + str(g_humid011) + '\n\n' + str(g_timeStamp) + 'Node 021
153     Temperature ' + str(g_temp021) + ' Humidity: ' + str(g_humid021) + '\n
154     \n'
155     self.outputText.setText(g_output)
156     self.commandTelegramText.setText('\nLast Status request from
157     Telegram: \n\n' + g_statusTime)
158
159
160 # Metodo para retornar log de horario
161 def status(update, context):
162     global g_statusTime
163     dateDateTimeObj = datetime.now()
164     dateObj = dateDateTimeObj.date()
165     timeObj = dateDateTimeObj.time()
166     g_statusTime = dateObj.strftime('%b %d %Y ') + timeObj.strftime(
167         '%H:%M:%S')
168     update.message.reply_text(g_output)
169
170 # Metodo inicializador da GUI
171 def GUI():
172     app = QtWidgets.QApplication(sys.argv)
173     window = Ui()
174     app.exec_()
175
176
177 # Metodo que envia os dados para o ThingSpeak
178 def thingSpeak():
179     global g_temp01, g_humid01, g_temp011, g_humid011, g_temp021,
180     g_humid021
181     while True:
182         conn = urlopen(baseURL + '&field1=%s&field2=%s&field3=%s&field4
183         =%s&field5=%s&field6=%s' % (g_temp01, g_humid01, g_temp011,
184         g_humid011, g_temp021, g_humid021))
185         #print (conn.read())
186         conn.close()
187         time.sleep(20)
188
189 def main():
190     #Thread para receber comandos do usuario por linha de comando
191     #input_thread = threading.Thread(target = receiveDecodeCommand)
192     #input_thread.daemon = True
193     #input_thread.start()
194
195     #ThingSpeak Thread
196     thingSpeak_thread = threading.Thread(target = thingSpeak)

```

```
186     thingSpeak_thread.daemon = True
187     thingSpeak_thread.start()
188
189     #Thread para receber dados
190     receive_thread = threading.Thread(target = receivePayload)
191     receive_thread.daemon = True
192     receive_thread.start()
193
194     #GUI Thread
195     gui_thread = threading.Thread(target = GUI)
196     gui_thread.daemon = True
197     gui_thread.start()
198
199     #Thread do Telegram
200     updater = Updater('939667550:AAG1IpIILnWjyjvcGQEULCCG_6AUNFFNZlg',
201     use_context = True)
202     updater.dispatcher.add_handler(CommandHandler('status', status))
203     updater.start_polling()
204
205     while True:
206         pass
207
208 if __name__ == "__main__":
    main()
```

Listing B.1 – Código Raspberry Pi

# APÊNDICE C – Código Arduino Uno

```

1 /* Bibliotecas para o radio */
2 #include <RF24Network.h>
3 #include <RF24.h>
4 #include <SPI.h>
5
6 /* Bibliotecas para sensor DHT11 */
7 #include <Adafruit_Sensor.h>
8 #include <DHT.h>
9 #include <DHT_U.h>
10
11 /* Entrada digital onde o sensor DHT11 esta conectado */
12 #define DHTPIN 3
13 /* DHT11 */
14 #define DHTTYPE      DHT11
15
16 /* Configuracao ligacao fisica */
17 RF24 radio(7,8);
18
19 RF24Network network(radio);
20 const uint16_t this_node = 021;      /* Endereco deste no sensor */
21 const uint16_t other_node = 00;      /* Endereco do no coordenador */
22
23 uint32_t readingDelay = 4000; /* Delay de leitura do sensor */
24
25 /* Estrutura do comando */
26 struct payload_t {
27     unsigned long command;
28     unsigned long parameter;
29 };
30
31 /* Estrutura dos dados*/
32 struct payload_to_send{
33     unsigned long temperature;
34     unsigned long humidity;
35 };
36
37 /* DHT11 */
38 DHT_Unified dht(DHTPIN, DHTTYPE);
39 uint32_t delayMS;
40 /* Estrutura do payload com temperatura e umidade */
41 payload_to_send payload_sensor = {30, 90};

```

```
42 /* Por padrao o Arduino e inicializado sempre como mestre (ativo) */
43 boolean slave = false;
44
45 /* Inicializando o sensor */
46 void initDHT(){
47     dht.begin();
48     sensor_t sensor;
49     dht.temperature().getSensor(&sensor);
50     dht.humidity().getSensor(&sensor);
51     delayMS = sensor.min_delay / 1000;
52 }
53
54 /* Metodo para realizar leituras do sensor */
55 void readDHT(){
56     sensors_event_t event;
57     dht.temperature().getEvent(&event);
58
59     /* Lendo a temperatura do sensor */
60     payload_sensor.temperature = event.temperature;
61     if (isnan(event.temperature)) {
62         Serial.println(F("Erro ao ler temperatura!"));
63     }
64
65     /* Lendo a umidade do sensor */
66     dht.humidity().getEvent(&event);
67     payload_sensor.humidity = event.relative_humidity;
68     if (isnan(event.relative_humidity)) {
69         Serial.println(F("Erro ao ler umidade!"));
70     }
71 }
72
73 /* Metodo criado para enviar os dados ao coordenador */
74 void sendData(uint8_t samples){
75     int i=0;
76     RF24NetworkHeader header2(other_node);
77     Serial.print("Enviando dado do no 01 para 00 ..");
78
79     for(i = 0; i < samples; i++){
80         bool ok = network.write(header2,&payload_sensor,sizeof(
81             payload_sensor));
82         if (ok){
83             Serial.print(" Temp: ");
84             Serial.print(payload_sensor.temperature);
85             Serial.print(" Umidade: ");
86             Serial.println(payload_sensor.humidity);
87         }
88     }
89 }
```

```
88     Serial.println(" falhou..");
89     delay(100);
90 }
91 }
92
93 void setup(void){
94     /* Inicializando a porta serial para debug */
95     Serial.begin(115200);
96     /* Inicializando SPI para comunicacao com sensor */
97     SPI.begin();
98     /* Inicializando radio */
99     radio.begin();
100    /* Selecionando o canal a ser utilizado e o endereoco do no */
101    network.begin(110, this_node);
102    /* Inicializando o sensor DHT11 */
103    initDHT();
104 }
105
106 /* Main loop */
107 void loop(void){
108     /* Atualizacao da rede */
109     network.update();
110
111     /* Verificando se existe pacote a ser lido */
112     while (network.available()){
113         RF24NetworkHeader headerR;
114         payload_t payload;
115         network.read(headerR,&payload,sizeof(payload));
116
117         Serial.print("Comando recebido: ");
118         Serial.print(payload.command);
119         Serial.print(" parametro: ");
120         Serial.print(payload.parameter);
121         Serial.print(" de: ");
122         Serial.println(headerR.from_node);
123
124     /* Comando para entrar em modo escravo (passivo) */
125     if (payload.command == 1){
126         slave = true;
127         Serial.println("Arduindo mudando para modo escravo (passivo) - responde por requisicoes RPI");
128     }
129     /* Comando para entrar em modo mestre (ativo) */
130     else if (payload.command == 2) {
131         slave = false;
132         readingDelay = payload.parameter * 1000; /* Convertendo segundos para milisegundos */
```

```
133     Serial.print("Arduino mudando para modo mestre (ativo) - Envia  
134     periodicamente a cada ");  
135     Serial.print(readingDelay);  
136     Serial.println("x segundos");  
137 /* Respondendo requisicao de leitura caso esteja no modo escravo (passivo) */  
138 else if (payload.command == 10 && slave) {  
139     Serial.println("Arduino recebeu requisicao de leitura do RPI");  
140     Serial.println(payload.parameter);  
141     sendData(payload.parameter);  
142 }  
143 else{ /* Caso contrario, ignorar comando */  
144     Serial.println("Comando invalido!");  
145 }  
146 }  
147 /* Se o arduino for mestre (ativo), enviar periodicamente os dados */  
148 if(!slave){  
149     readDHT();  
150     sendData(1);  
151 }  
152 delay(readingDelay);  
153 }  
154 }
```

Listing C.1 – Código Arduino Uno

# APÊNDICE D – Código Arduino Due

```

1 /* Bibliotecas para o radio */
2 #include <RF24Network.h>
3 #include <RF24.h>
4 #include <SPI.h>
5
6 /* Bibliotecas para sensor DHT11 */
7 #include <Adafruit_Sensor.h>
8 #include <DHT.h>
9 #include <DHT_U.h>
10
11 /* Entrada digital onde o sensor DHT11 esta conectado */
12 #define DHTPIN 3
13 /* DHT11 */
14 #define DHTTYPE      DHT11
15
16 /* Configuracao ligacao fisica */
17 RF24 radio(7,8);
18
19 RF24Network network(radio);
20 const uint16_t this_node = 01;      /* Endereco deste no sensor */
21 const uint16_t other_node = 00;     /* Endereco do no coordenador */
22
23 uint32_t readingDelay = 3200; /* Delay de leitura do sensor */
24
25 /* Estrutura do comando */
26 struct payload_t {
27   unsigned long command;
28   unsigned long parameter;
29 };
30
31 /* Estrutura dos dados*/
32 struct payload_to_send{
33   unsigned long temperature;
34   unsigned long humidity;
35 };
36
37 /* DHT11 */
38 DHT_Unified dht(DHTPIN, DHTTYPE);
39 uint32_t delayMS;
40 /* Estrutura do payload com temperatura e umidade */
41 payload_to_send payload_sensor = {30, 90};

```

```
42 /* Por padrao o Arduino e inicializado sempre como mestre (ativo) */
43 boolean slave = false;
44
45 /* Inicializando o sensor */
46 void initDHT(){
47     dht.begin();
48     sensor_t sensor;
49     dht.temperature().getSensor(&sensor);
50     dht.humidity().getSensor(&sensor);
51     delayMS = sensor.min_delay / 1000;
52 }
53
54 /* Metodo para realizar leituras do sensor */
55 void readDHT(){
56     sensors_event_t event;
57     dht.temperature().getEvent(&event);
58
59     /* Lendo a temperatura do sensor */
60     payload_sensor.temperature = event.temperature;
61     if (isnan(event.temperature)) {
62         Serial.println(F("Erro ao ler temperatura!"));
63     }
64
65     /* Lendo a umidade do sensor */
66     dht.humidity().getEvent(&event);
67     payload_sensor.humidity = event.relative_humidity;
68     if (isnan(event.relative_humidity)) {
69         Serial.println(F("Erro ao ler umidade!"));
70     }
71 }
72
73 /* Metodo criado para enviar os dados ao coordenador */
74 void sendData(uint8_t samples){
75     int i=0;
76     RF24NetworkHeader header2(other_node);
77     Serial.print("Enviando dado do no 01 para 00 ..");
78
79     for(i = 0; i < samples; i++){
80         bool ok = network.write(header2,&payload_sensor,sizeof(
81             payload_sensor));
82         if (ok){
83             Serial.print(" Temp: ");
84             Serial.print(payload_sensor.temperature);
85             Serial.print(" Umidade: ");
86             Serial.println(payload_sensor.humidity);
87         }
88     }
89 }
```

```
88     Serial.println(" falhou..");
89     delay(100);
90 }
91 }
92
93 void setup(void){
94     /* Inicializando a porta serial para debug */
95     Serial.begin(115200);
96     /* Inicializando SPI para comunicacao com sensor */
97     SPI.begin();
98     /* Inicializando radio */
99     radio.begin();
100    /* Selecionando o canal a ser utilizado e o endereoco do no */
101    network.begin(110, this_node);
102    /* Inicializando o sensor DHT11 */
103    initDHT();
104 }
105
106 /* Main loop */
107 void loop(void){
108     /* Atualizacao da rede */
109     network.update();
110
111     /* Verificando se existe pacote a ser lido */
112     while (network.available()){
113         RF24NetworkHeader headerR;
114         payload_t payload;
115         network.read(headerR,&payload,sizeof(payload));
116
117         Serial.print("Comando recebido: ");
118         Serial.print(payload.command);
119         Serial.print(" parametro: ");
120         Serial.print(payload.parameter);
121         Serial.print(" de: ");
122         Serial.println(headerR.from_node);
123
124     /* Comando para entrar em modo escravo (passivo) */
125     if (payload.command == 1){
126         slave = true;
127         Serial.println("Arduino mudando para modo escravo (passivo) - responde por requisicoes RPI");
128     }
129     /* Comando para entrar em modo mestre (ativo) */
130     else if (payload.command == 2) {
131         slave = false;
132         readingDelay = payload.parameter * 1000; /* Convertendo segundos para milisegundos */
```

```
133     Serial.print("Arduino mudando para modo mestre (ativo) - Envia  
134     periodicamente a cada ");  
135     Serial.print(readingDelay);  
136     Serial.println("x segundos");  
137 /* Respondendo requisicao de leitura caso esteja no modo escravo (passivo) */  
138 else if (payload.command == 10 && slave) {  
139     Serial.println("Arduino recebeu requisicao de leitura do RPI");  
140     Serial.println(payload.parameter);  
141     sendData(payload.parameter);  
142 }  
143 else{ /* Caso contrario, ignorar comando */  
144     Serial.println("Comando invalido!");  
145 }  
146 }  
147 /* Se o arduino for mestre (ativo), enviar periodicamente os dados */  
148 if(!slave){  
149     readDHT();  
150     sendData(1);  
151 }  
152 delay(readingDelay);  
153 }  
154 }
```

Listing D.1 – Código Arduino Due

# APÊNDICE E – Código Arduino Nano

```

1 /* Bibliotecas para o radio */
2 #include <RF24Network.h>
3 #include <RF24.h>
4 #include <SPI.h>
5
6 /* Bibliotecas para sensor DHT11 */
7 #include <Adafruit_Sensor.h>
8 #include <DHT.h>
9 #include <DHT_U.h>
10
11 /* Entrada digital onde o sensor DHT11 esta conectado */
12 #define DHTPIN 3
13 /* DHT11 */
14 #define DHTTYPE      DHT11
15
16 /* Configuracao ligacao fisica */
17 RF24 radio(7,8);
18
19 RF24Network network(radio);
20 const uint16_t this_node = 011;      /* Endereco deste no sensor */
21 const uint16_t other_node = 00;      /* Endereco do no coordenador */
22
23 uint32_t readingDelay = 3600; /* Delay de leitura do sensor */
24
25 /* Estrutura do comando */
26 struct payload_t {
27     unsigned long command;
28     unsigned long parameter;
29 };
30
31 /* Estrutura dos dados*/
32 struct payload_to_send{
33     unsigned long temperature;
34     unsigned long humidity;
35 };
36
37 /* DHT11 */
38 DHT_Unified dht(DHTPIN, DHTTYPE);
39 uint32_t delayMS;
40 /* Estrutura do payload com temperatura e umidade */
41 payload_to_send payload_sensor = {30, 90};

```

```
42 /* Por padrao o Arduino e inicializado sempre como mestre (ativo) */
43 boolean slave = false;
44
45 /* Inicializando o sensor */
46 void initDHT(){
47     dht.begin();
48     sensor_t sensor;
49     dht.temperature().getSensor(&sensor);
50     dht.humidity().getSensor(&sensor);
51     delayMS = sensor.min_delay / 1000;
52 }
53
54 /* Metodo para realizar leituras do sensor */
55 void readDHT(){
56     sensors_event_t event;
57     dht.temperature().getEvent(&event);
58
59     /* Lendo a temperatura do sensor */
60     payload_sensor.temperature = event.temperature;
61     if (isnan(event.temperature)) {
62         Serial.println(F("Erro ao ler temperatura!"));
63     }
64
65     /* Lendo a umidade do sensor */
66     dht.humidity().getEvent(&event);
67     payload_sensor.humidity = event.relative_humidity;
68     if (isnan(event.relative_humidity)) {
69         Serial.println(F("Erro ao ler umidade!"));
70     }
71 }
72
73 /* Metodo criado para enviar os dados ao coordenador */
74 void sendData(uint8_t samples){
75     int i=0;
76     RF24NetworkHeader header2(other_node);
77     Serial.print("Enviando dado do no 01 para 00 ..");
78
79     for(i = 0; i < samples; i++){
80         bool ok = network.write(header2,&payload_sensor,sizeof(
81             payload_sensor));
82         if (ok){
83             Serial.print(" Temp: ");
84             Serial.print(payload_sensor.temperature);
85             Serial.print(" Umidade: ");
86             Serial.println(payload_sensor.humidity);
87         }
88     }
89 }
```

```

88     Serial.println(" falhou..");
89     delay(100);
90 }
91 }
92
93 void setup(void){
94     /* Inicializando a porta serial para debug */
95     Serial.begin(115200);
96     /* Inicializando SPI para comunicacao com sensor */
97     SPI.begin();
98     /* Inicializando radio */
99     radio.begin();
100    /* Selecionando o canal a ser utilizado e o endereoco do no */
101    network.begin(110, this_node);
102    /* Inicializando o sensor DHT11 */
103    initDHT();
104 }
105
106 /* Main loop */
107 void loop(void){
108     /* Atualizacao da rede */
109     network.update();
110
111     /* Verificando se existe pacote a ser lido */
112     while (network.available()){
113         RF24NetworkHeader headerR;
114         payload_t payload;
115         network.read(headerR,&payload,sizeof(payload));
116
117         Serial.print("Comando recebido: ");
118         Serial.print(payload.command);
119         Serial.print(" parametro: ");
120         Serial.print(payload.parameter);
121         Serial.print(" de: ");
122         Serial.println(headerR.from_node);
123
124     /* Comando para entrar em modo escravo (passivo) */
125     if (payload.command == 1){
126         slave = true;
127         Serial.println("Arduino mudando para modo escravo (passivo) -");
128         responde por requisici o RPI");
129     }
130     /* Comando para entrar em modo mestre (ativo) */
131     else if (payload.command == 2) {
132         slave = false;
133         readingDelay = payload.parameter * 1000; /* Convertendo segundos
134         para milisegundos */

```

```
133     Serial.print("Arduino mudando para modo mestre (ativo) - Envia  
134     periodicamente a cada ");  
135     Serial.print(readingDelay);  
136     Serial.println("x segundos");  
137 /* Respondendo requisicao de leitura caso esteja no modo escravo (passivo) */  
138 else if (payload.command == 10 && slave) {  
139     Serial.println("Arduino recebeu requisicao de leitura do RPI");  
140     Serial.println(payload.parameter);  
141     sendData(payload.parameter);  
142 }  
143 else{ /* Caso contrario, ignorar comando */  
144     Serial.println("Comando invalido!");  
145 }  
146 }  
147 /* Se o arduino for mestre (ativo), enviar periodicamente os dados */  
148 if(!slave){  
149     readDHT();  
150     sendData(1);  
151 }  
152 delay(readingDelay);  
153 }  
154 }
```

Listing E.1 – Código Arduino Nano

# APÊNDICE F – Código Interface Gráfica

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">
3   <class>MainWindow</class>
4   <widget class="QMainWindow" name="MainWindow">
5     <property name="geometry">
6       <rect>
7         <x>0</x>
8         <y>0</y>
9         <width>795</width>
10        <height>465</height>
11      </rect>
12    </property>
13    <property name="windowTitle">
14      <string>TCC 2 - Daniel Hideaki Makita - RSSF para Monitoramento
15        Ambiental</string>
16    </property>
17    <property name="autoFillBackground">
18      <bool>false</bool>
19    </property>
20    <widget class="QWidget" name="centralwidget">
21      <widget class="QComboBox" name="commandBox">
22        <property name="geometry">
23          <rect>
24            <x>150</x>
25            <y>60</y>
26            <width>69</width>
27            <height>22</height>
28          </rect>
29        </property>
30        <item>
31          <property name="text">
32            <string>Slave</string>
33          </property>
34        </item>
35        <item>
36          <property name="text">
37            <string>Master</string>
38          </property>
39        </item>
40        <item>
41          <property name="text">
```

```
41      <string>Read</string>
42    </property>
43  </item>
44 </widget>
45 <widget class="QPushButton" name="sendButton">
46   <property name="geometry">
47     <rect>
48       <x>580</x>
49       <y>60</y>
50       <width>75</width>
51       <height>23</height>
52     </rect>
53   </property>
54   <property name="text">
55     <string>Send</string>
56   </property>
57 </widget>
58 <widget class="QLabel" name="commandLabel">
59   <property name="geometry">
60     <rect>
61       <x>60</x>
62       <y>60</y>
63       <width>71</width>
64       <height>21</height>
65     </rect>
66   </property>
67   <property name="font">
68     <font>
69       <family>Calibri</family>
70       <pointsize>12</pointsize>
71       <weight>75</weight>
72       <bold>true</bold>
73     </font>
74   </property>
75   <property name="text">
76     <string>Command</string>
77   </property>
78 </widget>
79 <widget class="QComboBox" name="nodeBox">
80   <property name="geometry">
81     <rect>
82       <x>310</x>
83       <y>60</y>
84       <width>69</width>
85       <height>22</height>
86     </rect>
87   </property>
```

```
88 <item>
89   <property name="text">
90     <string>01</string>
91   </property>
92 </item>
93 <item>
94   <property name="text">
95     <string>9</string>
96   </property>
97 </item>
98 <item>
99   <property name="text">
100    <string>17</string>
101   </property>
102 </item>
103 </widget>
104 <widget class="QLabel" name="nodeLabel">
105   <property name="geometry">
106     <rect>
107       <x>250</x>
108       <y>60</y>
109       <width>47</width>
110       <height>16</height>
111     </rect>
112   </property>
113   <property name="font">
114     <font>
115       <family>Calibri</family>
116       <pointsize>12</pointsize>
117       <weight>75</weight>
118       <bold>true</bold>
119     </font>
120   </property>
121   <property name="text">
122     <string>Node</string>
123   </property>
124 </widget>
125 <widget class="QTextBrowser" name="commandRaspberryText">
126   <property name="geometry">
127     <rect>
128       <x>40</x>
129       <y>120</y>
130       <width>331</width>
131       <height>121</height>
132     </rect>
133   </property>
134 </widget>
```

```
135 <widget class="QTextBrowser" name="outputText">
136   <property name="geometry">
137     <rect>
138       <x>420</x>
139       <y>120</y>
140       <width>331</width>
141       <height>281</height>
142     </rect>
143   </property>
144 </widget>
145 <widget class="QComboBox" name="parameterBox">
146   <property name="geometry">
147     <rect>
148       <x>490</x>
149       <y>60</y>
150       <width>41</width>
151       <height>22</height>
152     </rect>
153   </property>
154   <item>
155     <property name="text">
156       <string>1</string>
157     </property>
158   </item>
159   <item>
160     <property name="text">
161       <string>2</string>
162     </property>
163   </item>
164   <item>
165     <property name="text">
166       <string>3</string>
167     </property>
168   </item>
169   <item>
170     <property name="text">
171       <string>10</string>
172     </property>
173   </item>
174   <item>
175     <property name="text">
176       <string>20</string>
177     </property>
178   </item>
179   <item>
180     <property name="text">
181       <string>30</string>
```

```
182     </property>
183   </item>
184   <item>
185     <property name="text">
186       <string>60</string>
187     </property>
188   </item>
189 </widget>
190 <widget class="QLabel" name="parameterLabel">
191   <property name="geometry">
192     <rect>
193       <x>400</x>
194       <y>60</y>
195       <width>81</width>
196       <height>16</height>
197     </rect>
198   </property>
199   <property name="font">
200     <font>
201       <family>Calibri</family>
202       <pointsize>12</pointsize>
203       <weight>75</weight>
204       <bold>true</bold>
205     </font>
206   </property>
207   <property name="text">
208     <string>Parameter</string>
209   </property>
210 </widget>
211 <widget class="QLabel" name="CommandLabel">
212   <property name="geometry">
213     <rect>
214       <x>110</x>
215       <y>100</y>
216       <width>221</width>
217       <height>20</height>
218     </rect>
219   </property>
220   <property name="font">
221     <font>
222       <family>Calibri</family>
223       <pointsize>12</pointsize>
224       <weight>75</weight>
225       <bold>true</bold>
226     </font>
227   </property>
228   <property name="text">
```

```
229     <string>Commands from Raspberry Pi</string>
230   </property>
231 </widget>
232 <widget class="QLabel" name="outputLabel">
233   <property name="geometry">
234     <rect>
235       <x>530</x>
236       <y>100</y>
237       <width>131</width>
238       <height>16</height>
239     </rect>
240   </property>
241   <property name="font">
242     <font>
243       <family>Calibri</family>
244       <pointsize>12</pointsize>
245       <weight>75</weight>
246       <bold>true</bold>
247     </font>
248   </property>
249   <property name="text">
250     <string>Sensors Outputs</string>
251   </property>
252 </widget>
253 <widget class="QLabel" name="titleLabel">
254   <property name="geometry">
255     <rect>
256       <x>190</x>
257       <y>10</y>
258       <width>401</width>
259       <height>31</height>
260     </rect>
261   </property>
262   <property name="font">
263     <font>
264       <family>Calibri</family>
265       <pointsize>18</pointsize>
266       <weight>75</weight>
267       <bold>true</bold>
268     </font>
269   </property>
270   <property name="text">
271     <string>Raspberry Pi Wireless Sensor Network</string>
272   </property>
273 </widget>
274 <widget class="QLabel" name="CommandLabel_2">
275   <property name="geometry">
```

```
276     <rect>
277         <x>90</x>
278         <y>260</y>
279         <width>231</width>
280         <height>20</height>
281     </rect>
282 </property>
283 <property name="font">
284     <font>
285         <family>Calibri</family>
286         <pointsize>12</pointsize>
287         <weight>75</weight>
288         <bold>true</bold>
289     </font>
290 </property>
291 <property name="text">
292     <string>Commands from Telegram's Bot</string>
293 </property>
294 </widget>
295 <widget class="QTextBrowser" name="commandTelegramText">
296     <property name="geometry">
297         <rect>
298             <x>40</x>
299             <y>280</y>
300             <width>331</width>
301             <height>121</height>
302         </rect>
303     </property>
304 </widget>
305 <widget class="QPushButton" name="readButton">
306     <property name="geometry">
307         <rect>
308             <x>680</x>
309             <y>60</y>
310             <width>75</width>
311             <height>23</height>
312         </rect>
313     </property>
314     <property name="text">
315         <string>Read</string>
316     </property>
317 </widget>
318 </widget>
319 <widget class="QMenuBar" name="menubar">
320     <property name="geometry">
321         <rect>
322             <x>0</x>
```

```
323     <y>0</y>
324     <width>795</width>
325     <height>21</height>
326     </rect>
327   </property>
328   <widget class="QMenu" name="menuSave">
329     <property name="title">
330       <string>Save</string>
331     </property>
332     <addaction name="actionLog"/>
333   </widget>
334   <addaction name="menuSave"/>
335 </widget>
336   <action name="actionLog">
337     <property name="text">
338       <string>Log</string>
339     </property>
340   </action>
341 </widget>
342 <resources/>
343 <connections/>
344 </ui >
```

Listing F.1 – Código Interface Gráfica