



**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO CEARÁ
PRÓ-REITORIA DE ENSINO
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE
TELECOMUNICAÇÕES**

VALDINEI CARLOS OLIVEIRA

**INTERNET DAS COISAS: MONITORAMENTO REMOTO DE
SENSORES SEM FIO INTELIGENTES COM ZABBIX POR MEIO DE
UMA REDE LORA**

**FORTALEZA
2020**

VALDINEI CARLOS OLIVEIRA

INTERNET DAS COISAS: MONITORAMENTO REMOTO DE SENSORES SEM FIO INTELIGENTES COM ZABBIX POR MEIO DE UMA REDE LORA

Dissertação submetida à
Coordenação de Pós-Graduação
em Engenharia de Telecomunicações
do Instituto Federal do Ceará, como
requisito parcial para obtenção do
grau de Mestre em Engenharia de
Telecomunicações.

Área de concentração: Sistemas de
Comunicações

Orientador: Prof. Dr. José Wally
Mendonça Menezes

Fortaleza
2020

Dados Internacionais de Catalogação na Publicação

Instituto Federal do Ceará - IFCE

Sistema de Bibliotecas - SIBI

Ficha catalográfica elaborada pelo SIBI/IFCE, com os dados fornecidos pelo(a) autor(a)

O48i Oliveira, Valdinei Carlos.

Internet das coisas: monitoramento remoto de sensores sem fio inteligentes com Zabbix por meio de uma rede LoRa / Valdinei Carlos Oliveira. - 2020.

129 f. : il. color.

Dissertação (Mestrado) - Instituto Federal do Ceará, Mestrado em Engenharia de Telecomunicações, Campus Fortaleza, 2020.

Orientação: Prof. Dr. José Wally Mendonça Menezes.

1. Rede de sensores sem fio. 2. LoRa. 3. Monitoramento. 4. Coleta de dados. 5. Rádio-frequência. I. Título.

CDD 621.382

VALDINEI CARLOS OLIVEIRA

**INTERNET DAS COISAS: MONITORAMENTO REMOTO DE
SENSORES SEM FIO INTELIGENTES COM ZABBIX POR
MEIO DE UMA REDE LORA**

Dissertação de Mestrado apresentado à
coordenação do Curso de Pós-
Graduação em Engenharia de
Telecomunicações do Instituto Federal
do Ceará - como requisito parcial para
obtenção do título de Mestre em
Engenharia de Telecomunicações.

Aprovado em ____ de _____ de 2020.

Membros da Banca:

Prof. Dr. Aminadabe Barbosa de Sousa
Faculdade de Gestão & Negócios de Fortaleza - FGNF

Prof. Dr. Daniel Xavier Gouveia
Instituto Federal de Educação, Ciência e Tecnologia do Ceará – IFCE

Prof. Dr. Francisco José Alves de Aquino
Instituto Federal de Educação, Ciência e Tecnologia do Ceará – IFCE

Prof. Dr. José Wally Mendonça Menezes
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE

Dedico este trabalho a Deus, a minha esposa Nathalia e ao nosso primeiro filho, muito desejado, na esperança de que ele/ela venha com muita saúde.

Agradecimentos

A Deus, o Senhor dos Exércitos, Criador de todas as coisas, Autor e consumidor de nossa fé, primeiramente por ter me dado o dom da vida e ter proporcionado que eu pudesse vivê-la de uma maneira plena e com saúde, por ter me dado uma esposa e companheira para uma vida toda e dentro de alguns dias um filho, por ter me dado a possibilidade de trabalhar em algo que eu gosto e me realiza, por me ter dado a habilidade de perseverar, mesmo nos momentos que pareciam não haver mais saída alguma, por ter guardado a minha vida e a de minha família em vários outros momentos e por ter colocado no meu caminho, pessoas, algumas das quais eu não lembro sequer o nome, que colaboraram imensamente para que eu chegasse até aqui, apesar de todas as probabilidades, apontarem justamente o contrário.

A minha esposa Nathalia por ter suportado com paciência, toda essa caminhada iniciada lá em 2000, sempre ao meu lado, apoiando da melhor maneira possível, com o adendo de, mesmo no meio dessa situação incomum que vivenciamos, me presentear com o nosso primeiro filho, que a esta altura, está a poucas semanas de chegar. No decorrer desses 20 anos juntos, o mínimo que posso fazer é agradecer pelo esforço, pela compreensão, pelo companheirismo, por relegar até mesmos seus estudos para me acompanhar e pelo seu amor durante toda essa jornada.

Aos meus pais, por terem me passado valores e experiências, os quais eu carrego comigo até hoje e pelo amor e dedicação com o qual se debruçaram sobre a minha criação, que guardados os devidos aspectos e situações, eu espero passar aos meus filhos de maneira semelhante.

Ao Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE, extensivo aos seus quadros de técnicos e pessoal de apoio, por ter me proporcionado, inicialmente o conhecimento que eu precisava para o desempenho de minhas atividades no trabalho, e em seguida me mostrar que seria um local de descoberta de outros conhecimentos, bem além do que eu poderia imaginar naquele momento, que alcançaria um dia.

Ao professor José Wally Mendonça Menezes, por ter me orientado com paciência

nessa jornada, que se aproxima do seu encerramento, sendo inclusive uma das primeiras pessoas a me incentivar a iniciar o Mestrado, quando eu mesmo, achava que não tinha habilidade suficiente para aspirar tal título.

Ao professor Edson da Silva Almeida, por ter me auxiliado imensamente no decorrer da minha graduação, diante dos vários problemas que eu vivenciara, causados pela dificuldade de conciliar trabalho em regime integral e estudo, ainda mais, um curso de Engenharia.

Aos demais professores do IFCE, os quais sempre me apoiaram, colaborando para que eu conseguisse contornar as dificuldades encontradas, me tornando uma pessoa conhecedora do valor e da diferença que faz um bom professor na vida de um aluno.

Aos comandantes do 52º Centro de Telemática (52º CT), no período de 2015 a 2020, por sempre terem apoiado o meu estudo no IFCE, em especial ao Senhor General Carlos Eduardo da Mota Góes, que lá em 2015, acreditou que a capacitação do pessoal em ambientes externos ao 52º CT, seria de grande valia para a Instituição, autorizando que este militar iniciasse sua caminhada no IFCE, em um curso de Engenharia, agradeço inclusive, por ter intercedido para que este benefício permanecesse além do tempo do seu comando.

Aos meus chefes de divisão, chefes diretos, também no período de 2015 a 2020, por terem sido compreensivos no decorrer destes quase 06 anos, permitindo que eu me ausentasse em vários períodos para assistir aulas, fazer provas, apoiando e reforçando junto a chefia, o ganho obtido por meio de tal capacitação.

Aos meus companheiros de trabalho, superiores, pares e subordinados, por segurarem as pontas e me auxiliarem imensamente, quando eu não poderia mais fazê-lo sozinho. Nos momentos mais conturbados, que frequentemente ocorreram durante todo este período, mesmo nos menores atos, isso fez toda a diferença.

Não menos importante, a uma pessoa, que não é digna sequer de que eu cite seu nome nessas páginas, mas que, ao insinuar que eu deveria desistir do meu curso, dado as várias cadeiras que eu necessitava frequentar em outros cursos e em diferentes horários, na verdade renovou as minhas forças, para que continuasse e superasse as adversidades.

Por último, àqueles que por acaso eu tenha esquecido nessas breves palavras, fica também o meu agradecimento e a certeza de que em algum momento no tempo, vocês colaboraram para que essa vitória ocorresse. O meu muito obrigado a todos.

“Eu não falhei 10 mil vezes. Apenas encontrei 10 mil formas que não funcionam.”

Thomas Edison

Resumo

Esta dissertação tem por objetivo, apresentar o monitoramento de uma rede de sensores sem fio, instalada em *sites* remotos, *datacenters* e estações repetidoras, que estejam demasiado afastadas da sede, por meio do protocolo LoRa (Long Range), criado pela Semtech, utilizando hardware de baixo custo, em uma configuração que seja simples e portátil, buscando utilizar poucos protocolos de comunicação além do LoRa. O protocolo LoRa têm se destacado por funcionar em longas distâncias, ser altamente tolerante a ruídos e interferências, além do baixo consumo de energia. A Instituição no qual foi executado este projeto permitiu que se pudesse extrair todas as vantagens do LoRa por meio do monitoramento nos locais supracitados, da temperatura local, da tensão elétrica nos dispositivos de fornecimento autônomo de energia e controle de dispositivos legados através da comunicação máquina a máquina, via interação por meio do protocolo infra-vermelho. Os nós sensores realizam a coleta, enviam os dados a um nó agregador, que os armazena em seu cartão SD e os repassa a um segundo nó agregador, que funciona de maneira híbrida intermediando a transição da rede LoRa com a rede IP do servidor de aplicação, o qual se encontra armazenado localmente, proporcionando desse modo, um maior controle sobre os dados sensíveis da Instituição. Esse servidor de aplicação e monitoramento faz uso do software livre, reforçando ainda mais, a ideia de que Instituição tenha total liberdade acerca de como gerir as informações coletadas da maneira que melhor lhe aprouver, sem licenças, sem uso de *duty cycles* ou hospedagens em local externo. Esse trabalho então, busca através de uma pesquisa fundamentada, apresentar de uma maneira clara, concisa e por meio de exemplos verídicos, como todos estes eventos são coletados, registrados e enviados à sede, para que fiquem armazenados na aplicação de monitoramento, gerando conhecimento e memória acerca das atividades ocorridas nestes locais, justificando assim, a relevância desse projeto.

Palavras-chave: Rede de sensores sem fio. LoRa. Monitoramento. Coleta de dados. Rádio-frequência

Abstract

This dissertation aims to present the monitoring of a wireless sensor network, installed in remote sites, datacenters and repeater stations, which are too far from the headquarters, through the LoRa protocol (Long Range), created by Semtech, using hardware low cost, in a configuration that is simple and portable, seeking to use few communication protocols besides LoRa. The LoRa protocol has stood out for working over long distances, being highly tolerant of noise and interference, in addition to low energy consumption. The Institution in which this project was carried out allowed us to extract all the advantages of LoRa by monitoring the aforementioned locations, the local temperature, the electrical voltage in the autonomous power supply devices and the control of legacy devices through machine-to-machine communication. machine, via interaction via the infrared protocol. The sensor nodes perform the collection, send the data to an aggregator node, which stores it on your SD card and passes it on to a second aggregator node, which works in a hybrid way, intermediating the transition from the LoRa network to the IP network of the application server , which is stored locally, thus providing greater control over the Institution's sensitive data. This application server makes use of free software, further reinforcing the idea that the Institution has total freedom about how to manage the information collected in the way that best suits it, without licenses, without use of duty cycles or hosting in an external location . This work then seeks, through a well-founded research, to present in a clear, concise manner and through true examples how all these events are collected, recorded and sent to headquarters so that they are stored in the monitoring application, generating knowledge and memory about activities that took place in these places, thus justifying the relevance of this project.

Keywords: Wireless Sensor Network. LoRa. Monitoring. Data collect. Radio frequency.

Sumário

Lista de Figuras

Lista de Tabelas

Abreviações	17
1 Introdução	20
1.1 Motivação e objetivos	24
1.1.1 Objetivo Geral	24
1.1.2 Objetivos Específicos	25
1.2 Produção científica	25
1.3 Organização da Dissertação	26
2 Fundamentação Teórica	27
2.1 Sistemas de Monitoramento Rede	27
2.1.1 <i>Zabbix</i>	30
2.2 Sensores	32
2.2.1 Hardware básico de um sensor	33
2.2.2 Características dos sensores sem fio	34
2.2.3 Tipos de sensores	34
2.2.4 Redes de sensores sem fio (RSSF)	35
2.2.5 Arquitetura de comunicação das RSSF	35
2.2.6 Tipos de nós encontrados em uma RSSF	38

2.3	Microcontroladores	39
2.4	Consumo de Energia	41
2.5	Protocolos de Comunicação	44
2.5.1	Protocolos sem fio	45
2.5.2	Protocolos de baixo consumo de energia	45
2.5.3	<i>Sigfox</i>	48
2.5.4	LoRa	48
2.5.5	LoRaWAN	51
3	Materiais e Métodos	55
3.1	Estrutura	55
3.1.1	Princípio de funcionamento	55
3.1.2	Ferramentas de desenvolvimento e código-fonte	57
3.1.3	<i>Hardware</i> dos Nós	61
3.2	Instalação e locais de testes	66
3.3	Custos do projeto	67
4	Resultados	68
4.1	Protótipos	68
4.1.1	Nó agregador	68
4.1.2	Nó sensor	69
4.1.3	Nó integrador	70
4.1.4	Cuidados acerca da codificação dos protótipos	70
4.2	Testes realizados	71
4.2.1	Condições de realização dos testes	71
4.2.2	Teste de intensidade / qualidade do sinal - Teste RSSI/SNR . . .	75
4.2.3	Teste de perda de pacotes	78

4.2.4	Teste de consumo da bateria	80
4.2.5	Teste de duração da bateria	80
4.2.6	Teste de tensão e corrente fornecidos pelo painel solar	81
4.2.7	Teste de transmissão contínua e concorrência	81
4.2.8	Teste de Monitoramento das informações coletadas	83
4.2.9	Apresentação das informações no Sistema	84
5	Conclusões e Trabalhos Futuros	86
5.1	Trabalhos Futuros	87
	Referências Bibliográficas	88
	Apêndice A – Código-Fonte do Nó agregador - ESP 32	93
	Apêndice B – Código-fonte dos Nós Sensores - ESP 8266	106
	Apêndice C – Código-Fonte do Nó Integrador - ESP 8266	115
	Apêndice D – Código-Fonte do Nó 1 - Teste de Intensidade/Qualidade	120
	Apêndice E – Código-Fonte do Nó 2 - Teste de Intensidade/Qualidade	124

Lista de Figuras

Figura 1	Comandos de listagem dos componentes de <i>Hardware</i>	28
Figura 2	Mapa de monitoramento do <i>Zabbix</i>	31
Figura 3	Arquitetura do <i>Zabbix</i>	31
Figura 4	Visão geral de um Sensor Básico	33
Figura 5	Exemplo de uma RSSF	35
Figura 6	Arquitetura de uma RSSF	36
Figura 7	Pilha de Protocolos de uma RSSF	36
Figura 8	Tipos de nós em uma RSSF	39
Figura 9	Microcontrolador básico com seus sensores, atuadores e periféricos	39
Figura 10	Diagrama básico de Microcontrolador	40
Figura 11	Simbologia para célula e para bateria	41
Figura 12	Representação de uma célula Li-On	42
Figura 13	Sistema de Comunicação de dados	44
Figura 14	Comparativo entre tecnologias <i>wireless</i>	46
Figura 15	Cálculo de potência de um enlace	47
Figura 16	Topologia lógica Lora	49
Figura 17	Arquitetura de protocolos LoRa e LoRaWAN	49
Figura 18	O Frame LoRa	50
Figura 19	Pilha de protocolos LoRa e LoRaWAN	51
Figura 20	Formato do <i>frame</i> LoRaWAN com os tamanhos dos campos em bits	52

Figura 21	Distribuição espacial dos sensores monitorados	55
Figura 22	Diagrama de sequência do funcionamento e troca de mensagens	61
Figura 23	Estrutura dos nós sensores	61
Figura 24	Esp8266 NodeMcu	62
Figura 25	Esp32	63
Figura 26	Diagrama do Cartão SD	63
Figura 27	Visor Oled	64
Figura 28	Representação do módulo de energia	65
Figura 29	Diagrama do chip LoRa DRF1272f/1276f	65
Figura 30	Projeto do nó Agregador	68
Figura 31	Protótipo do nó Agregador	69
Figura 32	Projeto do nó Sensor	69
Figura 33	Projeto do nó Integrador	70
Figura 34	Protótipos utilizados no sistema de monitoramento	70
Figura 35	Teste Sala de comunicações - 16º andar	72
Figura 36	Teste entre DC 1 (à esquerda) e DC 2 (à direita)	72
Figura 37	Área geográfica dos testes <i>outdoor</i>	73
Figura 38	Local de instalação do nó agregador	74
Figura 39	Local de coleta externa	74
Figura 40	Local com vegetação	75
Figura 41	Teste <i>Indoor</i> RSSI	76
Figura 42	Teste <i>Indoor</i> SNR	77
Figura 43	Teste <i>Outdoor</i> RSSI	77
Figura 44	Teste <i>Outdoor</i> SNR	78
Figura 45	Teste <i>Indoor</i> de perda de pacotes	79
Figura 46	Teste de perda de pacotes <i>Outdoor</i>	79

Figura 47	Relação de tolerância SNR	80
Figura 48	Cartão SD do gateway com as informações dos nós sensores .	81
Figura 49	Teste de transmissão contínua e concorrência	82
Figura 50	Tela do nó integrador remetendo dados ao <i>Zabbix</i>	83
Figura 51	Monitoramento das informações	84
Figura 52	Mapa da Rede LoRa integrada com a Rede TCP/IP monitorada em tempo real	84

Lista de Tabelas

Tabela 1	Sensibilidade do receptor Semtech SX1276 LoRa em dBm em diferentes larguras de banda e fatores de propagação	51
Tabela 2	Tabela de custos	67

Abreviações

5G 5ª Geração

ANATEL Agência Nacional de Telecomunicações

ANS Acordo de Nível de Serviço

ARPA Advanced Research Projects Agency

BW Bandwidth

CAN Controller Area Network

CPU Central Process Unit

CR Code Rate

CRC Código de Redundância Cíclica

CSS Chirp Spread Spectrum

DC DataCenter

DSL Digital Subscriber Line

GPIO Generic Purpose Input/Output

GPL Generic Public License

HTML Hyper Text Markup Language

I2C Inter-Integrated Circuit

I2S Integrated Inter-IC Sound

ICMP Internet Control Message Protocol

ID identificação ou identity

IDE Integrated Development Environment

IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISM	Industrial, Scientific, Medical
LCD	Liquid Crystal Display
LED	Light Emissor Diode
Li-On	íons de Lítio
Li-Po	Lítio Polímero
LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power Wide Area Network
M2M	Machine to Machine
MAC	Media access Control
MHz	Mega Hertz
MQTT	MQ Telemetry Transport
NiCd	Níquel-Cádmio
NiMH	Níquel-Hidrogênio
Oled	Organic Light-Emitting Diode
P&G	Procter & Gamble
PWM	Pulse Wave Modulation
RF	Rádio Frequência
RFID	Identificação por radiofrequência
RSSF	Rede de Sensores Sem Fio
RSSI	Received Signal Strenght Indication

SF Spreading Factor

SMS Short Message System

SNMP Simple Network Management Protocol

SNR RelatÇão Sinal-Ruído

SPI Serial Peripheral Interface

SSH Security Shell

TCP Transmission Control Protocol

TTN The Things Network

UART Universal Asynchronous Receiver/Transmitter

UNB Ultra Narrow Band

V Volts

VPN Virtual Private Network

Wi-Fi Wireless Fidelity

CAPÍTULO 1

INTRODUÇÃO

"Não creio que haja uma emoção, mais intensa para um inventor do que ver suas criações funcionando. Essa emoção faz você esquecer de comer, de dormir, de tudo."

(Nicola Tesla)

*Internet*¹ das Coisas, termo comum hoje em dia, é uma tradução do termo *Internet of Things* (IoT). O especialista britânico em tecnologia Kevin Ashton, apresentou o termo IoT pela primeira vez em sua palestra na empresa *Procter & Gamble* (P&G) em 1999 (ASHTON, 2009). Na ocasião, ele afirmava que os computadores seriam capazes de obter dados do ambiente, sem intervenção humana e com a finalidade de otimizar atividades rotineiras.

A ITU (2012), em sua recomendação Y.2060, define Internet das Coisas como:

Infraestrutura mundial para a sociedade da informação que propicia a prestação de serviços avançados mediante a inter conexão de objetos físicos e virtuais devido a interação de diferentes tecnologias da informação e comunicação, presentes e futuras. (ITU, 2012, tradução nossa)²

Atzori, Iera e Morabito (2010) afirmam que o termo “*Internet das Coisas*” se refere a uma revolução tecnológica que tem como objetivo conectar, por meio de sensores, dispositivos usados do dia a dia, à rede mundial de computadores.

Magrani (2018) contudo, ressalta que existem fortes divergências em relação ao conceito de IoT, não havendo, portanto, um conceito único que possa ser considerado pacífico ou unânime.

Apesar das divergências, o termo IoT se consolidou (ASHTON, 2009) e tem crescido bastante. Para atender aos requisitos da IoT, é necessária uma infraestrutura

¹ Rede mundial de computadores que faz a ligação de centenas de computadores espalhados pelo mundo. Esses dispositivos fazem o armazenamento de informações e a transmissão entre as máquinas que estão interligadas na rede (KUROSE; ROSS, 2017).

² A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.

de Rede adequada que permita a integração entre os dispositivos. Comer (2015) afirma que a evolução dessa Rede foi um fator importante para o estabelecimento da IoT.

O desenvolvimento das redes de computadores seguiu um longo processo, suportado pelas diversas inovações tecnológicas que foram surgindo nesse intervalo de tempo. Pode-se destacar a Ethernet³, TCP/IP⁴, o Wi-Fi (*Wireless Fidelity*), a DSL (*Digital Subscriber Line*), o Bluetooth⁵, a Fibra óptica, o RFID⁶, o IPv6 (*Internet Protocol version 6*), dentre outras. Desse período, é possível destacar alguns marcos interessantes:

O conceito de transmissão sem fios e objetos coletando informações e cooperando entre si, vem de antes da Segunda Guerra Mundial. Em um artigo para uma revista, TESLA (1926) já se referia a algo que remetia à ideia atual de comunicação sem fio e de *Internet das Coisas*:

Quando a rede sem fio estiver implantada, a Terra inteira será convertida em um cérebro gigante, o que de fato, serão todas as coisas sendo partes orquestradas de um todo real. Nós seremos capazes de nos comunicar um com o outro instantaneamente, independente da distância. Não apenas isso, mas através da televisão e da telefonia nós poderemos ver e ouvir perfeitamente, como se estivéssemos frente a frente, mesmo estando a milhares de quilômetros; e os instrumentos através dos quais poderemos fazer isto, serão surpreendentemente simples, se comparados com o nosso telefone atual. Um homem poderá carregar um no bolso do colete. (TESLA, 1926, tradução nossa) ⁷

A partir dos anos 50, patrocinadas pelo Departamento de Defesa Norte Americano, as pesquisas em tecnologia começaram a evoluir, primeiro para suportar o esforço da Segunda Guerra Mundial e em seguida pelo início da corrida tecnológica de Rússia e Estados Unidos, no período da Guerra Fria (TANENBAUM; WETHERALL, 2011).

³Inventada em na década de 1970 por Bob Metcalfe e David Hoggs, usava um barramento coaxial para interconectar os nós. Padrão de rede local muito utilizado atualmente. (KUROSE; ROSS, 2017)

⁴Transmission Control Protocol, Protocolo inter-redes criado por Vinton Cerf e Robert Kahn na década de 70. (KUROSE; ROSS, 2017)

⁵Padrão sem fio para interconectar dispositivos utilizando rádios sem fios de curto alcance, baixa potência e baixo custo. O projeto foi denominado Bluetooth em homenagem a Harald Blaatand (Bluetooth) II (940-981), um rei Viking. (TANENBAUM; WETHERALL, 2011)

⁶O RFID (Identificação por radiofrequência) usa campos eletromagnéticos para reconhecer etiquetas anexadas a objetos. Estas etiquetas contém informações armazenadas. (WEIS, 2012)

⁷When wireless is perfectly applied the whole earth will be converted into a huge brain, which in fact it is, all things being particles of a real and rhythmic whole. We shall be able to communicate with one another instantly, irrespective of distance. Not only this, but through television and telephony we shall see and hear one another as perfectly as though we were face to face, despite intervening distances of thousands of miles; and the instruments through which we shall be able to do this will be amazingly simple compared with our present telephone. A man will be able to carry one in his vest pocket.

Nos anos 60, surge a primeira rede de computadores⁸ com o nome de ARPANET⁹. Inicialmente destinada às pesquisas militares e logo depois estendida às universidades Americanas (TANENBAUM; WETHERALL, 2011).

Em seguida, (Já) nos anos 70 foi criado o protocolo TCP/IP (CERF; KAHN, 1974) especificamente para manipular a comunicação entre redes interligadas (TANENBAUM; WETHERALL, 2011). Em 1983, o TCP/IP se torna o modelo oficial da *Internet* (KUROSE; ROSS, 2017).

No fim do século XX, a *Internet* oferecia boa parte do seu conteúdo através de páginas HTML (*Hyper Text Markup Language*) estáticas, que demoravam a carregar em razão da largura de banda ser pequena, principalmente no Brasil. Porém, com o avanço da tecnologia da infraestrutura de rede e com consequente aumento de banda, as páginas começaram a ficar mais dinâmicas e interativas (GABRIEL, 2010).

A partir do lançamento do *Iphone*, pela *Apple Inc*, em 29 de Junho de 2007, iniciou-se a revolução dos telefones inteligentes. A *Internet* já era difundida, mas seu acesso principal era por meio de computadores (*desktops*) e *notebooks* (IBGE, 2020).

Desde então, os fabricantes passaram a desenvolver modelos de telefones inteligentes, agora chamados de *Smartphones*, que na definição de Theoharidou, Mylonas e Gritzalis (2012) são:

Telefones celulares com recursos avançados, que executam um sistema operacional distinto, permitindo que os usuários estendam suas funcionalidades utilizando aplicativos de terceiros disponíveis em um repositório/loja de aplicativos. (THEOHARIDOU; MYLONAS; GRITZALIS, 2012, tradução nossa)¹⁰

O acesso sem fio em alta velocidade, a maior capacidade de processamento e a portabilidade foram fatores que contribuíram para a rápida hegemonia dos dispositivos portáteis (KUROSE; ROSS, 2017) (IBGE, 2020), oferecendo acesso quase instantâneo à informação (THEOHARIDOU; MYLONAS; GRITZALIS, 2012).

Com a mobilidade e a conectividade se tornando comum, a *Internet* das coisas começa a se tornar realidade. Ashton (2009), afirma que a limitação de tempo e da rotina faria com que as pessoas se conectassem à *Internet* de outras maneiras.

⁸Conjunto de máquinas interligadas que conversam e executam tarefas entre si (EDWARDS, 1996)

⁹Era uma rede pertencente à ARPA (*Advanced Research Projects Agency*)

¹⁰Cell phone with advanced capabilities, which executes an identifiable operating system allowing users to extend its functionality with third party applications that are available from an application repository.

Esse aumento na quantidade de dispositivos conectados, torna necessário uma maior preocupação com relação à segurança dos sensores e com a privacidade dos dados que eles armazenam. A integração de dispositivos para transferir todos os dados sensíveis também é crítica. Com bilhões de dispositivos conectados entre si, é importante garantir às pessoas que suas informações estão em um ambiente seguro (EVANS, 2011).

Para atender a enorme demanda de tráfego da *Internet* das Coisas, surge o 5G (5ª Geração), que promete velocidades de download de até 100 Mbps. O IPv6 também terá papel importante nessa infraestrutura, resolvendo um problema que seria grave para a IoT: a limitação de endereços IPv4, em razão de somente disponibilizar 04 (quatro) bilhões de endereços.

A integração de todas estas tecnologias, da IoT e em especial da *Internet* na vida das pessoas, é um dos grandes eixos da revolução digital, não mais restrita apenas a computadores e *smartphones*. A rede mundial está se espalhando pelos objetos de nossas casas, ambientes de trabalho e outros meios, trazendo com ela, todos os seus benefícios, oportunidades e desafios Evans (2011).

Ilustrando essa mudança de paradigma, Weiser (1991) declarou: “As tecnologias mais importantes são aquelas que desaparecem. Elas se integram à vida do dia a dia, ao nosso cotidiano e tornam-se indistinguíveis”.

A ideia é que as tarefas cotidianas, algumas de certa forma tediosas e repetitivas, sejam desempenhadas de forma autônoma, com certa “inteligência” por um dispositivo qualquer, interligado à *Internet*.

O dispositivo poderá até mesmo tomar decisões, previamente configuradas por um ser humano. A principal finalidade é que as “coisas” conectadas não dependam de interação humana para funcionar (COMER, 2016). Elas percebem o ambiente no qual estão inseridas e trocam dados entre si e com a rede mundial automaticamente. Isso é chamado de interação máquina a máquina (M2M) (TEIXEIRA *et al.*, 2014).

Em contrapartida, também existem inúmeros casos de objetos classificados como IoT, que permitem interação com o ser humano (COMER, 2015), tais como: *smartwatches*, dispositivos que monitoram o posicionamento geográfico e até os batimentos cardíacos de seu usuário.

No que tange à utilização, os dispositivos IoT interconectados podem ter várias aplicações úteis, dentre as quais podemos ressaltar:

- Casas, Carros e Cidades inteligentes;
- Operações automatizadas em locais remotos e/ou insalubres;
- Atividades Agrícolas e Medicina.

Outra aplicação da IoT é com sensores embarcados combinados com dispositivos de baixo custo, que possibilitam um monitoramento remoto de diversas atividades, principalmente aquelas que ofereçam certo grau de periculosidade ao ser humano.

1.1 Motivação e objetivos

Existem dispositivos legados (ar-condicionados, *no-breaks*, etc), que são relevantes para o funcionamento de uma instalação remota, mas que não estão nativamente conectados a uma rede de computadores. A equipe de monitoramento, nesse caso, devido a distância, não tem ciência do seu estado e/ou regime de funcionamento. Logo, por meio de um sistema de monitoramento mais efetivo, este problema pode ser sanado, auxiliando na execução das tarefas da equipe responsável.

1.1.1 Objetivo Geral

O objetivo do presente trabalho é criar um ambiente de monitoramento para os diversos ativos existentes na Organização, permitindo que a equipe responsável possa atuar nos incidentes com mais agilidade. Para que essa proposta seja bem sucedida, serão executados os passos a seguir.

Criação de uma rede de sensores distinta da rede de dados existente na Organização. Essa nova rede será composta por dispositivos IoT que realizarão o monitoramento dos equipamentos que não são nativamente ligados a rede e enviarão os dados para um sistema de monitoramento localizado na sede. Fazendo uso de protocolos de comunicação sem fio e com baixo consumo de energia.

Desenvolver um protótipo de dispositivo IoT, o qual se destina a coletar informações referentes à temperatura e energia e outros tipos de dados, atuando na coordenação do funcionamento de dispositivos legados, de maneira autônoma (M2M), em um site remoto com equipamentos de telecomunicações.

Este projeto será aplicado em um cenário real com a confecção de protótipos, com a possibilidade de ser implementado em outras situações que demandem o uso de diferentes sensores e tenha a necessidade de constante monitoramento, tais como: tanques de aquicultura, atividades agrícolas, ambientes com atmosfera insalubre, etc.

1.1.2 Objetivos Específicos

- Definir os tipos de sensores a serem escolhidos, conforme a necessidade da estação remota;
- Desenvolver o dispositivo IoT transmissor e receptor que consolidarão o resultado da coleta de dados do ambiente monitorado;
- Criar um sistema de alimentação do dispositivo IoT, que seja independente da infraestrutura instalada;
- Gerenciar a coleta de dados por meio dos sensores e as ações de resposta às situações previstas no projeto;
- Permitir o envio de dados do local remoto para a instituição por meio de um protocolo de comunicação sem fio;
- Promover a integração do sistema de monitoramento com a rede do dispositivo IoT remoto.

1.2 Produção científica

A produção de conhecimento é vital para o desenvolvimento acadêmico, impactando sobremaneira o campo social, cultural e econômico. Com o objetivo de compartilhar o aprendizado sobre redes de computadores e dispositivos IoT, obtidos no decorrer deste trabalho, foi realizada a publicação de um artigo de área correlata na Revista Research, Society and Development journal (RSD), no ano de 2020.

- OLIVEIRA, V. C. Simulador Eve-NG em projetos de redes heterogêneas: um estudo sobre a importância da simulação em redes de computadores. **Research, Society and Development**, [S. l.], v. 9, n. 11, p. e1199119562, 2020. DOI: 10.33448/rsd-v9i11.9562. Disponível em: <https://rsdjournal.org/index.php/rsd/article/view/9562>. Acesso em: 06 nov. 2020.

1.3 Organização da Dissertação

Este trabalho apresenta-se organizado da seguinte maneira:

Capítulo 2: Apresenta a fundamentação teórica na qual se baseia o projeto, observando as melhores práticas e recomendações da literatura consultada.

Capítulo 3: Apresenta a linha de ação tomada para a confecção do projeto e quais os métodos escolhidos para execução do projeto.

Capítulo 4: Apresenta os resultados apurados em cima dos dados coletados, proporcionando uma análise sobre relevância e eficácia do projeto.

Capítulo 5: Apresenta as conclusões obtidas com o projeto e oportunidades de melhorias futuras.

FUNDAMENTAÇÃO TEÓRICA

Este trabalho se baseia em quatro eixos fundamentais:

- O Sistema de monitoramento de rede: fará o acompanhamento dos ativos remotos e proporcionará a visualização dos *status* dos sensores, bem como os dados por eles coletados;
- Os Sensores e Microcontroladores: os sensores são chamados de dispositivos finais, isto é, aqueles que fazem a coleta de dados propriamente dita e estabelecerão a comunicação M2M entre si em caso de ocorrência de uma situação pré-configurada, bem como, via transmissor sem fio enviarão os dados coletados ao *gateway* ou concentrador, tudo isso devidamente coordenado pelo microcontrolador acoplado ao sensor ou ao conjunto de sensores;
- O consumo de energia e o sistema de alimentação: na IoT o baixo consumo de energia é fundamental, em razão da localização remota do sensor e das esporádicas operações de manutenção do dispositivo fixado. Isso também demanda um bom projeto de alimentação para o conjunto sensor-microcontrolador, que de preferência, advenha do próprio ambiente, por meio do vento ou da luz solar;
- O protocolo de comunicação: responsável por transportar os dados coletados pelos sensores até o *gateway* e depois até a infraestrutura de rede da empresa, onde os mesmos serão armazenados e postos em um sistema de monitoramento.

2.1 Sistemas de Monitoramento Rede

Com o crescimento da *Internet* e consequente popularização das redes *Ethernet* (802.3x) cabeadas e do *Wi-fi* (802.11), ficou mais fácil conectar à rede, os locais de

difícil acesso de uma instituição qualquer, a um custo razoavelmente baixo. Contudo, caso a quantidade de locais e *hosts*¹ a serem conectados fosse muito grande, se tornava impossível para uma pessoa diante de uma tela, tomar conta do que ocorria em vários lugares simultaneamente e ainda detectar incidentes (LIMA, 2014).

A solução implementada inicialmente pelos administradores de uma rede, era a utilização de um pequeno software, chamado de *script*, contendo alguns comandos do próprio sistema operacional, dentre eles o *hwinfo*², para listar os componente da máquina, conforme demonstrado na Figura 1.

Porém, à medida que mais *hosts* eram acrescentados e mais parâmetros precisavam ser checados, o pequeno *script* inicial, ia crescendo, de modo que aos poucos tornava-se de difícil manutenção (UYTTERHOEVEN; OLUPS, 2019).

Figura 1 – Comandos de listagem dos componentes de *Hardware*

```
valdinei@X510UKK:~$
valdinei@X510UKK:~$ hwinfo --short |more
cpu:
      Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2997 MHz
      Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2762 MHz
      Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2769 MHz
      Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2725 MHz
      Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2884 MHz
      Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2985 MHz
      Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2983 MHz
      Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2975 MHz
keyboard:
  /dev/input/event8      Maxxter Wireless Receiver
  /dev/input/event4      AT Translated Set 2 keyboard
mouse:
  /dev/input/mice        Maxxter Wireless Receiver
  /dev/input/mice        ELAN1200:00 04F3:303E Touchpad
monitor:
      LG Display LCD Monitor
graphics card:
      nVidia GM108M [GeForce 930MX]
      Intel VGA compatible controller
sound:
      Intel Audio device
storage:
      Intel Sunrise Point-LP SATA Controller [AHCI mode]
network:
  wlp2s0                 Intel Dual Band Wireless-AC 8265
network interface:
  cscotun0               Network Interface
  wlp2s0                 Ethernet network interface
  lo                     Loopback network interface
  docker0                Ethernet network interface
disk:
  /dev/sdb                SanDisk SD8SN8U1
  /dev/sda                ST1000LM035-1RK1
```

Fonte: autoria própria

Surgem então, os sistemas de monitoramento de rede com a finalidade de assegurar que os pontos a serem monitorados estejam funcionando conforme o planejado dentro do acordo de nível de serviço - ANS, e que possua alarmes de incidente, acionando rapidamente o indivíduo responsável, otimizando as tarefas de

¹ Nome dados aos dispositivos finais, computadores, em uma rede.

² Um aplicativo do sistema operacional *Linux*, que lista os componentes de hardware de uma máquina

resposta e controle de falhas.

Tudo isso, visando apoiar a tomada de decisões e a total resiliência em relação a um incidente ou catástrofe tão logo ele seja detectado, contribuindo para minimizar os impactos causados por possíveis interrupções (LIMA, 2014).

A infraestrutura de rede hoje para as empresas têm elevado grau de importância. A mesma possui missão crítica, não podendo parar de jeito algum. Logo, é necessário monitorar tanto os serviços prestados, como toda a infraestrutura que mantém este serviço (LIMA, 2014).

De acordo com Uytterhoeven e Olups (2019), as funcionalidades básicas a se esperar de um sistema de monitoramento são:

- Coleta de dados: comumente esta coleta é realizada usando vários métodos, tais como, SNMP(*Simple Network Management Protocol*)³, agentes nativos, *plugins*, *scripts* customizados, dentre outros;
- Armazenamento de dados: uma vez coletados os dados, não faz sentido simplesmente descartá-los, eles devem ser armazenados para possibilitar análises e comparações posteriores;
- Alertas: o dado coletado deve ser comparado com valores limites e em caso de extrapolação dos mesmos, alertas são enviados aos responsáveis por diferentes canais, tais como *email* ou SMS (*Short Message System*);
- Visualização: pessoas são melhores em distinguir dados visuais do que apenas números brutos, como os dados estão armazenados, fica mais fácil gerar informações mais visuais, como gráficos, a partir deles.

Uytterhoeven e Olups (2019) ainda ressaltam que um sistema de monitoramento a partir do momento que é implementado, vem a apresentar muitas funcionalidades, as quais são feitas por meio de configurações simples, alarmes de desastre e execução automática de comandos.

Em seguida, termina por alcançar outros ativos além da infraestrutura de redes, monitorando também discos de armazenamento, servidores, fases de *No-break*, podendo ainda, detectar várias categorias de problemas diferentes, a partir de um único ponto de configuração.

³Protocolo embarcado em *Switches* e *Roteadores*, que proporciona um servidor remoto coletar informações sobre o *status* dos mesmos.

Há vários sistemas de monitoramento de rede nos dias de hoje, dos quais podemos destacar o *Nagios*, *CACTI*, *ZenOSS*, *Spiceworks* (BLACK, 2008) e o *Zabbix*.

2.1.1 *Zabbix*

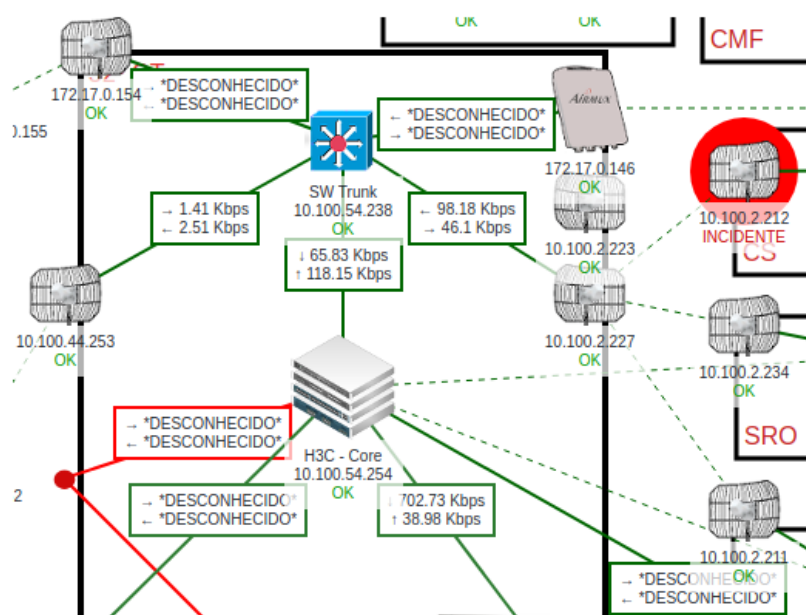
O *Zabbix* é uma aplicação criada por *Alexei Vladishev* em 1998 (ZABBIX, 2020). Desde então tem sido cada vez mais implementado nas empresas, em razão de ser uma solução do tipo "*all-in-one*", isto é, não necessita de *plugins* externos para desempenhar suas funções.

O *Zabbix* foi escolhido para ser a plataforma de monitoramento deste projeto, pelo fato de já estar implementado no local de monitoramento, ter fácil curva de aprendizagem e possibilitar a criação dos *scripts* customizados, além da funcionalidade de suportar o *Zabbix Sender / Trapper*, ambos necessários para a coleta dos parâmetros informados pelo sensores IoT.

O *Zabbix Sender*, de acordo com Zabbix (2020), é uma aplicação de linha de comando que envia dados coletados para o *Zabbix Server* e/ou *Zabbix Proxy*. No servidor *Zabbix*, um item do tipo *trapper Zabbix* deve ser criado com uma chave única para receber as informações do *sender*.

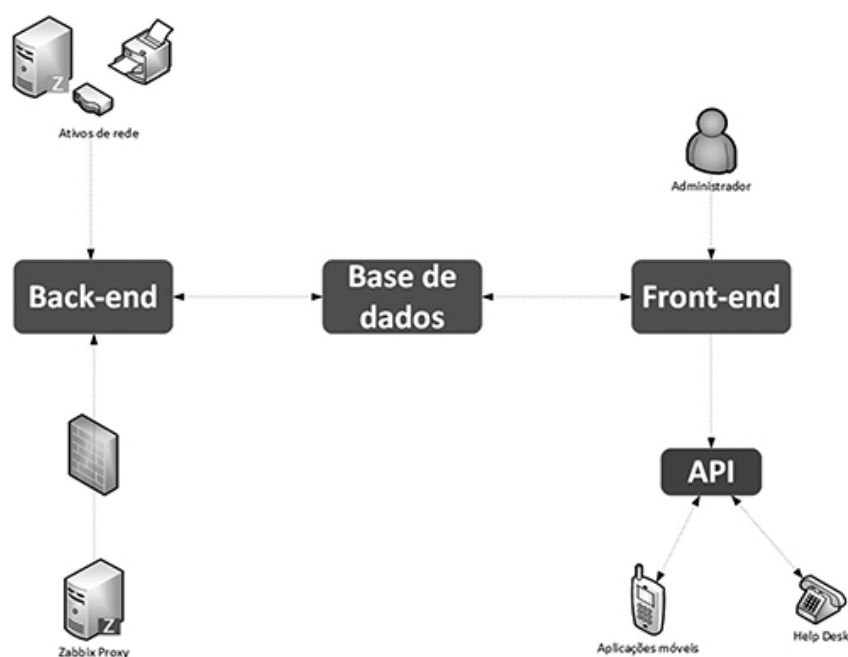
Para microcontroladores como *Arduíno* e ESP existe uma biblioteca chamada *ESP8266ZabbixSender.h* (SZABOLCS, 2016), que foi adaptada para permitir que o *Zabbix Sender* funcione nestes dispositivos. Esta biblioteca abstrai a complexidade do *Zabbix Sender* permitindo que o desenvolvedor consiga enviar os dados manipulando funções mais simples, que funcionam encapsuladas no código em linguagem C, da aplicação a ser desenvolvida para os nós.

O *Zabbix* é capaz de monitorar milhares de itens e apresentá-los em uma visão mais amigável, consolidada em um diagrama de ativos, tal como um mapa, representado na Figura 2. O alcance do monitoramento é limitado apenas pelos recursos de *hardware* servidor (UYTTERHOEVEN; OLUPS, 2019).

Figura 2 – Mapa de monitoramento do Zabbix

Fonte: autoria própria

A arquitetura do *Zabbix* se organiza no modelo de três camadas. A aplicação ou *back-end* - que faz a coleta dos dados dos ativos cadastrados no *Zabbix*; O banco de dados - responsável pela persistência dos dados e a *Interface Web - front-end*, que apresenta as informações de monitoramento à equipe responsável (UYTTERHOEVEN; OLUPS, 2019). A Figura 3 descreve essa arquitetura.

Figura 3 – Arquitetura do Zabbix

Fonte: adaptada de Lima (2014)

Lima (2014) afirma que o *Zabbix*, em seu *back-end* é composto de três elementos que realizam o monitoramento propriamente dito:

- *Zabbix Server*: É o responsável por receber e armazenar os dados coletados;
- *Zabbix Proxy*: É um elemento opcional, mas é responsável por fazer a coleta de clientes remotos, consolidar esta coleta e depois enviar ao *Zabbix Server* de uma única vez. Não precisa de tantos recursos e possui baixa manutenção, aliviando de certo modo, a carga de operações executadas pelo *Zabbix Server*;
- *Zabbix Agent*: É o dispositivo final que repassa as informações ao *Zabbix Proxy* ou ao *Zabbix Server*. Pode ser substituído por protocolos externos, tais como, o ICMP (Internet Control Message Protocol), SNMP, SSH (*Security Shell*), entre outros.

O *Zabbix* possui licença GPL (*Generic Public License*), é fácil de ser configurado e as informações são armazenadas em banco de dados. Toda a parte gráfica é adaptável e possui uma *interface web* de fácil configuração (ZABBIX, 2020).

Não possui a necessidade de *plugins* extras, tem sistemas de alerta com envio de notificação externa, além fornecer possibilidade de se obter dados por meio de *scripts* customizados, para os dispositivos que não permitam a coleta por meio dos itens pré-configurados em *templates* ou por SNMP (LIMA, 2014).

2.2 Sensores

Com objetivo de aplicar inteligência às coisas e permitir que as mesmas possam coletar e trocar informações acerca das pessoas e dos ambientes, surge a figura do sensor responsável por realizar a tarefa de coletar estes dados e encaminhá-los ao seu correto destino. São geralmente dispositivos pequenos, que em virtude de possuírem pouca capacidade computacional (TANENBAUM; WETHERALL, 2011), realizam funções específicas e tem como premissa fundamental, consumir pouca energia (COMER, 2016).

O sensores, também chamados de nós (TANENBAUM; WETHERALL, 2011), têm papel importante na Internet das Coisas. São responsáveis pela percepção do ambiente real e repassá-los ao mundo digital, seja coletando informações,

determinando posições no terreno, monitorando ambientes ou executando alarmes em caso de alguma atividade não autorizada (AKYILDIZ *et al.*, 2002).

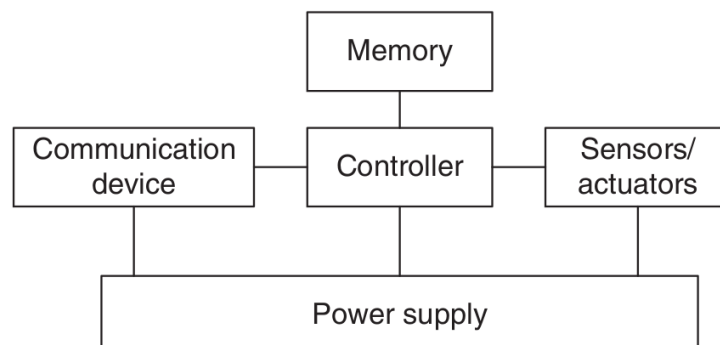
2.2.1 Hardware básico de um sensor

De acordo com Karl e Willig (2005), ao escolher um sensor para determinada atividade, alguns aspectos de *hardware* são levados em consideração para o desenvolvimento do projeto. Dentre os quais pode-se destacar: a capacidade de processamento; se os canais de comunicação são tecnologias acessíveis; o consumo de energia atende ao projeto; e principalmente o custo do *hardware*.

Um nó sensor básico compreende cinco componentes, conforme apresentado na Figura 4 (KARL; WILLIG, 2005):

- *Controller* (Controlador): tem por objetivo processar dados relevantes e executar os códigos inseridos;
- *Memory* (Memória): responsável por armazenar programas e dados;
- *Sensors and Actuators* (Sensores e atuadores): faz a interface com o mundo real, coletando parâmetros do ambiente ou executando ações pré-determinadas;
- *Communication* (Comunicação): é necessário para um dispositivo para enviar e receber as informações coletadas por um canal sem fio;
- *Power Supply* (Fonte de Energia): a fonte de energia é escassa, então às vezes, são necessárias baterias com alguma forma de recarga sendo obtida do ambiente, tal como vento ou células solares.

Figura 4 – Visão geral de um Sensor Básico



Fonte: adaptada de Karl e Willig (2005)

2.2.2 Características dos sensores sem fio

Akyildiz *et al.* (2002) relata que um sensor deve possuir as seguintes características:

- Ter baixo consumo de energia;
- Operar com altas densidades;
- Ter baixo custo e ser descartável;
- Ter autonomia para operar sem interferência humana;
- Ser flexível e se adaptar ao ambiente.

2.2.3 Tipos de sensores

Existem sensores para as mais diversas tarefas, sendo que cada um deles pode usar um princípio de funcionamento específico. Comer (2015), assim como, Tanenbaum e Wetherall (2011) retratam alguns desses sensores e exemplificam a sua aplicação em várias situações no ambiente real. Citam ainda que em um sistema embarcado, este formado por um conjunto de sensores, tem a capacidade de sentir e controlar o mundo físico em torno dele. Deste universo, podemos destacar:

- Sensores de temperatura;
- Sensores de umidade;
- Sensores de presença;
- Sensores de gás;
- Sensores de obstáculos;
- Sensores de ruído;
- Sensores de luminosidade;
- Sensores de tensão elétrica;
- Sensores de corrente.

Descrever o princípio de funcionamento de cada tipo de sensor, foge do escopo do presente trabalho, para maiores detalhes acerca dos mesmos, consultar Bell (2014).

2.2.4 Redes de sensores sem fio (RSSF)

Os sensores, dado suas limitações, na maioria das vezes não são instalados sozinhos, geralmente são implementados com vários outros, conforme a Figura 5, tornando-se um nó em uma rede de sensores (TANENBAUM; WETHERALL, 2011).

Uma Rede de sensores sem fio, na definição de Netto (2016), é:

Uma rede composta por grande número de nós sensores implantados numa região em particular para detectar e monitorar fenômenos físicos como temperatura, umidade, pressão e assim por diante. Este conjunto de sensores conduz a coleta dos dados, podendo ou não realizar algum processamento no próprio nó, transferindo a informação para um concentrador que é responsável pelo armazenamento, compilação e transferência destes dados à locais mais distantes. (NETTO, 2016)

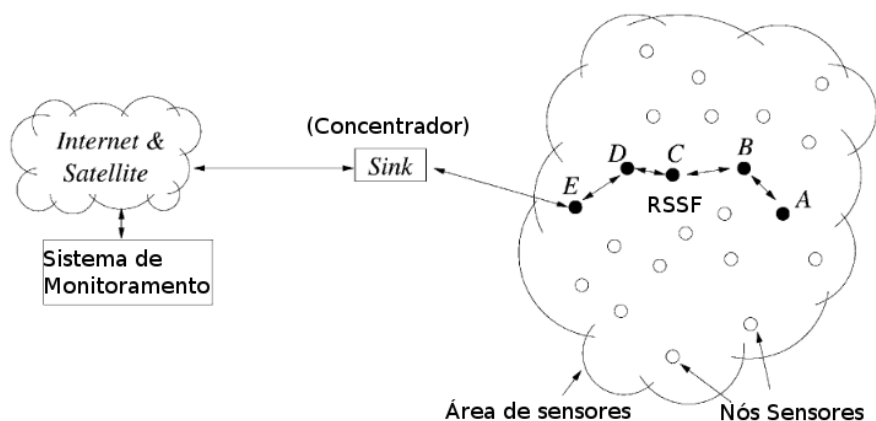
Figura 5 – Exemplo de uma RSSF



Fonte: adaptada de Tanenbaum e Wetherall (2011)

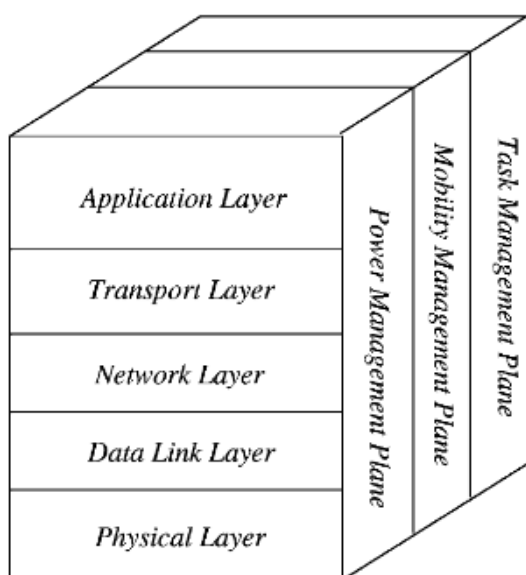
2.2.5 Arquitetura de comunicação das RSSF

Os nós (*Sensor Nodes*) em uma rede de sensores, geralmente estão localizados em uma região qualquer (*Sensor Field*), onde tem a capacidade de coletar dados. Uma vez que estão interligados a um nó concentrador (*Sink*), o nó sensor envia estes dados para ele, que por sua vez conduz os dados coletados à um gerenciador/monitor de tarefas (*Task Manager Node*) ou para Internet (AKYILDIZ *et al.*, 2002). A Figura 6 representa esta arquitetura.

Figura 6 – Arquitetura de uma RSSF

Fonte: adaptada de Akyildiz *et al.* (2002)

Akyildiz *et al.* (2002) relata que a pilha de protocolos usada pelo concentrador e os demais sensores, combina um uso adequado de energia e aprendizado de roteamento, integra os dados aos protocolos de rede, se comunica com eficiência energética por meio da rede sem fio e promove a cooperação entre os nós sensores.

Figura 7 – Pilha de Protocolos de uma RSSF

Fonte: adaptada de Akyildiz *et al.* (2002)

Akyildiz *et al.* (2002) descreve o funcionamento da pilha de protocolo apresentada na Figura 7, em um modelo de camadas:

- A camada de aplicação é implementada conforme os requisitos do projeto,

podendo ser desenvolvida uma aplicação que atenda as necessidades dos nós sensores;

- A camada de transporte mantém o fluxo de dados fim a fim;
- A camada de rede faz o roteamento por cada ativo intermediário (roteadores) até o destino;
- Como o ambiente é ruidoso e os sensores podem ser móveis, a camada de enlace é responsável por estar ciente das restrições de energia e ser capaz de contornar as colisões que ocorrerem com a transmissão de dados dos sensores vizinhos;
- A camada física se dedica às técnicas de modulação e demodulação, transmissão e recepção.

Netto (2016) afirma que existem também além das camadas acima, outras três que apoiam as tarefas de controle de energia, mobilidade e distribuição de tarefas entre os sensores:

- A camada de gerência de energia permite ao sensor saber como irá utilizar a energia que lhe foi disponibilizada, este serviço pode permitir o desligamento do sensor se este não estiver em uso;
- A camada de mobilidade controla os movimentos dos nós sensores e mantém atualizada a topologia da rede, descobrindo quem são os vizinhos de cada sensor. Esta camada está intimamente interligada ao controle de energia, as tarefas de descoberta de vizinhos custa muita energia;
- A camada de controle de tarefas distribui o que cada sensor deve fazer em uma determinada área de abrangência;

Essa rotina pode ser usada para redistribuir responsabilidades aos novos nós, atendendo a uma tarefa específica ou para recuperar um setor com problemas (auto recuperação) (NETTO, 2016).

Podemos observar a ampla gama de aplicação das RSSF. No futuro, isso fará das redes de sensores parte do cotidiano das pessoas. No entanto, as redes de sensores precisam satisfazer as certas restrições, como tolerância a falhas, escalabilidade,

custo, hardware, mudança de topologia, alteração de ambiente e consumo de energia. Pelo fato dessas restrições serem altamente rigorosas e específicas para as RSSF, se tornam necessárias novas técnicas de rede *adhoc*⁴ sem fio (AKYILDIZ *et al.*, 2002).

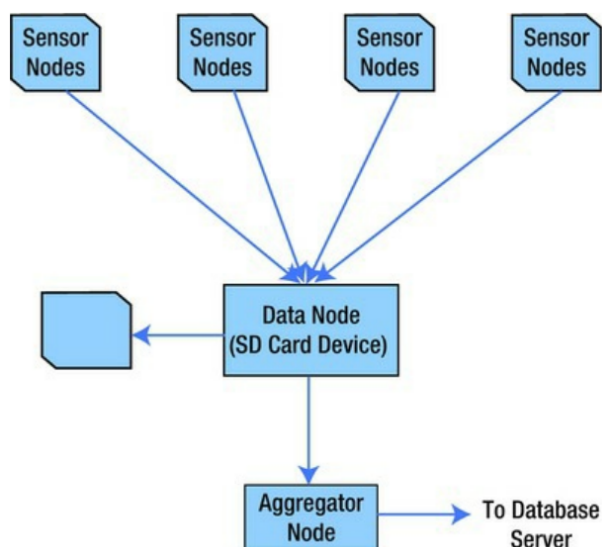
2.2.6 Tipos de nós encontrados em uma RSSF

De acordo com Bell (2014), os nós sensores podem ser compostos por um ou mais sensores, geralmente interligados com um microcontrolador, designados para realizar operações de maneira autônoma, desse modo eles podem ser classificados quanto ao seu uso, ajudando o planejamento do papel a ser desenvolvido pelo sensor em uma rede, por ocasião da sua implementação.

- **Nó Sensor Básico (*Basic Sensor Node*):** este tipo de sensor não armazena ou manipula nenhum dado coletado, apenas executa a tarefa de coletar os dados do ambiente e repassar a outro nó na rede;
- **Nó de dados (*Data Nodes*):** este sensor também é capaz de enviar dados para outro nó, mas sua característica principal é armazenar os dados coletados em um dispositivo de armazenamento interno, como *SD card*, para consultas posteriores e/ou apresentá-los por meio de um visor de cristal líquido (LCD) ou um conjunto de LED (*Light Emitter Diode*) - diodo Emissor de Luz;
- **Nó agregador ou *Sink* (*Aggregator Nodes*):** Os nós agregadores, reúnem um dispositivo de comunicação, um dispositivo de armazenamento e nenhum sensor. Os nós agregadores são usados para consolidar os dados coletados dos outros dois tipos de nós e enviá-los via rede cabeada, para armazenamento e apresentação dos mesmos em um sistema de monitoramento.

A finalidade dessa classificação é de que os dados sejam preservados, mesmo em caso de falhas em um ou mais pontos de comunicação com a rede de sensores. Tanto o nó de dados, quanto o nó agregador armazenam os dados coletados, possibilitando que os mesmos sejam repassados ao sistema de monitoramento no momento que a comunicação for reestabelecida (BELL, 2014). A Figura 8 ilustra essa organização.

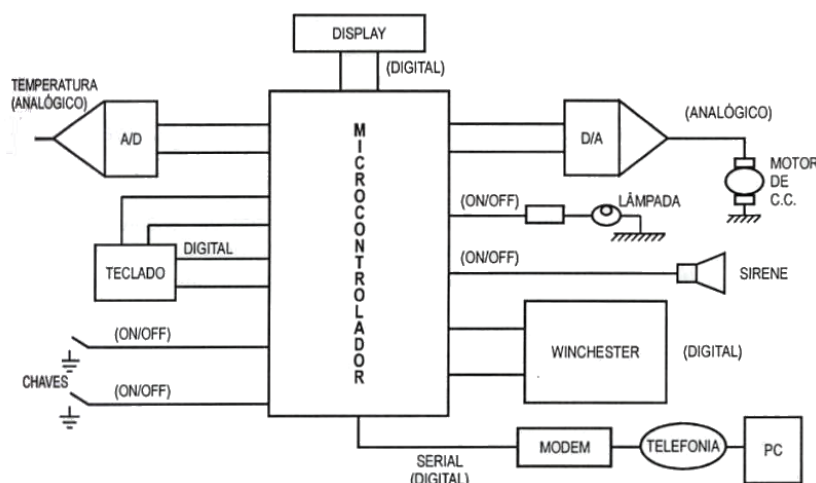
⁴Rede sem nenhum controle central e sem nenhuma comunicação com o mundo exterior. (KUROSE; ROSS, 2017)

Figura 8 – Tipos de nós em uma RSSF

Fonte: adaptada de Bell (2014)

2.3 Microcontroladores

Microcontroladores são criados com um circuito de apoio e implantados em pequenas placas de circuito impresso. São muito utilizados em sistemas embarcados para monitorar e controlar dispositivos, tornando-se ideal para o uso com rede de sensores (BELL, 2014). A Figura 9, apresenta a visão geral de um microcontrolador e seus circuitos de apoio.

Figura 9 – Microcontrolador básico com seus sensores, atuadores e periféricos

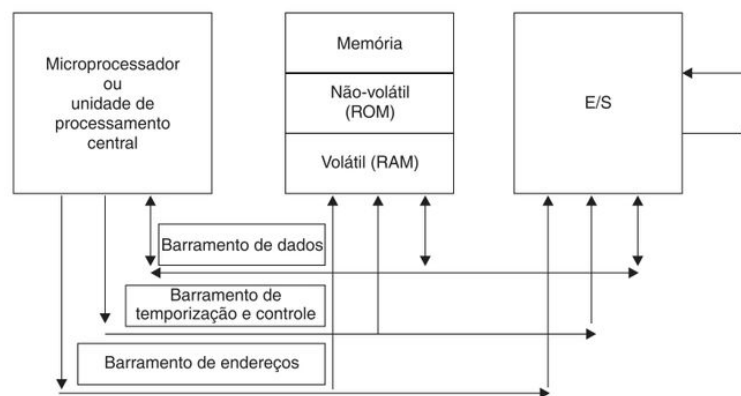
Fonte: adaptada de Nicolosi (2009)

Segundo Gimenez (2005), o microcontrolador, conforme a Figura 10, possui três blocos principais:

- A unidade central de processamento: também chamada de CPU (*Central Process Unit*), realiza a tarefa de processar dados que estão aguardando sua vez na memória, é também responsável pela tomada de decisões;
- Unidade de memória: armazenam os dados a serem processados, os programas e informações a serem guardadas após o processamento na CPU;
- Unidades de entrada e saída: Tem por finalidade proporcionar a interação com o mundo exterior.

Na Figura 10, é apresentado ainda como os três blocos se comunicam entre si, o que permite que o microcontrolador de fato funcione. São chamadas de vias de comunicação ou barramento.

Figura 10 – Diagrama básico de Microcontrolador



Fonte: adaptada de Gimenez (2005)

Os barramentos são na verdade as trilhas da placa (trilhas de cobre), por onde fluem as informações em paralelo. Basicamente são encontrados três tipos de barramento em um microcontrolador (GIMENEZ, 2005).

- Barramento de endereços: utilizado pela CPU para definir os endereços de memória do programa que terá sua vez na CPU. Esse barramento só tem uma direção, da CPU para a memória e para os dispositivos E/S;
- Barramento de controle: a CPU utiliza este barramento para gerir os sinais de controle e a direção do fluxo de informações de leitura e escrita nos dispositivos de E/S. Também é unidirecional;
- Barramento de Dados: A CPU usa este barramento para trocar informações com a memória acerca dos dados que serão enviados para serem processados. Este barramento é bidirecional.

O microcontrolador nesse projeto é o responsável por permitir que as informações do ambiente, uma vez coletadas pelos sensores, sejam consolidadas, armazenadas e dependendo do projeto, formatadas para serem enviadas pela rede.

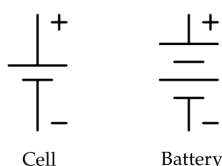
2.4 Consumo de Energia

A IOT possibilitou a conexão de qualquer dispositivo à rede e colaborou junto com as tecnologias *wireless* para que os mesmos operem sem estarem presos por fios (ASHTON, 2009). Em contrapartida, a conectividade e mobilidade oferecidos, fez surgir um alerta: o consumo de energia. Esse é um fator crítico para a IOT e deve ser minimizado (COMER, 2015), sendo necessário pensar em algo que forneça a eles autonomia de funcionamento (AUGUSTIN *et al.*, 2016).

Em um primeiro momento este problema é solucionado com a utilização de baterias, mas então surgem as indagações a respeito de qual bateria escolher (COMER, 2016). É preciso conhecer os tipos existentes no mercado e suas características.

Plett (2015) ressalta que uma célula (*Cell*), conforme a definição da *National Electrical Code, document NFPA-70*, é a menor unidade eletroquímica existente, usada para receber, armazenar e entregar energia elétrica. IEEE (1996) afirma que, comumente, a célula é chamada de bateria e que o termo "bateria" em seu uso correto define um conjunto de duas ou mais células conectadas eletricamente, seja em série ou paralelo, para produzir energia para um circuito de carga. A Figura 11 apresenta os símbolos para célula e bateria.

Figura 11 – Simbologia para célula e para bateria

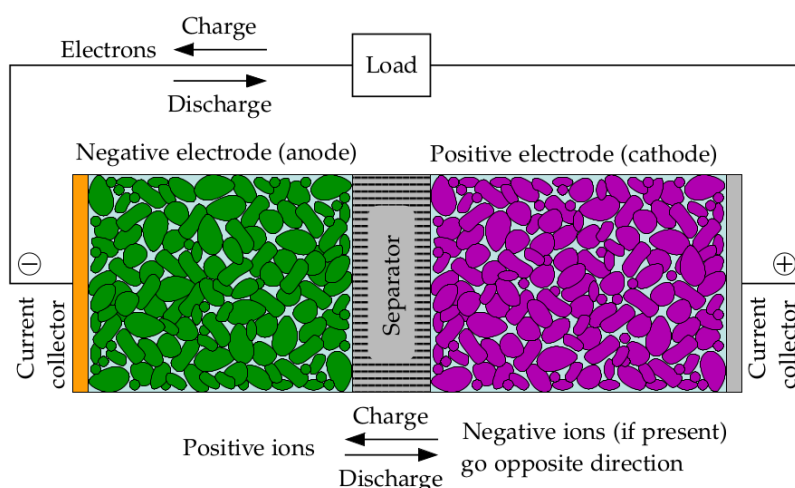


Fonte: adaptada de Plett (2015)

As células podem ser primárias, aquelas nas quais a reação eletroquímica não é reversível, este tipo de célula é utilizada apenas uma vez. Já as células chamadas secundárias ou recarregáveis são aquelas em que a reação eletroquímica é reversível, podendo ser descarregadas e recarregadas por muitas vezes (PLETT, 2015).

Uma célula de Li-On é apresentada na Figura 12, basicamente ela é formada por um anodo (parte negativa), um catodo (parte positiva), pelo eletrólito e um separador. Seu princípio de funcionamento se dá por uma reação química que transforma energia química em energia elétrica. Isso faz os elétrons fluírem do anodo para o catodo, quando alimenta a carga, e no caso das células recarregáveis, fluírem do catodo para o anodo, quando está sendo recarregada (TOLL, 2017).

Figura 12 – Representação de uma célula Li-On



Fonte: adaptada de Stetzel *et al.* (2015)

Há vários tipos de células disponíveis no mercado, com as mais diversas tecnologias e destinadas às mais diversas aplicações e projetos, tais como: níquel-cádmio (NiCd), prata-zinco, chumbo-ácido, níquel-hidrogênio (NiMH), alcalina, lítio polímero (Li-Po) e íons de lítio (Li-On) (PLETT, 2015).

Hoje as células Li-On recarregáveis são a principal fonte de energia utilizada em dispositivos embarcados, por possuírem melhor relação energética x densidade, o que ajuda na compactação dos protótipos a serem criados. Em razão das características químicas de seus componentes, principalmente do Lítio, este tipo de bateria demanda mais cuidados. Violações dos limites de segurança podem causar danos irreversíveis à célula e acidentes perigosos (TOLL, 2017).

Segundo Plett (2015), as células Li-On possuem algumas vantagens sobre as demais:

- Operam tensões maiores dos que as demais células recarregáveis, por volta de 3.7 V (Volts) contra 1.2 V das NiCd e similares;
- Devido às maiores tensões armazenadas em uma única célula Li-On, uma menor

quantidade de células pode alcançar tensões maiores e com isso o circuito de gerenciamento de bateria pode ser mais simples;

- As Células Li-On tem uma auto-descarga bem menor do que as demais células de outras tecnologias. Ao passo que as Ni-Cd e similares perdem de 1 a 5% da carga ao dia, as células Li-On mantêm sua carga por meses.

Por outro lado, apesar do relativo sucesso, Plett (2015) também ressalta algumas desvantagens das células Li-On:

- São mais caras, por serem fabricadas em menor volume do que as concorrentes;
- São muito sensíveis à sobrecarga de tensão, podendo explodir, por isso, incluem um circuito especial que desconecta a célula do circuito principal em caso de tentativa de operar acima da tensão de fábrica. Isto as tornam mais complexas;
- Não são disponibilizadas nos tamanhos padrão para pilhas (AAA, AA, C, D), como são as células NiCd, NiMH e similares. Com isso há um custo extra na aquisição de carregadores específicos pra esta tecnologia.

Novamente, vale ressaltar que, além da escolha da bateria, é preciso desenvolver técnicas que permitam que o dispositivo projetado use de maneira racional a pouca energia que lhe é disponibilizada. O baixo consumo em operação é fundamental para dispositivos IoT. Uma vez apurado o consumo do dispositivo, é definida a autonomia desejada, escolhendo-se a bateria mais adequada ao projeto (RAZA; KULKARNI; SOORIYABANDARA, 2017).

A energia consumida pelo conjunto vai determinar o tempo de vida útil da bateria. Essa durabilidade é fator crítico para sensores que estão em locais inacessíveis ou espalhados por uma grande área geográfica. Algumas tecnologias de comunicação sem fio privilegiam a largura de banda em detrimento do consumo de energia (COMER, 2016).

Outra abordagem interessante ao consumo racional de energia envolve segundo Comer (2016) o conceito de coleta de energia. Nele o dispositivo é alimentado diretamente por meio de uma fonte de energia coletada do ambiente ao redor, seja do sol, do vento ou de tarefas mecânicas repetitivas que seriam convertidas em energia elétrica.

Uma combinação de uso de energia coletada do ambiente com baterias, tem bastante potencial para a manutenção do funcionamento dos dispositivos IOT por longos períodos.

2.5 Protocolos de Comunicação

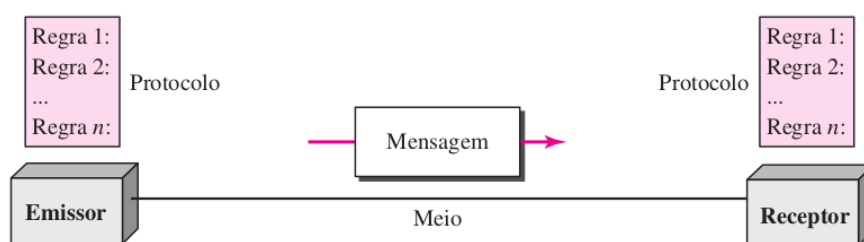
Comunicação segundo Forouzan (2013) significa compartilhar dados. Esse compartilhamento pode ser local, tal como duas pessoas conversando na rua ou remota, caracterizando uma troca de dados. E essa troca necessita de um meio de transmissão para ocorrer com sucesso.

Um sistema de comunicação de dados abrange o conjunto de cinco componentes, os quais permitem que a comunicação ocorra (FOROUZAN, 2013):

- Mensagem: são os dados a serem transmitidos;
- Emissor: é quem envia a mensagem;
- Receptor: é quem deve receber a mensagem;
- Meio de transmissão: caminho (meio) pelo qual a mensagem deve ser transmitida;
- Protocolo: conjunto de regras e normas que controlam como será feita a comunicação entre os dois lados interessados, ambos devem seguir estritamente o mesmo conjunto de regras. Sem esse acordo prévio, a comunicação não será possível.

A Figura 13 ilustra um sistema de comunicação básico:

Figura 13 – Sistema de Comunicação de dados



Fonte: adaptada de Forouzan (2013)

2.5.1 Protocolos sem fio

Um sistema de comunicação precisa de um meio e um protocolo para proporcionar a comunicação entre emissor e receptor. Esse meio pode ser com ou sem fio. Na IoT os protocolos de comunicação sem fio têm relevante destaque, pois são através deles, que se torna possível integrar os nós sensores remotos, de modo que eles possam trocar informações e instruções entre si (BELL, 2014).

Muitas vezes um só protocolo de comunicação não satisfaz todos os requisitos de um determinado projeto, seja por necessidade de largura de banda ou necessidade de mobilidade, logo se faz necessário utilizar mais de um protocolo, para prover a comunicação de dados (COMER, 2016).

O RSSI

Quando se estabelece um enlace sem fios, um dos parâmetros utilizados para verificar se a intensidade de um sinal, tanto transmitido, quanto recebido, se encontra dentro dos parâmetros previstos pelo *datasheet* do transmissor, é o RSSI (*Received Signal Strenght Indication*).

Com o RSSI pode-se precisar o local adequado para a instalação de um sensor e até prever se há necessidade de uma repetidora, em caso da potência do sinal transmitido não atingir o mínimo da sensibilidade exigida pelo receptor.

O SNR

Outro parâmetro importante ao se fazer um enlace sem fio é o SNR (Relação Sinal-Ruído), que permite verificar se a qualidade do sinal recebido permite o estabelecimento de um enlace de comunicação.

Estes dois parâmetros vão contribuir para a definição da viabilidade de um enlace por meio da comparação entre os valores coletados, deixando a cargo do projetista, a escolha do local que obtiver os melhores resultados.

2.5.2 Protocolos de baixo consumo de energia

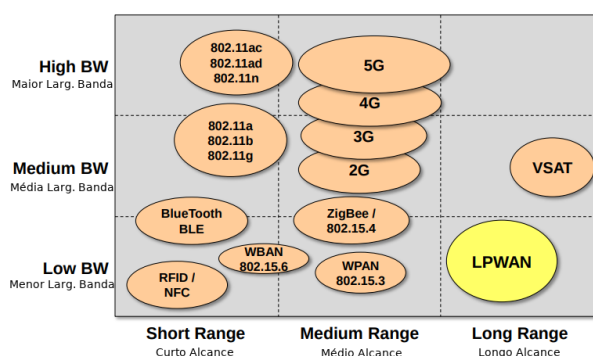
O uso de um protocolo de comunicação, que venha a consumir menos energia durante a rotina de transmissão e recepção dos dados, com uma maior capacidade

de cobertura, se tornou um fator importante para a escolha de sensores e demais dispositivos IoT em um projeto (RAZA; KULKARNI; SOORIYABANDARA, 2017).

Com a finalidade de cobrir essa lacuna, surgiram os protocolos de baixo consumo de energia, LPWAN (*Low Power Wide Area Network*) (RAZA; KULKARNI; SOORIYABANDARA, 2017). Tais protocolos têm grande utilidade, seja em uma área rural onde não há cobertura 3G e o custo de implantar uma rede *wi-fi* e similares se tornam altos, bem como em áreas industriais e em projetos de cidades inteligentes (QUERALTA *et al.*, 2019).

Possuem a capacidade de cobrir grandes áreas geográficas, da ordem de quilômetros, e conceder uma vida útil maior para as baterias atuais, da ordem de dez anos ou mais. A LPWAN promete acesso de baixo custo, baixo consumo e baixa vazão de dados. A Figura 14 apresenta um comparativo do alcance de algumas tecnologias consolidadas no mercado, com a LPWAN.

Figura 14 – Comparativo entre tecnologias *wireless*



Fonte: adaptada de Egli (2017)

Raza, Kulkarni e Sooriyabandara (2017) relembram que os protocolos LPWAN apesar das vantagens, possuem certas limitações quanto à latência e largura de banda, sendo mais recomendado para projetos que sejam tolerantes ao atraso, que não exijam grandes taxas de transmissão de dados, e façam uso do baixo consumo de energia, além de baixo custo. A maioria das trocas de mensagens seguem o fluxo nó-concentrador-servidor, contendo apenas um enlace sem fio.

Segundo Vangelista, Zanella e Zorzi (2015) e Centenaro *et al.* (2015), as tecnologias LPWAN se caracterizam por serem links de longa distância, por operarem na faixa sub-GHz não licenciada e possuírem uma topologia em estrela.

O LPWAN desempenha a tarefa de converter os protocolos IoT, para rede IP estruturada. Na Europa, a LPWAN operam principalmente na faixa entre 863 e 870

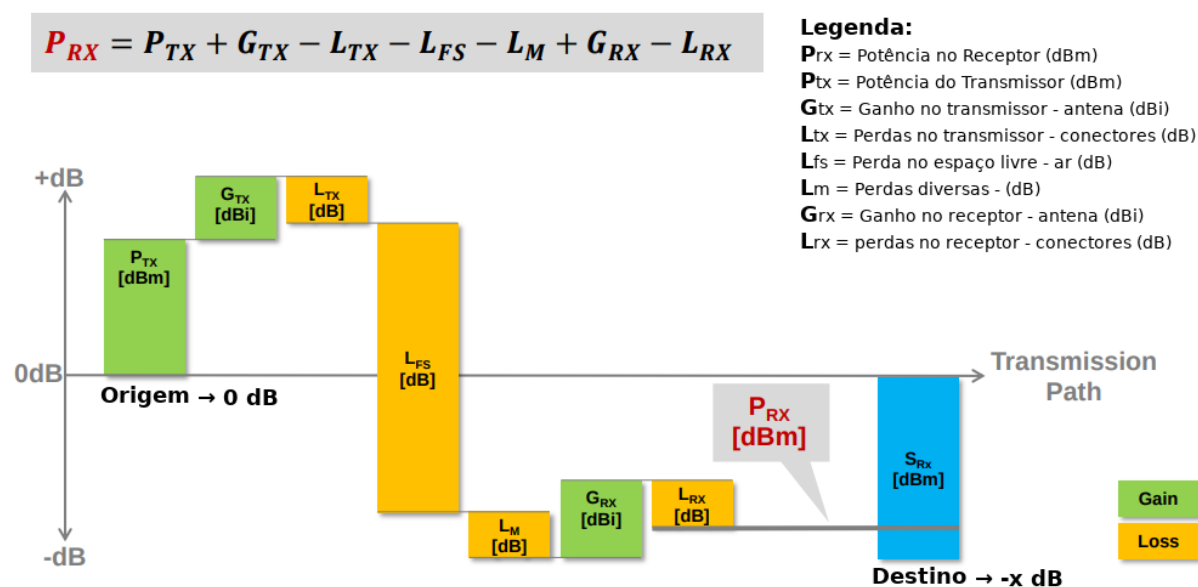
MHz e nos Estados Unidos operam na faixa compreendida entre 902 e 928 MHz. Dependendo da modulação, os LPWAN podem ser classificados em duas categorias:

- Ultra banda estreita: faz uso de canais de banda estreita com largura de 25 kHz;
- Banda Larga: usando uma banda mais larga, 125 kHz ou 250 kHz, utilizando técnicas de espalhamento de espectro para acesso múltiplo, buscando acomodar diversos usuários em um único canal.

Configurações de camada física, em especial a modulação e potência irradiada, são essenciais para o sucesso da comunicação do nó final com seu *gateway*.

Egli (2017) relata que na maioria dos sistemas de rádio, o cálculo do enlace é uma medida importante para o levantamento da potência necessária para estabelecer a comunicação entre dois pontos, quantificando a energia recebida no receptor e contabilizando ganhos e perdas ao longo do caminho de transmissão. O esquema de um cálculo de enlace é apresentado na Figura 15.

Figura 15 – Cálculo de potência de um enlace



Fonte: adaptada de Egli (2017)

Segundo Egli (2017), a sensibilidade do receptor é um ponto muito importante para um dispositivo LPWAN. A mesma é derivada do cálculo de potência do enlace. Um enlace de dados sem fio só pode ser fechado (quando fica possível a comunicação entre remetente e receptor) se a sensibilidade do receptor (S_{rx}) for igual ou inferior a potência recebida (P_{rx}).

Devido às longas distâncias entre os pontos nos enlaces LPWAN, a perda de espaço livre - caminho de perda (Lfs) é um fator que atua bastante na redução da potência recebida no receptor. Portanto, é essencial que os receptores LPWAN, particularmente os nós dos dispositivos, tenham uma sensibilidade do receptor da ordem de até -130 dBm (EGLI, 2017).

No cenário LPWAN algumas tecnologias têm se destacado, dentre as quais, convém destacar o *Sigfox* e a Lora/LoRaWAN (VANGELISTA; ZANELLA; ZORZI, 2015) (CENTENARO *et al.*, 2015) (AUGUSTIN *et al.*, 2016).

2.5.3 *Sigfox*

Segundo Augustin *et al.* (2016), *Sigfox* (<http://www.sigfox.com>) é uma variação do sistema celular, que permite que os nós se conectem por meio da banda ultra estreita ou *Ultra Narrow Band* (UNB). A tecnologia é proprietária, pertencente a empresa francesa *Sigfox* e até o momento, nenhuma especificação pública detalhada do protocolo foi disponibilizada (VANGELISTA; ZANELLA; ZORZI, 2015).

O *Sigfox* opera na faixa de frequência de 868 MHz, com o espectro dividido em 400 canais de 100 Hz. Podendo os nós enviarem até 140 mensagens por dia, com carga útil (*Payload*) de 12 bytes, a uma taxa de até 100 bps (AUGUSTIN *et al.*, 2016).

Ainda segundo a empresa, cada ponto de acesso pode se conectar com até um milhão de dispositivos finais, em uma extensão de 30 a 50 km nas áreas rurais e de 3 a 10 km nas áreas urbanas. Nas primeiras versões, o *Sigfox* suportava apenas comunicação unidirecional, isto é, do nó para o concentrador, mas atualmente a comunicação bidirecional também é suportada (CENTENARO *et al.*, 2015).

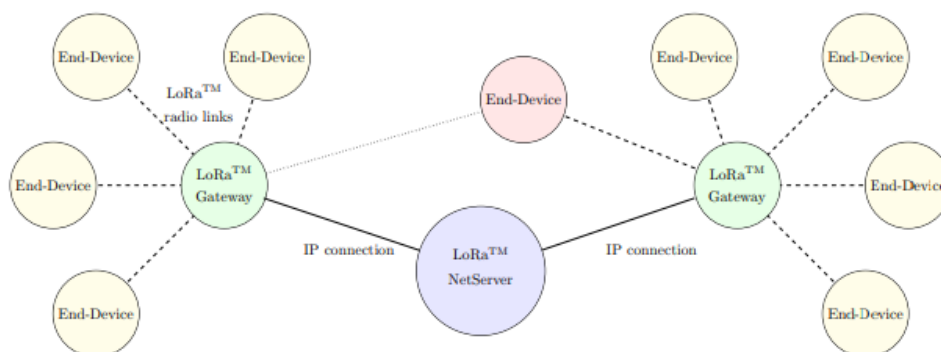
2.5.4 LoRa

De acordo com Augustin *et al.* (2016), LoRa significa "*Long Range*" e caracteriza-se por ser um sistema de comunicações sem fio de longo alcance, promovido pela *LoRa Alliance* (LORA-ALLIANCE, 2020). Seu principal uso é em sistemas que utilizam bateria, que necessitam de autonomia por grandes períodos, onde o consumo de energia é de suma importância.

Já Vangelista, Zanella e Zorzi (2015) destacam que a rede LoRa é uma topologia de "estrela de estrelas", onde os nós são conectados por um *link* LoRa diretamente

a um ou vários concentradores (*Gateways*), que fazem a retransmissão dos pacotes a ele enviados pelos nós e encaminham para o servidor de rede comum - *NetServer* associado. A topologia de uma rede LoRa é representada na Figura 16.

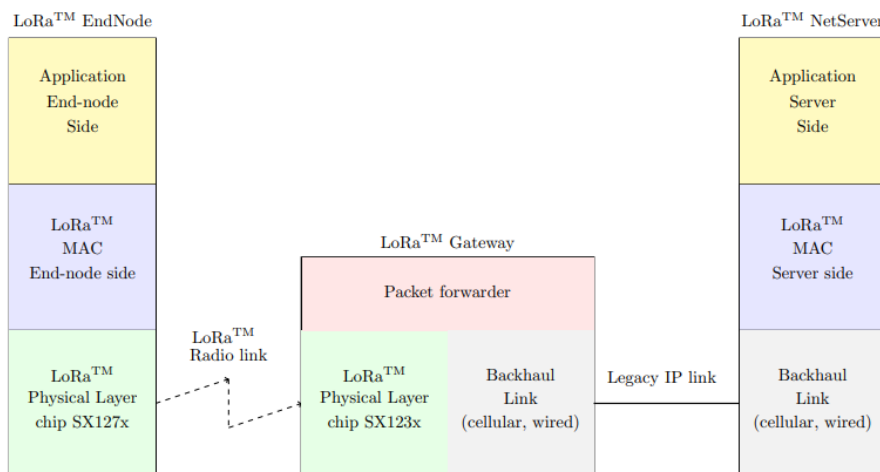
Figura 16 – Topologia lógica Lora



Fonte: adaptada de Centenaro *et al.* (2015)

O *NetServer* fica responsável por filtrar os pacotes duplicados e responder aos dispositivos finais, escolhendo um dos *gateways*, conforme algum critério pré-determinado. Os *gateways* são transparentes para os dispositivos finais, que logicamente, se conectam ao *NetServer*. Essa arquitetura é destacada na Figura 17.

Figura 17 – Arquitetura de protocolos LoRa e LoRaWAN



Fonte: adaptada de Vangelista, Zanella e Zorzi (2015)

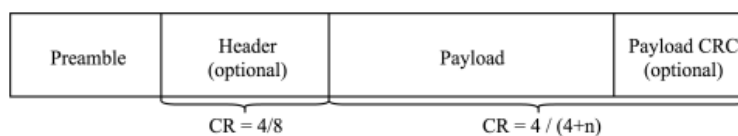
A camada física LoRa, desenvolvida pela *Semtech* (<https://www.semtech.com>), permite comunicações de longo alcance, baixa potência e baixa taxa de transferência, podendo ser utilizada nas bandas ISM (destinadas às aplicações na área Industrial, Científica e Médica) de 433, 868 ou 915 MHz, dependendo da legislação local.

O *Payload* de cada transmissão pode variar de 2 a 255 octetos e a taxa de dados pode chegar a 50 Kbps (AUGUSTIN *et al.*, 2016).

Um quadro LoRa começa com um preâmbulo, logo após há um cabeçalho opcional, que inclui um CRC (*Cycle code redundancy*) - código de redundância cíclica, para permitir que o receptor descarte *frames* com cabeçalhos inválidos.

O tamanho do *Payload* é armazenado usando um *byte*, limitando o *Payload* a 255 *bytes*. O cabeçalho é opcional para permitir desativá-lo em situações nas quais ele não é necessário. O *Payload* é enviado após o cabeçalho e no final do quadro há o CRC opcional (AUGUSTIN *et al.*, 2016). A Figura 18 apresenta o *frame* LoRa.

Figura 18 – O Frame LoRa



Fonte: adaptada de Augustin *et al.* (2016)

O LoRa pode geralmente se referir a duas camadas distintas: a camada física que usa a técnica de modulação por rádio - (CSS) *Chirp Spread Spectrum*; e um protocolo de camada MAC (*Media access Control*) - controle de acesso ao meio - LoRaWAN (Long Range Wide Area Network), embora a LoRa também use uma arquitetura de rede de acesso específica (VANGELISTA; ZANELLA; ZORZI, 2015).

A camada física LoRa possui alguns parâmetros configuráveis para a modulação CSS: largura de banda - *Bandwidth* (BW), fator de propagação - *Spreading Factor* (SF) e taxa de código - Code Rate (CR) que influenciam a taxa de bits da modulação, sua resistência ao ruído de interferência e sua facilidade de decodificação.

Estes parâmetros atuam na sensibilidade do decodificador, isto é, ao aumentar a largura de banda (BW), diminui-se a sensibilidade do receptor, enquanto um aumento do fator de propagação (SF), aumenta a sensibilidade do receptor.

Diminuir a taxa de código (CR) ajuda a reduzir a taxa de erro de pacote (PER) na presença de pequenas rajadas de interferência (AUGUSTIN *et al.*, 2016). A Tabela 1, com informações extraídas da folha de dados (*datasheet*) do SX1276 (SEMTECH, 2015), confirma essa afirmação.

Tabela 1: Sensibilidade do receptor Semtech SX1276 LoRa em dBm em diferentes larguras de banda e fatores de propagação

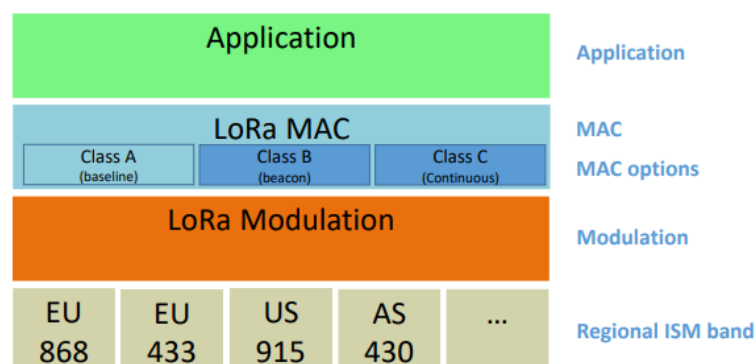
Larg. de Banda (BW)	Fator de propagação (SF)					
	7	8	9	10	11	12
125 kHz	-123	-126	-129	-132	-133	-136
250 kHz	-120	-123	-125	-128	-130	-133
500 kHz	-116	-119	-122	-125	-128	-130

Fonte: adaptada de Semtech (2015)

Augustin *et al.* (2016), afirma que em uma rede LoRa funcionando em conjunto com o protocolo LoRaWAN, os nós finais têm a funcionalidade de aumentar automaticamente o SF se a transmissão com o SF inferior vier a falhar, pois um maior SF aumenta a sensibilidade. Caso necessário, ainda pode-se usar a retransmissão.

2.5.5 LoRaWAN

A camada MAC definida pela *LoRa Alliance* é chamada LoRaWAN e, de acordo com as especificações, é um protocolo *ALOHA* controlado pelo LoRa *NetServer* (VANGELISTA; ZANELLA; ZORZI, 2015). A Figura 19 apresenta a pilha de protocolos LoRa e LoRaWAN.

Figura 19 – Pilha de protocolos LoRa e LoRaWAN

Fonte: adaptada de LoRa-Alliance (2020)

O LoRaWAN projetado principalmente para redes de sensores, nas quais estes (*end nodes*) trocam pacotes com o servidor com baixa taxa de dados e intervalos de tempo bem longos, algo da ordem de uma transmissão por hora ou mesmo dias (AUGUSTIN *et al.*, 2016).

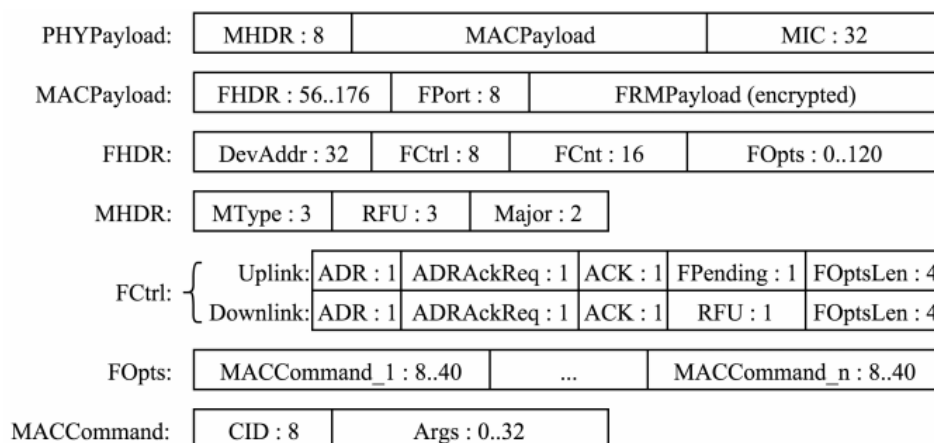
A rede LoRaWAN, conforme Augustin *et al.* (2016), classifica três classes de dispositivos: Classe A (para todos os dispositivos), Classe B (para *Beacon*) e Classe C (Ouve Continuamente), cada uma associada a um modo operacional diferente:

- Classe A: é o modo padrão das redes LoRa, devendo ser suportada por todos os dispositivos LoRa. Na rede de classe A, as transmissões são sempre iniciadas pelos nós finais, de maneira assíncrona;
- Classe B: os dispositivos finais de classe B, são sincronizados com o *NetServer* por meio de pacotes *beacon* que são transmitidos por *gateways* classe B, logo, podem receber dados de *downlink* ou pacotes de comando em janelas de tempo específicas, independentemente do tráfego de *uplink*;
- Classe C: É definida a nós finais que não tem restrições quanto ao consumo de energia, os quais podem manter a janela de recebimento sempre aberta.

O protocolo LoRaWAN não habilita as comunicações dispositivo a dispositivo, tal como acontece em uma rede *mesh*: nelas os *frames* são transmitidos apenas de um nó final para o servidor de rede ou vice-versa. A comunicação entre dispositivos finais, deve ser executada através do servidor de rede.

Augustin *et al.* (2016) detalha, o quadro LoRaWAN na camada física. Nele há o cabeçalho e o CRC mandatórios para mensagens de *uplink*. Por outro lado, os quadros de *downlink* possuem só o cabeçalho, mas não o CRC. Já a taxa de código (CR) a ser utilizada não é especificada. A Figura 20, apresenta o formato de um *frame* LoRaWAN:

Figura 20 – Formato do *frame* LoRaWAN com os tamanhos dos campos em bits



Fonte: adaptada de Augustin *et al.* (2016)

Augustin *et al.* (2016) apresenta a definição e as tarefas desempenhadas por cada campo do quadro LoRaWAN:

- DevAddr armazena o endereço do dispositivo;
- FPort é um campo de porta multiplexadora. Caso seja zero sinaliza que a carga contém comandos MAC. Se isso ocorrer, o campo FOptsLen fica com valor zero;
- FCnt armazena a contagem dos quadros;
- FOpts pega os comandos MAC em uma mensagem de dados;
- MIC código de integridade de mensagem criptográfica, calculado sobre os campos MHDR, FHDR, FPort e FRMPayload criptografado;
- MType é o tipo de mensagem, indicando se é uma mensagem de *uplink* ou de *downlink*. Também informa se é ou não uma mensagem confirmada. ACK são solicitados para as mensagens confirmadas;
- Major é a versão LoRaWAN; atualmente, só o valor zero é válido;
- ADR e ADRAckReq são responsáveis pelo mecanismo de adaptação da taxa de dados pelo servidor de rede;
- ACK reconhece o último quadro recebido;
- FPending indica que o servidor de rede possui dados adicionais para serem enviados e que o dispositivo final deve enviar outro quadro o mais rápido possível, para que ele abra as janelas de recebimento;
- FOptsLen é o comprimento do campo FOpts em *bytes*;
- CID é o identificador de comando MAC e Args são os argumentos opcionais do comando;
- FRMPayload é a carga útil, criptografada usando o AES com um comprimento de chave de 128 *bits*. O tamanho mínimo do cabeçalho MAC é 13 *bytes* e o máximo é 28 *bytes*. Sabendo disso, é possível calcular a capacidade máxima de canal disponível para cargas úteis de dados de aplicativos com determinados parâmetros de modulação.

Como os pacotes são enviados de um dispositivo para o servidor de rede e vice-versa, não há endereço de destino nos pacotes de *uplink* e não há endereço de origem nos pacotes de *downlink* (AUGUSTIN *et al.*, 2016).

Uma funcionalidade do LoRaWAN é a *Adaptive Data Rate*, que permite ao *NetServer* adaptar a taxa de transmissão de um dispositivo final alterando o índice SF, a fim de encontrar a melhor relação entre consumo de energia e robustez do *link*.

Outra característica importante, são os fortes mecanismos de segurança que envolvem uma chave de rede e uma chave de aplicativo, que são configuradas através de um procedimento de ativação aérea (*over-the-air activation*) ou por meio de uma ativação por procedimento de personalização (*activation-by-personalization*), na qual os parâmetros de segurança são configurados no dispositivo na hora da implantação (VANGELISTA; ZANELLA; ZORZI, 2015).

LoRa-Alliance (2020) destaca alguns desses comandos, chamados de comandos MAC, que podem personalizar alguns parâmetros do dispositivo final. Abaixo são destacados alguns deles:

- *LinkCheckReq*: pode ser enviado por um dispositivo final para testar sua conectividade;
- *LinkADRRReq*: esse comando do *Netserver* pode controlar a taxa de dados e a potência de saída usada pelo dispositivo, bem como o número de vezes que cada pacote não confirmado deve ser enviado;
- *DutyCycleReq*: define ciclo de trabalho do dispositivo, (*RXTimingSetupReq*, *RXParamSetupReq*) para modificar os parâmetros das janelas de recebimento e (*NewChannelReq*) para alterar os canais usados pelo dispositivo;
- *DevStatusReq*: verifica o nível da bateria e a qualidade de recepção de um dispositivo.

Centenaro *et al.* (2015) declara que a tecnologia LoRa possui vantagens frente às outras tecnologias Wireless e isso pode ser o diferencial no mercado de IoT. Dentre algumas dessas vantagens, pode-se destacar a modulação Chirp usada na camada física que permite robustez nas comunicações e transmissões de longo alcance.

CAPÍTULO 3

MATERIAIS E MÉTODOS

Neste capítulo é apresentado o processo de desenvolvimento do monitoramento da rede de sensores sem fio, utilizando o protocolo LoRa e o *Zabbix*.

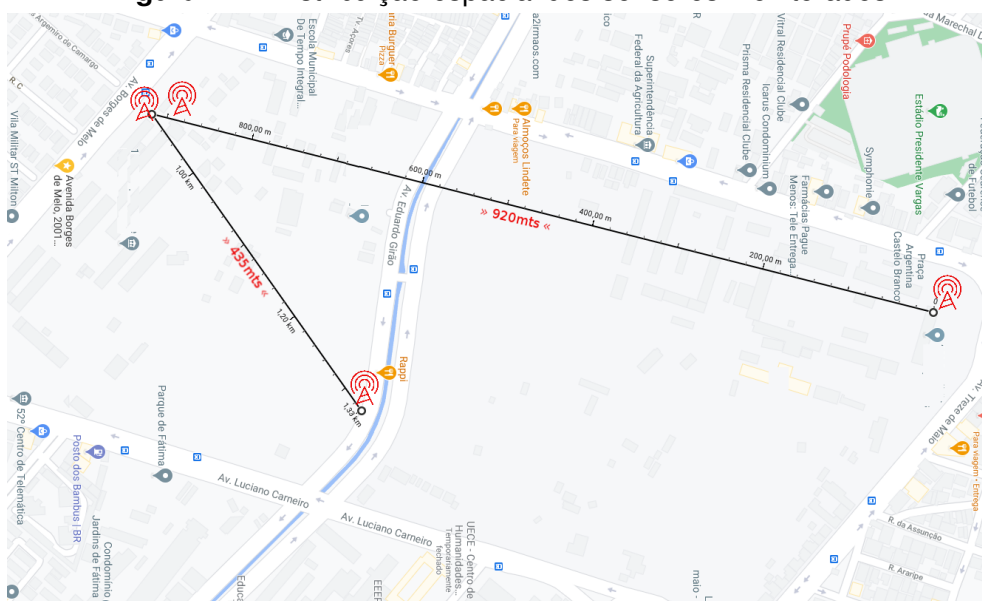
3.1 Estrutura

3.1.1 Princípio de funcionamento

A escolha por uma rede nova de monitoramento que fosse independente da rede já instalada, se deve ao fato de que em caso de interrupção da rede principal, perde-se a coleta de informações sobre o *status* do ambiente.

Uma rede diferente permite que as coletas continuem funcionando. Assim, a equipe responsável pode apurar com mais detalhes acerca do que está causando os incidentes no local.

Figura 21 – Distribuição espacial dos sensores monitorados



Fonte: Próprio autor

Conforme representado na Figura 21, foram determinados quatro pontos, nos quais os sensores serão testados, coletando dados e realizando a comunicação máquina a máquina com dispositivos não IoT.

Os nós sensores se comunicam com o nó agregador (em posição central) usando o protocolo LoRa em 915 MHz, faixa do espectro, que no Brasil, não requer licença da ANATEL para operação (ANATEL, 2017, 2018).

Nesse projeto, ainda cabe ressaltar que optou-se por sair da configuração comum muito utilizada com os dispositivos IoT que fazem uso dos protocolos LoRa/LoRaWAN.

Em tal padrão de configuração existem os nós sensores (*Nodes*) que realizam as coletas de dados relevantes, um *gateway* específico, o qual somente repassa os pacotes enviados pelos sensores, o *NetServer* que é um elemento de rede que proporciona o acesso à *Internet* e o Servidor de aplicação, que fica em algum local do globo e que pode ser acessado de qualquer ponto do planeta, desde que se tenha uma conexão de dados (LORA-ALLIANCE, 2020).

A configuração padrão foi descartada pelo fato da Organização onde o projeto foi implantado, não se sentir a vontade de ter dados sensíveis de instalações sob sua responsabilidade, hospedados em outro local. Local este, sob o qual não possui responsabilidade ou conhecimento pleno acerca das políticas que envolvem o armazenamento e o acesso a tais dados.

Dessa maneira, foram feitas algumas modificações na arquitetura, de modo que o armazenamento das informações e geração de suporte para apoio às decisões, ficarão hospedadas em um servidor de monitoramento local, dentro das instalações da própria Organização.

Se fez necessário ainda implementar alguns controles para possibilitar o tratamento de concorrência e tolerância a possíveis falhas de transmissão (com uso de recursos que possibilitem a confirmação de envio aos nós e políticas de retransmissões), em razão do uso do LoRa puro.

O chip LoRa utilizado é um transmissor comum, modelo DRF 1276f (DORJI, 2015), modelo genérico do SX 1276 (SEMTECH, 2015). Em razão do alto custo de aquisição do nó *gateway* e apesar das restrições de um nó comum, foi possível utilizar esses nós, tal como um *gateway*, sem perdas significativas de recursos e funcionalidades, dado o número reduzido de nós distribuídos no terreno.

Este servidor poderá ser acessado para consultas mediante uma conexão VPN -

Virtual Private Network, restritas ao pessoal responsável estritamente necessário.

Nessa configuração, em busca de eficiência no uso de energia, foi estabelecido que os sensores fiquem ativos o mínimo de tempo possível, cabendo ao nó agregador enviar uma mensagem aos nós sensores, solicitando todos os dados que eles possuem.

Por meio dessa chamada, os nós sensores saem do estado de *standby* e realizam em tempo pré-determinado uma coleta de dados local. Uma vez feita essa coleta, em momentos distintos, configurados pelo administrador, cada nó envia os valores ao nó agregador, que processa os dados recebidos, armazena-os no cartão de dados local e remete um ACK (*acknowledgment*) ao nó sensor, para que este incremente o contador de pacotes.

Em seguida, o *gateway* remete os dados a um outro nó, também via protocolo LoRa, utilizando uma função específica chamada `LoRa.enableInvertIQ()`, (MISTRY, 2016), que foi implementada para permitir somente a comunicação entre estes dois *gateways*, evitando que os nós sensores recebam tal mensagem desnecessariamente. Caso não receba o ACK deste outro nó, o nó agregador remeterá o dado novamente.

Este outro nó, que por sua vez já está localizado próximo ou dentro do *DataCenter*, com sua alimentação externa (também possui uma pilha Li-On 18650) sendo realizada pela infraestrutura local, desempenha a função do *NetServer* tradicional e tem a tarefa de repassar os pacotes oriundos da rede LoRa, para uma rede TCP/IP por meio do seu transmissor 802.11 embarcado.

Usando a aplicação *Zabbix Sender*, o *NetServer* consegue enviar as informações diretamente ao Servidor de Aplicação local, que possui o *Zabbix Server* instalado. Finalmente, tem-se o servidor de aplicação funcionando localmente, o qual armazena, plota gráficos e exibe as informações coletadas em uma interface gráfica, provendo o apoio a decisão e oferecendo maior confiabilidade e disponibilidade ao administrador local, responsável pelo sistema.

3.1.2 Ferramentas de desenvolvimento e código-fonte

O código fonte foi desenvolvido na aplicação Arduíno IDE (*Integrated Development Environment*) - Ambiente de Desenvolvimento Integrado, que é um *software* gratuito desenvolvido pela Arduíno.CC e compatível com uma grande variedade de placas, inclusive o ESP da Expressif.

Para utilizar o ESP nesta IDE, basta fazer a inserção de uma URL - endereço eletrônico, de uma página do projeto ESP 8266 e ESP 32. Na tela principal, clique em "Arquivos- "Preferências". Na nova tela aberta, há uma caixa chamada "URLs adicionais para gerenciadores de placas". Nesse local, basta inserir as seguintes URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json e https://dl.espressif.com/dl/package_esp32_index.json e clicar em "Ok".

Assim, se torna possível usar os ESP de maneira análoga às placas *Arduíno*, não sendo necessário a instalação de novos *softwares*, devido ao uso do ESP no projeto. A escolha definitiva de usar o ESP em detrimento dos *Arduínos* foram, basicamente, por dois motivos:

- Velocidade de clock: o clock do *Arduíno* com o chip ATmega328p é de 16MHz (ARDUÍNO.CC, 2015), já o clock do Esp 8266 é de 80MHz (ESPRESSIF-SYSTEMS, 2020b) e o do ESP 32 é de 240MHz (ESPRESSIF-SYSTEMS, 2020a), apesar dessa diferença significativa, que é importante para a quantidade de execução de tarefas, a faixa de preço dos ESP são bem próximas do *Arduíno*;
- Tensão de saída e nível lógico: o *Arduíno* trabalha com 5 volts tanto para alimentação quanto para nível lógico (ARDUÍNO.CC, 2015), já os ESP trabalham com 3,3 Volts. Como a maioria dos demais módulos (chip LoRa, *display* Oled, Cartao SD e alguns sensores), também funcionam com 3,3 Volts, isso veio a contribuir como mais um ponto a favor para o ESP, evitando assim o uso de mais um dispositivo, que seria um conversor lógico TTL de 5 para 3,3 Volts.

Também foram utilizadas na composição do código fonte as bibliotecas SD.h, *IRRemote.h*, SPI.h, LoRa.h, *Wire.h*, SSD1306.h, SD.h e RtcDS3231.h. Assim, foi possível codificar o regime de trabalho do microcontrolador e seus módulos acessórios, conforme a necessidade levantada pela Instituição.

No código do nó agregador, nó integrador e dos nós finais, temos a função *setupLoRa()*, que basicamente usa as funções da biblioteca LoRa.h, responsável por realizar todas as etapas de inicialização do chip LoRa DRF1276f, configurar o uso correto da rádio-frequência nas transmissões e executar a interrupções de *textitsoftware*, por ocasião do recebimento de um novo pacote LoRa (MISTRY, 2016).

Nos três tipos de nó, também temos as funções *sendMessage(parâmetro)* e *onReceive(parâmetro)*. A *sendMessage(parâmetro)* é responsável por pegar

as informações do parâmetro e enviá-las ao outro nó utilizando as funções *LoRa.beginPacket()*, *LoRa.write()* e *LoRa.endPacket*, também da biblioteca *LoRa.h*.

A função *onReceive*(parâmetro) é chamada assim que um pacote é detectado pelo chip *LoRa* que envia um sinal de interrupção ao *ESP*. Em seguida é verificado através da função *LoRa.read()*, se o pacote é de *broadcast* ou destinado àquele nó especificamente.

Nesse ponto, a função tem uma leve diferença caso esteja no nó agregador/nó integrador ou nó final. Iniciando pelo nó final, é verificado qual o tipo de pacote através do conteúdo do *payload* (se é uma requisição de dados ou se é um pacote de *ACK*), com isso uma *flag* de controle, chamada sucesso é sinalizada como "*true*".

Observando o nó agregador, a função *onReceive*(parâmetro) realiza além das funções relatadas no nó final, a tarefa de atribuir o conteúdo do *payload* a uma estrutura especial chamada *struct*, a qual agrega as variáveis que guardam as informações coletadas dos sensores. Além de verificar se o pacote de resposta coincide com o último enviado, armazenando o total de pacotes confirmados e perdidos.

O nó integrador faz exatamente o contrário do nó agregador, ele recebe do nó agregador, as informações da *struct*, retira as informações contidas ali e as coloca na estrutura da função *ZabbixSender()*, que remete os dados ao *Zabbix*.

O nó agregador possui a função *sendToGW*(estrutura), na qual ele pega a estrutura informada pelo nó final e também utilizando as funções *LoRa.beginPacket()*, *LoRa.write()* e *LoRa.endPacket*, a repassa ao segundo *gateway* - nó integrador.

Outra função exclusiva do nó agregador é a função *setupSD()*, que inicia o cartão *SD* para acesso do *ESP 32*, a *setupOled()*, que inicializa as tarefas para funcionamento do *Oled* utilizando a biblioteca *SSD1306.h* e a função *armazenaSD()* que guarda no cartão já em estado funcional, as informações passadas pelos nós finais, fazendo uso da biblioteca *SD.h*.

Há ainda a função *sendACK()*, que após receber com sucesso os dados do nó final, envia uma mensagem informando a ele que recebeu todos dados. Com isso o nó final incrementa seu contador de pacotes.

Nos nós finais, têm-se ainda as funções *readSensorTempLocal()*, *verificaTempAC()*, *SendIR()*, *coletaTensão()* e *coletaDados()*.

A função *readSensorTempLocal()* faz uso das bibliotecas *OneWire.h* e *DallasTemperature.h* e vai executar as funções na qual utiliza o sensor DS18B20 para coletar a temperatura do ambiente. Elas funcionam com o sensor digital e utilizam apenas um pino, economizando assim, os pinos do ESP 8266, que já são escassos.

A função *sendIR()*, que é provida pelas bibliotecas *IRremoteESP8266.h* e *IRsend.h* é responsável por controlar o emissor IR para efetuar a emissão de sinal infra-vermelho para o aparelho de ar-condicionado instalado no ambiente.

A função *verificaTempAC()* trabalha em conjunto com *readSensorTempLocal()* e *sendIR()*, utilizando a primeira para consultar a temperatura. Em caso da temperatura estar abaixo ou acima do considerado ideal, *sendIR()* é usada.

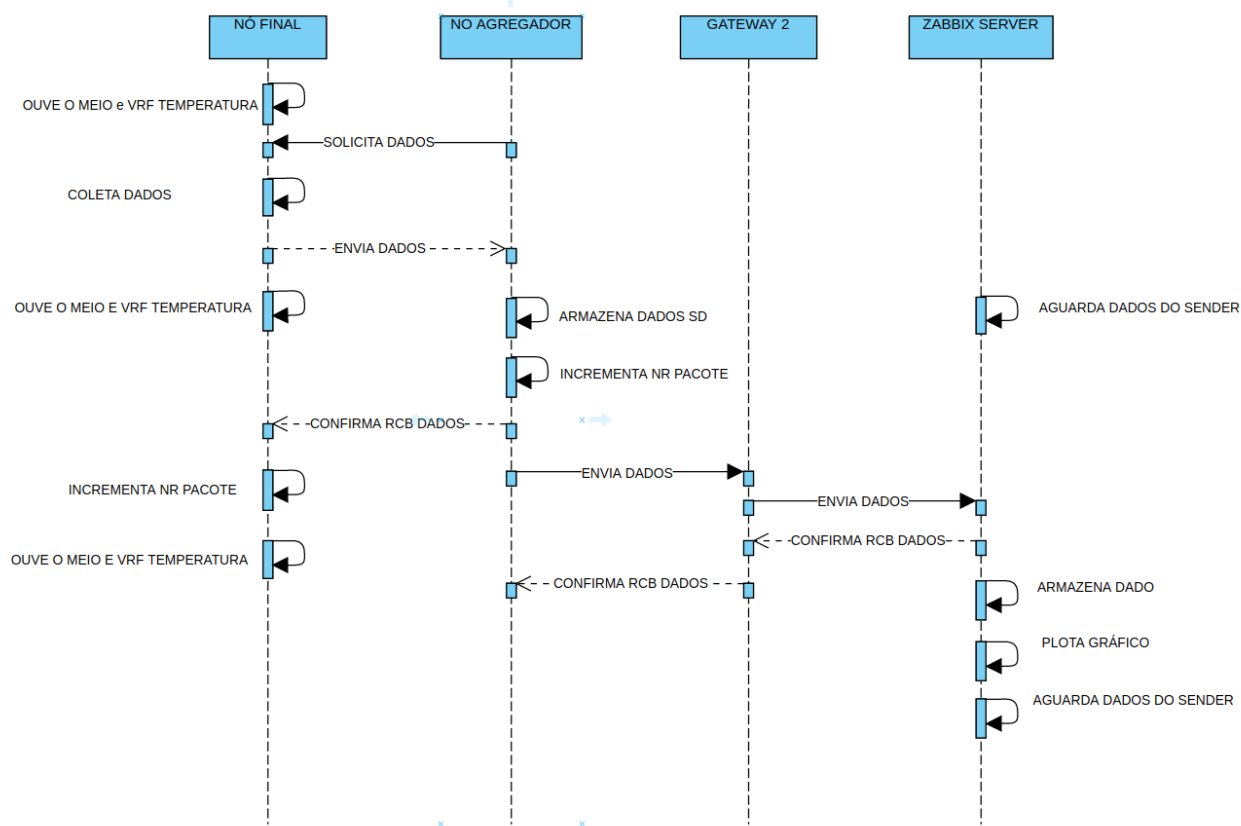
Ela atua no aparelho de ar-condicionado, corrigindo a temperatura, de modo a utilizar o aparelho de maneira mais econômica, respeitando o regime de trabalho previsto em manual específico. Aqui se concretiza a comunicação M2M.

A função *coletaTensão()* utiliza um sinal alto ou baixo no pino D0 ou GPIO 16 - *Generic Purpose Input/Output* para realizar o chaveamento de um relé de 5v, de modo a alternar a coleta no pino AD0. Isso permite que ora se colete a tensão da bateria, com um sinal baixo, e em seguida, com um sinal alto, a coleta da tensão no painel solar. Essa função não necessita de nenhuma biblioteca específica, além das básicas do *Arduíno* IDE.

A função *coletaDados*, encapsula todas as funções acima, chamando-as em sequência e armazenando suas informações na estrutura que consolida as variáveis de coleta, a *struct tDados*.

Por último, a função *loop()*, padrão do *Arduíno* IDE, vai chamar *coletaDados()* e *sendMessage()*, colocar o chip LoRa novamente em modo *receive* e aguardar novas requisições.

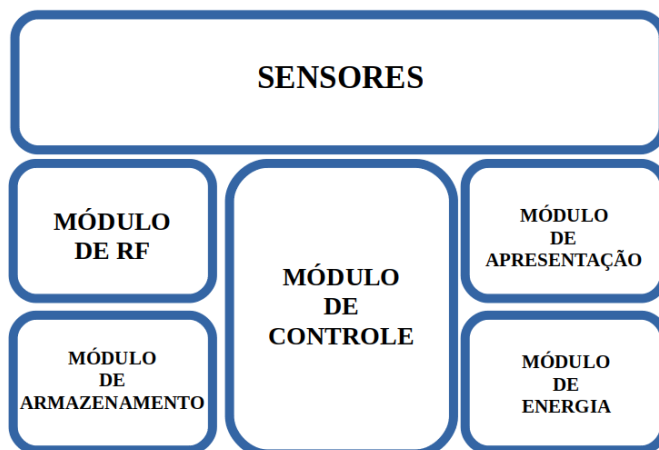
Uma vez conhecidas as principais funções, segue-se o detalhamento do ciclo de funcionamento e troca de mensagens entre os nós que compõe o projeto, apresentado na Figura 22.

Figura 22 – Diagrama de sequência do funcionamento e troca de mensagens

Fonte: Próprio autor

3.1.3 Hardware dos Nós

Os nós sensores, possuem o módulo de controle, os sensores, o módulo de RF (Rádio Frequência), o de armazenamento, o de apresentação e o módulo Energia.

Figura 23 – Estrutura dos nós sensores

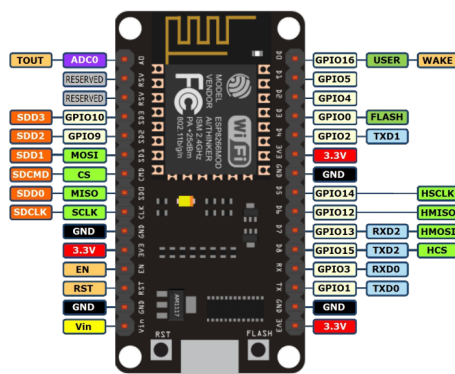
Fonte: Próprio autor

Devido à flexibilidade da unidade de controle (microcontrolador), mais sensores podem ser adicionados conforme a necessidade, limitados apenas pela quantidade de pinos e memória RAM disponíveis para seu funcionamento. Na Figura 23 é apresentada a estrutura básica dos nós, com o microcontrolador e seus periféricos.

- **Unidade de Controle:** as unidades de controle são compostas pelos Módulos Esp8266 e Esp32, ambos fabricados pela *Expressif Systems*.

O ESP 8266, conforme a Figura 24, de acordo com ESPRESSIF-SYSTEMS (2020b), possui Wi-Fi embutido, tem 16 conectores GPIO, barramentos I2C (*Inter-Integrated Circuit*), SPI (*Serial Peripheral Interface*), 2 interfaces, UART (*Universal Asynchronous Receiver/Transmitter*), entrada ADC, saída PWM e sensor interno de temperatura, CPU que opera em 80 MHz, arquitetura RISC de 32 *bits*, 32 *kBytes* de RAM para instruções, 96kBytes de RAM para dados, 64 *kBytes* de ROM para *boot*, possui uma memória Flash SPI Winbond W25Q40BVNIG de 512 *kBytes*. O núcleo é baseado no IP *Diamond Standard LX3* da *Tensilica*, o ESP utilizado está em uma placa *NodeMCU v3 Lolin*.

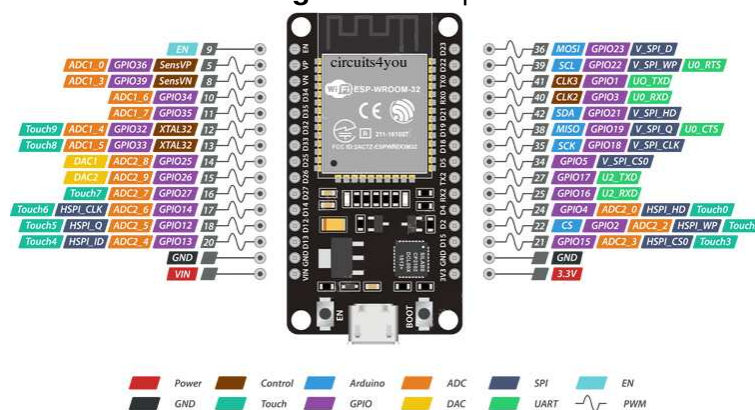
Figura 24 – Esp8266 NodeMcu



Fonte: Adaptado de randomnerdtutorials.com (2019)

O ESP 32 é apresentado na Figura 25. Bem mais poderoso do que seu antecessor, conforme ESPRESSIF-SYSTEMS (2020a), possui como processador principal, o microprocessador *Tensilica Xtensa* 32-bit LX6, com 1 ou 2 núcleos, frequência de até 240 MHz, possui um coprocessador auxiliar, que consome bem pouca energia, conectividade *wireless* Wi-Fi: 802.11 b/g/n, *bluetooth*: v4.2 BR/EDR e *bluetooth Low Energy* (BLE), memória ROM de 448 kB, memória SRAM de 520 kB, RTC slow SRAM: 8 kB, RTC fast SRAM: 8 kB, eFuse: 1 *kbit* – 256 *bits* são usados para sistema (endereço MAC e configurações do *chip*), e os 768 *bits* restantes são reservados para aplicações incluindo criptografia da *Flash* e *Chip-ID*.

Figura 25 – Esp32



ESP32 Dev. Board Pinout

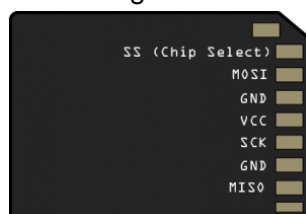
Fonte: Adaptado de Arduining.com (2018)

Possui memória *flash* externa com suporte de até 16 MB. Possui periféricos de comunicação com suporte a DMA. 10 GPIOs com suporte a toque capacitivo, 16 canais de conversor SAR ADCs (conversor analógico-digital) de 12-bits, 2 canais de 8 *bits* (conversor digital-analógico), 2 *interfaces* I2C, 2 *interfaces* UART, controlador CAN 2.0 (*Controller Area Network*), 4 *interfaces* SPI, I2S (*Integrated Inter-IC Sound*), RMII (a parte *Ethernet* do ESP32), 16 canais de PWM (*Pulse Wave Modulation*) - modulação por largura de pulso).

Possui conectividade IEEE 802.11 com suporte a protocolos de segurança WFA, WPA/WPA2, *boot* seguro, criptografia de *Flash*, aceleração de Criptografia em *Hardware* usando: AES, SHA-2, RSA, ECC e RNG, neste projeto o modelo utilizado é o ESP-WROOM-32.

- **Unidade de Armazenamento:** a unidade de armazenamento foi concebida utilizando um modelo proposto por Mischianti (2019), na qual um cartão SD comum, conforme a Figura 26, (*Security Digital*) de menor capacidade, é empregado no protótipo. Sendo conectado à unidade de controle via protocolo SPI, realizando a função de persistir os dados alocados na memória SRAM.

Figura 26 – Diagrama do Cartão SD



Fonte: Adaptado de Mischianti (2019)

- **Módulo de apresentação:** o Oled (*Organic Light-Emitting Diode*), ilustrado na Figura 27, significa diodo orgânico que emite luz, é responsável por apresentar o que está ocorrendo no sensor. Possui 128x64 *pixels*, não necessita de *backlight*, o que é uma grande vantagem sobre os visores LCD, pois consomem menos energia. Possui também, a *interface* i2c de comunicação com microcontrolador e as seguintes especificações: tensão de operação entre 2,2 e 5,5 V, o controlador é o *chip* SSD1306, com resolução de 128x64 e dimensões compactas de 30 x 27mm.

Figura 27 – Visor Oled



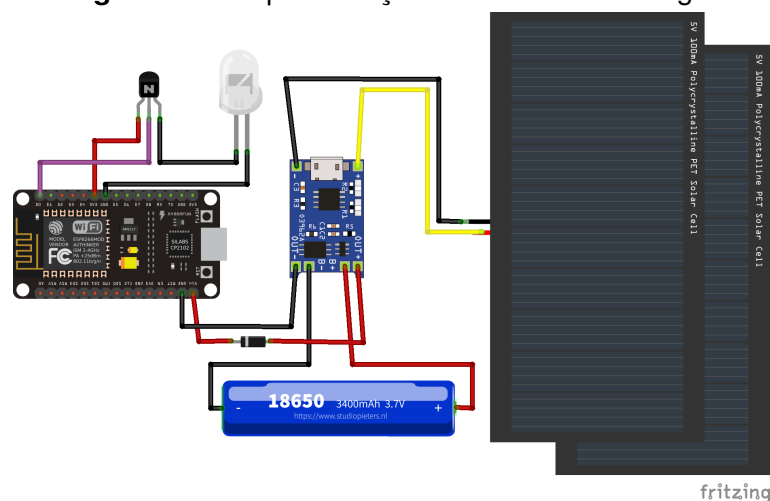
Fonte: Autoria própria

- **Módulo de energia:** energia é algo crítico para um dispositivo IoT e nesse projeto não é diferente. Os ESP (8266 ou 32) devem consumir o mínimo possível e que também, seja uma carga menor do que aquela que é fornecida pelo painel solar para as pilhas, permitindo assim a recarga da mesma em tempo hábil nos momentos em que não houver luz solar.

Neste projeto o módulo de energia tem em sua composição, dois painéis solares de 06 (seis) volts conectado em paralelo para dobrar a sua capacidade de fornecimento de corrente. Desse modo, os painéis conseguem fornecer mais carga, para a bateria Li-On 18650 (reaproveitada de um notebook) e com capacidade de 2000mAh.

Esse ciclo de carga é devidamente gerenciado por uma placa controladora modelo TP4056, a qual corta a entrada de carga quando a pilha atinge em torno de 4,18 Volts e também corta a saída de carga quando a pilha fica abaixo de 3,4 Volts, preservando assim a vida útil da mesma. A Figura 28, descreve o diagrama do sistema elétrico do projeto.

Figura 28 – Representação do módulo de energia



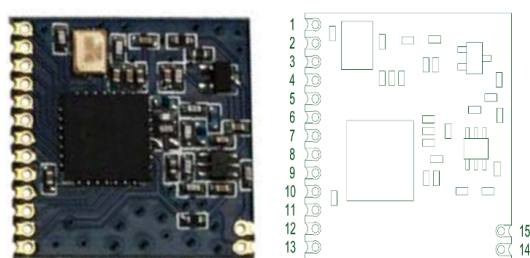
Fonte: Próprio autor

- **Módulo de RF:** esse módulo é responsável por transportar os dados coletados pelo sensores até o local de destino, onde os mesmos serão armazenados e utilizados para suporte à decisão. Nesse projeto o modelo utilizado é o DRF 1276f (DORJI, 2015), modelo genérico e de menor custo de aquisição em relação ao SX 1276 da *Semtech* (SEMTECH, 2015).

Possui também as seguintes especificações: frequência de 915MHz, aceita modulação FSK/GFSK/MSK/LoRa, conexão por SPI, *Sensitivity* de até -137dBm, potência de saída de 20dBm, por *default* é de 17dBm. Taxa de dados < 300 kbps, valor suficiente para a coleta de dados de texto, 127dB de *range* RSSI, imunidade a ruídos, detecção de preâmbulo de pacote, bit *synchronizer* embutido.

Aceita pacotes de até 256 *bytes* com CRC. Temperatura de trabalho entre -40°C e 80°C, sensor de temperatura embutido, corrente de standby <1uA e tensão de operação entre 1.8 e 3.6 V. A Figura 29 apresenta o modelo utilizado.

Figura 29 – Diagrama do chip LoRa DRF1272f/1276f



Fonte: adaptado de Dorji (2015)

- **Sensores:** os sensores são uma parte bastante relevante deste trabalho, haja vista que eles são os responsáveis por monitorar e coletar informações

relevantes do ambiente externo.

Dos sensores utilizados, podemos destacar o DS18B20 que possui alimentação de 3,3 volts a 5 volts, consumo de 1mA, faixa de temperatura de -55° a 125°C, precisão de 0,5°C e resolução de 9 a 12 bits.

Este sensor precisa um resistor de 4,7kΩ entre seu pino de alimentação e o pino de dados. Possibilita a medição de temperaturas negativas e positivas, com bastante precisão. Tem um consumo bastante reduzido, o que vem de acordo com o desejado no projeto.

O Emissor IR TSAL6200, funciona em conjunto com o sensor DS18B20, atuando no aparelho de ar-condicionado. Funciona semelhantemente a um LED, a única diferença é que o TSAL6200 emite sinais IR, ao invés de luz visível, quando recebe pulsos elétricos vindos do ESP.

Possui as seguintes especificações: comprimento de onda: 940 nm, corrente direta: 100 mA, tensão no sentido direto: 1,35 Volts, temperatura de operação:-55° a 100°C, tempo de queda: 800 ns, diâmetro: 5mm.

Esses dois dispositivos serão responsáveis por controlar a temperatura da sala e agir no dispositivo não IoT de ar-condicionado, para manter a temperatura de acordo com o exigido, pelos equipamentos ali existentes.

3.2 Instalação e locais de testes

Os testes para verificar a viabilidade deste projeto ocorreram em dois locais distintos. O primeiro local planejado foi uma sala com equipamentos de comunicação da Instituição, localizado no 16º andar de um prédio na mesma cidade e uma sala no 6º andar neste mesmo prédio, para realizar um teste de como seria a qualidade do enlace em ambientes internos - *indoor*, haja vista que há a pretensão de se instalar sensores que se comuniquem entre si em dependências sem visada direta.

O segundo local de teste planejado, foi a sede da organização, primeiramente em um local a 434 metros do *DataCenter* principal, o segundo ponto está a 916 metros do *DataCenter*, há ainda um terceiro ponto, a 3000 metros do *DataCenter*, o qual inicialmente não será utilizado em situação real, mas foi testado para verificar o alcance máximo do sinal com uma antena simples.

Para os testes foi utilizada uma máquina virtual *Zabbix* na versão 4.0,

disponibilizada gratuitamente pela empresa VerdanaDesk em seu *site* (SALES, 2018), desempenhando nesse projeto, um papel semelhante ao de um LoRa *Server*.

3.3 Custos do projeto

A implantação deste projeto com seus protótipos demandaram um desembolso de recurso financeiro para a sua aquisição, conforme descrito na Tabela 2.

Tabela 2: Tabela de custos

Item	Descrição	Qtd	Vr. Unit. (R\$)	Vr. Total (R\$)
1	Microcontrolador ESP 32 - Dev Rom	1	76,90	76,90
2	Microcontrolador ESP 8266 - Node MCU	3	30,00	90,00
3	Sensor DS18B20	2	16,90	33,80
4	Emissor IR TSAL6200	2	1,20	2,40
5	Módulo LoRa DRF 1276f	4	52,50	210,00
6	Módulo TP4056	4	6,90	27,60
7	Display Oled 128x64	1	41,90	41,90
8	Módulo RTC 3231	1	18,90	18,90
9	Placa ilhada 5x7 cm	6	2,90	11,60
10	Painel Fotovoltaico 6v 100mA	6	25,00	150,00
11	Display LCD 16x2	1	19,90	19,90
12	Resistores (100, 220, 560, 1k, 4,7k 10k) Ω	24	0,70	15,40
13	Transistor NPN BC337	6	0,30	1,80
14	Transistor PNP BC327	2	0,25	0,50
15	Relé 5v	4	4,90	19,60
16	Caixa Plástica multiuso	3	8,90	26,70
Total (R\$) =				749,40

Conforme a Tabela 2, o custo do projeto ficou em torno de R\$ 750,00 para a implantação, algo em torno de US\$ 150 dólares. O valor não é muito alto quando se considera que este custo é dividido por quatro nós. Desse modo, acredita-se que a implantação não causará muitos impactos negativos no orçamento.

RESULTADOS

Este capítulo trata dos resultados obtidos a partir dos testes realizados em ambiente *indoor* e *outdoor*, para determinar a viabilidade do projeto proposto.

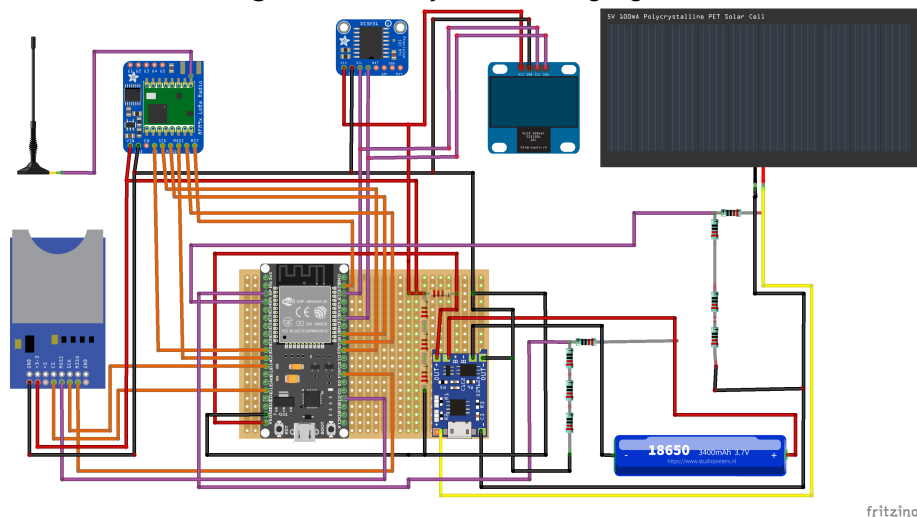
4.1 Protótipos

Para este projeto foram desenvolvidos três tipos de nós sensores: os nós finais, um nó agregador e um outro nó, o qual foi chamado de integrador, que tem a função de encaminhar os dados vindos da rede LoRa para a rede TCP/IP, buscando alcançar o sistema de monitoramento.

4.1.1 Nó agregador

Dispositivo que tem a função de solicitar os dados aos nós sensores por meio do código em linguagem C, que ao receber os dados repassa a um outro nó. A Figura 30 apresenta o modelo projetado na aplicação *Fritzing*.

Figura 30 – Projeto do nó Agregador

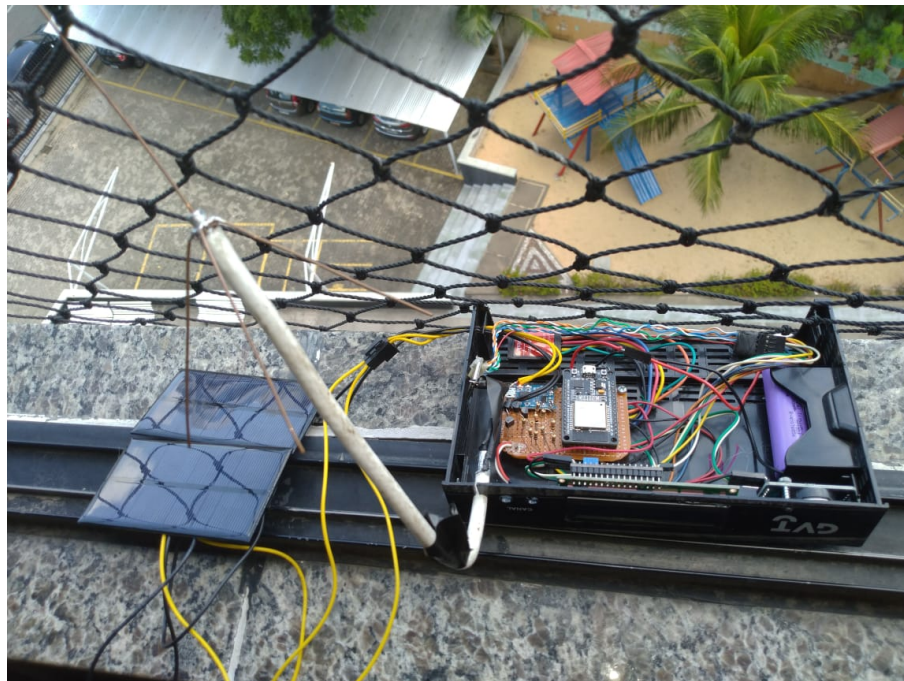


Fonte: Próprio autor

fritzing

Figura 31, apresenta o protótipo, baseado no projeto, desenvolvido e funcional. O código-fonte do nó agregador se encontra no Apêndice A.

Figura 31 – Protótipo do nó Agregador

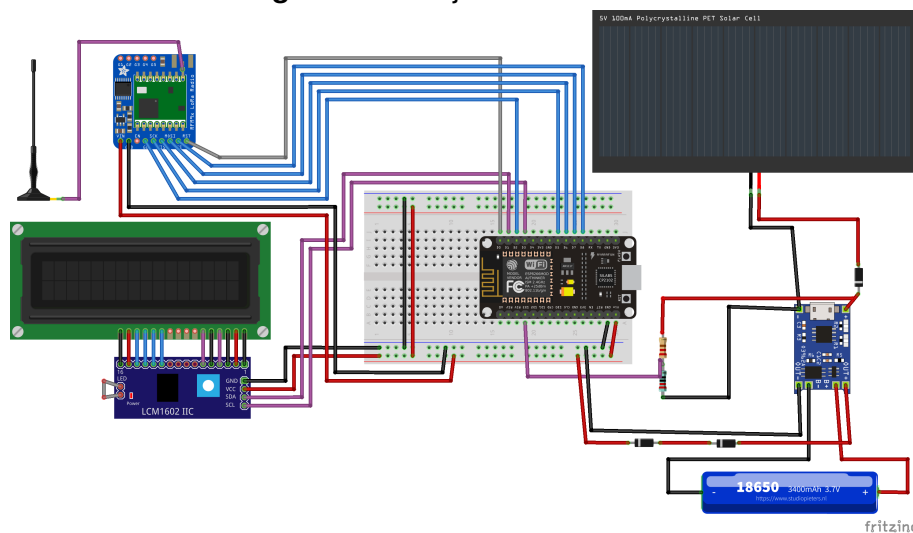


Fonte: Próprio autor

4.1.2 Nó sensor

Dispositivo que tem a função de ordenar ou executar a coleta de dados do ambiente por meio de sensores. Segue seu projeto desenvolvido no *Fritizing*, conforme a Figura 32. O código-fonte do nó final é apresentado no Apêndice B.

Figura 32 – Projeto do nó Sensor

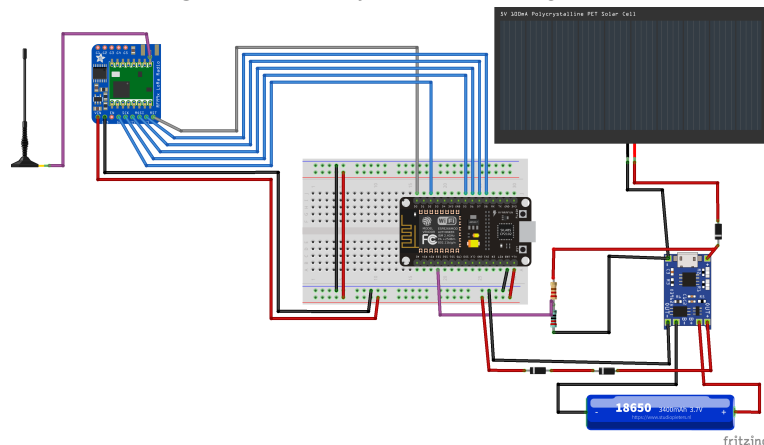


Fonte: Próprio autor

4.1.3 Nó integrador

É na verdade, um segundo *gateway*, que faz a integração entre a rede LoRa e a Rede IP, por meio do Wi-Fi (802.11). Segue seu projeto desenvolvido no *Fritizing*, conforme a Figura 33. O código-fonte do nó integrador é apresentado no Apêndice C.

Figura 33 – Projeto do nó Integrador

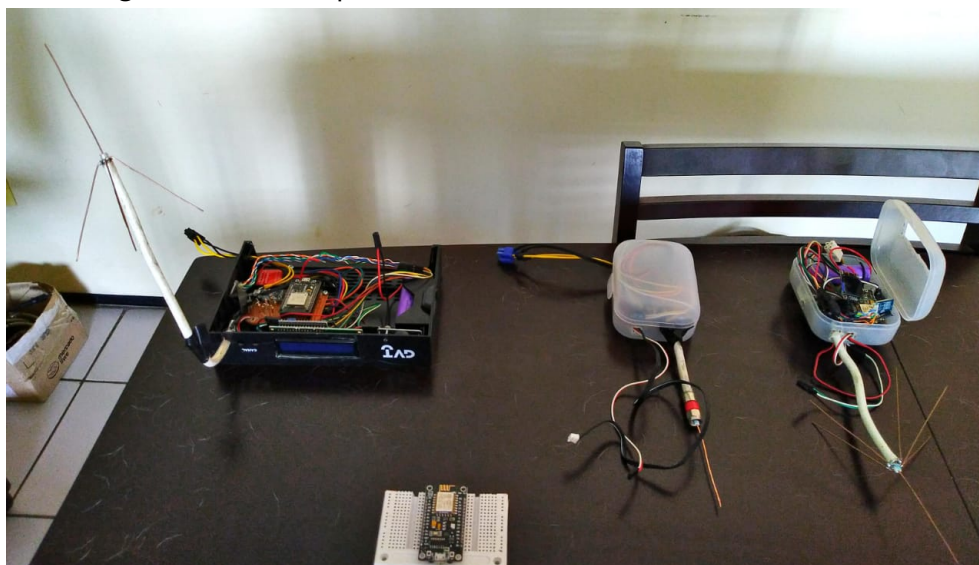


Fonte: Próprio autor

4.1.4 Cuidados acerca da codificação dos protótipos

Em razão do custo, não foi utilizado o nó concentrador padrão do mercado, com os modelos com os *chips* SX 1301, SX 1302 e SX 1308 da *Semtech* (LIANDO *et al.*, 2018). Neste caso, um dos *chips* transmissores foram utilizados como *gateway*. A Figura 34, apresenta os três tipos protótipos utilizados.

Figura 34 – Protótipos utilizados no sistema de monitoramento



Fonte: Próprio autor

Apesar de bem mais baratos, utilizar transmissores como *gateway*, traz a tona, algumas restrições, tais como o fato de só ouvir um canal (SF) por vez. Desse modo, ele só pode interagir com um transmissor de cada vez, além de não ser aceito, nem recomendado como nó *gateway*, pelos servidores de aplicação, tais como a TTN - *The Things Network*. Exceto se o dispositivo for configurado para o modo legado, que ainda impõe a limitação de somente aceitar nós LoRaWAN.

Outro cuidado observado nesse teste, foi transmitir as informações dentro de um único pacote, evitando explicitamente a fragmentação do pacote transmitido, uma vez que reunir pacotes fragmentados demandaria mais processamento e técnicas de controle, aumentando a complexidade do projeto.

Assim, as informações coletadas, acrescidas das informações de cabeçalho, deveriam ser limitadas a 255 bytes (LIANDO *et al.*, 2018). Felizmente, as informações utilizadas na *struct* que foi criada para encapsular os vários tipos de dados a serem transmitidos entre os nós, ficaram abaixo desse limite, algo em torno de 60 bytes.

4.2 Testes realizados

Para demonstrar a viabilidade de enlaces sem fio, baseados na tecnologia LoRa, foram realizados 03 (três) tipos de testes: intensidade de sinal, qualidade de sinal e perda de pacotes. Estes focados no sucesso da transmissão de dados.

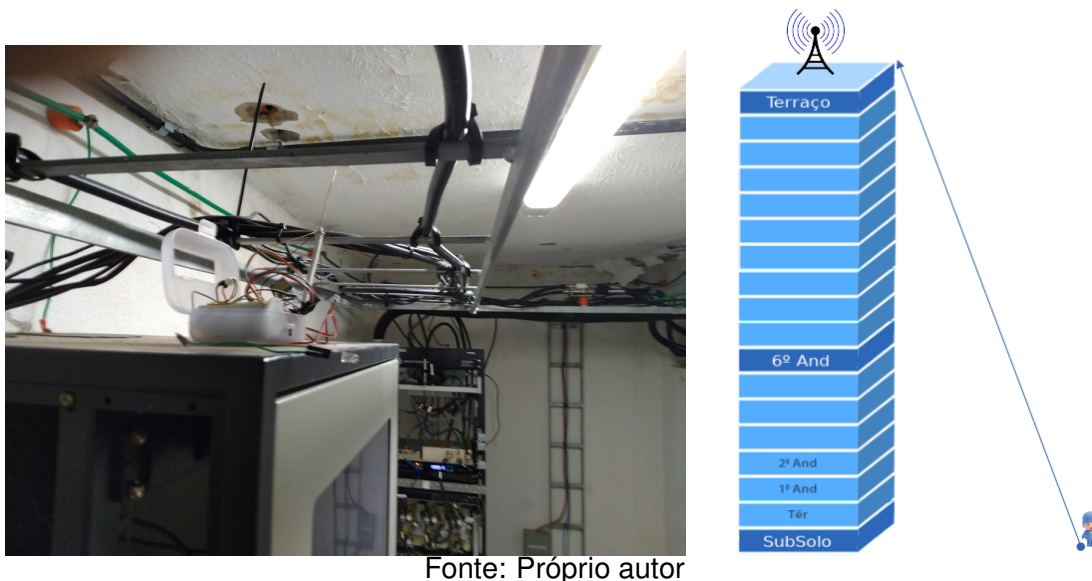
Em seguida, foram feitos testes relativos a parte elétrica. Após uma análise, verificou-se que devido à uma funcionalidade do LoRa, implementada pela *Semtech*, os testes de transmissão poderiam ser realizados simultaneamente.

4.2.1 Condições de realização dos testes

Os testes de intensidade e qualidade, baseados no RSSI e SNR, respectivamente, foram realizados em duas condições:

- Em ambiente interno (*indoor*): realizado a fim de verificar o desempenho do protocolo, em ambientes fechados, com o sinal cruzando várias paredes ou lajes.

No primeiro teste, o nó sensor foi posto na sala de telecomunicações, no 16º andar, e o *gateway* em uma sala no 6º piso e no Térreo, conforme Figura 35.

Figura 35 – Teste Sala de comunicações - 16º andar

O resultado obtido é que foi possível detectar o sinal enviado, tanto no 6º andar, quanto no térreo e no subsolo.

Também foi realizado um segundo teste, entre dois *datacenters* - DC 1 e DC 2, com distância de 90 metros, conforme Figura 36. Este teste teve por finalidade, verificar o funcionamento em um local com bastante ruído eletromagnético, em razão de possuir equipamentos enviando sinal RF de outros protocolos, *nobreaks* de grande capacidade, dentre outros. Mas uma vez houve sucesso na transmissão dos dados.

Figura 36 – Teste entre DC 1 (à esquerda) e DC 2 (à direita)

- Em ambiente externo (*outdoor*): neste teste, iniciou-se a transmissão de dados com a distância de 180m, passando 410m, 600m, 620m, 750m e 860m, todas com o receptor no nível do solo, e com o nó agregador no terraço de um prédio de 16 andares.

Nestes locais não havia uma visada totalmente direta com o emissor, por estar em uma região densamente habitada e no nível do solo. Contudo os resultados foram promissores e possivelmente com a instalação de dois rádios em uma torre alta com visada direta, há a possibilidade dos resultados serem ainda melhores.

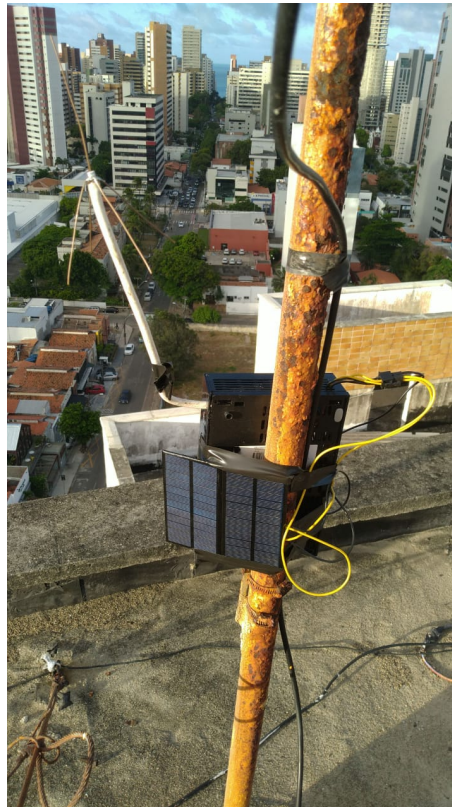
A Figura 37 apresenta uma visão geral dos locais onde foram realizados os testes externos. Convém ressaltar que se buscou locais com razoável visualização do prédio no qual estava o nó agregador, com presença de alguns obstáculos, tais como de veículos grandes e vegetação.

Figura 37 – Área geográfica dos testes *outdoor*



Fonte: Próprio autor

O nó agregador foi instalado em um ponto no alto do edifício, conforme a Figura 38, de modo a permitir que o sinal pudesse alcançar maiores distâncias, apesar da localidade possuir bastantes construções, que podem impactar a qualidade do sinal emitido.

Figura 38 – Local de instalação do nó agregador

Fonte: Próprio autor

A Figura 39 apresenta o ponto mais distante para o teste do sinal. Localizado aproximadamente 850 metros do nó agregador, de modo a testar o comportamento do enlace, mesmo diante de barreiras físicas e possíveis interferências existentes.

Figura 39 – Local de coleta externa

Fonte: Próprio autor

Houve ainda a coleta do sinal em um local distante com razoável vegetação, retratado pela Figura 40. Neste local o desempenho não foi tão bom quanto os demais testes. Acredita-se que o motivo primordial que impactou o desempenho do enlace, foi a distância em relação ao emissor que estava dentro de uma sala fechada, a 370 metros de distância e também devido à vegetação ao redor.

Figura 40 – Local com vegetação



Fonte: Próprio autor

4.2.2 Teste de intensidade / qualidade do sinal - Teste RSSI/SNR

O protocolo LoRa é um protocolo de camada física puro. A maioria das ferramentas que possibilitam o controle sobre a transmissão, precisam ser implementadas em software, ficando a cargo do desenvolvedor da solução.

Contudo, os dispositivos que usam o protocolo LoRa, tem implementados a funcionalidade de informar o RSSI e o SNR a cada pacote transmitido, por meio das funções *LoRa.packetRssi()* e *LoRa.packetSnr()*, constantes da biblioteca *LoRa.h*.

Logo, isso pôde ser utilizado para apurar informações sobre o sinal recebido, sendo um parâmetro importante para o projeto em tela.

Convém observar que o limite da sensibilidade do LoRa muda conforme o SF aplicado. Desse modo, uma das premissas utilizadas, foi realizar testes em todos os SF possíveis e analisar o resultado obtido.

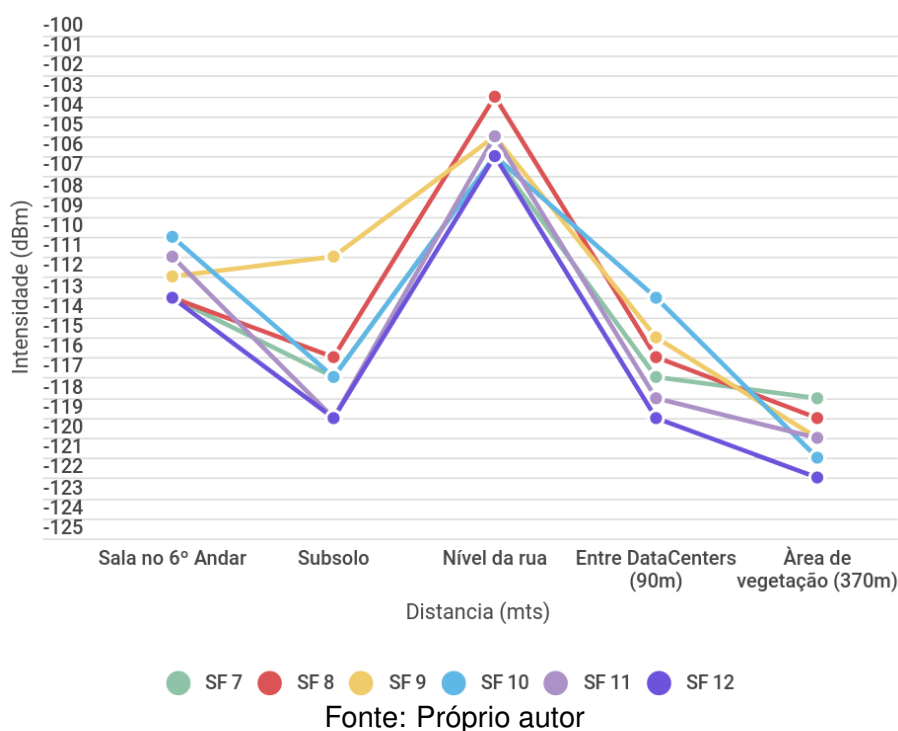
Assim foi criado na própria arduíno IDE, um código, descrito no Apêndice D, no qual o nó agregador dispara em cada SF (entre 7 a 12) pacotes com uma *string* de texto a cada 6 segundos, até um total de 200 pacotes.

O nó final, cujo código está descrito no Apêndice E, armazena a mensagem enviada, o número do pacote, o modo de SF, o RSSI e o SNR (estes dois informados pelo LoRa). Além de somar o RSSI e o SNR pra criar uma média, bem como, incrementa o valor do contador de mensagem a cada pacote recebido.

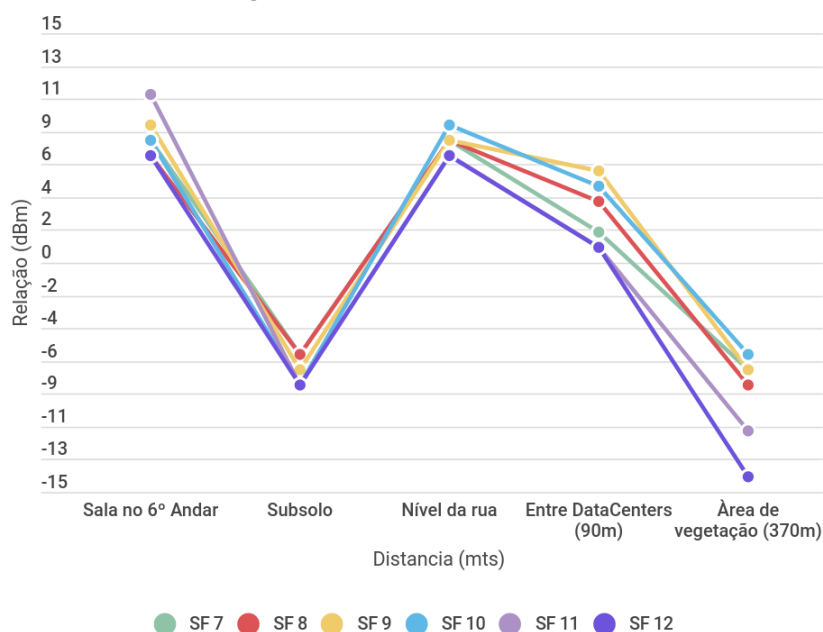
Ao final desse período, o nó transmissor ordena ao seu nó final que mude o SF para o seguinte e contabilize os resultados dos dados acima. Ao mudar, o nó final avisa ao nó agregador que fez a mudança e está em condições de receber novos pacotes. Diante da confirmação, o nó agregador muda seu SF e começa a disparar mais 250 pacotes no SF seguinte e assim segue até o SF12, quando ao final retorna ao SF7, infinitamente. São apresentados em seguida, os resultados coletados nos testes realizados.

Os gráficos dos resultados coletados, referentes ao testes de intensidade do sinal (RSSI), em ambiente interno, é apresentado na Figura 41.

Figura 41 – Teste Indoor RSSI



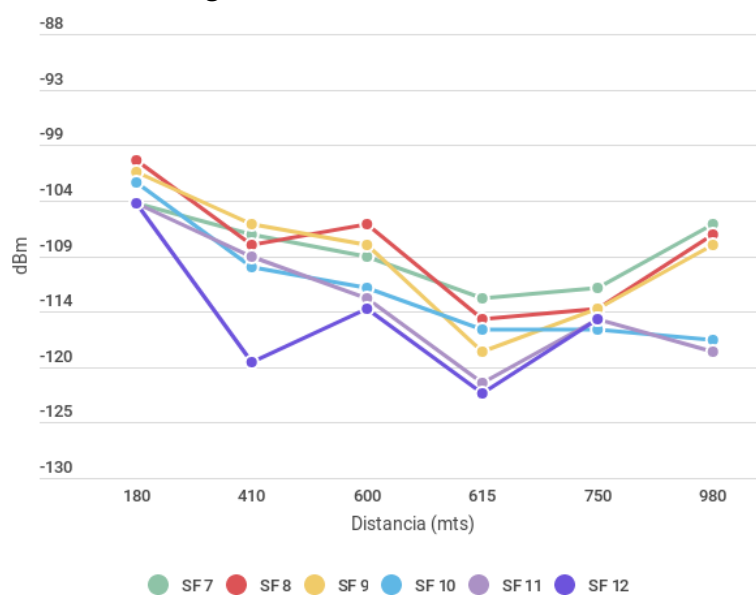
A Figura 42, ilustra os gráficos dos resultados do teste de qualidade do sinal (SNR) em ambiente interno.

Figura 42 – Teste *Indoor* SNR

Fonte: Próprio autor

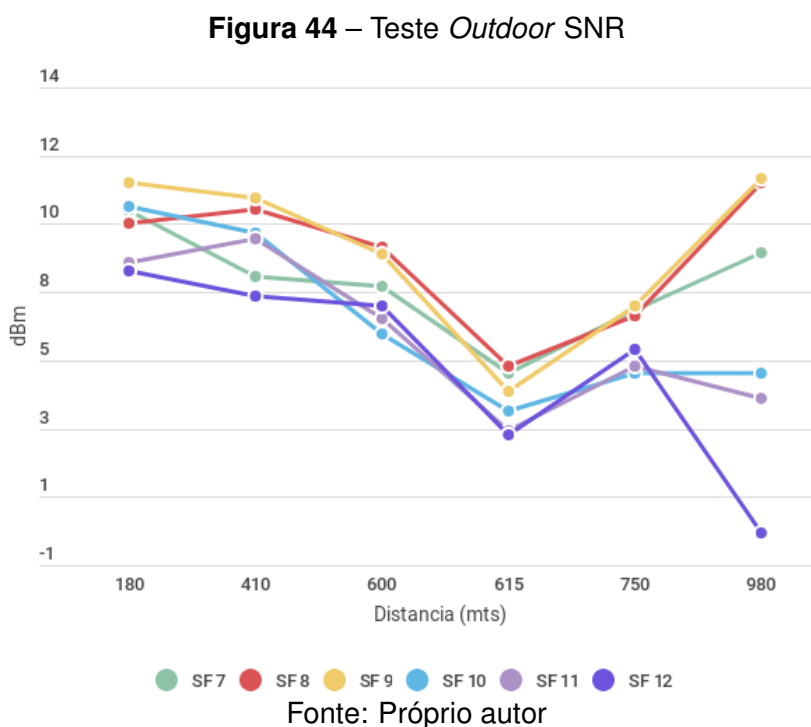
Pode-se observar nos testes realizados, que a potência recebida e a qualidade do sinal aumentam quando em visada livre. Por isso com o receptor na rua, mesmo 30 metros abaixo, o sinal chega com razoável qualidade.

Em uma área com vegetação mais densa e/ou com bastante interferência, a qualidade do sinal será impactada negativamente. Contudo, devido a resiliência do protocolo LoRa, quase nenhuma informação foi perdida durante a execução destes testes. A Figura 43, apresenta o resultado do teste de intensidade (RSSI) *outdoor*.

Figura 43 – Teste *Outdoor* RSSI

Fonte: Próprio autor

Na Figura 44 estão os resultados coletados no teste de qualidade do sinal (SNR) em ambiente externo.



Aqui observa-se que o sinal coletado possui intensidade semelhante nos vários SF medidos, porém em ambiente ruidosos o SF 12 sofre um desvio maior em relação aos demais SF tanto no RSSI, quanto na relação SNR. Contudo, este tipo de sinal consegue chegar a maiores distâncias.

4.2.3 Teste de perda de pacotes

Observa-se neste teste, que à medida que o sinal enfraquece, seja devido à distância e às interferências no sinal transmitido, a taxa de perda de pacotes aumenta.

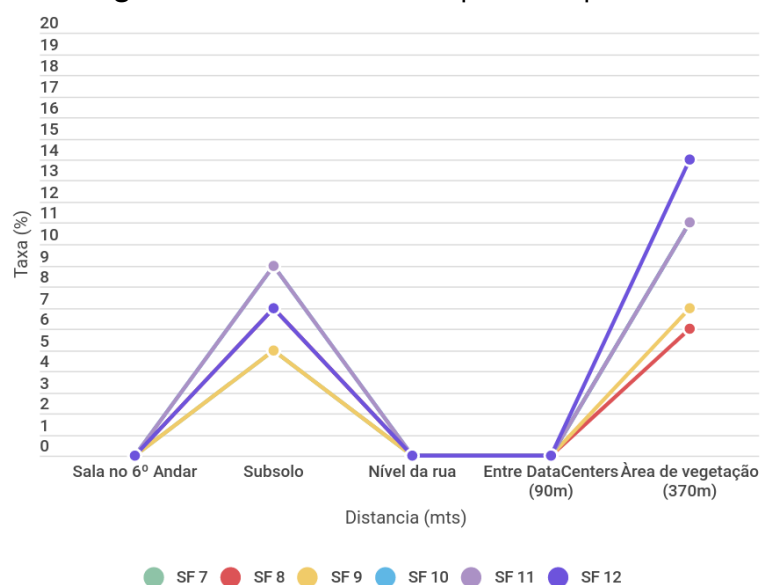
Logo a perda de pacotes pôde ser avaliada concomitantemente com a execução do teste de RSSI e do SNR, desde que o código em linguagem C desenvolvido, contemple essa possibilidade e no caso do teste deste projeto, foi possível que isso fosse implementado.

No código criado, o nó agregador que também é o emissor, envia uma mensagem de texto. O nó sensor receptor recebe a mensagem e armazena o RSSI e SNR, dentre outras informações, mas para a perda de pacotes, a função utilizada busca apenas verificar se o número de ID (Identificação) do pacote recebido é igual ao somatório de

pacotes recebidos, armazenado em outra variável.

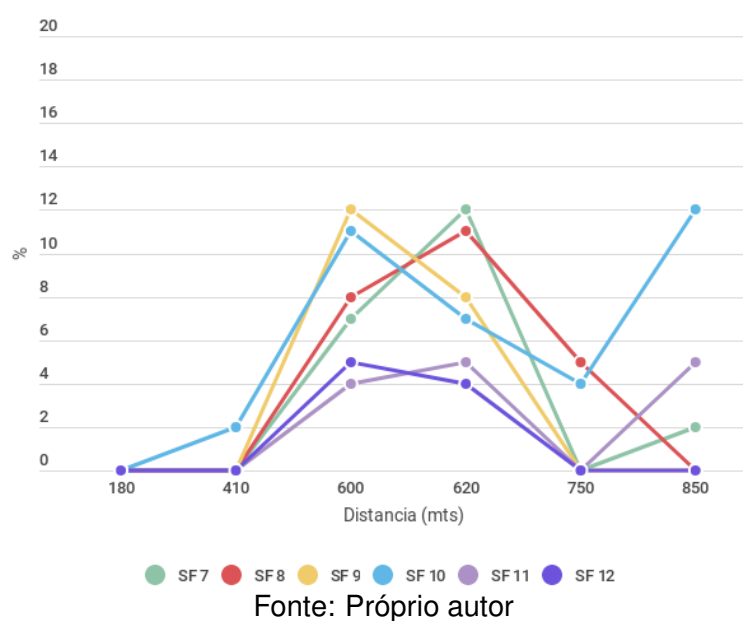
Caso haja diferença, sendo o ID um valor maior que o somatório de pacotes, é porque um pacote foi perdido e por isso não foi incluído neste mesmo somatório de pacotes, permitindo assim, implementar uma verificação da taxa de sucesso na entrega dos pacotes. A Figura 45 apresenta os resultados obtidos no teste de perda de pacote em ambiente interno.

Figura 45 – Teste *Indoor* de perda de pacotes



A Figura 46 informa os resultados dos testes de perda de pacote *indoor*.

Figura 46 – Teste de perda de pacotes *Outdoor*



As taxas de perda de pacote, em ambos os ambientes, se mantiveram pequenas apesar dos locais escolhidos para medição serem altamente suscetíveis ao ruído. Isso demonstra a robustez do protocolo, lembrando que em seu *datasheet*, é informado o alto nível de sensibilidade ao sinal e sua capacidade de trabalhar com relações SNR negativas, conforme Figura 47, o que colabora para que a mensagem enviada se mantenha inteligível ao chegar ao seu destino.

Figura 47 – Relação de tolerância SNR

<i>SpreadingFactor</i> (RegModulationCfg)	Spreading Factor (Chips / symbol)	LoRa Demodulator SNR
6	64	-5 dB
7	128	-7.5 dB
8	256	-10 dB
9	512	-12.5 dB
10	1024	-15 dB
11	2048	-17.5 dB
12	4096	-20 dB

Fonte: Adaptado de Semtech (2015)

4.2.4 Teste de consumo da bateria

O teste de consumo da bateria se confunde com o consumo do dispositivo, uma vez que somente ele é alimentado pela pilha Li-On 18650 de 2000mAh, na saída do módulo de carga TP 4056: o consumo em A (Amperes) na maior parte do tempo avaliado, foi de 100mA com picos de 150mA em curtos momentos, principalmente, durante a troca de mensagens entre emissor e receptor para mudança de SF.

4.2.5 Teste de duração da bateria

O teste de bateria foi realizado simultaneamente com o teste de transmissão *indoor*, uma vez que a bateria foi usada como única fonte de alimentação do dispositivo, sem o auxílio do painel solar.

Durante este teste o nó transmitia e recebia dados a cada 05 (cinco) segundos durante 24 horas. Neste regime de trabalho, o resultado obtido foi que a bateria inicialmente com 4,12 Volts teve duração de 14 horas, até o desligamento total do conjunto. Ao realizar a medição desta bateria com um multímetro, verificou-se que restou uma carga de 2,51 Volts.

O desligamento automático é o controle exercido pelo módulo TP 4056, que corta a alimentação ofertada, quando abaixo de 2,7 V para evitar danos na bateria de Li-On.

4.2.6 Teste de tensão e corrente fornecidos pelo painel solar

A verificação do funcionamento painel solar foi executada com o uso de dois multímetros, um conectado entre o painel solar e o circuito de controle de carga, o TP 4056 e outro verificando as tensões entre os terminais.

Convém ressaltar que a tensão oscila à medida que a incidência da luz solar varia, devido ao movimento do sol e que a corrente fornecida começa com um valor maior e começa a cair conforme a pilha carrega, devido à ação do controlador de carga, visando não superaquecer a bateria Li-On.

As variações apuradas foram entre 5,58 Volts, por volta das 9 horas da manhã, passando por 7,28 Volts ao meio dia e chegando 6,28 volts por volta das 16hs. A corrente fornecida durante os períodos de coleta, variou entre 126 e 140mA.

4.2.7 Teste de transmissão contínua e concorrência

Este teste verificou a troca contínua de mensagens entre agregador e seus nós finais, com controle de concorrência, armazenamento e confirmações de recebimento. Este teste reproduz a meta do projeto, pois consegue mensurar a troca de mensagens no caminho percorrido do nó sensor até o sistema de monitoramento escolhido. A Figura 48 mostra a etapa de armazenamento dos dados.

Figura 48 – Cartão SD do gateway com as informações dos nós sensores

MLLT-2.txt [Somente leitura] (LORA /media/valdinei/LORA)

Arquivo Editar Ver Pesquisar Ferramentas Documentos Ajuda

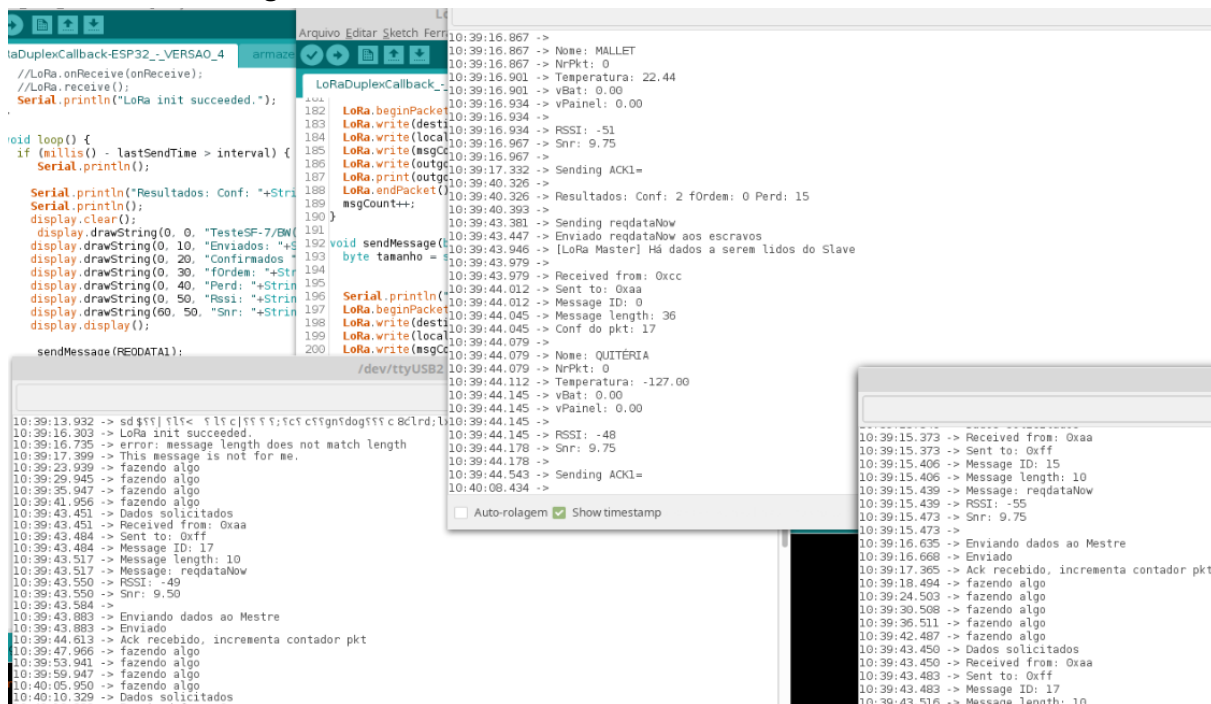
Fonte: Próprio autor

Em testes iniciais, o ESP 32 entrava em loop de reinicialização ao configurar qualquer SF diferente do padrão (SF 7). Desse modo, o microcontrolador foi substituído por um ESP 8266, que se comportou de maneira mais estável e implementava as mudanças de SF em tempo de execução. Posteriormente, o trecho de código que causava a falha no ESP 32 foi corrigida e o mesmo voltou a funcionar a contento.

Uma vez superada esta etapa, os testes foram realizados com o nó agregador e o nó integrador funcionando de maneira coordenada. Assim que chegavam pacotes ao segundo *gateway*, ele rapidamente os remetia ao *Zabbix*, que confirmava o recebimento. Neste teste foi escolhido o SF 7 para ser utilizado nos nós a uma distância de até 1 quilômetro.

O teste foi realizado com visada direta e nenhum pacote (contendo algo em torno de 60 bytes de informação), foi perdido. A Figura 49, apresenta a transmissão contínua entre o nó agregador e dois nós sensores, com o primeiro gerenciando a concorrência de acesso dos outros dois.

Figura 49 – Teste de transmissão contínua e concorrência



Fonte: Próprio autor

4.2.8 Teste de Monitoramento das informações coletadas

No teste de monitoramento buscou-se verificar se o servidor de aplicação estava recebendo as informações do segundo *gateway*, via rede IP (Wi-Fi e cabeada), oriundos da rede LoRa.

Com a implantação do *Zabbix Sender* no segundo *gateway*, a tarefa de enviar dados ao *Zabbix Server* foi facilitada, bastando criar os *hosts* no Servidor e quando chegam dados da rede LoRa, o integrador já executa o código do *Zabbix Sender*, embarcado, que envia as informações ao local de destino pré configurado. A figura 50 apresenta as coletas realizadas enviadas pelo *Sender*:

Figura 50 – Tela do nó integrador remetendo dados ao *Zabbix*

```

/dev/ttyUSB0
16:24:55.125 -> Waiting for Wi-Fi connection
16:24:55.125 -> Connected!
16:24:55.125 -> request = ZBXD * {"request":"sender data","data":[{"host":"IOTBOARD_01","key":"tempSl","value":"23.80"},{"|
16:24:55.159 -> result = ZBXD Z {"response":"success","info":"processed: 5; failed: 0; total: 5; seconds spent: 0.000200"}
16:25:00.177 -> ZABBIX SEND: OK
16:25:10.175 -> Waiting for Wi-Fi connection
16:25:10.175 -> Connected!
16:25:10.175 -> request = ZBXD * {"request":"sender data","data":[{"host":"IOTBOARD_00","key":"tempSl","value":"23.80"},{"|
16:25:10.274 -> result = ZBXD Z {"response":"success","info":"processed: 5; failed: 0; total: 5; seconds spent: 0.000102"}
16:25:15.296 -> ZABBIX SEND: OK
16:25:25.276 -> Waiting for Wi-Fi connection
16:25:25.276 -> Connected!
16:25:25.276 -> request = ZBXD * {"request":"sender data","data":[{"host":"IOTBOARD_01","key":"tempSl","value":"23.80"},{"|
16:25:25.365 -> result = ZBXD Z {"response":"success","info":"processed: 5; failed: 0; total: 5; seconds spent: 0.000194"}
16:25:30.360 -> ZABBIX SEND: OK
16:25:40.329 -> Waiting for Wi-Fi connection
16:25:40.329 -> Connected!
16:25:40.329 -> request = ZBXD * {"request":"sender data","data":[{"host":"IOTBOARD_00","key":"tempSl","value":"23.80"},{"|
16:25:40.495 -> result = ZBXD Z {"response":"success","info":"processed: 5; failed: 0; total: 5; seconds spent: 0.000195"}
16:25:45.484 -> ZABBIX SEND: OK
16:25:55.471 -> Waiting for Wi-Fi connection
16:25:55.471 -> Connected!
16:25:55.504 -> request = ZBXD * {"request":"sender data","data":[{"host":"IOTBOARD_01","key":"tempSl","value":"23.80"},{"|
16:25:55.537 -> result = ZBXD Z {"response":"success","info":"processed: 5; failed: 0; total: 5; seconds spent: 0.000214"}
16:26:00.557 -> ZABBIX SEND: OK
16:26:10.531 -> Waiting for Wi-Fi connection
16:26:10.531 -> Connected!
16:26:10.531 -> request = ZBXD * {"request":"sender data","data":[{"host":"IOTBOARD_00","key":"tempSl","value":"23.80"},{"|
16:26:10.564 -> result = ZBXD Z {"response":"success","info":"processed: 5; failed: 0; total: 5; seconds spent: 0.000203"}
16:26:15.584 -> ZABBIX SEND: OK
16:26:25.562 -> Waiting for Wi-Fi connection
16:26:25.562 -> Connected!
16:26:25.595 -> request = ZBXD * {"request":"sender data","data":[{"host":"IOTBOARD_01","key":"tempSl","value":"23.80"},{"|
16:26:25.628 -> result = ZBXD Z {"response":"success","info":"processed: 5; failed: 0; total: 5; seconds spent: 0.000227"}
16:26:30.643 -> ZABBIX SEND: OK

```

Fonte: Próprio autor

Nesse momento, as informações coletadas pelo sensor remoto, estão armazenadas e sendo apresentadas no sistema de monitoramento. A Figura 51 ilustra essa situação.

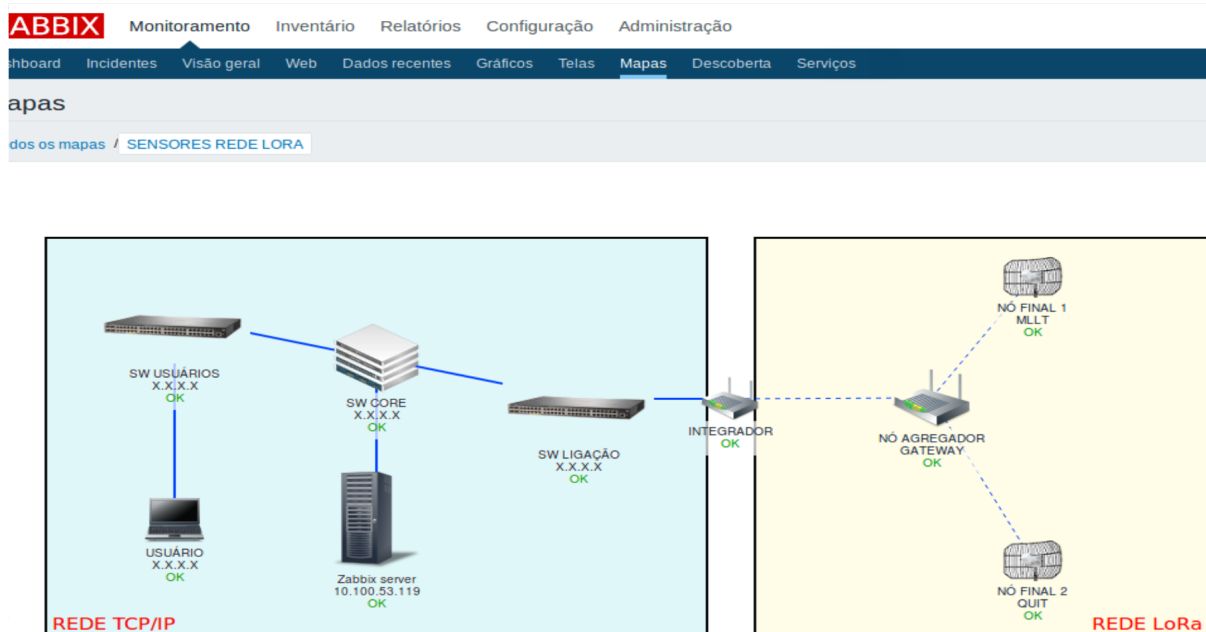
Figura 51 – Monitoramento das informações

Grupos de hosts	informe aqui o argumento para pesquisa	Selecionar	Nome	
Hosts	IOTBOARD_00 x IOTBOARD_01 x informe aqui o argumento para pesquisa	Selecionar	Exibir itens sem dados	☒
Aplicação		Selecionar	Mostrar detalhes	☐
Aplicar Limpar				
Nome ▲			Última checagem	Último valor
- other - (5 Itens)				
Nr de Acessos			13-09-2020 16:26:40	2
Temperatura da Sala			13-09-2020 16:26:40	23.8 °C
Temperatura do AC			13-09-2020 16:26:40	19 °C
Tensão da Bateria			13-09-2020 16:26:40	3.99 V
Tensão do Painel Solar			13-09-2020 16:26:40	6.7 V
- other - (5 Itens)				
Nr de Acessos			13-09-2020 16:26:55	2
Temperatura da Sala			13-09-2020 16:26:55	23.8 °C
Temperatura do AC			13-09-2020 16:26:55	19 °C
Tensão da Bateria			13-09-2020 16:26:55	3.99 V
Tensão do Painel Solar			13-09-2020 16:26:55	6.7 V

Fonte: Próprio autor

4.2.9 Apresentação das informações no Sistema

A Figura 52 apresenta os dados coletados, em uma visão de mapa no *Zabbix*. Esta é situação almejada, a qual proporciona uma visão integrada da rede LoRa remota, para a equipe responsável pelo sistema monitoramento.

Figura 52 – Mapa da Rede LoRa integrada com a Rede TCP/IP monitorada em tempo real

Fonte: Próprio autor

Uma vez realizados os testes e havendo a apresentação dos resultados, verificou-se que foi possível coletar os dados de determinado ambiente mediante o uso de sensores controlados por microcontroladores e transportar estes dados até o sistema de monitoramento, através de uma rede LoRa.

No decorrer dos testes foram observadas algumas dificuldades que demandaram pequenas adaptações na execução da metodologia. Porém tais adaptações não chegaram a causar impactos no projeto original.

A integração entre a rede LoRa e a rede TCP/IP, também foi realizada com sucesso e de um modo razoavelmente mais simples, com a criação de um elemento extra, o nó integrador e o uso do *Zabbix Sender*.

Foi possível observar o funcionamento real do conceito de Internet das Coisas, baseado na literatura consultada e com implementações específicas, devido a natureza do projeto. Tudo isso, de um modo mais simples, com baixo consumo de energia e bem efetiva.

CONCLUSÕES E TRABALHOS FUTUROS

A ideia de estabelecer uma Rede de Sensores sem fio para monitorar dados de locais remotos, tem se tornado uma necessidade comum nos dias atuais, dado que a cada dia, as pessoas buscam estar mais conectadas. Com isso, as organizações tem uma necessidade de estabelecer conexões de longa distância e daí vem a necessidade de monitorar a estrutura de dados principal com esta rede distinta.

Dentre os requisitos, deseja-se uma rede estável, de baixa manutenção e de baixo custo. Atendendo a isto, foi proposto que esta lacuna poderia ser preenchida por um sistema de monitoramento *open-source*, com sensores confeccionados em dispositivos baratos, tendo como meio de transmissão uma rede LoRa.

O LoRa possui várias vantagens, sendo seu diferencial, o longo alcance, baixo consumo e uma boa documentação. Somado a isso, a possibilidade de ser alimentado por um sistema composto por baterias Li-On e painéis solares de baixa tensão.

A interconexão do sistema de monitoramento localizado na rede IP com os sensores que estão na rede LoRa, foi simplificada devido a flexibilidade dos módulos ESP 32 e 8266. Eles permitiram que fosse embarcado um código com o *Zabbix-Sender*, que vai enviar as mensagens diretamente ao *Zabbix Server*, evitando assim, o uso de outros protocolos tais como o MQTT¹ (*MQ Telemetry Transport*) ou o protocolo LoRaWAN, cujos requisitos eram incompatíveis com o estabelecido pela Organização onde foi implementado o projeto.

A utilização do protocolo LoRa puro, armazenando os dados dentro da própria Organização de maneira mais simples foi um elemento facilitador para esta última fase do projeto. Além de necessitar usar somente um dos módulos ESP de maneira híbrida, com o aproveitamento do módulo Wi-Fi embarcado no mesmo.

E com isso, por meio dos resultados coletados, foi possível observar que o funcionamento é viável, ocorrendo de maneira coordenada, superando as limitações

¹Protocolo de mensagens para sensores e pequenos dispositivos móveis

de ruído e bloqueios físicos, tão comuns nas grandes cidades.

Ainda há a possibilidade de que, caso haja locais com distâncias maiores do que 01 (um) quilômetro, ser possível o uso de repetidores compactos e com pouco consumo de energia, para que o sinal alcance distâncias ainda maiores.

O grande diferencial trazido por este projeto na implantação do monitoramento da RSSF por meio do LoRa, foi permitir que a equipe responsável pelo monitoramento da rede passe a controlar e coletar os dados de dispositivos legados e de locais externos à Organização. Os ativos que antes não eram acompanhados e tinham seus *status* praticamente desconhecidos da equipe, agora podem ser monitorados e analisados posteriormente.

5.1 Trabalhos Futuros

Como possíveis trabalhos futuros utilizando este projeto como base, pode-se citar:

- Implementação de atualizações dos módulos sensores, via rede, sem a necessidade de deslocamento de um técnico até o local onde está o sensor;
- Melhoria do código C utilizado nos dispositivos para torná-los tolerantes a falhas com um maior uso do *watch dog timer* do ESP;
- Criação de um *web server* no nó integrador para melhor acesso dos dados em tempo real da rede LoRa, haja vista que os dados no *Zabbix* não refletem a situação instantânea dos sensores;
- Estudar futuramente o uso do LoRaWAN, que possui recursos de controle de fluxo, controle de janela e confirmação, mas de maneira que os dados fiquem armazenados localmente, em alternativa ao formato atual;
- Implementar técnicas de criptografia, que é um fator muito importante, já que o LoRa permite que outros dispositivos não pertencentes àquela rede, possam capturar o que está sendo transmitido;
- Melhorias na implementação da comunicação M2M entre os sensores com a tarefa de enviar sinais aos dispositivos legados, por meio da aquisição de insumos com maior qualidade e raio de utilização.

Referências Bibliográficas

AKYILDIZ, I. F.; SU*, W.; SANKARASUBRAMANIAM, Y.; CAYIRCI, E. Wireless sensor networks: a survey. **Computer Networks**, v. 38, p. 393–422, 2002. Disponível em: <<https://bwn.ece.gatech.edu/surveys/sensornets.pdf>>. Acesso em: 22 jun. 2020.

ANATEL, BRASIL. Agência Nacional de Telecomunicações. **Ato nº 1448**. 2017. Disponível em: <<https://www.anatel.gov.br/legislacao/index.php/component/content/article?id=1139>>. Acesso em: 20 set. 2020.

ANATEL, BRASIL. Agência Nacional de Telecomunicações. **Altera o Regulamento sobre Equipamentos de Radiocomunicação de Radiação Restrita**. 2018. Disponível em: <<https://www.anatel.gov.br/legislacao/resolucoes/2018/1220-resolucao-705>>. Acesso em: 20 set. 2020.

ARDUINING.COM. **Esp 32 Pinout Reference**. 2018. Disponível em: <<https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>>. Acesso em: 24 jun. 2020.

ARDUÍNO.CC. **Datasheet ATmega328p**. 2015. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf>. Acesso em: 08 jun. 2020.

ASHTON, Kevin. That ‘internet of things’ thing. **RFID Journal**, 2009. Disponível em: <<http://www.rfidjournal.com/article/view/4986>>. Acesso em: 13 abr. 2020.

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. The internet of things: a survey. **Computers Networks**, 2010. Disponível em: <<https://www.cs.mun.ca/courses/cs6910/IoT-Survey-Atzori-2010.pdf>>. Acesso em: 03 jun. 2020.

AUGUSTIN, Aloÿs; YI, Jiazi; CLAUSEN, Thomas Heide; TOWNSLEY, William. A study of lora: Long range low power networks for the internet of things. **Sensors**, v. 16, p. 1466, 10 2016. Disponível em: <https://pdfs.semanticscholar.org/7e78/f4d96a9c27d0ae6f3685999c3c4470cab1f1.pdf?_ga=2.203326808.878468302.1595125388-4194>. Acesso em: 18 jul. 2020.

BELL, Charles. **Beginning Sensor Networks with Arduino and Raspberry Pi**. 1st. ed. [S.l.]: Apress., 2014.

BLACK, Tomas Lovis.

Comparação de Ferramentas de Gerenciamento de Redes — UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, INSTITUTO DE INFORMÁTICA, 2008. Disponível em: <<https://lume.ufrgs.br/bitstream/handle/10183/15986/000695315.pdf>>. Acesso em: 21 jun. 2020.

CENTENARO, M.; VANGELISTA, L.; ZANELLA, A.; ZORZI, M. Long-range communications in unlicensed bands: the rising stars in the iot and smart city scenarios. **IEEE Wireless Communications**, v. 23, 10 2015. Disponível em: <https://www.researchgate.net/profile/Andrea_Zanella/publication/282603371_Long-Range_Communications_in_Unlicensed_Bands_the_Rising_Stars_in_the_IoT>. Acesso em: 28 jun. 2020.

CERF, V.; KAHN, R. A protocol for packet network interconnection. **IEEE Transactions on Communications**, v-22, n.5, p. 627–641, 1974. Disponível em: <<https://www.cs.princeton.edu/courses/archive/fall06/cos561/papers/cerf74.pdf>>. Acesso em: 23 mai. 2020.

COMER, Douglas E. **Interligação de Redes com TCP/IP**. 6th. ed. [S.l.]: Elsevier Editora, 2015. ISBN 9788535278637.

COMER, Douglas E. **Redes de Computadores e a Internet**. 6th. ed. [S.l.]: Bookman, 2016. ISBN 9788582603734.

DORJI, Applied Technologies. **868MHz LoRa SX1272 Module DRF1272F/1276, V1.01**. 2015. Disponível em: <<http://www.dorji.com/docs/data/DRF1272F.pdf>>. Acesso em: 20 set. 2020.

EDWARDS, Paul N. **The Closed World**. [S.l.]: MIT Press, 1996. ISBN 9780262550284.

EGLI, Peter R. **LPWAN - Low Power Wide Area Networks**. 2017. Disponível em: <<http://peteregli.ch/content/iot/iot.html>>. Acesso em: 19 jul. 2020.

ESPRESSIF-SYSTEMS. **Datasheet Esp 32**. 2020. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp-wroom-s2_datasheet__en.pdf>. Acesso em: 17 jun. 2020.

ESPRESSIF-SYSTEMS. **Datasheet Esp 8266**. 2020. Disponível em: <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. Acesso em: 17 jun. 2020.

EVANS, Dave. **The Internet of Things - How the Next Evolution of the Internet Is Changing Everything**. Abril 2011. Disponível em: <https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf>. Acesso em: 12 jun. 2020.

FOROUZAN, Behrouz A. **Data Communications and Networking**. 5th. ed. [S.l.]: McGraw-Hill Companies, Inc., 2013. ISBN 978-0-07-337622-6.

GABRIEL, Martha. **Marketing na Era digital**. 1st. ed. [S.l.]: Novatec Editora., 2010. ISBN 9788575222577.

GIMENEZ, Salvador P. **Microcontroladores 8051**. [S.l.]: Pearson Education do Brasil, 2005. ISBN 8587918281.

IBGE, INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. Pesquisa nacional por amostra de domicílios contínua: acesso à internet e posse de telefone móvel celular para uso pessoal 2018. **IBGE**, 2020. Disponível em: <https://biblioteca.ibge.gov.br/visualizacao/livros/liv101705_informativo.pdf>. Acesso em: 23 abr. 2020.

IEEE. Ieee recommended practice for emergency and standby power systems for industrial and commercial applications. **IEEE Std 446-1995 [The Orange Book]**, p. 133, 1996. Disponível em: <<https://pdfs.semanticscholar.org/c72f/efd3fc4270c365dc945bc35a294ce7e04c52.pdf>>. Acesso em: 18 jul. 2020.

ITU, International Telecommunication Union. Recommendation itu-t y.2060, overview of the internet of things. **ITU**, 2012. Disponível em: <https://www.itu.int/rec/dologin_pub.asplang=e&id=REC-Y.2060-201206-PDF-E&type=items>. Acesso em: 18 jun. 2020.

KARL, Holger; WILLIG, Andreas. **Protocols and Architectures for Wireless Sensor Networks**. [S.l.]: John Wiley Sons, 2005. ISBN 9780470095102.

KUROSE, James F.; ROSS, Kevin. W. **Computer Networking: A Top-Down Approach**. 7th. ed. [S.l.]: Pearson Education., 2017. ISBN 978-0-13-359414-0.

LIANDO, JANSEN C.; GAMAGE, AMALINDA; TENGOURTIUS, AGUSTINUS W.; LI, MO. Known and unknown facts of lora: Experiences from a large scale measurement study. **Association for Computing Machinery**, v. 0, 2018. Disponível em: <<https://www.ntu.edu.sg/home/limo/papers/TOSN-LoRa.pdf>>. Acesso em: 27 jun. 2020.

LIMA, Janssen dos Reis. **Monitoramento de Redes com Zabbix**. 1st. ed. [S.l.]: Brasport., 2014. ISBN 9788574526515.

LORA-ALLIANCE. **LoRaWAN Specification V1.0.3**. 2020. Disponível em: <<https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf>>. Acesso em: 20 jul. 2020.

MAGRANI, Eduardo. **A internet das coisas**. 1st. ed. [S.l.]: FGV Editora., 2018. ISBN 9788522520053.

MISCHIANI, Renzo. **Come usare la scheda SD con esp 8266 e Arduino**. 2019. Disponível em: <<https://www.mischianti.org/it/2019/12/15/come-usare-la-scheda-sd-con-esp8266-e-arduino/>>. Acesso em: 20 ago. 2020.

MISTRY, Sandeep. **arduino-LoRa API**. 2016. Disponível em: <<https://github.com/sandeepmistry/arduino-LoRa/blob/master/API.md>>. Acesso em: 02 jun. 2020.

NETTO, Guilherme T. Redes de sensores sem fio: Revisão. 2016. Disponível em: <<http://netto.ufpel.edu.br/phd/lib/exe/fetch.php?media=survey.pdf>>. Acesso em: 24 jun. 2020.

NICOLOSI, Denys E. C. **Microcontrolador 8051 detalhado**. 9th. ed. [S.l.]: Érica, 2009. ISBN 9788571947214.

PLETT, Gregory L. **Battery Management Systems Volume I: Battery Modeling**. [S.l.]: Artech House., 2015. ISBN 9781630810238.

QUERALTA, J. P.; GIA, T. N.; ZOU, Z.; TENHUNEN, H; WESTERLUND, T. Comparative study of lpwan technologies on unlicensed bands for m2m communication in the iot: beyond lora and lorawan. **Procedia Computer Science**, Elsevier, v. 155, p. 343–350, 2019. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S1877050919309639>>. Acesso em: 28 jun. 2020.

RANDOMNERDTUTORIALS.COM. **Esp 8266 Pinout Reference**. 2019. Disponível em: <<https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/>>. Acesso em: 24 jun. 2020.

RAZA, U.; KULKARNI, P.; SOORIYABANDARA, M. Low power wide area networks: An overview. **IEEE Communications Surveys Tutorials**, v. 19, n. 2, p. 855–873, 2017. Disponível em: <<https://arxiv.org/pdf/1606.07360.pdf>>. Acesso em: 28 jun. 2020.

SALES, Halexandro de Freitas. **Máquina virtual com GLPi, Zabbix e Grafana**. 2018. Disponível em: <<https://verdanadesk.com/maquina-virtual-com-glpi-zabbix-e-grafana/>>. Acesso em: 14 ago. 2020.

SEMTECH. **LoRa SX1276/77/78/79 Datasheet, Rev. 4**. 2015. Disponível em: <http://www.semtech.com/images/datasheet/sx1276_77_78_79.pdf>. Acesso em: 20 jul. 2020.

STETZEL, Kirk D.; ALDRICH, Lukas L.; TRIMBOLI, M. Scott; PLETT, Gregory L. Electrochemical state and internal variables estimation using a reduced-order physics-based model of a lithium-ion cell and an extended kalman filter. **Journal of Power Sources**, v. 278, 3 2015. Disponível em: <<https://www.sciencedirect.com/science/article/abs/pii/S0378775314020023>>. Acesso em: 18 jul. 2020.

SZABOLCS, Kajdocsi. **ESP8266ZabbixSender**. 2016. Disponível em: <<https://github.com/zaphodus/ESP8266ZabbixSender>>. Acesso em: 21 set. 2020.

TANENBAUM, Andrew S.; WETHERALL, David. **Redes de Computadores**. 5th. ed. [S.l.]: Pearson Education do Brasil., 2011. ISBN 788576059240.

TEIXEIRA, Fernando; PEREIRA, Fernando; VIEIRA, Gustavo; FONSECA, Pablo; WONG, Hao; NOGUEIRA, José Marcos; OLIVEIRA, Leonardo. Siot

– defendendo a internet das coisas contra exploits. 05 2014. Disponível em: <<http://sbrc2014.ufsc.br/anais/files/trilha/ST14-1.pdf>>. Acesso em: 14 jun. 2020.

TESLA, NICOLA. When woman is boss. in: An interview with nikola tesla by john b. kennedy. **Colliers Magazine**, 1926. Disponível em: <<http://www.tfcbooks.com/tesla/1926-01-30.htm>>. Acesso em: 13 jun. 2020.

THEOHARIDOU, M.; MYLONAS, A.; GRITZALIS, D. A risk assessment method for smartphones. **Information Security and Privacy Research.**, Springer-Verlag Berlin Heidelberg, v. 376, p. 443–456, 2012. Disponível em: <<http://www2.aueb.gr/users/amylonas/docs/SEC-12SmartphoneRiATechRe.pdf>>. Acesso em: 18 jun. 2020.

TOLL, Micah. **DIY Lithium Batteries How to Build Your Own Battery Packs**. [S.l.]: Toll Publishing., 2017.

UYTTERHOEVEN, Patrik; OLUPS, Rihards. **Zabbix 4 Network Monitoring**. 3rd. ed. [S.l.]: Packt Publishing., 2019. ISBN 9781789340266.

VANGELISTA, Lorenzo; ZANELLA, Andrea; ZORZI, Michele. Long-range IoT technologies: The dawn of LoRa™. Springer International Publishing, p. 51–58, 2015. Disponível em: <https://doi.org/10.1007%2F978-3-319-27072-2_7>. Acesso em: 18 jul. 2020.

WEIS, Stephen. Rfid (radio-frequency identification). In: BIDGOLI, Hossein (Ed.). **Handbook of Computer Networks**. John Wiley & Sons, 2012. v. 3, p. 974–984. Disponível em: <<https://onlinelibrary.wiley.com/doi/book/10.1002/9781118256107>>. Acesso em: 29 jun. 2020.

WEISER, Mark. The computer for the 21st century. **ACM SIGMOBILE Mobile Computing and Communications**, 1991. Disponível em: <<https://www.ics.uci.edu/~corps/phaseii/Weiser-Computer21stCentury-SciAm.pdf>>. Acesso em: 18 jun. 2020.

ZABBIX, ZABBIX SIA. **Zabbix Documentation 5.0**. 2020. Disponível em: <<https://www.zabbix.com/documentation/current/pt/manual>>. Acesso em: 20 jun. 2020.

APÊNDICE A – Código-Fonte do Nó agregador - ESP 32

```

1  /*
2    LoRa Duplex communication wth callback
3    created 28 April 2017  by Tom Igoe
4
5    Modificado em 10 Outubro 2020 by Valdinei Oliveira
6  */
7
8  #include <SPI.h>                // include libraries
9  #include <LoRa.h>
10 #include <Wire.h>
11 #include <SSD1306.h>
12 #include <LiquidCrystal_I2C.h>
13 #include <SD.h>
14 #include <RtcDS3231.h>          //Library Manager
15 #include <esp_task_wdt.h> //Biblioteca do watchdog
16
17 #ifdef ARDUINO_SAMD_MKRWAN1300
18 #error "This example is not compatible for Arduino MKR WAN 1300!"
19 #endif
20
21 //--->ESP32
22 #define LORA_SCK 5 // GPIO5 SCK
23 #define LORA_MISO 19 // GPIO19 MISO
24 #define LORA_MOSI 27 // GPIO27 MOSI
25 #define LORA_SS 18 // GPIO18 CS
26 #define LORA_RST 23 // GPIO14 RESET
27 #define LORA_DI00 26 // GPIO26 IRQ(Interrupt Request)
28

```

```
29 #define SDA 21    // GPIO 0
30 #define SCL 22    // GPIO 5
31
32 //CARTAO SD
33 #define SD_CS 13
34 #define SD_SCK 14 // GPIO5 SCK 17
35 #define SD_MISO 04 // GPIO19 MISO INV 13
36 #define SD_MOSI 15 // GPIO27 MOSI INV 12
37 #define BAND 915E6 //Freq. radio: 433E6, 868E6, 915E6
38
39 //Constante para informar ao Slave que queremos os dados
40 const String REQDATA1 = "reqdataNow";
41 //Constante que o Slave retorna junto com os dados para o Master
42 const String ACKDATA1 = "ACK1=";
43
44 float LAT = -3.432177;
45 float LON = -38.203456;
46 float VBat = 0;           // armazena a tensao Vusb
47 float VSol = 0;           // armazena a tensao Vusb
48 long rcbUltimaVez = 0;
49
50 SSD1306 display(0x3c, 21, 22);
51 // Inicializa o display no endereco 0x27
52 LiquidCrystal_I2C lcd(0x27, 16, 2);
53 File myFile;              //arquivo do SD
54 SPIClass sd_spi(HSPI);
55 RtcDS3231<TwoWire> Rtc(Wire); //variavel que ctrl relógio
56 RtcDateTime horaAtual;     //obj que armazena hora
57
58 String outgoing;           // outgoing message
59 int msgCount = 1;          // count of outgoing messages
60 byte localAddress = 0xAA;  // address of this device
61 byte destination;         // destination to send to
62 long lastSendTime = 0;     // last send time
63 int interval = 60000;      // interval between sends
64
65 // read packet header bytes:
66 int recipient = 0;         // recipient address
```

```
67 byte incomingMsgId = 0;          // incoming msg ID
68 byte incomingLength = 0;        // incoming msg length
69
70 typedef struct{
71     String nome;
72     int nrPkt = 0;
73     float tempSala=0;
74     //int nrAcessos=0;              //PIR
75     String presetAC;
76     float lat = LAT;
77     float lon = LON;
78     //int alt = ALT;
79     float vBateria=0;
80     float vPlSolar=0;
81     int disco;
82     int Rssi=0;
83     float Snr=0;
84 }tDados;
85
86 tDados myD4d0s;
87 tDados d4d0sNodes;
88
89 int perdidos = 0;
90 int foraDeOrdem = 0;
91 int confirmados = 0;
92 boolean sucesso = false;
93 byte sender;
94
95 void aguardaSemDelay(long tempoMs){
96     long mylastSendTime = millis();
97     while (millis() - mylastSendTime < tempoMs){
98         //para ganhar tempo
99     }
100 }
101
102 void setupLoRa(){                  //inicia SPI
103     SPI.begin(LORA_SCK, LORA_MISO, LORA_MOSI, LORA_SS);
104     LoRa.setPins(LORA_SS, LORA_RST, LORA_DIO0); // seta os pinos
```

```
105
106 //LoRa.setTxPower(20);           // TX power em dB, padrao 17
107 //LoRa.setSpreadingFactor(10);   //spreading factor, padrao: 7
108 //LoRa.setSignalBandwidth(250E3);//Muda a larg de banda do sinal
109                                 //padrao: 125E3.
110                                 //Suportado:62.5E3, 250E3 e 500E3.
111 //LoRa.setCodingRate4(6);        //Muda o CR do radio, Padrao: 5
112                                 //Suportado: entre 5 and 8.
113
114 //Inicializa o LoRa
115 while (!LoRa.begin(BAND)){        //Se nao iniciar, mostra msg
116     display.clear();
117     display.drawString(0, 0, "Erro ao inicializar o LoRa!");
118     Serial.println("Erro ao inicializar o LoRa!!!");
119     display.display();
120     //while (1);
121 }
122 //Ativa o crc
123 //LoRa.enableCrc();
124 Serial.println("LoRa OK!!!");
125 //LoRa.onReceive(onReceive);
126 //LoRa.receive();
127 Serial.println("LoRa init succeeded.");
128 }
129
130 void setupDisplay(){
131     display.init();
132     display.flipScreenVertically();
133     display.setFont(ArialMT_Plain_10);
134     display.setTextAlignment(TEXT_ALIGN_LEFT);
135 }
136
137 void setupRelogio(){
138     Rtc.Begin();                      // Obtem a data
139     RtcDateTime data_compilacao = RtcDateTime(__DATE__, __TIME__);
140     if (!Rtc.IsDateTimeValid()) {      // Vrf se desatualizado
141         Serial.println("Relogio esta desatualizado!!!");
142         Rtc.SetDateTime(data_compilacao); // acerta o relógio
```

```
143     }
144     if (!Rtc.GetIsRunning()) {                // Ativa o RTC
145         Rtc.SetIsRunning(true);
146     }
147     RtcDateTime now = Rtc.GetDateTime();
148     if (now < data_compilacao) {              // Verifica data
149         Rtc.SetDateTime(data_compilacao);
150     }
151     Rtc.Enable32kHzPin(false);
152     Rtc.SetSquareWavePin(DS3231SquareWavePin_ModeNone);
153     Serial.println("Relogio ativado\n");
154 }
155
156 void getHora(){
157     if (!Rtc.IsDateTimeValid()) {             //Bateria ok?
158         Serial.println("Relogio esta desatualizado!!!");
159     }
160     horaAtual = Rtc.GetDateTime();            // Obtem o horario atual
161 }
162
163 String formatDate(const RtcDateTime& dt, String format) {
164     String d = dt.Day() < 10 ? "0" + String(dt.Day()) :
165     String(dt.Day()) ;
166     String m = dt.Month() < 10 ? "0" + String(dt.Month()) :
167     String(dt.Month()) ;
168     String y = String(dt.Year()) ;
169     format.replace("d",d);
170     format.replace("m",m);
171     format.replace("y",y);
172     return format;
173 }
174
175 String formatTime(const RtcDateTime& dt, String format) {
176     String h = dt.Hour() < 10 ? "0" + String(dt.Hour()) :
177     String(dt.Hour()) ;
178     String m = dt.Minute() < 10 ? "0" + String(dt.Minute()) :
179     String(dt.Minute()) ;
180     String s = dt.Second() < 10 ? "0" + String(dt.Second()) :
```

```
181   String(dt.Second()) ;
182   format.replace("h",h);
183   format.replace("m",m);
184   format.replace("s",s);
185   return format;
186 }
187
188 String myDate(const RtcDateTime& dt) {
189   String myDate = (String(dt.Day())+"/"+String(dt.Month())+"/"+
190   String(dt.Year()));
191   return myDate;
192 }
193
194 String myHora(const RtcDateTime& dt) {
195   String myDate = (String(dt.Hour()+":"+String(dt.Minute())+": "+
196   String(dt.Second())));
197   return myHora;
198 }
199
200 void setupCartaoSD(){
201   Serial.print("Inicializando o cartao SD...");
202   SPI.begin();
203   sd_spi.begin(SD_SCK, SD_MISO, SD_MOSI, SD_CS);
204   SD.begin(SD_CS, sd_spi);           // Inicia SD Card
205
206   if (!SD.begin(SD_CS,sd_spi)) {
207     //if (!SD.begin(SD_CS)) {
208       Serial.println("SD card initialization failed");
209       return;
210     }
211   else {
212     Serial.println("SD card esta pronto");
213   }
214   Serial.println("Inicializacao terminada");
215
216   //uint8_t cardType = SD.cardType();
217   uint8_t cardType = SD.cardType();
218
```

```
219     if(cardType == CARD_NONE){
220         Serial.println("Nenhum cartao no slot");
221         return;
222     }
223     Serial.print("Tipo do SD Card: ");
224     if (cardType == CARD_MMC){
225         Serial.println("MMC");
226     }
227     else if(cardType == CARD_SD){
228         Serial.println("SDSC");
229     } else if(cardType == CARD_SDHC){
230         Serial.println("SDHC");
231     } else {
232         Serial.println("Desconhecido");
233     } //recebe esp livre no cartao
234     uint64_t cardSize = SD.cardSize() / (1024 * 1024);
235     Serial.printf("SD Card Size: %lluMB\n", cardSize);
236 }
237
238 int arquivaCartaoSD(tDados Recebido){
239     myFile = SD.open("/"+Recebido.nome+"-2.txt", FILE_APPEND);
240
241     if (myFile) { // se foi aberto, escreve
242         getHora();
243         Serial.println("O arquivo aberto para escrita.");
244         //formatacao no arquivo: linha a linha
245         myFile.print(myDate(horaAtual));
246         myFile.print(" | ");
247         myFile.print(myHora(horaAtual));
248         myFile.print(" | ");
249         myFile.print(Recebido.nrPkt);
250         myFile.print(" | ");
251         myFile.print(Recebido.nome);
252         myFile.print(" | ");
253         myFile.print(String(Recebido.tempSala));
254         myFile.print(" | ");
255         myFile.print(String(Recebido.vBateria));
256         myFile.print(" | ");
```

```
257     myFile.println(String(Recebido.vPlSolar));
258     myFile.close(); //fecha o arquivo apos o uso
259     Serial.println("Terminado");
260
261     memset(&Recebido, 0, sizeof(Recebido)); //limpa struct
262     return 1;
263 }
264 else { // senao nao grava.
265     Serial.println("Falha ao abrir o arquivo "+Recebido.nome+".txt");
266     return 0;
267 }
268 }
269
270 void setup() {
271     Serial.begin(9600); // initialize serial
272     while (!Serial);
273     Wire.begin(0, 5); //inicia I2C
274     Serial.println("LoRa Master");
275     // override the default CS, reset, and IRQ pins (optional)
276     pinMode(LED_BUILTIN, OUTPUT);
277     setupDisplay();
278     lcd.begin(16, 2); // inicia lcd
279     lcd.init();
280     lcd.setBacklight(HIGH);
281     display.clear();
282     display.drawString(0, 0, "Master");
283     display.display();
284     setupRelogio();
285     setupLoRa();
286     setupCartaoSD();
287     Serial.println("LoRa init succeeded.");
288 }
289
290 void loop() {
291     if (millis() - lastSendTime > interval) {
292         Serial.println();
293
294         Serial.println("Resultados: Conf: "+String(confirmados)+
```

```
295     " Perd: "+String(perdidos));
296     Serial.println();
297     display.clear();
298     display.drawString(0, 0, "TesteSF-7/BW(125E3)/CR-5");
299     display.drawString(0, 10, "Enviados: "+String(msgCount));
300     display.drawString(0, 20, "Confirmados "+String(confirmados));
301     display.drawString(0, 30, "fOrdem: "+String(foraDeOrdem));
302     display.drawString(0, 40, "Perd: "+String(perdidos));
303     display.drawString(0, 50, "Rssi: "+String(d4d0sNodes.Rssi));
304     display.drawString(60, 50, "Snr: "+String(d4d0sNodes.Snr));
305     display.display();
306
307     sendMessage(REQDATA1);
308     //Serial.println("Sending " + REQDATA1);
309     Serial.println("Enviado "+REQDATA1 +" aos escravos" );
310     lastSendTime = millis();           // timestamp the message
311     //interval = random(2000) + 21777;   // 2-3 seconds
312     perdidos = msgCount-confirmados;
313     interval = 60000;
314 }
315 // parse for a packet, and call onReceive with the result:
316 onReceive(LoRa.parsePacket());
317
318 if(sucesso){
319     aguardaSemDelay(500);
320     int result = arquivoCartaoSD(d4d0sNodes);
321
322     if(result == 1){
323         sendAck(ACKDATA1);
324         delay(800);
325         sendToGw2(d4d0sNodes);           // remete ao nó integrador
326         result=0;
327         sucesso = false;
328         memset(&d4d0sNodes, 0, sizeof(d4d0sNodes));
329     }
330     else {
331         arquivoCartaoSD(d4d0sNodes);
332     }
```

```
333     }
334
335     if(millis() - rcbUltimaVez > 260000){
336         Serial.println("Sem msg a mais de 260s, resetando");
337         esp_task_wdt_init(1, true);
338         esp_task_wdt_add(NULL);
339         while(1);
340     }
341
342     if(millis()-rcbUltimaVez >= 29800 && millis()-rcbUltimaVez <= 30000 ){
343         delay(210);
344         myD4d0s.nome = "KAME";
345         RtcTemperature temp = Rtc.GetTemperature();
346         myD4d0s.tempSala = temp.AsFloatDegC() - 20;
347         myD4d0s.lat = LAT;
348         myD4d0s.lon = LON;
349         myD4d0s.disco = SD.cardSize();
350         coletaTensao();
351         myD4d0s.vBateria = VBat;
352         myD4d0s.vPlSolar = VSol;
353         int result2 = arquivaCartaoSD(myD4d0s);
354
355         if(result2 == 1){
356             delay(800);
357             sendToGw2(myD4d0s);
358             result2=0;
359         }
360     }
361     onReceive(LoRa.parsePacket());
362 }
363
364 void sendMessage(String outgoing) {
365     Serial.println("Sending " + outgoing);
366     destination = 0xFF;           // destination to send to
367     LoRa.beginPacket();           // start packet
368     LoRa.write(destination);      // add destination address
369     LoRa.write(localAddress);     // add sender address
370     LoRa.write(msgCount);         // add message ID
```

```
371   LoRa.write(outgoing.length());    // add payload length
372   LoRa.print(outgoing);              // add payload
373   LoRa.endPacket();                  // finish packet and send it
374   msgCount++;                       // increment message ID
375 }
376
377 void sendAck(String outgoing) {
378   Serial.println("Sending " + outgoing + "para: " + d4d0sNodes.nome);
379   if(d4d0sNodes.nome == "GOKU"){
380     destination = 0xBB;
381   }
382   else {
383     destination = 0xCC;
384   }
385
386   LoRa.beginPacket();
387   LoRa.write(sender);
388   LoRa.write(localAddress);
389   LoRa.write(msgCount);
390   LoRa.write(outgoing.length());
391   LoRa.print(outgoing);
392   LoRa.endPacket(true);
393   msgCount++;
394 }
395
396 void sendToGw2(tDados msg){
397   byte tamanho = sizeof(msg);
398   destination = 0xEE;
399   Serial.println("A Enviar dados para GW2");
400   Serial.println(" ");
401   LoRa.beginPacket();
402   LoRa.write(destination);
403   LoRa.write(localAddress);
404   LoRa.write(msgCount);
405   LoRa.write(tamanho);
406   LoRa.write(( unsigned char *)&msg , sizeof(msg));
407   LoRa.endPacket();
408 }
```

```
409     lcd.clear();
410     lcd.setCursor (0,0);
411     lcd.print("Finaliz  Envio  GW2!");
412     Serial.println("Finaliz  Envio ao GW2!");
413     Serial.println(" ");
414     display.display ();
415 }
416
417 void onReceive(int packetSize) {
418     if (packetSize == 0) return;           // sai da funcao
419     char byte_recebido;
420     char * ptInformacaoRecebida = NULL;
421     int recipient = LoRa.read();           // endereco destino
422     sender = LoRa.read();                 // remetente
423     int incomingMsgId = LoRa.read();       // ID msg recebida
424     byte incomingLength = LoRa.read();     // Tamanho msg Recebida
425     int pktOrigem = LoRa.read();
426
427     //Se a msg nao for para este endereco ou o de broadcast
428     if (recipient != localAddress && recipient != 0xFF) {
429         Serial.println("This message is not for me.");
430         return;                           // sai da funcao
431     }
432     digitalWrite(LED_BUILTIN, HIGH);      // liga o led interno
433     Serial.println("[LoRa Master] Chegaram dados do Slave");
434     ptInformacaoRecebida = (char *)&d4d0sNodes; //Recebe os dados
435     while (LoRa.available()) {
436         byte_recebido = (char)LoRa.read();
437         *ptInformacaoRecebida = byte_recebido;
438         ptInformacaoRecebida++;
439     }
440     digitalWrite(LED_BUILTIN, LOW);       // desliga o led interno
441     rcbUltimaVez = millis();
442     d4d0sNodes.Rssi = LoRa.packetRssi();
443     d4d0sNodes.Snr = LoRa.packetSnr();
444     if(pktOrigem == (msgCount-1)){
445         confirmados++;
446     }
```

```
447     else{
448         if(pktOrigem < (msgCount-1)){
449             foraDeOrdem++;
450             confirmados++;
451         }
452         else{
453             perdidos++;
454         }
455     }
456     // Se a mensagem for para este nó, imprime os detalhes - debug
457     Serial.println("xxx Nome: " + String(d4d0sNodes.nome)+" xxxxxx");
458     Serial.println("Received from: 0x" + String(sender, HEX));
459     Serial.println("Sent to: 0x" + String(recipient, HEX));
460     Serial.println("Message ID: " + String(incomingMsgId));
461     Serial.println("Message length: " + String(incomingLength));
462     Serial.println("Conf do pkt: " + String(pktOrigem));
463     Serial.println("NrPkt: " + String(d4d0sNodes.nrPkt));
464     Serial.println("Temperatura: " + String(d4d0sNodes.tempSala));
465     Serial.println("vBat: " + String(d4d0sNodes.vBateria));
466     Serial.println("vPainel: " + String(d4d0sNodes.vPlSolar));
467     Serial.println("RSSI: " + String(LoRa.packetRssi()));
468     Serial.println("Snr: " + String(LoRa.packetSnr()));
469     Serial.println(" ");
470
471     lcd.setCursor(0,0);
472     lcd.print("T:"+String(confirmados*2)+"/"+String(msgCount));
473     lcd.setCursor(12,0);
474     lcd.print(d4d0sNodes.nome);           //max 5 letras
475     lcd.setCursor(0,1);
476     lcd.print("R:"+String(d4d0sNodes.Rssi));
477     lcd.setCursor(6,1);
478     lcd.print("S:"+String(d4d0sNodes.Snr));
479     lcd.setCursor(13,1);
480     lcd.print("T:"+String(d4d0sNodes.tempSala));
481
482     sucesso = true;
483 }
```

APÊNDICE B – Código-fonte dos Nós Sensores - ESP 8266

```

1  /*
2    created 28 April 2017
3    by Tom Igoe
4  */
5  #include <SPI.h>                // include libraries
6  #include <LoRa.h>
7  #include <Wire.h>
8  #include <SSD1306.h>
9  #include <LiquidCrystal_I2C.h>
10 #include <OneWire.h>            //BIBLIOTECA NECESSARIA PARA O DS18B20
11 #include <DallasTemperature.h> //BIBLIOTECA NECESSARIA PARA O DS18B20
12
13 #include <Arduino.h>
14 #include <IRremoteESP8266.h>
15 #include <IRsend.h>
16
17 #ifdef ARDUINO_SAMD_MKRWAN1300
18 #error "This example is not compatible for Arduino MKR WAN 1300!"
19 #endif
20
21 #define SLAVE
22 #define SCK D5 // GPIO5 SCK
23 #define MISO D6 // GPIO19 MISO
24 #define MOSI D7 // GPIO27 MOSI
25 #define SS D8 // GPIO18 CS
26 #define RST 3 // GPIO16 - usarei o reset do esp
27 #define DI00 D2 // GPIO26 IRQ(Interrupt Request)
28 #define BAND 915E6 //Freq do r dio: 433E6, 868E6, 915E6

```

```
29
30 //DEFINE O PINO DO SENSOR DS18B20
31 #define sensorDS18B20 D4
32 #define PinTranADC 16
33
34 //Constante para informar ao Slave que queremos os dados
35 const String REQDATA = "reqdataNow";
36 //Constante que o Slave retorna junto com os dados para o Master
37 const String ACKDATA = "ACK1=";
38 OneWire ourWire(sensorDS18B20);
39 DallasTemperature sensors(&ourWire); //PASSA A TEMPERATURA PARA O DallasT
40 long rcbUltimaVez = 0;
41 SSD1306 display(0x3c, D3, D1);
42 LiquidCrystal_I2C lcd(0x27, 16, 2); // Inicializa o display no endereco
43 float LAT = -3.737945;
44 float LON = -38.500863;
45 float temperatura = 0;
46 String Status="";
47 String outgoing; // outgoing message
48 int msgCount = 0; // count of outgoing messages
49 byte localAddress = 0xBB; // address of this device
50 byte destination = 0xAA; // destination to send to
51 long lastSendTime = 0; // last send time
52 int interval = 10000; // interval between sends
53
54 struct tDados {
55     String nome;
56     int nrPkt = 0;
57     float tempSala=0;
58     //int nrAcessos=0; //PIR
59     String presetAC = " ";
60     float lat = LAT;
61     float lon = LON;
62     //int alt = ALT;
63     float vBateria=0;
64     float vPlSolar=0;
65     int disco=0;
66     int Rssi=0;
```

```
67   float Snr=0;
68 };
69 struct tDados myD4d0s;
70 String funcao2 = "LIGAR";
71 boolean sucesso = false;
72 int incomingMsgId;
73
74 void aguardaSemDelay(long tempoMs){
75     long mylastSendTime = millis();
76     while (millis() - mylastSendTime < tempoMs){
77         //funcao ganhar tempo
78     }
79 }
80
81 void readSensorTempLocal() {
82     sensors.requestTemperatures();
83     temperatura = sensors.getTempCByIndex(0);
84 }
85
86 void setup() {
87     Serial.begin(9600);                                // initialize serial
88     while (!Serial);
89     Serial.println("LoRa Slave 1");
90     sensors.begin();
91     Wire.begin(0, 5);
92
93     pinMode(LED_BUILTIN, OUTPUT);
94     lcd.begin(16, 2);
95     lcd.init();
96     lcd.setBacklight(HIGH);
97     setupLora();
98     readSensorTempLocal();
99     lcd.setCursor(0,0);
100    lcd.print("Slave 1");
101    lcd.setCursor(0,1);
102    lcd.print(temperatura);
103    LoRa.onReceive(onReceive);
104    LoRa.receive();
```

```
105   Serial.println("LoRa init succeeded.");
106 }
107
108 void loop() {
109   if (millis() - lastSendTime > interval) {
110     Serial.println("fazendo algo");
111     lastSendTime = millis();           // timestamp the message
112     interval = 30000;
113     LoRa.receive();                   // go back into receive mode
114   }
115   if (sucesso){
116     coletaDadosSensores();
117     sendMessage(incomingMsgId);
118     sucesso = false;
119     LoRa.receive();
120   }
121   if(millis() - rcbUltimaVez > 300000){
122     Serial.println("Sem msg a mais de 300s, resetando");
123     while(1);
124   }
125 }
126
127 void sendMessage2(String outgoing) {
128   aguardaSemDelay(800);
129   LoRa.beginPacket();                // start packet
130   LoRa.write(destination);           // add destination address
131   LoRa.write(localAddress);          // add sender address
132   LoRa.write(msgCount);              // add message ID
133   LoRa.write(outgoing.length());     // add payload length
134   LoRa.print(outgoing);              // add payload
135   LoRa.endPacket();                  // finish packet and send it
136   msgCount++;                        // increment message ID
137 }
138
139 void sendMessage(byte pktOrigem) {
140   byte tamanho = sizeof(myD4d0s);
141   delay(3577);
142   Serial.println("Enviando dados ao Mestre");
```

```
143   LoRa.beginPacket();
144   LoRa.write(destination);
145   LoRa.write(localAddress);
146   LoRa.write(msgCount);
147   LoRa.write(tamanho);
148   LoRa.write(pktOrigem);
149   LoRa.write((unsigned char *)&myD4d0s, sizeof(myD4d0s));
150   LoRa.endPacket(true);
151   Serial.println("Enviado");
152   msgCount++;
153 }
154
155 float coletaTensao(boolean bateria){
156   //some constants - algumas constantes
157   const unsigned int analogInPin = A0; // pino analogico
158   const float analogLimit = 3.3;      // Limite da entrada analogica
159   const float R1 = 15000;
160   const float R2 = 18000;
161   // used variables - variaveis utilizadas
162   unsigned int RAWanalogInput = 0; // armazena o valor RAW de A0
163   float analogInputVoltage = 0;    //armazena a tensao convertida de A0
164   if (bateria == false){
165     digitalWrite(PinTranADC, LOW);
166   }
167   else {
168     digitalWrite(PinTranADC, HIGH);
169   }
170   delay(400);
171
172   RAWanalogInput = analogRead(analogInPin); // le a entrada analogica
173   // converte o valor RAW em tensao
174   analogInputVoltage = ((float)RAWanalogInput * (analogLimit/1023));
175   analogInputVoltage = analogInputVoltage - (analogInputVoltage*0.07);
176
177   if (bateria == true){ //calcula a tensao Vusb
178     VBat = ((analogInputVoltage) * ((R1 + R2) / R2));
179     Serial.println(VBat);
180     return VBat;
```

```
181 }
182 else {                                     //calcula a tensao Vusb
183     VSol = ((analogInputVoltage) * ((R1 + R2) / R2));
184     Serial.println(VSol);
185     return VSol;
186 }
187 }
188
189 void setupLora (){
190     LoRa.setPins(SS, RST, DI00);
191     if (!LoRa.begin(915E6)) {              // initialize radio at 915 MHz
192         Serial.println("LoRa init failed. Check your connections.");
193         while (true);                      // if failed, do nothing
194     }
195 }
196
197 void onReceive(int packetSize) {
198     if (packetSize == 0) return;           // if there's no packet, return
199     // read packet header bytes:
200     int recipient = LoRa.read();           // recipient address
201     byte sender = LoRa.read();             // sender address
202     incomingMsgId = LoRa.read();           // incoming msg ID
203     byte incomingLength = LoRa.read();     // incoming msg length
204     String incoming = "";                 // payload of packet
205     while (LoRa.available()) {             // can't use readString(), so
206         incoming += (char)LoRa.read();     // add bytes one by one
207     }
208     if (incomingLength != incoming.length()) { // check length for error
209         Serial.println("error: message length does not match length");
210         return;                            // skip rest of function
211     }
212     // if the recipient isn't this device or broadcast,
213     if (recipient != localAddress && recipient != 0xFF) {
214         Serial.println("This message is not for me.");
215         return;                            // skip rest of function
216     }
217     if (incomingLength == REQDATA.length()){
218         Serial.println("Dados solicitados");
```

```

219     Serial.println("Received from: 0x" + String(sender, HEX));
220     Serial.println("Sent to: 0x" + String(recipient, HEX));
221     Serial.println("Message ID: " + String(incomingMsgId));
222     Serial.println("Message length: " + String(incomingLength));
223     Serial.println("Message: " + incoming);
224     Serial.println("RSSI: " + String(LoRa.packetRssi()));
225     Serial.println("Snr: " + String(LoRa.packetSnr()));
226     Serial.println();
227     rcbUltimaVez = millis();
228     sucesso = true;
229 }
230 if(incomingLength == ACKDATA.length()){
231     Serial.println("Ack recebido, incrementa contador pkt");
232     msgCount++; // increment message ID
233 }
234 }
235
236 /* IRremoteESP8266: IRsendDemo - demonstrates sending IR codes with IRsend.
237  * Version 1.1 January, 2019
238  * Based on Ken Shirriff's IrsendDemo Version 0.1 July, 2009,
239  * Copyright 2009 Ken Shirriff, http://arcfn.com
240  */
241 void sendIR(String funcao){
242     const uint16_t kIrLed = 3; // ESP8266 GPIO. Recommended: 4 (D2).
243     IRsend irsend(kIrLed); // Set the GPIO to be used to sending the message
244
245     uint16_t rawData[211] = {9064, 4464, 582, 1666, 584, 1662, 582, 528,
246 //ler tamanho do array
247
248 //Mesg Desc.: Power: On, Mode: 1 (Cool), Temp: 23C, Fan: 1 (High), Swing(V):
249     uint16_t rawData2[211] = {9018, 4492, 554, 1692, 556, 1692, 558, 552,
250
251 //Mesg Desc.: Power: Off, Mode: 1 (Cool), Temp: 23C, Fan: 1 (High), Swing(V):
252     uint16_t rawData3[211] = {9048, 4464, 588, 1660, 586, 1660, 586, 524,
253
254 //Ligar TV
255     uint16_t rawData4[203] = {4566, 4494, 588, 1660, 590, 1660, 588, 1660,
256

```

```
257 //Desligar TV
258     uint16_t rawData5[135] = {4568, 4492, 590, 1662, 586, 1662, 588, 1662,
259
260 //desliga o ac
261     uint16_t rawDataDesliga[211] = {9032, 4452, 588, 1656, 586, 1658, 588,
262
263 //liga AC
264     uint16_t rawDataLiga[211] = {9030, 4456, 588, 1656, 586, 1656, 588, 5
265
266 //baixa o AC a 17
267     uint16_t rawDataBaixa[211] = {9030, 4454, 586, 1656, 586, 1656, 588,
268
269 //normaliza o AC a 21
270     uint16_t rawDataNormaliza[211] = {8998, 4488, 552, 1690, 530, 1714, 5
271
272     String msg6 = "M:1|17C|Fan:1|T:On";
273     String msg7 = "M:1|21C|Fan:2|T:On";
274
275     if(funcao == "LIGAR") {
276         Serial.println("Ligando tv");
277         irsend.sendRaw(rawDataLiga, 211, 38); // Send a data capture at 38K
278         digitalWrite(LED_BUILTIN, HIGH);
279     }
280
281     if(funcao == "DESLIGAR") {
282         Serial.println("Ligando tv");
283         irsend.sendRaw(rawDataDesliga, 211, 38);
284         digitalWrite(LED_BUILTIN, LOW);
285     }
286
287     if(funcao == "BAIXAR") {
288         Serial.println("Agindo no AC - Baixar");
289         Serial.println(msg6);
290         irsend.sendRaw(rawDataBaixa, 211, 38);
291         digitalWrite(LED_BUILTIN, LOW);
292     }
293
294     if(funcao == "NORMALIZAR") {
```

```
295     Serial.println("Agindo no AC - Normalizar");
296     Serial.println(msg7);
297     irsend.sendRaw(rawDataNormaliza, 211, 38);
298     digitalWrite(LED_BUILTIN, LOW);
299 }
300 LoRa.receive();
301
302 void verificaTempAC(){
303
304     if(temperatura >= 23) {
305         funcao2 = "BAIXAR";
306     }
307     else{
308         if(temperatura <= 20) {
309             funcao2 = "NORMALIZAR";
310         }
311         else {
312             Serial.println("Temperatura NORMAL");
313             funcao2 = "NADA";
314         }
315     }
316     if(funcao2 != "NADA") {
317         sendIR(funcao2);
318     }
319 }
320
321 void coletaDadosSensores(){
322     boolean gate;
323     readSensorTempLocal();
324     gate = true;
325     myD4d0s.vBateria = coletaTensao(gate);
326     gate = false;
327     myD4d0s.vPlSolar = coletaTensao(gate);
328     myD4d0s.nrPkt = msgCount; // nr do envio
329     myD4d0s.tempSala = temperatura; //coletado por readDataLocal()
330     myD4d0s.nome = "GOKU"; // N.A
331 }
```

APÊNDICE C – Código-Fonte do Nó Integrador - ESP 8266

```

1 #include <ESP8266ZabbixSender.h>
2 #include <SPI.h>
3 #include <LoRa.h>
4
5 //xxxxxxxxxxxxxxxxxxxxx DEFINES NECESSARIOS AO LORA xxxxxxxxxxxxxxxxxxxxxxxx//
6
7 #define LORA_SCK D5 // GPIO5 SCK
8 #define LORA_MISO D6 // GPIO19 MISO
9 #define LORA_MOSI D7 // GPIO27 MOSI
10 #define LORA_SS D8 // GPIO18 CS
11 #define LORA_RST 3 //D0 // GPIO14 RESET
12 #define LORA_DIO0 D2 // GPIO26 IRQ(Interrupt Request)
13 #define DS18B20 D4
14 #define BAND 915E6 //Frequencia do radio - exemplo : 433E6, 868E6, 915E6
15
16 ESP8266ZabbixSender zSender;
17
18 /* WiFi settings */
19 String ssid = "##XXXXXX##";
20 String pass = "YYYYYYYYYY";
21
22 /* Zabbix server setting */
23 #define SERVERADDR 192, 168, 0, 20 // Zabbix server Address
24 #define ZABBIXPORT 10051 // Zabbix erver Port
25
26 boolean checkConnection();
27 void onReceive();
28

```

[illegible]

```
67 void setupLoRa(){
68     //Inicializa a comunicacao
69     LoRa.setPins(LORA_SS, LORA_RST, LORA_DIO0);
70     while (!LoRa.begin(BAND)){
71         Serial.println("Erro ao inicializar o LoRa!!!");
72     }
73     Serial.println("LoRa OK!!!");
74 }
75 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//
76
77 //so funciona no loop
78 void enviaAoZabbix(tDados recebido){
79     zSender.Init(IPAddress(SERVERADDR), ZABBIXPORT, recebido.nome);
80     checkConnection(); // Check wifi connection
81     zSender.ClearItem(); // Clear ZabbixSender's item list
82     zSender.AddItem("tempS1", (String)recebido.tempSala);
83     zSender.AddItem("Lat", (String)recebido.lat);
84     zSender.AddItem("Long", (String)recebido.lon);
85     zSender.AddItem("vBat", (String)recebido.vBateria);
86     zSender.AddItem("vPlSolar", (String)recebido.vPlSolar);
87     zSender.AddItem("disco", (String)recebido.disco);
88     zSender.AddItem("rssi", (String)recebido.Rssi);
89     zSender.AddItem("snr", (String)recebido.Snr);
90
91     if (zSender.Send() == EXIT_SUCCESS) {
92         Serial.println("ZABBIX SEND: OK");
93     } else {
94         Serial.println("ZABBIX SEND: NG");
95     }
96     delay(800); // wait 0,8sec
97 }
98
99 void setup() {
100     Serial.begin(115200);
101     WiFi.mode(WIFI_STA);
102     WiFi.disconnect();
103     delay(100);
104     WiFi.begin(ssid.c_str(), pass.c_str());
```

```
105  while (!checkConnection()) {
106  }
107  SPI.begin();                      //inicializa SPI
108  setupLoRa();                      //Chama a configuracao inicial do LoRa
109  LoRa.onReceive(onReceive);
110  LoRa.receive();
111  Serial.println("LoRa init succeeded.");
112 }
113
114 void loop() {
115     if(sucesso){
116         enviaAoZabbix(d4d0s);
117         memset(&d4d0s, 0, sizeof(d4d0s));
118         sucesso = false;
119     }
120 }
121
122 void onReceive(int packetSize) {
123     if (packetSize == 0) return;
124     char byte_recebido;
125     char * ptInformacaoRecebida = NULL;
126     // read packet header bytes:
127     int recipient = LoRa.read();
128     byte sender = LoRa.read();
129     int incomingMsgId = LoRa.read();
130     byte incomingLength = LoRa.read();
131
132     if the recipient isnt this device or broadcast,
133     if (recipient != localAddress) {
134         Serial.println("This message is not for me.");
135         return;
136     }
137
138     Serial.print("[LoRa Master] Ha dados a serem lidos do Master1");
139     ptInformacaoRecebida = (char *)&d4d0s;
140
141     while (LoRa.available()) {
142         byte_recebido = (char)LoRa.read();
```

```
143     *ptInformacaoRecebida = byte_recebido;
144     ptInformacaoRecebida++;
145 }
146
147 if(d4d0s.nome == "KAME"){
148     d4d0s.Rssi = LoRa.packetRssi();
149     d4d0s.Snr = LoRa.packetSnr();
150 }
151
152 Serial.println("recebido!");
153 Serial.println("Received from: 0x" + String(sender, HEX));
154 Serial.println("Sent to: 0x" + String(recipient, HEX));
155 Serial.println("Message ID: " + String(incomingMsgId));
156 Serial.println("Message length: " + String(incomingLength));
157 Serial.println("RSSI: " + String(d4d0s.Rssi));
158 Serial.println("Snr: " + String(d4d0s.Snr));
159 Serial.println();
160 sucesso = true;
161 }
162
163 boolean checkConnection() {
164     int count = 0;
165     Serial.print("Waiting for Wi-Fi connection");
166     while (count < 300) {
167         if (WiFi.status() == WL_CONNECTED) {
168             Serial.println();
169             Serial.println("Connected!");
170             return (true);
171         }
172         delay(500);
173         Serial.print(".");
174         count++;
175     }
176     Serial.println("Timed out.");
177     return false;
178 }
```

APÊNDICE D – Código-Fonte do Nó 1 - Teste de Intensidade/Qualidade

```

1  #include <SPI.h>
2  #include <LoRa.h>
3  #include <Wire.h>
4  #include <SSD1306.h>
5
6  #define SCK D5                // GPIO5 SCK
7  #define MISO D6              // GPIO19 MISO
8  #define MOSI D7              // GPIO27 MOSI
9  #define SS D8                // GPIO18 CS
10 #define RST D0               // GPIO16 -
11 #define DI00 D2              // GPIO26 IRQ(Interrupt Request)
12 #define BAND 915E6           //Freq do radio (433E6, 868E6, 915E6)
13 SSD1306Wire display(0x3c, D3, D1);
14
15 String outgoing;
16 int msgCount = 1;
17 byte localAddress = 0xBB;
18 byte destination = 0xFF;
19 long lastSendTime = 0;
20 int interval = 5000;
21 int modo = 8;
22 int modoRcb = 0;
23 boolean sucesso = false;
24 int enviados = 0;
25 int maxPkt = 100;
26 String message="";
27 long ultima_vez = 0;
28

```

```
29 void aguardaSemDelay(long tempoMs){
30     long mylastSendTime = millis();
31     while (millis() - mylastSendTime < tempoMs){
32         //para ganhar tempo
33     }
34 }
35
36 void setup() {
37     Serial.begin(9600);
38     while (!Serial);
39     Wire.begin(0, 5);           //inicia I2C e seta pinos
40     display.init();
41     display.flipScreenVertically();
42     Serial.println("LoRa Duplex with callback");
43     LoRa.setPins(SS, RST, DI00);
44
45     if (!LoRa.begin(BAND)) {
46         Serial.println("LoRa init failed. Check your connections.");
47         while (true);
48     }
49     LoRa.enableCrc();
50     LoRa.onReceive(onReceive);
51     LoRa.receive();
52     Serial.println("LoRa init succeeded.");
53 }
54
55 void loop() {
56     String message="";
57     String message2="";
58     if (millis() - lastSendTime > interval) {
59
60         String message2 = "Mudar de SF";    // Envia mensagem
61         if(msgCount > maxPkt){
62             if (modo > 6) {
63                 if (modo > 12) {
64                     modo = 7;
65                 }
66                 Serial.println(ultima_vez);
```

```
67         if(millis() - ultima_vez > 3*interval*maxPkt ){
68             Serial.println("Resetando");
69             //ESP.wdtDisable();           //reset por hardware
70             while(1){};                  //chama wdt
71         }
72         Serial.println("Enviando SF "+String(modos));
73         sendMessage(message2, modos);    //normal
74         LoRa.setSpreadingFactor(modos);
75         LoRa.receive();
76     }
77 }
78 else {
79     String message = "Pacote "+String(msgCount);
80     sendMessage(message, 0);
81     LoRa.receive();
82     Serial.println("Sending " + message);
83 }
84 lastSendTime = millis();
85 //interval = 4000;           // 2-3 seconds
86 LoRa.receive();
87 }
88 }
89
90 void sendMessage(String outgoing, int valor) {
91     LoRa.beginPacket();
92     LoRa.write(destination);
93     LoRa.write(localAddress);
94     LoRa.write(msgCount);
95     LoRa.write(valor);           //informa o modo SF
96     LoRa.write(outgoing.length());
97     LoRa.print(outgoing);
98     LoRa.endPacket();
99     msgCount++;
100 }
101
102 void onReceive(int packetSize) {
103     if (packetSize == 0) return;
104     // read packet header bytes:
```

```
105  int recipient = LoRa.read();
106  byte sender = LoRa.read();
107  int incomingMsgId = LoRa.read();
108  modoRcb = LoRa.read();
109  byte incomingLength = LoRa.read();
110  String incoming = "";
111  while (LoRa.available()) {
112      incoming += (char)LoRa.read();
113  }
114  if (incomingLength != incoming.length()) {
115      Serial.println("error: message length does not match length");
116      return;
117  }
118  if (recipient != localAddress && recipient != 0xFF) {
119      Serial.println("This message is not for me.");
120      return;
121  }
122  Serial.println("Received from: 0x" + String(sender, HEX));
123  Serial.println("Sent to: 0x" + String(recipient, HEX));
124  Serial.println("Message ID: " + String(incomingMsgId));
125  Serial.println("Message length: " + String(incomingLength));
126  Serial.println("Message: " + incoming);
127  Serial.println("RSSI: " + String(LoRa.packetRssi()));
128  Serial.println("Snr: " + String(LoRa.packetSnr()));
129  Serial.println();
130
131  if(modoRcb == modo){          //receiver comunica
132      sucesso = true;
133      Serial.println("Master em SF: "+String(modo));
134      sucesso = false;
135      msgCount = 0;
136      modo++;                  //so muda o modo quando tem resposta
137      ultima_vez = millis();
138  }
139 }
```

APÊNDICE E – Código-Fonte do Nó 2 - Teste de Intensidade/Qualidade

```
1 #include <SPI.h>
2 #include <LoRa.h>
3 #include <Wire.h>
4 #include <SSD1306.h>
5
6 #define SCK D5
7 #define MISO D6
8 #define MOSI D7
9 #define SS D8
10 #define RST 3
11 #define DI00 D2
12 #define BAND 915E6
13
14 SSD1306 display(0x3c, 0, 5);
15 String outgoing;
16 int msgCount = 0;
17 byte localAddress = 0xBB;
18 byte destination = 0xFF;
19 long lastSendTime = 0;
20 int interval = 2000;
21 byte modoMestre = 1;
22 String message="";
23 int sModo = 7;
24 int incomingMsgId = 0;
25 int qtdPkt = 0;
26 float media_rssi = 0;
27 float media_snr = 0;
28 String incoming = "";
```

```
29 long ultima_vez = 0;
30
31 void aguardaSemDelay(unsigned long tempoMs){
32     long mylastSendTime = millis();
33     while (millis() - mylastSendTime < tempoMs){
34         //para ganhar tempo
35     }
36 }
37
38 void setup() {
39     Serial.begin(9600);
40     while (!Serial);
41     Wire.begin(0, 5);
42     Serial.println("LoRa Duplex with callback");
43
44     // override the default CS, reset, and IRQ pins (optional)
45     LoRa.setPins(SS, RST, DI00);
46     setupDisplay();
47     if (!LoRa.begin(BAND)) {
48         Serial.println("LoRa init failed. Check your connections.");
49         while (true);
50     }
51     LoRa.enableCrc();
52     LoRa.onReceive(onReceive);
53     LoRa.receive();
54     Serial.println("LoRa init succeeded.");
55 }
56
57 void loop() {
58     if (millis() - lastSendTime > interval) {
59         //Serial.println(String(millis())+" "+String(ultima_vez));
60         if(millis() - ultima_vez > 25000){
61             ESP.wdtDisable();
62             while(1){};
63         }
64         String message = "Fazendo algo!";
65         Serial.println(message);
66         lastSendTime = millis();
```

```
67     interval = random(2000) + 1000;
68     LoRa.receive();
69 }
70 //Apaga o display
71 display.clear();
72 display.drawString(0, 0, "Modo Atual "+String(sModo));
73 display.drawString(0, 20, "Msg: "+incoming);
74 //Desenha a barra de progresso
75 display.drawProgressBar(0, 33, 120, 8,(qtdPkt*10));
76 display.drawString(0, 43, "RSSI "+String(LoRa.packetRssi()));
77 display.drawString(56, 43, "MdRssi: "+String(media_rssi/qtdPkt));
78 display.drawString(0, 53, "SNR "+String(LoRa.packetSnr()));
79 display.drawString(56, 53, "MdSnr: "+String(media_snr/qtdPkt));
80 display.display();
81 LoRa.receive();
82 }
83
84 void sendMessage(String outgoing, int valor) {
85     LoRa.beginPacket();
86     LoRa.write(destination);
87     LoRa.write(localAddress);
88     LoRa.write(msgCount);
89     LoRa.write(modoSestre);
90     LoRa.write(outgoing.length());
91     LoRa.print(outgoing);
92     LoRa.endPacket(true);
93     msgCount++;
94 }
95
96 void onReceive(int packetSize) {
97     if (packetSize == 0) return;
98
99     int recipient = LoRa.read();
100     byte sender = LoRa.read();
101     incomingMsgId = LoRa.read();
102     modoSestre = LoRa.read();
103     byte incomingLength = LoRa.read();
104     incoming = "";
```

```
105 while (LoRa.available()) {
106     incoming += (char)LoRa.read();
107 }
108 if (incomingLength != incoming.length()) {
109     Serial.println("error: message length does not match length");
110     return;
111 }
112 if (recipient != localAddress && recipient != 0xFF) {
113     Serial.println("This message is not for me.");
114     return;
115 }
116 if(incoming != ""){
117     ultima_vez = millis();    //armz a ultima vez que recebeu um pct
118 }
119 qtdPkt++;                    //contabiliza os pkt recebidos
120 media_rssi = ((media_rssi + LoRa.packetRssi())); //armz media de rssi
121 media_snr = ((media_snr + LoRa.packetSnr()));   //armz media de snr
122
123 Serial.println("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
124 Serial.println("Received from: 0x" + String(sender, HEX));
125 Serial.println("Sent to: 0x" + String(recipient, HEX));
126 Serial.println("Message ID: " + String(incomingMsgId));
127 Serial.println("SF do Mestre: " + String(modoSf));
128 Serial.println("Message length: " + String(incomingLength));
129 Serial.println("Message: " + incoming);
130 Serial.println("RSSI: " + String(LoRa.packetRssi()));
131 Serial.println("Snr: " + String(LoRa.packetSnr()));
132 Serial.println();
133
134 if (modoSf > 6) {
135     sModo = modoSf;
136     String message2 = "Confirmado!"; // conf rcb
137     sendMessage(message2, modoSf);
138     aguardaSemDelay(600);
139     LoRa.setSpreadingFactor(modoSf);
140     Serial.println("Slave em SF: "+String(modoSf));
141     qtdPkt = 0; //zera o nr pct rcb depois de env ao mestre
142     media_rssi = 0;
```

```
143     media_snr = 0;
144     modoMestre = 0;
145 }
146 }
147
148 void apresentaResultados(){
149     if(incomingMsgId == 99){
150         Serial.println("xxxxxxxxxxxxxxxxx RESULTADOS xxxxxxxxxxxxxxxxxxxxx");
151         Serial.println("TesteSF "+String(sModo));
152         Serial.println("NrPkt "+String(incomingMsgId));
153         Serial.println("TT Rcb: "+String(qtdPkt));
154         Serial.println("Perdidos: "+String((incomingMsgId+1) - qtdPkt));
155         Serial.println("RSSI "+String(LoRa.packetRssi()));
156         Serial.println("MdRssi: "+String(media_rssi/qtdPkt));
157         Serial.println("SNR "+String(LoRa.packetSnr()));
158         Serial.println("MdSnr: "+String(media_snr/qtdPkt));
159         Serial.println();
160         //aguardaSemDelay(1000);
161
162         display.clear();
163         display.drawString(0, 0, "TESTE SF "+String(sModo)+" RESULTS");
164         display.drawString(0, 20, "NrPkt "+String(incomingMsgId));
165         display.drawString(65, 20, "TT Rcb: "+String(qtdPkt));
166         display.drawString(0, 30, "Perdidos: "+String(incomingMsgId - qtdPkt));
167         display.drawString(0, 40, "RSSI "+String(LoRa.packetRssi()));
168         display.drawString(56, 40, "MdRssi: "+String(media_rssi/qtdPkt));
169         display.drawString(0, 50, "SNR "+String(LoRa.packetSnr()));
170         display.drawString(56, 50, "MdSnr: "+String(media_snr/qtdPkt));
171         display.display();
172     }
173 }
```