UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Ricardo Mazza Zago

Sistema de baixo custo para monitoramento da geração de energia solar com conexão para Internet das Coisas

Campinas

2018

UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Ricardo Mazza Zago

Sistema de baixo custo para monitoramento da geração de energia solar com conexão para Internet das Coisas

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Eletrônica, Microeletrônica e Optoeletrônica.

Orientador: Prof. Dr. Fabiano Fruett

Este exemplar corresponde à versão final da tese defendida pelo aluno Ricardo Mazza Zago, e orientada pelo Prof. Dr. Fabiano Fruett

_____

Campinas

2018

Agência(s) de fomento e nº(s) de processo(s): CAPES, 1588669
ORCID: https://orcid.org/0000-0003-0175-1976

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

Z132s

Zago, Ricardo Mazza, 1991-
Sistema de baixo custo para monitoramento da geração de energia solar com conexão para Internet das Coisas / Ricardo Mazza Zago. – Campinas, SP : [s.n.], 2018.

Orientador: Fabiano Fruett.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de sensores sem fio. 2. Energia solar. 3. Sistemas embarcados (Computadores). I. Fruett, Fabiano, 1970-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: A low-cost solar generation monitoring system with connection to Internet of Things
Palavras-chave em inglês:
Wireless sensor network
Solar energy
Embedded systems (Computers)
Área de concentração: Eletrônica, Microeletrônica e Optoeletrônica
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Fabiano Fruett [Orientador]
Marcelo Gradella Villalva
Homero Mauricio Schneider
Data de defesa: 20-02-2018
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO

Candidato: Ricardo Mazza Zago RA: 118570

Data da Defesa: 20 de fevereiro de 2018

Título da Tese: "Sistema de baixo custo para monitoramento da geração de energia solar com conexão para Internet das Coisas".

Prof. Dr. Fabiano Fruett (Presidente, FEEC/UNICAMP)
Dr. Homero Mauricio Schneider (CTI)
Prof. Dr. Marcelo Gradella Villalva (FEEC/UNICAMP)

A ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no processo de vida acadêmica do aluno.

Aos meus pais, irmãos e minha namorada pelo apoio incondicional.

Agradecimentos

Ao meu orientador, Prof. Fabiano Fruett, pelo apoio, orientação, conversas, por sempre ter sua porta aberta.

À Carlo Giuliano, pelo suporte técnico nos mais diversos temas durante o desenvolvimento do projeto e suas sugestões.

A todos os amigos do laboratório LSM pelas sugestões e suporte.

Ao Prof. Marcelo Villalva e a seu aluno de doutorado Marcos Reis pelo suporte quando ao uso do LEPO (Laboratório de Eletrônica de Potência).

Ao meu país, que apesar dos problemas sociais, por meio da educação pública de qualidade me fez crescer.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro.

"A mente que se abre a uma nova ideia jamais volta ao seu tamanho original."
(Oliver Wendell Holmes Sr.)

Resumo

Este trabalho apresenta o desenvolvimento de uma rede de sensores sem fio para medição de geração de painéis solares utilizando componentes de prateleira com custo reduzido. O sistema se baseia no conceito de internet das coisas e foi especialmente desenvolvido para medir e quantificar a geração de sistemas fotovoltaicos. Um concentrador foi desenvolvido, para gerência da rede sem fio, que também armazena os dados localmente e hospeda uma página HTML para acesso via rede local. Por sua vez, o acesso via internet é feito através do serviço Ubidots, onde apenas os dados já consolidados são enviados, diminuindo o trafego de rede. O sistema foi projetado para medir saídas DC do painel solar até 60 V de tensão e 30 A de corrente, com baixo consumo de energia. Um aplicativo para uso em computadores pessoais é utilizado para leitura dos bancos de dados e exibição para consolidação dos dados de concentradores, permitindo gerar relatórios de períodos em formatos XLS, CSV e PDF. A acurácia dos nós de medição foi aferida em uma comparação com um osciloscópio utilizando uma ponteira de corrente. O sistema foi implantado em campo para teste e a geração do painel mensurada. Desta maneira, concluímos que o sistema é adequado para microgeração, principalmente doméstica, agregando valor ao painel solar, possibilitando a seu usuário informações que o auxiliam a futuras expansões ou manutenções de seu parque solar.

Palavras-chaves: Rede de sensores sem fio; Geração solar; Sistemas embarcados.

Abstract

This work presents the development of a wireless sensor network for measuring the generation of solar panels using low cost off-the-shelf components. The system is based on the internet of things concept and was specially developed to measure and quantify the generation of photovoltaic systems. A hub was developed for wireless network management, which also stores data locally and hosts an HTML page for local network access. In turn, internet access is done through the Ubidots service, where only the already consolidated data is sent, reducing network traffic. The system is designed to measure solar panel DC outputs up to 60 V voltage and 30 A current, with low power consumption. An application for use on personal computers was developed for reading databases and viewing data from hubs for consolidating, allowing the user to generate period reports in XLS, CSV and PDF formats. The accuracy of the measuring nodes was measured in a comparison with an oscilloscope using a current tip. The system was deployed in the field for testing and the results analyzed. In this way, we conclude that the system is suitable for microgeneration, mainly domestic, adding value to the solar panel, allowing its user information that helps him to future expansions or maintenance of his solar park.

Keywords: Wireless sensor network; Solar generation; Embedded systems.

Lista de ilustrações

Figura 1 – Custo de módulos solares ao longo do tempo, dados da Bloomberg (5). 18
Figura 2 – Conversão de vários tipos de energia em elétrica. 23
Figura 3 – Consumo de energia nos Estados Unidos (1776-2015) (20). 24
Figura 4 – Matriz energética mundial. 25
Figura 5 – Espectro de irradiação solar com massa absoluta do ar de 1,5 a.u. (AM1.5). 27
Figura 6 – Representação de uma junção p-n com indicação do campo elétrico e região de depleção. 29
Figura 7 – Representação do diodo de *bypass*. 30
Figura 8 – Curva I-V típica de um módulo solar. 31
Figura 9 – Sistema da empresa alemã SMA Solar WebBox. 33
Figura 10 – Diagrama de instalação de um sistema da Solar Edge[1]. 34
Figura 11 – Placas consideradas para o concentrador. 45
Figura 12 – Placas consideradas para os nós. 47
Figura 13 – Modelos do Google OnHub.[2] 49
Figura 14 – Vista aérea do Hospital de Clinicas da Unicamp.[3] 50
Figura 15 – Topologias de rede. 51
Figura 16 – Topologia de uma rede ZigBee Mesh. 52
Figura 17 – Esquema de uma rede Digimesh. 53
Figura 18 – Placa “output-board kit” da panStamp.[4] 54
Figura 19 – XBee S2C ZigBee. 56
Figura 20 – Esquemático do *shield* do Arduino para XBee. 56
Figura 21 – Arduino utilizando o *shield* para conexão ao XBee. 57
Figura 22 – Placa utilizada para conexão do XBee a computadores e ao Raspberry Pi. 57
Figura 23 – Esquemático do XBee Explorer Stick v1.0. 57
Figura 24 – Reguladores chaveados utilizados no projeto. 63
Figura 25 – Esquemático do regulador de tensão utilizado para tensões até 60 V. . 64
Figura 26 – Amperímetro Alicate YHDC 20A SCT-013.[5] 64
Figura 27 – Sensores de corrente. 65
Figura 28 – Esquemático do sensor hall Allegro ACS712. 66
Figura 29 – Gráfico da corrente (A) pela tensão de saída do sensor (V). 66
Figura 30 – Circuito amplificador utilizado para o sensor de corrente. 68
Figura 31 – Resultados da simulação do circuito amplificador do sensor de corrente. (a) Tensão de entrada do amplificador, (b) Tensão de saída do amplificador e (c) Ganho do amplificador. 69

Figura 32 – Esquemático da placa de tratamento de sinais. 70
Figura 33 – Projeto da placa e ela fabricada com componentes 70
Figura 34 – Impressora 3D Sethi3D AiP A3. 71
Figura 35 – Padrão de impressão 3D utilizando o *software* Repetier. 72
Figura 36 – Projeto da caixa e caixa já montada com os componentes dentro. . . . 72
Figura 37 – Nó 5 enviando dados via *unicast* para o no central utilizando a topologia *Mesh*. 78
Figura 38 – Estrutura genérica de um *frame* de dados utilizados no XBee. 79
Figura 39 – Procedimento de amostragem dos dados. 80
Figura 40 – Fluxograma do funcionamento do *firmware*. 82
Figura 41 – Fluxograma do funcionamento do *software* do concentrador. 83
Figura 42 – Arquivo de configurações do concentrador. 84
Figura 43 – Interface de usuário para visualização de dados e geração de relatórios. 86
Figura 44 – Layout da segunda versão da placa no Kicad. 90
Figura 45 – Esquemático da segunda versão da placa no Kicad. 91
Figura 46 – Visão 3D da segunda versão da placa. 92
Figura 47 – Projeto e caixa já fabricada. 92
Figura 48 – Formação da rede sem fio Zigbee visualizada pelo *software* XCTU. . . . 95
Figura 49 – Emulador de painel solar Ametek TerraSAS ETS 600/25. 96
Figura 50 – Osciloscópio LeCroy MSO 44MXs-B. 96
Figura 51 – Tensão e corrente medidas pelo nó do sistema e utilizando o osciloscópio. 97
Figura 52 – Potência medida pelo nó do sistema e utilizando o osciloscópio. 97
Figura 53 – Sistema instalado em campo. 98
Figura 54 – Geração medida pelo sistema no dia 23/08/2017. 98
Figura 55 – Geração medida pelo sistema no dia 24/08/2017. 99
Figura 56 – Geração medida pelo sistema no dia 25/08/2017. 99
Figura 57 – Página Web do Ubidots com dados enviados por um nó da rede. 100
Figura 58 – Página Web do Ubidots com painel de informações exibindo informações de tensão, corrente e potência de um painel. 100
Figura 59 – Aplicativo Ubidots para *Smartphones* Android. 101
Figura 60 – Interface gráfica do programa XCTU utilizado para alterar configurações e atualizar o *firmware* do XBee. 118
Figura 61 – Gerador de *frames* do *software* XCTU. 120
Figura 62 – *Frame* gerado para enviar a mensagem "Hello World!". 121

Lista de tabelas

Tabela 1 – Características de corrente e tensão de alguns modelos de painéis solares: 32
Tabela 2 – Comparação entre placas de desenvolvimento. 46
Tabela 3 – Características Arduino Uno R3 e da NXP FRDM-KL25Z. 48
Tabela 4 – Comparação entre opções de rede. 55
Tabela 5 – Resultados do ensaio do sensor de corrente. 67
Tabela 6 – Consumo de energia dos componentes utilizados na montagem dos protótipos. 73
Tabela 7 – Custo dos componentes contidos em cada nó. 74
Tabela 8 – Custo dos componentes contidos no concentrador. 74
Tabela 9 – Canais disponíveis por versão de rádio XBee. 78
Tabela 10 – Principais tipos de comandos AT: 79
Tabela 11 – Campos presentes no banco de dados. 84
Tabela 12 – Características utilizadas no emulador de painel solar: 96
Tabela 13 – Consumo do nó removendo componentes um a um. 102
Tabela 14 – Consumo individual dos principais componentes dos nós 102
Tabela 15 – Variação da eficiência do conversor DC-DC alterando a tensão de alimentação. 103
Tabela 16 – Custo dos componentes dos nós em dólares. 106

Lista de abreviaturas e siglas

ANEEL Agência Nacional de Energia Elétrica.

AES *Advanced Encryption Standard.*

BOM *Bill of Material*, lista de materiais.

CISC *Complex Instruction Set Computer.*

CSV *Comma-separated values*, valores separados por vírgula.

DSI *Display Serial Interface.*

DSP *Digital Signal Processor*, processamento digital de sinal.

Firmware Software básico de baixo nível responsável pelo controle do hardware.

FPGA *Field-programmable gate array.*

GPIO General Purpose Input/Output.

GSM Global System for Mobile Communications.

I²C Inter-Integrated Circuit.

IoT *Internet of Things* (Internet das Coisas).

JSON *JavaScript Object Notation.*

MPPT *Maximum Power Point Tracking.*

MISO *Master Input, Slave Output.*

MPPT *Maximum Power Point Tracking.*

MOSI *Master Output, Slave Input.*

NFC *Near Field Communication.*

OTA *Over the Air*, atualização de *software* realizada em um dispositivo não conectado a um computador.

PCB *Printed Circuit Board*, Placa de circuito impresso.

PCBA *Printed Circuit Board Assembly*, Placa de circuito impresso com componentes soldados.

PWM *Pulse Width Modulation*, Modulação por largura de pulso.

RAM *Random Access Memory.*

RISC *Reduced Instruction Set Computer.*

RFID *Radio-Frequency Identification.*

RJ45 *Registered jack 45*, conector usualmente utilizado para conexão à internet de computadores e servidores.

ROM *Read only Memory.*

RS-232 *Recommended Standard 232*, padrão de comunicação serial.

SCL *Serial Clock.*

SDA *Serial Data.*

SoC *System-on-a-chip.*

SPI *Serial Peripheral Interface Bus.*

SS *Slave Select.*

SSH *Secure Shell.*

VPN *Virtual private network.*

UART *Universal asynchronous receiver/transmitter.*

USB *Universal Serial Bus.*

UWP *Universal Windows Platform.*

Wi-Fi É uma tecnologia de redes sem fio local, baseadas no padrão IEEE 802.11, comumente utilizada para conexão de computadores e telefones portáteis à internet.

Sumário

1 Introdução 18
1.1 Objetivo Geral 20
1.2 Objetivos Específicos 20
2 Revisão Bibliográfica 21
2.1 Rede de Sensores Sem Fio 21
2.2 Panorama Energético 23
2.3 Células fotovoltaicas 26
2.3.1 História 26
2.3.2 Radiação solar 26
2.3.3 Física das células fotovoltaicas 27
2.3.4 Parâmetros da célula fotovoltaica 29
2.3.5 Organização macroscópica das células fotovoltaicas 30
2.3.6 Curva I-V 31
2.3.7 Métodos de MPPT 31
2.3.8 Análise das características de painéis solares 32
2.4 Soluções comerciais de medição 32
2.5 Medição de geração solar em laboratórios 34
2.6 Resoluções normativas da ANEEL 36
2.7 Computação em nuvem 37
2.8 Conclusões do capítulo 38
3 Hardware do Sistema 40
3.1 Placas de desenvolvimento, SoC e Microcontroladores 41
3.1.1 Interface de comunicação GPIO e protocolos UART, SPI e I^2C. . . 42
3.1.1.1 SPI 42
3.1.1.2 I^2C 43
3.2 Placa de desenvolvimento para o concentrador 43
3.3 Placa de desenvolvimento para os nós sensores 46
3.4 Rede sem fio 48
3.4.1 Dimensionamento da aplicação 49
3.4.2 Padrão de mercado e tecnologias proprietárias 49
3.4.3 Topologia de rede em estrela ou malha 50
3.4.4 IEEE 802.11 e Bluetooth 51
3.4.5 IEEE 802.15.4 51
3.4.6 ZigBee 51
3.4.7 Thread 52
3.4.8 Digimesh 53

3.4.9 LoRa . . . 53
3.4.10 panStamp . . . 54
3.4.11 Comparação entre soluções de rede . . . 55
3.4.12 Conexão do XBee ao Raspberry Pi e Arduino . . . 56
3.5 Sistema Operacional . . . 57
3.5.1 Para o concentrador . . . 58
3.5.1.1 Linux . . . 58
3.5.1.2 Windows 10 IoT . . . 58
3.5.1.3 Android Things (Google Brillo) . . . 59
3.5.1.4 Comparação . . . 59
3.5.2 Para os nós . . . 59
3.5.2.1 Zephyr . . . 60
3.5.2.2 Contiki . . . 61
3.5.2.3 FreeRTOS . . . 61
3.6 Alimentação dos nós sensores . . . 62
3.7 Medição de Tensões e Correntes . . . 64
3.8 Placa projetada . . . 66
3.8.1 Caixa fabricada utilizando uma impressora 3D . . . 69
3.9 Consumo de energia estimado do sistema . . . 72
3.10 Custo . . . 73
3.11 Conclusões do capítulo . . . 74
4 Integração do software com o hardware . . . 76
4.1 Configuração da rede sem fio Zigbee . . . 76
4.1.1 Canais de rede sem fio . . . 77
4.1.2 Frame de dados no modo API do XBee . . . 78
4.2 Firmware desenvolvido . . . 79
4.3 *Software* desenvolvido para o concentrador . . . 81
4.4 Interface para o usuário analisar dados . . . 85
4.5 Segurança . . . 86
4.5.1 Segurança em redes sem fio . . . 87
4.5.2 Segurança no padrão Zigbee . . . 88
4.5.3 Processo de fabricação de *hardware* . . . 89
4.5.4 Projeto de versão atualizada . . . 90
4.6 Conclusões do capítulo . . . 92
5 Resultados . . . 94
5.1 Conexão dos nós . . . 94
5.2 Validação das medições do sistema . . . 95
5.3 Estudo de campo . . . 97
5.4 Consumo de energia medido do nó . . . 102

5.5 Conclusões do capítulo . . . 103
6 Conclusões finais . . . 105
6.1 Trabalhos Futuros . . . 107
Publicações . . . 109

Referências . . . 110

Anexos 117
ANEXO A Configurações do rádio XBee . . . 118
A.1 Configurações do XBee . . . 118
A.2 Detalhamento de um *frame* XBee de dados . . . 120
ANEXO B Transformando um programa em serviço no Linux . . . 122
ANEXO C Códigos fonte . . . 125
C.1 Concentrador . . . 125
C.1.1 main.py . . . 125
C.1.2 server.py . . . 132
C.1.3 access_database.py . . . 134
C.1.4 cloud.py . . . 137
C.1.5 plotador.py . . . 138
C.1.6 main_interface_db.py . . . 141
C.1.7 aux_functions.py . . . 148
C.2 Firmware . . . 149

1 Introdução

Energias renováveis a cada dia se tornam mais importantes para o futuro. Pesquisas indicando a realidade do aquecimento global apenas aumentam a importância do uso de fontes renováveis e não poluentes (1). Segundo o *Key World Energy Statistics* de 2017 publicado pela Agência de Energia Internacional (2), no ano de 2015, 31,7% da energia consumida no mundo adveio do petróleo, 28,1% de carvão mineral, 21,6% de gás natural e 4,9% de nuclear, ou seja, mais de 85% da energia consumida foi gerada a partir de fontes não renováveis. Isto liberou 32.294 Mt de CO_2 na atmosfera.

Diversos países têm investidos de forma crescente em geração solar, a Alemanha, por exemplo, tem uma capacidade instalada de 39700 MW (3), praticamente a mesma quantidade (4) que o Brasil tem de instalada e fiscalizada de usinas termelétricas. A capacidade instalada de geração solar em países como China, Japão e Estados Unidos, cresceu respectivamente, 15150, 11000, 7300 MW, apenas em 2015 (3).

O custo de módulos solares caiu muito nos últimos anos, em dados da Bloomberg utilizados em um relatório por pesquisadores do Laboratório Nacional de Energia Renováveis dos Estados Unidos (5), o custo dos módulos caiu aproximadamente 40% entre o período de início de 2015 e início de 2017. Estas informações são compiladas no gráfico e reproduzidas com adaptações na Figura 1.

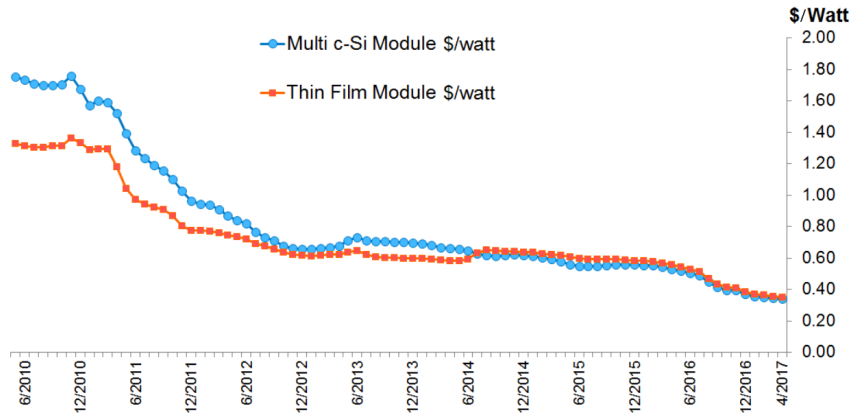


Figura 1 – Custo de módulos solares ao longo do tempo, dados da Bloomberg (5).

No Brasil, o primeiro leilão público onde as energias renováveis não disputaram por preço com as não renováveis ocorreu em 2014, sendo vendida a capacidade instalada de 1.048 MW, entrando no sistema integrado nacional a partir de 2017 (6). Para o mercado residencial e comercial, a Associação Brasileira de Energia Solar Fotovoltaica (Absolar) estima no ano de 2017 um aumento na capacidade instalada de 235 MW para 1000 MW, um crescimento de 325% (7). Com o aumento do preço de energia devido à falta de

chuvas de implementação das bandeiras tarifárias, o custo de instalação de painéis solares se tornou mais competitivo.

Além disso, em 2012, através da Resolução Normativa número 482, a ANEEL estabeleceu as condições gerais para o acesso de microgeração e minigeração aos sistemas de distribuição de energia. Isto regulamentou a forma com que usuários domésticos que instalassem painéis solares em casa e obtivessem créditos em suas contas para quando a produção excedesse o consumo (8).

Existem também sistemas que não estão conectados na rede, chamado de *off-grid* pode não necessitar de um inversor, caso a carga seja de corrente continua. Mesmo que necessite, este inversor não precisa das funções de controle e proteção utilizadas em sistemas *grid-tie*, como proteção contra o efeito "ilha"(9), possuindo menor complexidade e consequentemente menor custo. No entanto, o custo do sistema, no geral, é mais elevado, pela necessidade de baterias. Independentemente de o sistema estar ou não ligado à rede elétrica, ele é benéfico para o meio ambiente, já que faz com que outras fontes sejam menos demandas, e ainda socialmente, já que leva energia a áreas remotas.

Mesmo em sistemas ligados a rede elétrica, caso se utilize um inversor simples, sem funções de medição, apenas a energia enviada e recebida pela rede é mensurada pelo medidor da concessionaria. Isto é pior ainda em sistema *off-grid*, que geralmente utilizando inversores de baixo custo que não possuem capacidade de medição. Assim, o usuário do painel, seja ele residencial ou empresarial, não sabe quanta energia foi gerada e consequentemente quanto dinheiro deixou de ser gasto na conta de luz, além disso, o próprio governo não dispõe destas informações para a criação de políticas públicas.

Com o mercado de equipamentos e aplicativos para IoT (*Internet of Things*) em plena expansão, investimentos de grandes multinacionais de bilhões de dólares impulsionam este mercado, como o efetuado recentemente pela IBM de US$ 3,0 bilhões (10) ou pela Samsung de US$ 1,2 bilhões (11). É possível que dentro de alguns anos tenhamos em casa "centrais de controle", versões mais poderosas da atual Amazon Echo ou do recém anunciado Google Home (12). Estes dispositivos seriam responsáveis por fornecer a conexão à internet, sugestões e lembres, totalizar os dados de geração do painel solar, da rede elétrica por *Smart Grid*, etc.

Estas centrais de controle necessitarão de aparelhos de medição que tenham o custo mais baixo possível para ler estes dados e enviar ao concentrador. Mesmo uma prefeitura poderia fazer uso para um programa de instalação de painéis solares, cuja economia de energia esteja atrelada a um pequeno desconto no IPTU, caso esta rede tivesse alcance o suficiente.

O desenvolvimento deste projeto possibilitará a criação de ferramentas para a obtenção de dados para orientar futuras políticas públicas na área de energias renováveis

que sejam acessíveis, precisas, fáceis de usar e instalar, impulsionando a implementação de energias renováveis na matriz energética nacional.

1.1 Objetivo Geral

Desenvolver uma rede *mesh* de dispositivos para medição da quantidade de energia elétrica gerada e o estado de operação de paneis solares utilizando componentes comerciais de baixo custo.

1.2 Objetivos Específicos

- Desenvolver uma rede robusta em malha de dispositivos de medição com escalabilidade para atendimento de áreas com centenas de painéis solares;
- Medir corrente e tensão na saída de cada painel solar para calcular a energia gerada;
- Que o sistema tenha um consumo máximo de 1% da energia gerada pelo painel fotovoltaico e custe no máximo 10% do preço de um painel fotovoltaico;
- Desenvolver protótipos do sistema e validá-los em campo;
- Projetar uma interface de exibição dos dados aos usuários de fácil uso, intuitiva, consolidando os dados para exibição em um *dashboard*, que esteja disponível para o usuário através de uma página da internet ou aplicativo celular.

2 Revisão Bibliográfica

Neste Capítulo será feita uma revisão bibliográfica de assuntos pertinentes ao desenvolvimento da dissertação de mestrado.

2.1 Rede de Sensores Sem Fio

Uma rede de sensores sem fio (RSSF), do inglês *Wireless Sensor Network (WSN)*, é uma rede de pequenos dispositivos, chamados de nós sensores, que monitoram uma determinada área e enviam a informação por uma rede sem fio geralmente para um concentrador, que pode utilizar a informação localmente, armazená-la ou enviar para outra rede, como a Internet (13). Os nós podem ser todos do mesmo tipo ou podem ser diferentes utilizando o mesmo protocolo de rede.

Os nós desta rede podem ser alimentados por bateria ou utilizar métodos de *energy harvesting*, como módulos solares. De qualquer maneira, seu consumo deve ser mínimo, já que o fornecimento de energia é limitado. O maior consumidor de energia é o rádio, sendo utilizados geralmente rádios de baixa taxa de transmissão (10-100 kbps) e pequeno alcance (menos de 100 m). Outra maneira de diminuir o consumo é deixar os nós o maior tempo possível no modo de baixo consumo de energia.

Existem diversos tipos de redes de sensores, tais quais (13):

- Terrestres: geralmente contém um grande número de nós (centenas ou milhares), onde os nós funcionam também como retransmissores de informação. Aplicações comuns são sensoriamento e monitoramento ambiental e monitoramento industrial;
- Subterrâneo: sensores são aplicados em cavernas, minas ou no subterrâneo, os concentradores são colocados acima do solo. Devido à dificuldade de recarga de bateria, métodos para economia de energia são extremamente importantes. Geralmente são utilizados para monitoramento de agricultura, no subterrâneo do solo, água e minerais e monitoramento de fronteira.
- Submersa: geralmente os nós subaquáticos são caros, portanto pouco empregado. Pode-se usar veículos submersos para recolher os dados ou comunicação sem fio por meio de ondas acústicas, porém contando com baixa taxa de transmissão. Aplicações incluem monitoramento de poluição, vigilância e exploração subaquática, prevenção de desastre, monitoramento sísmico, etc.

Redes de sensores são amplamente utilizadas na indústria e na academia. Alguns exemplos de projetos utilizando redes de sensores:

- Em Ruiz-Garcia et al.(14) é apresentado uma rede de sensores utilizando Zigbee para a medição das condições de transporte de produtos frescos em caminhões refrigerados em longas distâncias. Foi feito um estudo de caso em uma viagem de Mercia, na Espanha, para Avinhão, na França, onde 4 sensores forem colocados espalhados no baú do caminhão. Eles monitoraram a temperatura, umidade relativa, paradas e aberturas de porta.

- Casey, Lim e Dozier(15) apresentam uma proposta de rede de sensores e atuadores para mitigar os efeitos de um Tsunami. A rede consiste em 3 tipos de nós: os sensores que medem a pressão da água, os de comando que analisam os dados coletados e os de barreira para diminuir os efeitos do Tsunami. Os autores também propões algoritmos para predição da direção da onda e protocolos de rede.

- Li et al.(16) desenvolveram um sistema baseado em rede de telefonia móvel para medição do nível d'água. Foi desenvolvido um sistema operacional próprio, nomeado de ZKOS, que possui a capacidade de atualização OTA, ou seja, sem acesso físico ao sistema é possível atualizar todo o *firmware*. O sistema mede o nível de água via um conversor analógico digital de 12 bits tendo precisão de 4.882 mm, após *oversampling*. Este sistema é útil para verificar alagamentos e notificar mais rapidamente do que o sistema anteriormente utilizado.

- Mudumbe e Abu-Mahfouz(17) desenvolveram um sistema para medição de consumo de água utilizando uma rede de sensores sem fio, que também pode ser utilizado para detecção de vazamentos. Eles fazem uso de um módulo da Dizic que possui suporte ao padrão IEEE 802.15.4, microcontrolador ARM Cortex M, 128 KB de memória ROM e 8 KB de memória RAM. Foi utilizado o protocolo 6LoWPAN e o *software* Pandora FMS adaptado para exibir as informações.

- Jalali, El-khatib e McGregor(18) apresentam uma arquitetura de redes para cidades inteligentes, que faz uso de três camadas: a de sensoriamento, onde é utilizado RFID como NFC e rede de sensores sem fio. A camada de rede, onde são conectadas cada uma das aplicações. Por último, a camada de aplicação, que é responsável pelo processamento de apresentação dos dados. Foi feito um estudo na área de saúde utilizando um sistema com Raspberry Pi e Arduino, que mede o ritmo cardíaco, pressão sanguínea, posição, oxigenação, fluxo de ar, temperatura corporal, entre outros, enviando a informação para o telefone celular via rede sem fio Bluetooth, que por sua vez envia para um servidor que procura padrões nos dados utilizando.

Mais conceitos como topologias e padrões de rede sem fio e microcontroladores utilizados serão apresentados posteriormente.

2.2 Panorama Energético

As necessidades energéticas de sociedades modernas aumentam cada dia, para diferentes propósitos. Seja para processos industriais, preparação de alimentos, conforto, entretenimento, entre outros. Energia é utilizada para refrigeração do ambiente com ar condicionado, o que historicamente aumentou muito a produtividade do trabalhador. Também é utilizada para aquecimento, seja de água para banho, de alimentos ou de toda a casa em lugares muito frios.

No exemplo anterior, para aquecimento pode ser utilizada a queima de gás ou óleo diesel, uma resistência elétrica ou luz solar. Em qualquer um destes, a energia não é criada nem destruída, mas sim transformada de uma forma para outra.

É dito que uma fonte de energia é primária quando está em seu estado natural, sem conversões ou transformações feitas pelo homem. Alguns exemplos são: carvão, petróleo, radiação solar, vento, água corrente e urânio. Quando esta energia é convertida para uma forma a ser transportada ou enviada para uso, tais quais eletricidade, gasolina ou óleo diesel e vapor, é dito que é uma forma de energia secundária ou final (19).

A conversão de energia de uma forma a outra tem um papel importante na matriz energética mundial, na Figura 2 é feita uma adaptação de Freris e Infield(1), onde são mostradas as maneiras de se produzir energia elétrica. A principal maneira é passando a energia de química para térmica, mecânica e elétrica, com perdas em todas essas conversões. A maior perda está na conversão de térmica para mecânica, que tem eficiência de no máximo 60%, limitada pelo ciclo de Carnot (1).

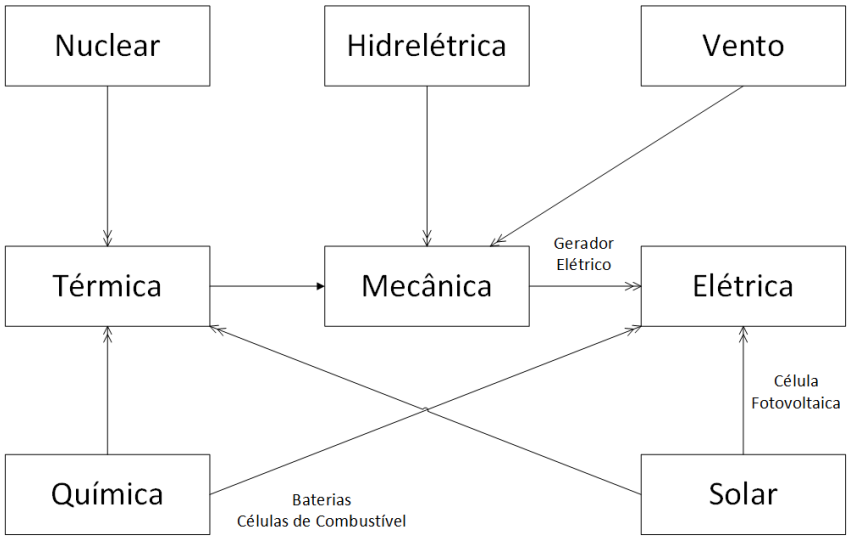


Figura 2 – Conversão de vários tipos de energia em elétrica.

Também na Figura 2 é possível visualizar que existem duas maneiras de gerar energia elétrica a partir do sol. A primeira é diretamente usando células fotovoltaicas, que convertem a luz solar em corrente elétrica. A segunda utiliza luz solar concentrada para aquecer vapor e girar turbinas, consequentemente produzindo energia elétrica, tecnologia chamada de *Concentrated solar power*.

A Administração de Informação de Energia dos Estados Unidos (EIA) fez um interessante estudo com o consumo energético americano desde 1776 até dias atuais (20). Estes dados são mostrados na Figura 3, extraídos do site da EIA sem adaptações. Pode ser visto que inicialmente somente biomassa era utilizada, provavelmente para aquecimento e preparação de alimentos. Apenas na segunda metade do século XIX o carvão passou a ser utilizado e no começo do XX, o petróleo.

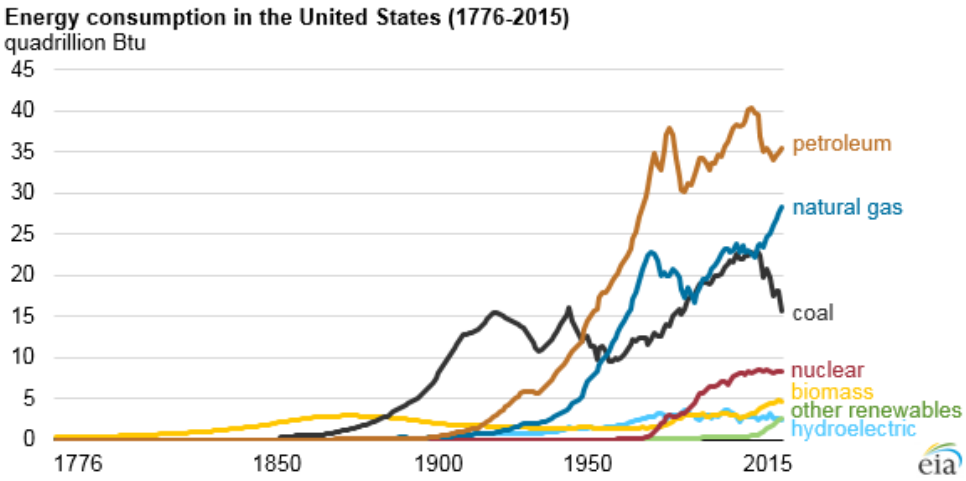


Figura 3 – Consumo de energia nos Estados Unidos (1776-2015) (20).

Com o crescimento da econômica mundial, o uso de energia também deve crescer, já que, no geral, quanto maior a capacidade de geração, distribuição e conversão de energia um país possui, maior é seu desenvolvimento econômico e tecnológico. No entanto, é evidente que um país pode fazer maiores investimentos em uso eficiente de energia, reduzindo seu consumo em comparação aos demais de desenvolvimento equivalente.

O relatório da Agência de Energia Internacional prevê um crescimento de mais de 70% na demanda por energia elétrica até 2040, com energias renováveis se tornando, no lugar do carvão mineral, a maior fonte de energia elétrica no começo da década de 30. Por volta de 550 milhões de pessoas ainda estarão sem acesso à energia elétrica, em sua maioria na África subsaariana (21).

Uma energia é dita renovável quando pode ser reposta por processos naturais em ritmo comparável ou maior que seu consumo. Por esta definição energias hidráulica, eólica e solar são renováveis. Na energia hidráulica, com chuvas em locais elevados ou montanhosos é possível gerar energia através de hidro geradores devido a energia potencial desta água em altitude elevada. Na eólica a energia cinética dos ventos é convertida em

elétrica utilizando turbinas especialmente desenvolvidas. Em última instância, ambas as energias advêm da solar, pois as nuvens são formadas de água inicialmente evaporada pelo sol e os ventos são resultado da diferença de pressão na atmosfera induzida pela radiação solar (19).

Cada país tem suas particularidades no que se refere a matriz energética, ou seja, as fontes de energia utilizadas. Dependendo de fatores históricos, disponibilidade de recursos naturais, de rios, ou mesmo a falta deles. O Brasil, por sua vez, tem uma elevada quantidade de usinas hidrelétricas, onde as usinas térmicas são acionadas apenas em momentos de seca ou necessidade de maior geração. Outros países, por sua vez, não têm muitos rios aproveitáveis para geração ou sua capacidade já foi utilizada praticamente em sua totalidade. Os Estados Unidos é um exemplo disso, apesar de praticamente do mesmo tamanho do Brasil, é dependente de usinas térmicas (66%) e nucleares (20%) (22), em contraste ao Brasil que possuí 61% de sua geração em hidrelétricas (4).

Em Key World Energy Statistics 2017 (2) da Agência de Energia Internacional são estudados a distribuição de uso de cada tipo de energia na matriz energética mundial. Na Figura 4 é mostrada esta distribuição. Como já explicitado, mais de 85% da geração advém de fontes não renováveis.

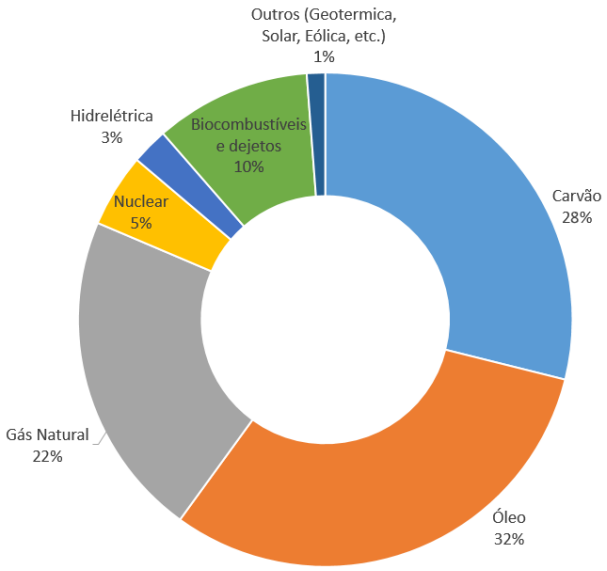


Figura 4 – Matriz energética mundial.

Em Solar FAQs (23) é feito um estudo teórico da geração solar. O autor analisa que a capacidade teórica máxima de geração de energia solar na Terra em uma hora e trinta minutos (480 EJ) é maior que o consumo mundial total de energia em 2001 (430 EJ), onde seria necessário apenas 0,17% da superfície da terra para gerar tal energia ao longo de todo ano. Além disso, para suprir a demanda energética americana de 2001 seria necessário cobrir 2,0% da superfície terrestre americana, o que é aproximadamente 30 vezes o espaço disponível nos telhados de construções americanas.

2.3 Células fotovoltaicas

2.3.1 História

Em 1839, Edmond Becquerel descobriu que duas placas de latão imersas em líquido produziam corrente continua quando iluminadas com luz solar. Na década de 1870, Smith, Adams e Day descobriram o efeito fotoelétrico no selênio. Charles Fritts no ano de 1883 cobriu uma amostra de selênio amorfa com uma fina camada de ouro, reportando a produção de corrente quando exposto ao sol. Existia grande ceticismo em relação a estas descobertas, logo uma amostra foi enviada para Werner Siemens na Alemanha, que confirmou os resultados, porém a eficiência na conversão era de apenas 1% (24).

Em 1954, Chapin, do Bell Labs, inventou a chamada *silicon single-crystal solar cell*, com 6% de eficiência, que se elevaria nos anos seguintes para 15%. Com o lançamento do primeiro satélite Sputnik em 1957, os módulos solares se tornaram ideais fontes de energia para exploração espacial, já que são leves e demandam pouca manutenção. Hoje são amplamente utilizadas na Estação Espacial Internacional (24).

Até a primeira crise do petróleo, em 1973, a indústria de módulos solares permaneceu pequena, com a confiabilidade em primeiro plano e custo em segundo. Depois de 1973, com o foco na fabricação, o custo dos painéis caiu muito. Além disso, passaram a ter maior resistência as intempéries do tempo. A eficiência máxima subiu de 10% para mais de 30%, quando considerando células de tripla junção (24).

2.3.2 Radiação solar

Maxwell, no século XIX, com a formulação de sua teoria eletromagnética previu que a luz seria uma onda eletromagnética, a partir da comparação entre os valores de $1/\sqrt{\mu_0\epsilon_0}$ e da velocidade da luz disponíveis na época. Em 1900 foi descoberto por Max Planck, que trabalhava com o problema da emissão de um corpo negro, que a energia poderia apenas ser absorvida ou emitida em pequenos "pacotes" que ficaram conhecidos como Quantum.

Einstein, em seu famoso *paper* sobre o efeito fotoelétrico, pelo qual recebeu o prêmio Nobel em 1921, percebeu que uma onda eletromagnética incidente sobre um metal poderia liberar elétrons quando a frequência da radiação fosse maior que um certo valor mínimo. Aumentando a intensidade da luz aumenta o número de elétrons, mas não a energia de cada um. Chegando à conclusão que a frequência da luz é proporcional a energia do fóton e que ela é quantizada, sendo dada pela Equação 2.1 (25).

$$E = hf \tag{2.1}$$

O termo E da Equação 2.1 é a energia dada em Joules, h a constante de Plank que é igual a $6,63 \times 10^{-34}$ J $\cdot$ s e f a frequência da radiação eletromagnética (25).

A maior parte do espectro de emissão solar está entre 300 nm e 2500 nm (24) e a luz visível entre 390 nm e 700 nm (25). Na Figura 5 é mostrado o espectro de emissão solar medido na superfície da Terra, criada com auxílio do MATLAB, baseada em dados da American Society for Testing and Materials Terrestrial Reference Spectra for Photovoltaic Performance Evaluation para AM1.5 (26).

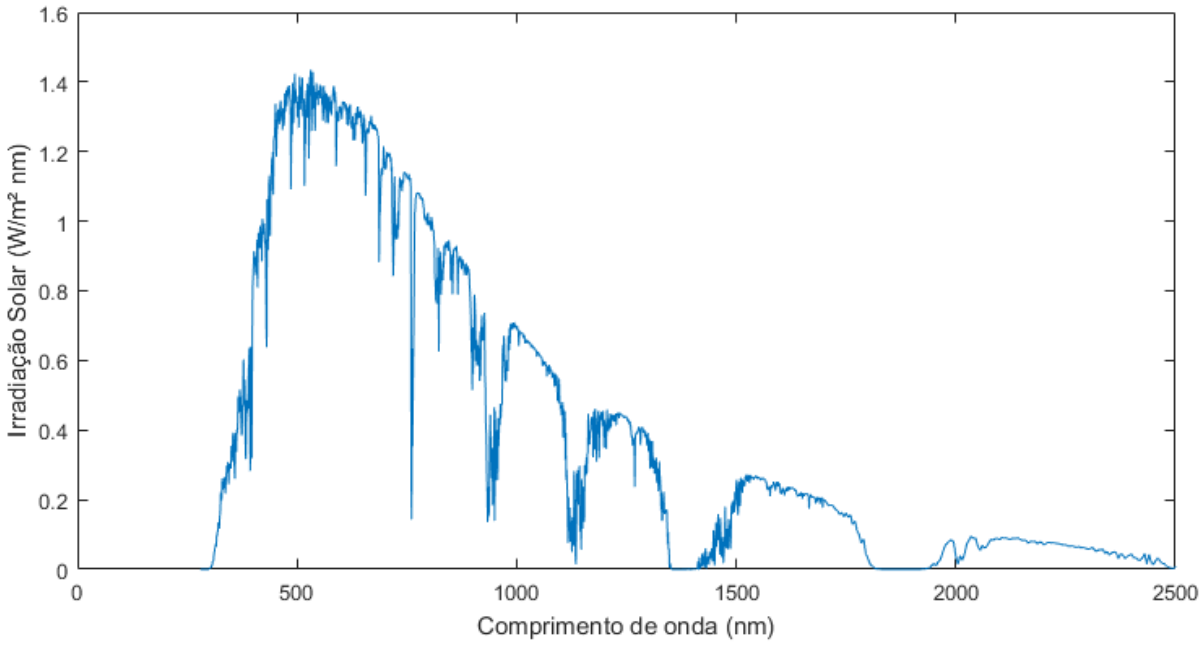


Figura 5 – Espectro de irradiação solar com massa absoluta do ar de 1,5 a.u. (AM1.5).

O espectro solar fora da atmosfera é chamada de AM0, pois não ocorre nenhuma atenuação devido à atmosfera, logo o número zero. Quando a radiação solar passa pela atmosfera ela é atenuada, portanto é importante saber qual distância a radiação solar percorreu na atmosfera para conhecer a atenuação. Quando o sol está no zênite é a menor distância que os raios solares devem percorrer. A razão entre o caminho e o menor caminho possível é chamada de massa ótica do ar e é dada pela Equação 2.2, a condição padrão para teste é AM1.5.

$$AM \approx \frac{1}{\cos\theta} \tag{2.2}$$

2.3.3 Física das células fotovoltaicas

Uma célula fotovoltaica é basicamente uma junção p-n, figurativamente uma união de duas peças de silício (ou outro material semicondutor), com dopagens de elementos diferentes em cada um dos lados. Na prática é utilizado apenas uma peça e o processo

de dopagem usado para criar estas duas zonas. Consequentemente existe uma quantidade maior de cargas positivas em um dos lados e negativas no outro, criando uma diferença de potencial. Esta diferença faz com que cargas geradas pela luz solar no material sejam atraídas mais para um lado, criando uma corrente elétrica quando conectada a uma carga externa.

Quando esta rede cristalina de um material semicondutor está próxima do zero absoluto (0 K) não há elétrons na banda de condução, logo o material é um isolante. Conforme a temperatura aumenta, elétrons ganham energia e são liberados de suas ligações, criando um par elétron lacuna. Uma lacuna pode ser vista como um portador de carga positiva, sendo a falta de um elétron. É chamada de recombinação quando um elétron passa a ocupar o lugar de lacuna (27).

Existe um número médio de pares elétrons lacunas em uma estrutura cristalina pura de silício, que varia conforme a temperatura. A proporção de elétrons e lacunas numa rede cristalina pode ser modificada adicionando impurezas ao silício. Adicionando um átomo como boro ou gálio com 3 elétrons na camada de valência são criadas mais lacunas, pois fica "faltando" um elétron, sendo chamada de dopagem tipo "P", de positivo. Um átomo como fósforo ou arsênio com 5 elétrons na camada de valência faz com que fique sobrando um elétron, aumentando a concentração de elétrons em relação a lacunas, sendo chamado de material tipo "N" (27).

Existem dois métodos de movimentação de portadores de carga, a deriva e a difusão. Na deriva é aplicada uma diferença de potencial no material e é criado um campo elétrico, que é responsável pelo movimento dos portadores de cargas, elétrons e lacunas. A velocidade média de movimento dos portadores é proporcional ao campo elétrico:

$$v = \mu E \tag{2.3}$$

Onde μ é chamado de mobilidade e varia conforme o material e portador. No silício para o tipo N é $\mu_n = 1350$ cm^2/(V.s) e para o tipo P é $\mu_p = 480$ cm^2/(V.s) (27).

Na difusão, portadores em áreas com maior densidade se movem para regiões de menor densidade. Considerando que não existe um campo elétrico externo aplicado, o movimento continua até que a densidade de portadores do material seja uniforme.

Uma junção p-n, também chamada de diodo, possui duas zonas com dopagens do tipo p e tipo n. Um lado tem uma densidade maior de lacunas e o outro de elétrons, devido a esta diferença de densidade de portadores, por difusão, esta concentração tende a se uniformizar com a migração das cargas, surge um campo elétrico que contrabalanceia o efeito da difusão, criando uma região chamada de "depleção". Este efeito é mostrado na Figura 6.

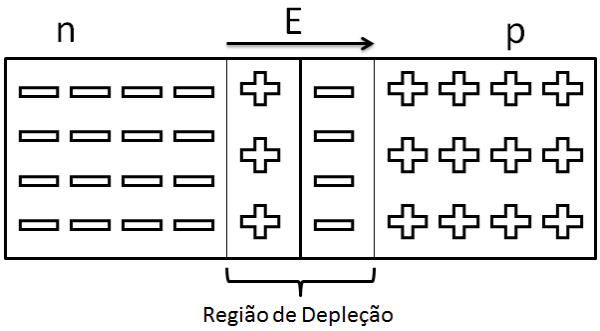


Figura 6 – Representação de uma junção p-n com indicação do campo elétrico e região de depleção.

Quando uma junção p-n é iluminada, o número de portadores minoritários, elétrons no tipo p e lacunas no tipo n, aumenta muito. Isto resulta em um fluxo de portadores atravessando a região de depleção. Este fluxo de portadores fotogerados resultado na chamada corrente de fotogeração. Apenas portadores de carga gerados em sua região minoritária e na região de depleção contribuem para a corrente fotogerada. Portanto, o projeto da célula fotovoltaica deve levar em consideração este fato, para que a espessura da área de absorção não seja maior que a região que eles contribuem para a corrente (19).

2.3.4 Parâmetros da célula fotovoltaica

Todas as características dos painéis solares devem ser analisadas utilizando as condições de teste padrão, isto é, irradiação solar de 1000 W/m^2, AM1.5 e temperatura constante em 25 °C (19), conforme a norma ASTM E927-10 (28).

A densidade de corrente de curto circuito, J_{SC} é dependente de propriedades óticas da célula fotovoltaica. No caso ideal é igual a J_{ph}. Em silício cristalino utilizando o espectro AM1.5 a máxima corrente possível é de 46 mA/cm^2, enquanto células comerciais superam 35 mA/cm^2.

Uma das características dos módulos solares é sua tensão de circuito aberto (V_{OC}), dada pela Equação 2.4, onde k_B é a constante de Boltzmann, T a temperatura e J_0 a densidade de corrente de saturação, que depende da taxa de recombinação (19).

$$V_{OC} = \frac{k_B T}{q} ln\left(\frac{J_{ph}}{J_0} + 1\right) \approx \frac{k_B T}{q} ln\left(\frac{J_{ph}}{J_0}\right) \tag{2.4}$$

O fator de preenchimento (FF) é a razão entre a potência gerada pelo módulo solar e o produto da tensão de circuito aberto e da corrente de curto circuito.

$$FF = \frac{P_{Max}}{J_{SC} V_{OC}} \tag{2.5}$$

A eficiência da conversão (η) é a razão entre a potência máxima gerada e a potência incidente, conforme os padrões de medição.

$$\eta = \frac{P_{max}}{I_{in}} = \frac{J_{SC} V_{OC} FF}{I_{in}} \tag{2.6}$$

Os parâmetros típicos de uma célula de cristal de silício são: $J_{SC} \approx 35$ mA/cm^2, V_{OC} até 0,65 V, FF entre 0,75 e 0,80. (19). Módulos comerciais como a Maxpower CS6U da Canadian Solar tem eficiência de até 17,49%[1].

2.3.5 Organização macroscópica das células fotovoltaicas

Cada célula fotovoltaica é capaz de gerar uma pequena quantidade de energia. Geralmente painéis comerciais são formados de 60, 72 ou 96 células, colocadas em série para aumentar a tensão de saída (19). Devido a deficiência no casamento das células individuais, que ocorrem por motivo de imperfeições na fabricação, por causa da soldagem das interconexões, que ocasiona aumento da resistência em série, ao EVA e vidro não serem totalmente transparentes, tipicamente 91% de transparência, a eficiência do modulo é inferior ao das células individuais. Tomando como exemplo, a folha de especificações do módulo Sanyo HIT-N240SE10, indica uma eficiência para as células de 21,6% e para o painel de 19,0% (19).

A geração de uma célula pode ser menor que as demais, seja devido a iluminação que não é necessariamente igual em todas ou a uma célula com defeito. É utilizado um diodo de *bypass* para evitar que estas células que geram menos que as demais passem a dissipar energia, dando origem a pontos quentes (*hotspots*). Isto poderia até mesmo danificar o módulo solar. Uma representação da aplicação deste diodo é mostrada na Figura 7.

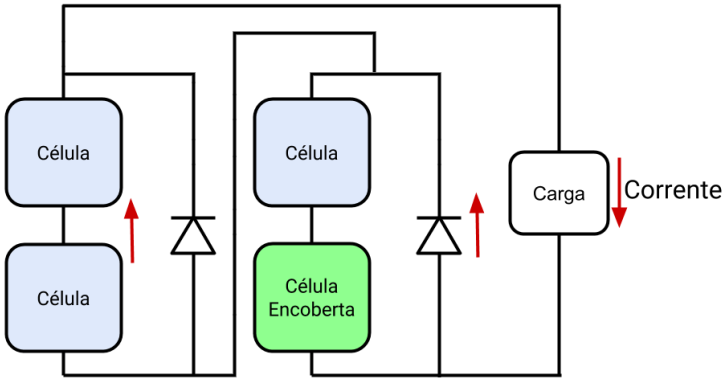


Figura 7 – Representação do diodo de *bypass*.

[1] <http://www.canadiansolar.com/fileadmin/user_upload/downloads/datasheets/v5.5/Canadian_Solar-Datasheet-MaxPower-CS6U-M-v5.51en.pdf>.

2.3.6 Curva I-V

Os painéis solares possuem uma curva característica de corrente por tensão, mostrada na Figura 8. Esta curva demonstra diversas características fundamentais dos módulos solares, como: corrente de curto circuito, onde os terminais do painel são curto circuitados e consequentemente a tensão de saída é zero, e tensão de circuito aberto, onde os terminais são deixados em aberto, não existindo corrente. O produto da tensão pela corrente da origem a curva de potência do painel, demonstrando, claramente, o único ponto ótimo de geração do painel (P_{max}), que maximiza a energia gerada.

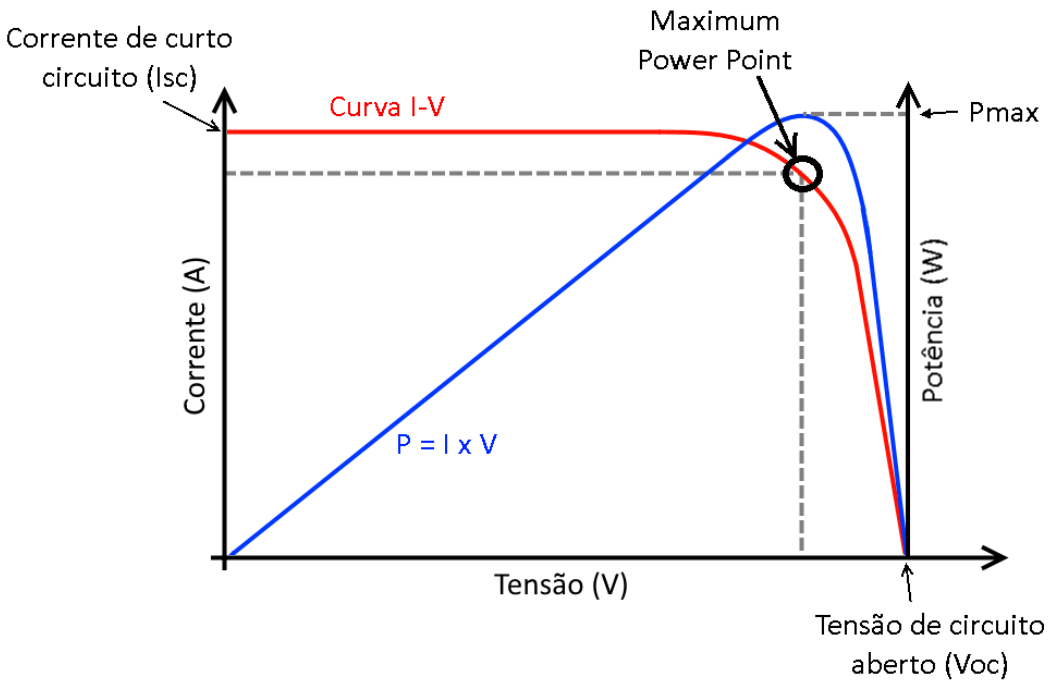


Figura 8 – Curva I-V típica de um módulo solar.

A curva se altera com mudanças na irradiância ou na temperatura. A irradiância influencia a corrente de curto circuito, que aumenta ou diminui conforme a incidência solar, no entanto, pouco altera a tensão de circuito aberto. Já a temperatura, por sua vez, influencia a tensão de circuito aberto, que diminui conforme a temperatura aumenta, consequentemente reduzindo também a geração de energia.

2.3.7 Métodos de MPPT

Como já visto, painéis solares possuem apenas um ponto ótimo de operação para maior geração de potência, que varia com a radiação solar, temperatura, sombreamento parcial, envelhecimento do painel, etc. De acordo com a teoria de circuitos elétricos, a resistência de fonte e carga devem ser iguais para maximizar o a transferência de potência. Portanto, o inversor deve procurar um ponto ótimo de operação utilizando um método chamado de MPPT (*Maximum Power Point Tracking*) (29).

Um dos métodos de MPPT mais utilizados é o de Perturbação e Observação (P&O), onde o *duty cicle* do conversor DC/DC (na entrada do inversor) tem uma pequena modificação, aumentando ou diminuindo a tensão de operação. A potência na saída é recalculada, caso a variação seja positiva a tensão de operação deve seguir a mesma direção, caso contrário, a tensão deve ser alterada para a direção oposta. Outros métodos são *Hill Climbing* (HC), *Incremental Conductance* (InCond), entre outros (30).

2.3.8 Análise das características de painéis solares

O dispositivo desenvolvido precisa operar em uma faixa de tensão e corrente a depender das condições de instalação e do modelo de painel utilizado. Para definir esta faixa de operação foram escolhidos painéis de diversas marcas disponíveis para venda no Brasil e suas especificações técnicas analisadas. As principais características são exibidas na Tabela 1. Tomando como base estas especificações, o dispositivo desenvolvido deve ser capaz de atender uma tensão de saída do painel de ao menos 25 V, utilizando uma fonte de custo menor, ou de 50 V, a depender do tipo de painel utilizado, onde o de 50 V é um dispositivo mais generalista, e uma corrente de 10 A para ter uma margem de segura de operação, evitando operar perto do limite da alimentação ou do sensor de corrente, onde um fico de tensão ou corrente poderia danificar o circuito.

Tabela 1 – Características de corrente e tensão de alguns modelos de painéis solares:

Painel	Potência (W)	Tensão Pot. Máx. (V)	Corrente Pot. Máx. (A)	Tensão C.A. (V)	Corrente C.C. (A)
Yingli JS150	150	18,50	8,12	22,90	8,61
Kyocera KD140SX	140	17,70	7,91	22,10	8,68
Komaes KMP150	150	18,28	8,21	21,90	8,93
Canadian Solar CS6K 260P	260	30,40	8,56	37,50	9,12
Canadian Solar CS6K 275P	275	31,00	8,88	38,00	9,45
UPSolar M150P	150	18,06	8,07	22,90	8,47
UPSolar M315P	315	36,50	8,63	46,20	8,90

2.4 Soluções comerciais de medição

Alguns inversores possuem a capacidade de medir a potência gerada, entre outras informações, e transmiti-las utilizando uma interface serial ou sem fio. A empresa alemã SMA Solar Technology AG desenvolveu um sistema com um concentrador chamado de Sunny WebBox, que é capaz de receber informações de até 50 inversores utilizando a tecnologia Bluetooth. Os dados são enviados para a nuvem, estando disponíveis através de um *login* e senha (31, 32). O sistema também permite exportar os dados no formato CSV para posterior análise em outros *softwares*.

Figura 9 – Sistema da empresa alemã SMA Solar WebBox.

Esta solução necessita do uso de equipamentos compatíveis (em geral de inversores da mesma marca de modelos específicos), para ser possível realizar a comunicação. Além disso, no exemplo de aplicação da solução, disponível em (31), cada andar ou setor da casa necessita de um repetidor Bluetooth para realizar a comunicação com os inversores no telhado, mesmo que estejam a apenas 2 andares de diferença.

Em 2017, foram identificadas diversas brechas de segurança que permitiriam um invasor entrar nas configurações destes sistemas. Como estes sistemas podem ser ligados à rede, em um país onde boa parte da geração advenha da solar, permitiria com um ataque coordenado derrubar a rede elétrica (33). Desta forma é de extrema importância se atentar as questões de segurança no desenvolvimento destes sistemas.

A empresa israelense SolarEdge[2] desenvolve equipamentos auxiliares para geração solar, desde inversores, soluções com bateria e monitoramento da geração. Ela se destaca pela sua solução que instala um conversor DC-DC em cada um dos painéis, aplicando o método de MPPT individualmente. Esta solução é mostrada na Figura 10. Outras fabricantes, no geral, utilizando um método de MPPT em vários painéis de uma vez, o que diminui a eficiência, já que existem variações que alteram o ponto ótimo de operação entre os painéis e a solução irá apenas encontrar o ponto ótimo para os vários painéis, não individualmente. A SolarEdge estima um aumento de produção fotovoltaica entre 2% e 25% quando utilizando sua solução.

A saída destes conversores é ligada em um barramento de corrente contínua e posteriormente em um inversor. A SolarEdge também possui uma solução de baterias para sistemas ligados a rede elétrica, chamado de StorEdge[4]. A solução pode fornecer energia a cargas pré-selecionadas em hipótese de um apagão. Além disso, em localidades onde existe diferenciação no custo da energia pela hora do consumo, a solução faz a otimização diminuindo a conta de luz.

[2] <www.solaredge.com>.
[3] <https://www.solaredge.com/products/pv-monitoring>.
[4] <www.solaredge.com/products/storedge>.

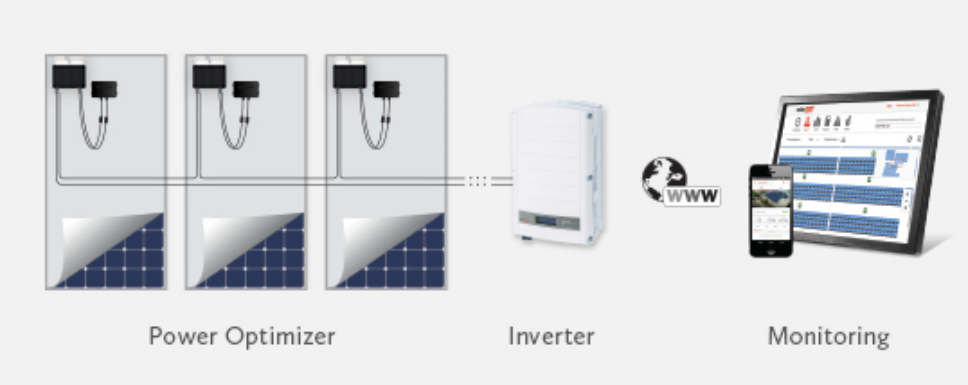


Figura 10 – Diagrama de instalação de um sistema da Solar Edge[3].

Sua solução de monitoramento faz o monitoramento de cada um dos conversores, desta maneira a geração individual de cada painel é mensurada. Ela possui aplicativos para plataformas móveis, como iPhones e Android. O sistema detecta falhas na geração e avisa o usuário automaticamente, possibilitando acesso em tempo real aos dados e análise da causa da falha.

A empresa alemã Meteocontrol desenvolve toda linha de sensores para geração solar, como para monitoramento de geração e falhas de *strings*, para medição de irradiância solar, temperatura ambiente e do módulo fotovoltaico. Seus sensores, ao contrário da solução da SolarEdge, fazem o monitoramento do sistema ao nível de *string*, não de painel solar, desta maneira eles não conseguem isolar eventuais problemas ao nível de painel. A vantagem é que eles podem ser instalados, segundo o fabricante, em inversores e *data loggers* de qualquer marca, não exigindo modelos específicos. A Meteocontrol também fabrica *data loggers*, que enviam ou não os dados para serviços em nuvem e funcionam com inversores de outras marcas.

2.5 Medição de geração solar em laboratórios

A medição de geração solar diretamente na saído do painel é efetuada utilizando um voltímetro e um amperímetro, a implementação destes instrumentos/sensores pode variar desde utilizar um resistor *shunt* para medição de corrente, um circuito integrado especializado para tal, que utilize, por exemplo, o efeito Hall.

O trabalho de Mostafa e Khan(34) apresentam um dispositivo para a medição da geração solar que exibe os resultados em um aplicativo no sistema operacional Microsoft Windows. É utilizado um resistor *shunt* de pequena resistência para transformar a corrente elétrica em tensão e um amplificador de instrumentação para amplificar esta tensão para uma posterior conversão via um conversor analógico digital. Este conversor é conectado a um microcontrolador que recebe um comando do computador a cada 5 segundos, enviando a leitura de tensão e corrente de forma assíncrona para o computador através de um

conversor da porta UART do microcontrolador, para uma RS232 emulada via USB com um *baud rate* de 19200 bps. Neste trabalho não existe nenhum tipo de conexão sem fio entre o microcontrolador e o computador, portanto os dados são enviados via cabo.

Em 2013, Kabalci, Gorgun e Kabalci(35) desenvolveram um sistema para monitoramento de energias renováveis. Seu protótipo foi utilizado para mensuração de geração de energias eólica e solar, que injetavam potência em um barramento de corrente contínua. Para medição da corrente foi utilizado um sensor da LEM, que possui um fundo de escala maior que o necessário, desta maneira foi necessário a utilização de um amplificador operacional para amplificar o sinal antes de enviá-lo para o conversor analógico digital do microcontrolador. O *software* de monitoramento foi desenvolvido no Microsoft Visual C# e se comunica diretamente com o microcontrolador.

Em 2014, Siregar e Soegiarto(36) desenvolveram um sistema para monitoramento de lâmpadas de postes alimentadas por energia solar. Durante o período com sol a bateria é carregada e em momentos de necessidade descarregada para manter a iluminação ligada. O sistema faz a leitura da potência enviada ou recebida pela bateria e a gerada pelo painel solar, desta maneira, descontada as perdas do controlador, é possível aferir também o consumo da lâmpada. As informações são enviadas para uma central utilizando SMSs enviados por uma conexão GSM.

Em 2016, Erasmus e Bagula(37) desenvolveram um sistema para mensuração da tensão e corrente em baterias utilizadas em sistemas *off-grid*, desta maneira também estimando a quantidade de energia gerada. Eles desenvolveram um aparato de medição, uma página web para visualização dos dados, além de enviar uma mensagem SMS para avisar que a carga da bateria está acabando.

Em 2017, Dhole et al.(38) desenvolveu um sistema para medição de geração solar, temperatura e intensidade solar. Para a medida de tensão foi utilizado um divisor resistivo, para corrente novamente um resistor shunt, temperatura um sensor LM35 e intensidade de luz um Resistor Dependente de Luz (do inglês LDR, *Light Dependent Resistor*) utilizando um conversor analógico digital ligado a um microcontrolador PIC16F877A. O sistema dispõe de um LCD de 16x2, além de uma conexão serial de 2,4 GHz para enviar dados ao computador via texto. Este trabalho apesar de contar com uma ligação sem fio, não é apresentado nada relativo a uma rede de sensores, apenas o *hardware* básico necessário para realizar a medição, também sendo importante destacar que o sistema é alimentado por uma fonte, não pela energia do próprio painel.

Em 2017, Patil, Vijayalashmi e Tapaskar(39) desenvolveram um sistema para medição de geração solar e envio das informações para a nuvem. A medição de tensão fez uso de um divisor resistivo para diminuir a tensão na saída do painel e realizar a leitura utilizando o conversor analógico digital. A medição de corrente, por sua vez, utilizou um sensor ACS712, que possui uma sensibilidade de 66 mV/A, ocasionando uma baixa reso-

lução de corrente, já que o sinal não foi amplificado. As leituras foram enviadas para o serviço em nuvem Thingspeak através de uma placa de desenvolvimento Raspberry Pi.

2.6 Resoluções normativas da ANEEL

A partir de 2010, a ANEEL (Agência Nacional de Energia Elétrica) realizou consultas públicas para a formulação da chamada Resolução Normativa Nº 482, publicada em 17 de abril de 2012 (8), que passou a vigorar 9 meses após a sua edição. Esta resolução estabelece as condições gerais para conexão de unidades microgeração e minigeração distribuídas ao sistema de distribuição da concessionária local, além de medidas de compensação para o caso de excedente de geração. Uma nova resolução em 2015, após novas consultas públicas, foi editada, a Nº 687, que alterou alguns itens da anterior (40).

Os principais pontos que a Resolução Nº 482, já atualizados com as mudanças da Nº 687, são:

- É considerada microgeração, centrais que tenham potência instalada menor ou igual a 75 kW e minigeração até 5 MW para fontes renováveis não hídricas;
- Foi instituído um sistema de compensação, onde a geração excedente pode ser emprestada gratuitamente a concessionária e posteriormente compensada com o consumo em até 60 meses;
- Regulamenta a compensação de créditos em condomínios, desde que não utilize vias públicas ou propriedades de terceiros;
- Permite a criação de consórcios ou cooperativas, dentro de uma mesma área de concessão ou permissão, para compensação entre os membros;
- Permite que um consumidor pessoa física ou jurídica atendido por uma mesma concessionária com unidade de microgeração ou minigeração obtenha créditos em um local e utilize a compensação para outra unidade;
- Na conta deve ser cobrado do consumidor o consumo mínimo do tipo de sua conexão;
- A distribuidora é a responsável tecnicamente e financeiramente pelo sistema de medição, mas o consumidor é responsável pelos custos de adequação do medidor normal para o medidor para geração distribuída;
- A partir de 2017 a distribuidora deve ter um sistema eletrônico para o consumidor fazer a requisição para se tornar um microgerador ou minigerador.

Esta resolução foi o primeiro passo para incentivar a microgeração de energia elétrica no Brasil. Ainda existia a questão do Imposto Sobre Circulação De Mercadorias

E Serviços (ICMS), que era cobrado duas vezes, uma quando a energia era enviada para a concessionária e outra quando a energia era utilizada pelo consumidor. O Conselho Nacional de Política Fazendária (CONFAZ) do Ministério da Fazenda autorizou a partir de 2015, que vários estados parassem com essa dupla cobrança, com cada vez mais estados entrando na lista. Atualmente os únicos estados que ainda possuem a dupla cobrança são: Santa Catarina, Paraná, Espírito Santo e Amazonas (41).

2.7 Computação em nuvem

Um dos paradigmas atuais é a migração da computação em servidores locais para servidores na nuvem, ou seja, mantidos por uma empresa que presta este serviço para uma infinidade de outras empresas e consumidores. A vantagem nesta abordagem é que não é necessário manter servidores ociosos localmente para atender a uma demanda momentânea. Este mesmo serviço na nuvem pode escalar para um número maior de servidores em momentos de necessidade, como em eventos especiais, como numa grande promoção de vendas, onde a loja diminui o preço dos produtos aumentando o número de acessos simultâneos ao site.

Grandes empresas como a Amazon, com o AWS (Amazon Web Services)[5], IBM, com o Bluemix[6], Microsoft, com o Azure[7], e Google Cloud[8] possuem serviços para hospedar máquinas virtuais em seus servidores na nuvem. Dentre estes serviços, alguns possuem funções focadas em IoT, no entanto, no geral são pagos e oferecem apenas seu uso por um curto período.

Dois serviços menores se destacam quando se pensa em IoT. O primeiro é o Thingspeak [9], que possui uma relação muito próxima com a Mathworks, empresa que desenvolve o MATLAB, permitindo até a análise posterior dos dados enviados à nuvem no MATLAB. O segundo sistema é o Ubidots[10], uma plataforma que possui uma interface de fácil uso para usuários finais, com aplicativo de visualização de dados para *smartphones*. Diversos projetos de IoT fazem uso destas plataformas, dentre eles:

- Kumar et al.(42) apresenta um sistema de monitoramento de pacientes. Este sistema mede a temperatura, pulso, posição corporal e faz um eletrocardiograma (utilizando um circuito integrado da Analog Devices AD8232) e envia estes dados para perfis individualizados na plataforma Thingspeak utilizando uma porta ethernet ou um rádio com microcontrolador ESP8266;

[5] <https://aws.amazon.com>.
[6] <https://www.ibm.com/cloud/>.
[7] <https://azure.microsoft.com>.
[8] <https://cloud.google.com/>.
[9] <https://thingspeak.com/>.
[10] <https://ubidots.com>.

- Chandra et al.(43) desenvolveram um sistema de monitoramento de geradores de energia à diesel. Neste sistema é medido o nível de combustível, utilizando um sensor ultrassônico, e a tensão e frequência de saída do gerador. Para envio das informações ao Thingspeak foram testados uma porta ethernet e conexão GSM à internet;

- Rios, Romero e Molina(44) apresentam um sistema de controle de motores que monitora a velocidade de rotação do motor e permite a atuação pela internet utilizando o site Ubidots;

- Escobar e Salinas(45) desenvolveram um sistema de monitoramento de pulso, utilizando uma placa LinkIt ONE adaptada para medir a pulsação cardíaca e analisar se a pessoa está tendo arritmia. Caso positivo os dados do batimento e informação do GPS presente na placa são enviados do serviço Ubidots;

- O trabalho de Bolivar e Silva(46) apresenta um sistema para medição da radiação solar. Ele faz uso de um sensor TAOS TCS-230 conectado a um microcontrolador PIC, que envia as informações via I2C para um Raspberry Pi. Estas informações são, por sua vez, enviadas via protocolo UDP para um computador executando LabVIEW, que também é responsável por fazer o envio ao serviço Ubidots.

2.8 Conclusões do capítulo

Neste capítulo é apresentada uma revisão bibliográfica de temas relacionados a rede de sensores sem fio. Posteriormente feita uma análise do panorama energético mundial, demonstrando a viabilidade da geração solar, já que se apenas uma pequena fração da incidência solar na terra fosse utilizado para geração de energia, ela poderia suprir todas as necessidades humanas.

Alguns equipamentos comerciais foram analisados, como painéis, reguladores de carga e inversores, que serão importantes para a definição das características de corrente e tensão requeridas para nosso sistema. Consequentemente limitando a escolha de possíveis fontes de alimentação e sensores de corrente.

Sistemas comerciais como os vendidos pela empresa alemã SMA Solar Technology AG e a americana SolarEdge são comentados. Eles implementam interfaces de visualização da geração e exportação de dados parecidas como as que serão desenvolvidas ao longo do projeto. A solução da SolarEdge implementa um conversor DC-DC que funciona como método de MPPT na saída de cada painel solar, desta maneira aumentando a eficiência do sistema.

A resolução normativa formulada em 2012 pela ANEEL, atualizada em 2015, que regulamentou a micro e mini geração solar, venda e compensação de energia com a con-

cessionaria, foi um grande incentivo para a instalação residencial de geração solar, que resultou num expressivo aumento nos últimos anos da geração fotovoltaica no Brasil.

3 Hardware do Sistema

Neste capítulo são detalhados os componentes de *hardware* usados na implementação dos protótipos. Inicia com uma introdução à tecnologia e posteriormente são discutidos os critérios técnicos utilizados em sua escolha.

O custo dos componentes para sistemas embarcados vem diminuindo acentuadamente, possibilitando o projeto e venda de produtos que a poucos anos atrás seriam totalmente inviáveis do ponto de vista econômico. Chris Anderson apresenta um interessante conceito (47): os grandes investimentos feitos pela indústria dos *Smartphones* em pesquisa e desenvolvimento de produtos de baixo custo, trouxeram para outras áreas de produtos componentes de baixo custo e de fácil alcance, seja no mercado de processadores, memórias, tecnologias de redes sem fio e baterias. Ele chama isto de *"the peace dividend of the smartphone wars"*. Logo, o dinheiro gasto em P&D para um tipo de produto em específico trouxe inovação para um imenso número de outras indústrias.

Desta maneira temos uma infinidade de opções de componentes e *softwares* para uso no projeto. É importante ressaltar que, apesar das escolhas de projeto serem detalhadas individualmente, como é um problema de engenharia, elas estão fortemente acopladas. A escolha de um padrão de rede sem fio pode ser dependente da placa de desenvolvimento oferecer suporte este padrão, ter as conexões de entrada e saída necessários, do sistema operacional a ser utilizado ter os módulos (*drivers*) requeridos para sua utilização. Portanto, a escolha de um componente pode ter influência em outros não diretamente relacionados.

Durante a escolha e execução do projeto, diversos novos *hardwares*, padrões de mercado e *softwares* foram lançados ou anunciados para posterior lançamento. Por este motivo, possivelmente um *hardware* ou *software* utilizado durante o desenvolvimento do projeto, no momento da apresentação pode não ser mais a versão mais recente ou a melhor opção para o projeto, que no momento da escolha o era.

Aqui, vamos apresentar e discutir os componentes de *hardware* e/ou *software* normalmente utilizados e seus critérios de escolha, de maneira que mesmo que não sejam utilizados no trabalho, possam ser considerados para outros trabalhos futuros em projetos envolvendo IoT. O uso de microcontroladores, reguladores de tensão, sensores, projeto de placas, é comum a sistemas embarcados, assim as soluções aqui analisadas também podem ser empregadas.

3.1 Placas de desenvolvimento, SoC e Microcontroladores

Um celular, um aparelho de TV, um micro-ondas ou mesmo um satélite têm em comum uma arquitetura parecida. Todos contam com um processador principal, que pode ser um SoC ou um microcontrolador, memória de acesso aleatório (RAM), para execução dos programas, memória de gravação permanente (ROM), que contém os programas que não são apagados após desligamentos, e os periféricos conectados a interfaces de expansão.

Um SoC (*System on a Chip*) é um circuito integrado que integra vários componentes de um computador, além de possíveis funções como entradas/saídas digitais e analógicas, de rádio frequência, tudo isso em apenas um único encapsulamento, que pode ou não conter mais de um *die*, capaz de executar sistemas operacionais complexos como Microsoft Windows e Linux, com exceção de memória que geralmente é utilizada em circuitos separados. Um microcontrolador também é constituído por diversos componentes em apenas um encapsulamento, como memória RAM, memória ROM, entradas e saídas, conversores digitais analógicos e analógicos digitais, no entanto, geralmente contando com poucos kB de memória e processadores com capacidade muito limitada.

A maioria dos microprocessadores utiliza a arquitetura Von Neumann, onde o programa a ser executado e os dados são armazenados na mesma memória, no caso a RAM. A arquitetura conhecida como Harvard armazena o programa em uma memória permanente (ROM) e os dados em outra memória (RAM) (48). No geral, mesmo na arquitetura Von Neumann, o programa não é modificado em tempo de execução, por incorrer em demasiada complexidade e problemas de segurança. Como o conteúdo da RAM é perdido com o desligamento do computador, na Von Neumann ele é copiado de outro meio como um disco rígido ou memória Flash para a RAM e então o processador passa a executar o programa. Repare na diferença: na Harvard ele é executado diretamente na ROM, enquanto na Von Neumann ele é copiado para a RAM antes. A ROM geralmente é uma memória Flash ou EEPROM, que tem uma quantidade limitada de gravações e pode ser necessário condições especiais para gravação, como uma maior tensão de alimentação, portanto limitando a capacidade de atualizar o *Firmware* em tempo de execução. Arquiteturas de microcontroladores como Atmel AVR e Microchip PIC fazem uso da arquitetura Harvard (48).

Existem dois conceitos de instruções para processadores, a chamada de RISC (*Reduced Instruction Set Computing*, que possui um conjunto reduzido de instruções, e a CISC (Complex Instruction Set Computing) um conjunto que possui instruções mais complexas, além de maior quantidade de instruções. Enquanto o CISC foca em adicionar instruções complexas, que demoram mais para serem executadas, porém realizando mais trabalho, o RISC foca em instruções mais simples, sendo executadas em menor tempo, porém necessitando de mais de uma instrução para fazer algumas operações que apenas uma CISC é capaz. Considere-se que um conjunto RISC tem uma complexidade menor

e consequentemente um consumo de energia inferior, mas é evidente que outros fatores influenciam no consumo de energia, como o processo de fabricação do circuito.

No geral, arquiteturas RISC tem algumas poucas instruções para acesso a memória e as demais instruções manipulam os dados já nos registradores, enquanto as instruções CISC podem fazer as operações com os dados ainda em memória, tendo, portanto, a execução de algumas instruções com ciclo consideravelmente maior do que outras, devido a latência elevada das memórias. Outra diferença é que quando se trabalha com periféricos (como um relógio RTC, sensores, etc) no RISC estes são mapeados em endereços de memória, acessados com instruções comuns para acesso a memória. No caso do CISC o processador, pode ter instruções especiais para tratar estes periféricos (48).

A escolha por um SoC ou um microcontrolador dependerá da aplicação, onde uma mais complexa que exija um sistema operacional completo deverá utilizar uma placa de desenvolvido com um SoC, caso do concentrador da rede de sensores sem fio. Já uma aplicação mais simples, que podem ou não necessitar de um sistema operacional exigindo menor capacidade de processamento, menor consumo de energia e baixo custo, deverá utilizar um microcontrolador, caso dos nós sensores.

3.1.1 Interface de comunicação GPIO e protocolos UART, SPI e I^2C.

Os dispositivos conectados ao SoC/Microcontrolador podem utilizar entradas e saídas, chamadas de GPIO ou, caso forem mais complexos, devem utilizar um protocolo de comunicação como UART, SPI ou I^2C.

GPIO (*General-Purpose Input/Output*) são simplesmente entradas ou saídas com características elétricas pré-determinadas pelo fabricante do sistema. Considerando um sistema arbitrário, que opere com 3,3 V, entradas entre 0 e 1 V podem ser consideradas nível lógico baixo, entre 2,5 e 3,3 V nível lógico alto e maiores que 1 V e menores que 2,5 V indeterminado (esses limiares variam conforme o dispositivo e fabricante).

O protocolo UART (*Universal Asynchronous Receiver/Transmitter*) é um protocolo assíncrono que permite grande flexibilidade, com diferentes taxas de transmissão, tensões de operação, operar em modo *simplex* (em apenas uma direção), *full duplex* (nas duas direções ao mesmo tempo) e *half duplex* (nas duas direções, uma por vez).

3.1.1.1 SPI

O SPI é um protocolo de barramento síncrono, com comunicação *full duplex*, utilizando uma arquitetura *master-slave*. Existe um dispositivo mestre (apenas ele pode iniciar a comunicação), sendo geralmente o microcontrolador, e um número pré-definido máximo de escravos. São utilizados no mínimo 4 conexões (49):

- SCK: *Clock* gerado pelo mestre;
- MOSI: Dados enviados pelo mestre para o escravo;
- MISO: Dados enviados pelo escravo para o mestre;
- SS: Seleciona o escravo.

Cada escravo deve estar conectado a uma saída do SS do mestre, logo a quantidade de conexões é 3 mais o número de escravos. As portas utilizam conexão *Push-Pull*, ou seja, existe um transistor entre a linha e o terra e outro ligando na alimentação. Desta maneira, quando se quer aterra a linha, apenas um dos transistores é ativado, da mesma maneira que para colocar a linha em estado lógico alto.

3.1.1.2 I^2C

O protocolo I^2C foi desenvolvido pela Philips, atual NXP. É um barramento *half duplex*, que utiliza uma arquitetura *master-slave*. Ele pode ter um número arbitrário de mestres e até 127 escravos (desde que os endereços dos periféricos não sejam coincidentes e as características do circuito permitam), utilizando apenas duas conexões físicas (50):

- SCL: *Clock* gerado pelo mestre;
- SDA: os dados a serem transmitidos.

Todos os dispositivos escutam a linha SCL e apenas quando são endereçados passam a se comunicar com o mestre. Se existirem dois ou mais mestres em um mesmo barramento, procedimentos de arbitragem são empregados para evitar conflitos (50).

A conexão das portas utiliza um circuito chamado de *Open-Drain*, ou seja, existe um transistor entre o terra e a linha e um resistor de *pull-up* entre a linha e a alimentação. Assim, quando o transistor está desligado, a linha está em estado alto; quando o transistor é ativado a linha é colocada em estado lógico baixo. Repare que quando a linha está aterrada, o resistor de *pull-up* está conduzindo corrente, coisa que não ocorre em *Push-Pull*, pois o transistor que liga a linha com a alimentação não estará conduzindo.

3.2 Placa de desenvolvimento para o concentrador

O concentrador será a base do sistema, responsável por iniciar a rede a ser utilizada, receber e totalizar os dados os armazenando e enviando-os para internet, hospedando um servidor responsável por responder requisições de dados do navegador, etc. É essencial que esse concentrador tenha um baixo consumo de energia, pois estará em operação no

mínimo durante toda operação do sistema, idealmente deverá estar ligado 24 horas por dia.

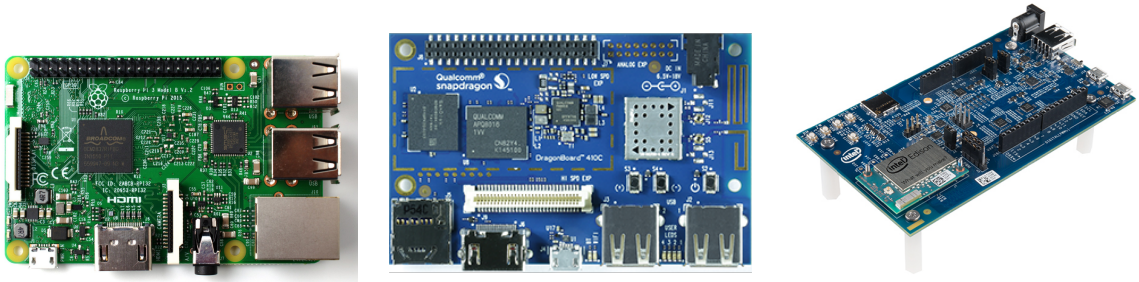
Um fator essencial para a escolha de um SoC ou mesmo microcontrolador, é a existência de uma placa de desenvolvimento. No geral, a primeira versão dos protótipos de um dispositivo embarcado é construída em uma bancada de testes, a existência de uma placa de desenvolvimento ajuda nesta montagem, não sendo necessários o projeto e montagem de uma placa apenas para demonstrar o conceito. Também ajuda a iniciar o desenvolvimento de *software* em paralelo com o *hardware*. Outro ponto é que quando *software* e *hardware* são desenvolvidos ao mesmo tempo para um dispositivo, não se é possível ter certeza em qual dos dois eventuais falhas estão. Com uma placa de desenvolvimento é possível testar o *software* nela, garantindo, ou ao menos diminuindo muito, que a falha seja de *hardware*. Não é incomum que produtos com desenvolvimento mais amador sejam montados e enviados para o consumidor final utilizando a placa de desenvolvimento.

Muitas empresas como a Broadcom, Microchip, NXP, Qualcomm, Texas Instruments oferecem diversas placas de desenvolvimento. No geral possuem um custo elevado, mas com o passar do tempo e demanda dos laboratórios de pesquisa e de projetistas amadores, diversos projetos apareceram, como o Raspberry Pi. O foco de seu lançamento em 2011 era promover o ensino de computação e eletrônica em escolas utilizando um *hardware* de baixo custo (51). Diversos outros produtos similares foram lançados ou se tornaram mais conhecidos devido à grande cobertura da mídia do Raspberry Pi, tais quais: Banana Pi[1], Orange Pi[2], ODROID[3], BeagleBone Black[4], C.H.I.P.[5], dentre outros. O Raspberry Pi passou a ser utilizado como plataforma de desenvolvimento de aplicações IoT, praticamente criando um mercado.

Para nossa aplicação serão avaliadas 3 placas, o Raspberry Pi[6], Qualcomm Dragonboard 410c[7] e Intel Edison[8], exibidas na Figura 11. Na Tabela 2 é feita uma comparação quantitativa e posteriormente analisada para escolha de qual delas será utilizada.

Todas são placas contam com a capacidade de processamento necessária, o Raspberry Pi e a Qualcomm Dragonboard utilizam processadores de arquitetura ARM Cortex-A53 com 1,2 GHz, de arquitetura RISC, enquanto a placa Intel Edison utiliza um x86 com 500 MHz, de arquitetura CISC. Todas as placas contam com 1 GB de RAM, quantidade suficiente para execução de um sistema operacional completo.

[1] <http://www.bananapi.org/>.
[2] <http://www.orangepi.org/>.
[3] <http://www.hardkernel.com/main/products/prdt_info.php>.
[4] <https://beagleboard.org/black>.
[5] <https://getchip.com/>.
[6] <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf>.
[7] <https://www.arrow.com/en/products/dragonboard410c/arrow-development-tools>.
[8] <http://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison_pb_331179002.pdf>.



(a) Raspberry Pi 3 (b) Qualcomm Dragonboard (c) Intel Edison com Arduino Uno Compatible Board

Figura 11 – Placas consideradas para o concentrador.

A placa da Intel não possui um sistema de vídeo, sendo assim ela deve ser utilizada exclusivamente pela rede, não sendo possível utilizá-la com um monitor ou uma tela. O Raspberry Pi tem, além da saída HDMI, uma DSI para conexão de telas, surgindo a possibilidade de o concentrador contar com uma interface *touchscreen* para o usuário final.

Em questão de acesso a redes sem fio todas as placas são igualmente capazes, suportado os padrões IEEE 802.11 B, G e N e Bluetooth 4.0 no caso da Intel e 4.1 nas demais. Apenas o Raspberry conta com uma porta de rede RJ45, comumente utilizada para conexão à internet com fio. Em relação a interfaces de comunicação para conexões a periféricos, na aplicação é demandada apenas a capacidade de conexão ao padrão de rede escolhido. Como todas contam com ao menos uma GPIO, UART, SPI e I^2C, todas com suas particularidades são igualmente capazes para a aplicação.

Na questão de custo, o Raspberry Pi custa metade do preço dos outros dois, é facilmente encontrado para compra no Brasil, não necessitando pagar os caros custos de transporte de encomendas internacionais. Além disso, a Element14, umas das fabricantes das placas, tem um serviço de customização, permitindo a remoção ou adição de interfaces de comunicação, potencialmente reduzindo ainda mais o custo em versões para fabricação em larga escala[9]. Todas as placas são capazes de executar o sistema operacional Linux. O Raspberry e a placa da Qualcomm também rodam o Windows 10 IoT.

Principalmente pela questão do baixo custo e potencial customização, a placa de desenvolvimento escolhida para a aplicação foi o Raspberry Pi 3. Ao final do projeto, depois da placa já comprada, ela foi lançada oficialmente no Brasil, com certificação da Anatel, pelo preço de R$ 199,90[10].

[9] <https://www.element14.com/community/docs/DOC-76955/l/raspberry-pi-customization-service>.

[10] <https://www.filipeflop.com/produto/raspberry-pi-3-model-b/>.

Tabela 2 – Comparação entre placas de desenvolvimento.

Placa	Raspberry Pi 3	Qualcomm Dragonboard 410c	Intel Edison usando Arduino Uno Compatible Board
Processador/SOC	4x ARM Cortex-A53 (BCM2837)	4x ARM Cortex-A53 (Snapdragon 410)	2x Atom e Intel Quark
Arquitetura	ARMv8-A	ARMv8-A	Intel x86
Frequência	1,2 GHz	1,2 GHz	Atom 500 MHz/ Quark 100MHz
GPU	VideoCore IV	Adreno 306	N/A
RAM	1 GB	1 GB	1 GB
ROM	Somente Cartão SD	8 GB eMMC/Cartão SD	4 GB eMMC/Cartão SD
Wi-Fi 802.11	Padrão B/G/N	Padrão B/G/N	Padrão B/G/N
Bluetooth	4.1	4.1	4.0
RJ45	Sim	Não	Não
USB	4	4	1
GPIO	24	12	11
UART	1	2	1
SPI	2	1	1
I^2C	1	4	2
Sistema Operacional	Linux e Windows 10 IoT	Linux, Android, Windows 10 IoT e Google Brillo	Linux e Google Brillo
Preço (US$)	40	75	80

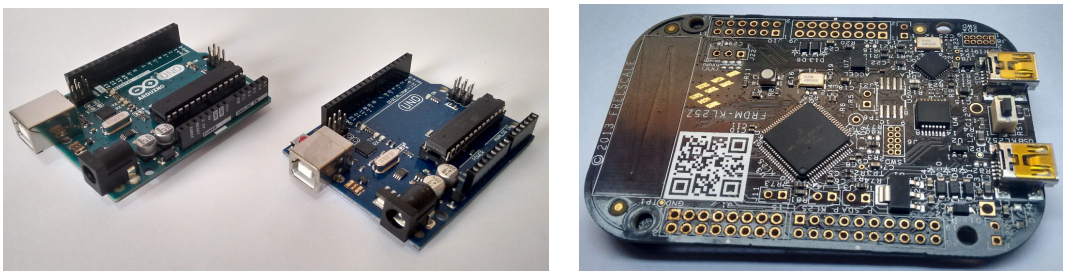
3.3 Placa de desenvolvimento para os nós sensores

Uma placa de desenvolvimento para os nós, devem possuir microcontrolador, memória RAM e ROM (onde o *firmware* reside), pinos de entrada e saída. O *firmware* é escrito pelo usuário e realizará operações previamente definidas conforme a aplicação. Consideramos principalmente dois fatores para a escolha do *hardware*: o valor, pois todo nosso instrumento deve custar no máximo uma pequena fração do preço de um painel fotovoltaico; e ser *hardware* aberto, ou seja, ter os esquemáticos para replicação disponíveis.

Existem diversas arquiteturas de microcontroladores no mercado. Talvez o nome mais conhecido seja a linha PIC da Microchip, também os ATmega da Atmel, empresa adquirida pela Microchip em 2016 (durante o desenvolvimento do projeto). Microcontroladores com a arquitetura ARM, mais poderosos, estão se tornando comuns, utilizando núcleos da série Cortex-M, partindo do simples M0 até o M4, que progressivamente permi-

tem usos que demandem maior capacidade de processamento, alguns possuindo instruções de DSP. Outros microcontroladores comuns são a série Intel 8051, que atualmente são fabricados por diversas companhias, e a linha MSP430, da Texas Instruments, dentre muitos outros.

Placas Arduino geralmente possuem microcontroladores ATmega ou ARM, enquanto a NXP Freedom Board possui um ARM M0+. Elas são mostradas na Figura 12, se enquadram nestes dois requisitos, possuem baixo custo e seus esquemáticos estão disponíveis. Isto permite construir um protótipo utilizando-as como uma placa de desenvolvimento e, através de *shield*[11], fazer a comunicação com transceptor responsável pela conexão à rede sem fio. Posteriormente, poderia ser criada uma placa, apenas com os componentes realmente necessários, como o microcontrolador ATmega328P (52) ou o Kinetis KL25 (53), no caso da Freedomboard, um *socket* para encaixar o módulo de comunicação, integrando todos os componentes necessários para um futuro produto.



(a) Arduinos (b) NXP FRDM-KL25Z

Figura 12 – Placas consideradas para os nós.

O Arduino conta com o *software* homônimo para escrita do *firmware*, enquanto na Freedomboard pode-se usar o *software* KDS[12] ou a plataforma web da ARM chamada mbed[13]. A principal vantagem do da Freedom Board é o suporte à *breaking points*[14]. As principais características de ambas as placas são mostradas na Tabela 3.

O modelo escolhido para nosso protótipo foi uma placa equivalente ao Arduino Uno R3, que é uma das várias alternativas fabricadas por outras empresas, que utilizam os mesmos componentes, devido ao esquemático aberto, com o conceito de *Open Hardware*[15]. A versão fabricada na Itália sai por R$ 120,00. O Arduino também é um pouco menor, o que facilita no projeto da caixa.

[11] Placa intermediária responsável pela conexão entre a placa de desenvolvimento e outra placa a ser utilizada como uma para comunicação, fazendo as alterações de tensões e outras adaptações necessárias.

[12] <www.nxp.com/kds>.

[13] <https://developer.mbed.org/>.

[14] Durante a programação é possível criar pontos de interrupção no código fonte, durante a execução o programa é parado nestes pontos. O desenvolvedor do *firmware* pode ver o estado de cada uma das variáveis e registradores da CPU, facilitando o *debug* de possíveis problemas.

[15] Um conceito parecido com o de *softwares* de código aberto (*Open Source*), onde os esquemáticos e *layouts* de placa estão disponível para qualquer um replicar e modificar.

Tabela 3 – Características Arduino Uno R3 e da NXP FRDM-KL25Z.

Microcontrolador	ATmega328P (52)	Kinetis KL25
Tensão de operação	5 V	5 V
Tensão de entrada	7-12 V	4,3-9 V
Pinos de entrada e saída digitais	14 (6 permitem PWM)	52 (6 permitem PWM)
Pinos de entrada analógicas	6	6
Memória Flash	32 KB	128 KB
SRAM	2 KB	16 KB
Frequência de operação do microcontrolador	16 MHz	48 MHz
Comprimento	68.6 mm	81 mm
Largura	53.4 mm	53 mm
Preço	R$ 60,00	R$ 149,90

3.4 Rede sem fio

Comunicação sem fio vem sendo utilizada desde a última década do século 19, com o telegrafo sem fio de Marconi (54). O uso difundido do rádio no começo do século 20, culminando no SCR-536, considerado o primeiro *Walkie Talkie*, utilizado durante a Segunda Guerra pelos aliados (55). Em 1983, a Motorola lança o DynaTAC, o primeiro celular (56), dando início a comunicação sem fio para a população em geral, que ficou conhecida como 1G (primeira geração).

As tecnologias de comunicação evoluíram com as gerações a seguir. Com 2G, lançado em 1991, os sinais de rádio passaram a ser digitais, permitindo serviços de dados como mensagens de texto e multimídia. O 3G, em 2001, além de aumentar as taxas de transmissão, seu objetivo foi permitir ligações em vídeo e acesso à internet (57).

Padrões com usos específicos foram lançados ao longo dos anos, como: IEEE 802.11, utilizado para acesso a internet sem fio residencial e empresarial, Bluetooth para troca de arquivos entre dispositivos a curtas distâncias e transmissão de áudio, NFC para comunicações com tags RFID. Padrões com foco em sistemas embarcados, automação residencial como: IEEE 802.15.4 (ZigBee, Thread), Digimesh, LoRa, panStamp e Thread. Todos estes padrões foram considerados para a aplicação. A seguir analisaremos, justificando nossa escolha pelo XBee Zigbee.

Dispositivos comerciais como o Google OnHub, mostrado na Figura 13, suportam os padrões IEEE 802.11, Bluetooth e, futuramente com uma atualização de *software*, 802.15.4 (Thread)[16].

[16] <https://on.google.com/hub/>.
[17] <https://on.google.com/hub/static/images/marquee_new.jpg>.

Figura 13 – Modelos do Google OnHub.[17]

3.4.1 Dimensionamento da aplicação

A escolha de uma solução de rede sem fio está diretamente relacionada ao tamanho do local onde será utilizada. Como, no geral, painéis solares são instalados no telhado de casas ou de empreendimentos comerciais/industriais, sendo esta aplicação o foco do dispositivo. Como referência de uma possível área de aplicação do dispositivo precisamos escolher um local onde ele poderia ser instalado para definição do alcance da rede. Para isso escolhemos o Hospital de Clinicas da Unicamp, que é mostrado na Figura 14. Sua escolha se justifica pela grande área com que

Como seu tamanho é de aproximadamente 170 m por 125 m, é necessária uma tecnologia capaz de cobrir 105 m, supondo a estação no centro do hospital e cobertura radial.

3.4.2 Padrão de mercado e tecnologias proprietárias

Algumas empresas desenvolvem produtos baseados em um padrão de mercado, ou seja, existe uma associação que desenvolve um padrão e várias empresas colocam produtos no mercado que são compatíveis entre si. Caso um não seja tecnicamente viável, tenha custo restritivo para a aplicação ou a empresa que o desenvolve perca o interesse no mercado, pode-se usar alternativas. Um ótimo exemplo disso são roteadores de internet com o padrão IEEE 802.11, fornecidos com *chipsets* de diversas empresas, tais como: Broadcom, Intel, Qualcomm, Realtek, Texas Instruments[19].

Algumas tecnologias são proprietárias e apenas uma empresa vende produtos para tal aplicação. Caso a empresa saia do mercado ou mesmo diminua investimentos, o usuário poderá ficar sem opções ou com uma tecnologia inferior ao padrão aberto, devendo fazer uma custosa migração. Portanto, considerar padrões abertos é importante, apesar de não primordial. Um exemplo de caso onde foi necessário mudar todo plano de negócios de

[18] Google Earth: <https://earth.google.com/>.
[19] <https://wikidevi.com/wiki/List_of_Wi-Fi_Chipset_Vendors>.

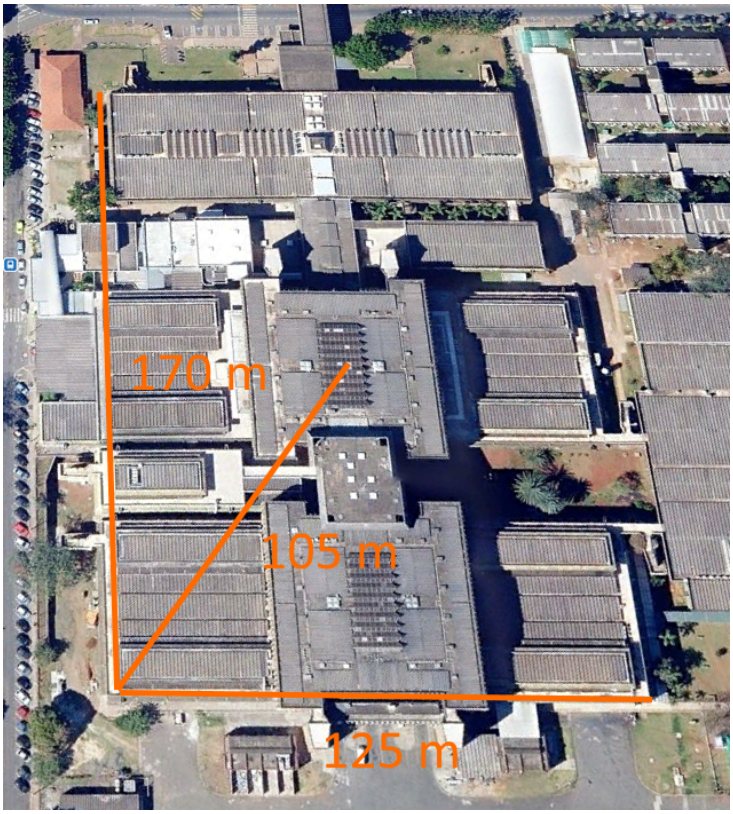


Figura 14 – Vista aérea do Hospital de Clinicas da Unicamp.[18]

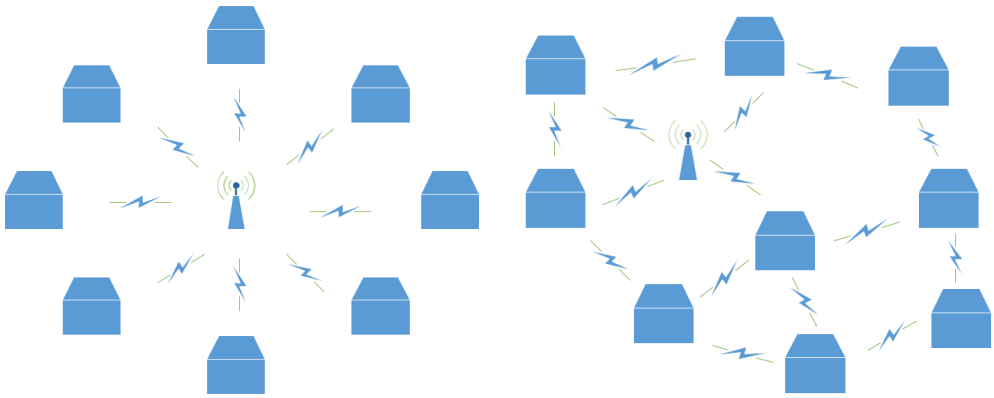
uma empresa devido a saída do mercado da fabricante de soluções sem fio da tecnologia foi o caso da Nextel. A tecnologia iDEN para transmissão rádio para telefonia celular, desenvolvida pela Motorola e atualmente utilizada no Brasil pela Nextel vai ser desligada em março de 2018, dentre os motivos alegados está a impossibilidade da compra de novos dispositivos para a tecnologia[20].

3.4.3 Topologia de rede em estrela ou malha

Em redes, são comuns dois tipos de arquiteturas de rede: a estrela, onde todos os nós se conectam a apenas ao concentrador, e redes de malha (*mesh*), onde os próprios nós podem ser roteadores de pacotes para os demais nós. As Figuras 15a e 15b demostram as topologias estrela e *mesh*, respectivamente.

Como necessitamos de uma cobertura de, ao menos, 105 metros, em uma rede no formato estrela, a comunicação entre a antena central e os nós deve ser capaz de cobrir tal distância diretamente. Uma rede *mesh*, devido a capacidade de os pacotes serem reenviados pelos nós intermediários, pode ter uma distância de comunicação direta entre os nós menor, porém utilizando os nós para roteamento, é possível atingir tal distância. A desvantagem da rede *mesh* é a maior complexidade no gerenciamento da rede, o que ocasi-

[20] <https://www.telegeography.com/products/commsupdate/articles/2017/10/26/nextel-to-shut-down-brazilian-iden-network-in-march-2018/>.



(a) Topologia Estrela (b) Topologia Malha

Figura 15 – Topologias de rede.

ona overhead, já que mais dados devem ser adicionados a transmissão para endereçamento devido a topologia da rede.

3.4.4 IEEE 802.11 e Bluetooth

O padrão IEEE 802.11 e o Bluetooth utilizam a topologia estrela e não são capazes de distâncias da ordem de 100 metros, portanto devem ser logo desconsideradas para nossa aplicação. O IEEE 802.11 é um padrão utilizado em Computadores, *Smartphones* e *SmartTVs* para acesso à internet. Utiliza principalmente a frequência de 2,4 GHz e é capaz de velocidades de mais de 150 Mbps, porém com consumo de energia proibitivo para nossa aplicação. O Bluetooth além do baixo alcance, menor que 10 metros, permite um número limitado de nós, o que inviabilizaria um local com muitos painéis.

3.4.5 IEEE 802.15.4

É um padrão do IEEE (*Institute of Electrical and Electronics Engineers*) publicado em 2003, tem taxa de transmissão entre 20 e 250 kbps, para frequências de 868 MHz, 915 MHz e 2,4 GHz, respectivamente, modos de endereçamento de 16 e 64 bits, suporte a dispositivos com baixa latência. É a base dos padrões ZigBee e Thread[21].

3.4.6 ZigBee

ZigBee é um padrão de mercado implementado por diversas empresas, tais quais: Microchip, Texas Instruments, NXP, entre outras. Segue o padrão IEEE 802.15.4 em relação a frequências e taxas de transmissão. Suas principais características são:

- Capacidade de até 65000 nós;

[21] <http://www.ieee802.org/15/pub/TG4.html>.

- Topologias estrela, ponto a ponto e *mesh*;
- Criptografia AES de 128 bits na comunicação.

Na Figura 16 é mostrado a topologia de uma rede ZigBee. Existem 3 tipos de dispositivo em uma rede ZigBee, coordenador, roteador e dispositivo final (58).

- Coordenador: Apenas um por rede, responsável pelos protocolos de inicialização, definição de canais, chaves de criptografia, etc;
- Roteador: Capaz receber pacotes e encaminhar para outros nós;
- Dispositivo final: apenas se conecta a roteadores e ao coordenador. É capaz de entrar em modo de economia de energia e receber os pacotes endereçados a ele que ficaram no buffer dos outros nós posteriormente.

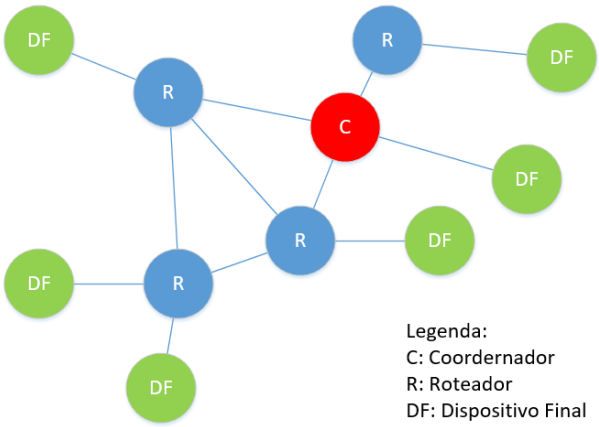


Figura 16 – Topologia de uma rede ZigBee Mesh.

Utilizando o conceito de uma rede em malha é possível cobrir sem grandes problemas uma distância de 100 metros, considerando que individualmente cada nó tem um alcance variando de 60 metros para ambientes com paredes e até 3,2 Km considerando a versão Pro com visada direta[22].

Produtos comerciais como as lâmpadas Philips Hue utilizam Zigbee para a comunicação[23].

3.4.7 Thread

É um protocolo baseado no proprietário utilizado pela Nest, empresa comprada em 2014 pelo Google, que vem sendo desenvolvido com suporte de outras empresas, tais

[22] <http://www.digi.com/products/xbee-rf-solutions/rf-modules/xbee-zigbee#specifications>.
[23] <http://www.zigbee.org/what-is-zigbee/connectedlighting/>.

quais Samsung e ARM, para se tornar um padrão de mercado para IoT. Assim como o ZigBee é baseado no protocolo IEEE 802.15.4, suportando redes *mesh* com até 250 nós, com a adição de endereçamento por IPv6[24].

No início do projeto ainda era uma solução em desenvolvimento, sem opções comerciais, portanto ainda não aplicável para nosso sistema.

3.4.8 Digimesh

É um protocolo proprietário desenvolvido pela Digi International, que é a única empresa a fornecer rádios para esta tecnologia, suas características são muito parecidas com o ZigBee, porém a construção da rede é mais simples, já que todos os nós são iguais. Na figura 17 é demonstrada essa característica. Suas principais vantagens são[25]:

- Permite colocar em modo de baixo consumo todos os nós;
- Expandir a rede é fácil, pois não existe diferenciação entre os nós;
- Alcance teórico de até 40 milhas, utilizando o rádio de maior capacidade da Digi.

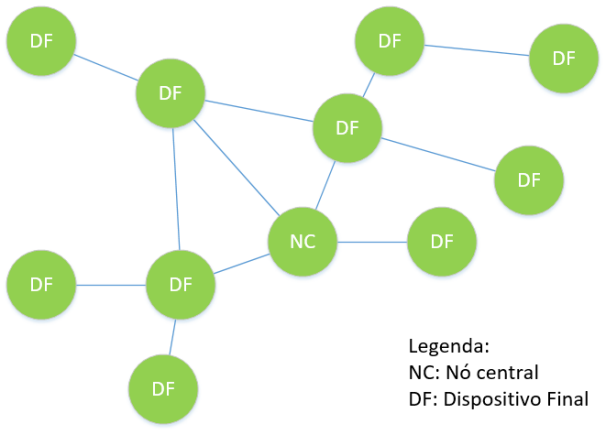


Figura 17 – Esquema de uma rede Digimesh.

3.4.9 LoRa

Seu nome deriva de *Long Range* (Longo Alcance), uma técnica e modulação de espalhamento espectral. É uma tecnologia patenteada pela Semtech, com rádios também desenvolvidos pela Microchip. É geralmente utilizada em conjunto com LoRaWAN, um protocolo de comunicação com topologia estrela. Possui grande alcance, de 3 a 4 km em áreas urbanas e até 12 km em rurais.

[24] <https://developers.nest.com/documentation/weave/thread-wifi>.
[25] <http://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf>.

Atualmente o LoRaWAN não é uma boa opção para aplicações comerciais, pois todos os canais utilizam as mesmas frequências, ocorrendo, portanto, problemas de colisões de pacotes. Um *framework* está em desenvolvimento para permitir aplicações privadas de redes LoRaWAN em redes públicas[26].

Atualmente, LoRa seria ideal para uma aplicação onde existiria apenas um servidor na região e todos os nós conversariam apenas com este servidor. Em nosso sensor, poderia ocasionar problemas caso em um mesmo bairro diversas pessoas o aplicassem, utilizando cada casa um servidor para contabilizar os dados. Uma aplicação interessante seria em *Smartgrids*, onde os dados de consumo de cada casa seriam totalizados por uma antena central da companhia de energia.

3.4.10 panStamp

O panStamp é um padrão totalmente aberto, cujos esquemáticos e códigos fontes estão disponíveis. É desenvolvido por uma empresa espanhola de mesmo nome, que utiliza o rádio CC1101 da Texas Instruments. Se baseia em um protocolo de rede próprio chamado SWAP, para utilização sensores e atuadores[27].

A versão NVR 2, conta com o mesmo microcontrolador utilizado em algumas versões do Arduino, sendo compatível com sua interface de programação. A depender da versão, é possível alcançar mais de 2 km de distância em ambientes abertos com visada direta. Esta grande distância é o que possibilitaria nossa aplicação, pois não existe o suporte a redes no formato mesh[28]. Existem placas já montadas para serem utilizadas com sensores, como mostrada na Figura 18.

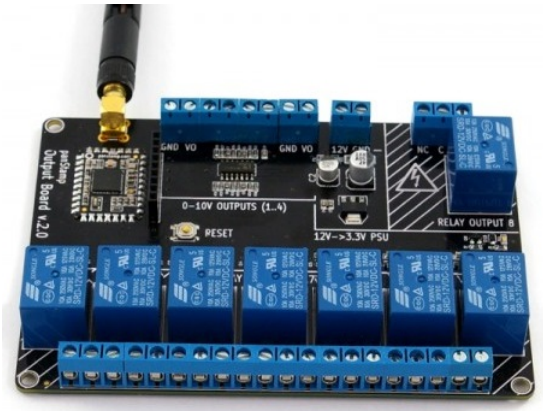


Figura 18 – Placa "output-board kit" da panStamp.[29]

[26] <http://www.link-labs.com/what-is-lorawan/>.
[27] <https://github.com/panStamp/panstamp/wiki/Technical-details>.
[28] <http://pt.slideshare.net/epokh1/panstamp-a-developer-introduction>.
[29] <http://panstamp.com/store/img/p/1/9/1/191-thickbox_default.jpg>.

3.4.11 Comparação entre soluções de rede

Tendo descartado o IEEE 802.11 e Bluetooth devido a limitações de distância, o Thread por ainda estar em desenvolvimento na época da escolha, sobraram 4 padrões a serem considerados: ZigBee, Digimesh, LoRa e panStamp, que são comparados na tabela 4.

Tabela 4 – Comparação entre opções de rede.

Atributo	XBee® ZigBee	Digimesh	LoRa	panStamp
Frequência	2.4 GHz, 900 e 868 MHz	2.4 GHz, 900 MHz	433, 868, 915 MHz	433, 868, 905, 915, 918 MHz
Taxa de transmissão máxima	250 kbps	250 kbps	38,4 kbps	115 kbps
Alcance máximo teórico	3.2 Km	1,6 Km	Mais que 15 km	200 m
Consumo em transmissão de dados	33 mA	50 mA	40 mA	36 mA
Suporte a redes *mesh*	Sim	Sim	Não (LoRaWAN)	Não
Padrão Aberto?	Sim	Não	Não	Sim

LoRa é um padrão que, apesar de satisfazer nossos requerimentos, não seria bem aplicado devido a limitações em se manter diversas redes na mesma região. O Digimesh é um padrão fechado, onde apenas uma empresa o suporta, sendo improvável que se torne um padrão de mercado.

O panStamp é uma alternativa interessante, porém a falta de suporte à topologia *mesh* pode dificultar sua aplicação em grandes áreas, pois ele tem um alcance de 200 metros com visada direta.

O ZigBee é baseado no padrão IEEE 802.15.4, que por sua vez, através de seus derivados, muito provavelmente se tornará o padrão de mercado de IoT de fato. Na época da escolha, foi considerado que a longo prazo poderia ser possível migrar para outra tecnologia baseada no padrão IEEE 802.15.4, como o Thread ou alguma outra tecnologia que se consolidar. Assim seria possível fazer a leitura de valores do próprio roteador da casa, sem a necessidade de um concentrador. Portanto a nossa escolha recaiu sobre o ZigBee, utilizando a implementação da XBee® ZigBee S2 e S2C, mostrado na Figura 19. É interessante notar que a nova versão da solução utilizada, o XBee® ZigBee S2D, permite a escolha entre os padrões Zigbee e Thread (59) ao se alterar o *firmware*, apesar de ainda não disponível no encapsulamento que utilizamos.

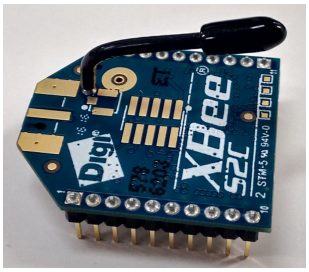


Figura 19 – XBee S2C ZigBee.

3.4.12 Conexão do XBee ao Raspberry Pi e Arduino

Na Figura 20 é mostrado o esquemático do *shield* do Arduino para conexão ao XBee. Este *shield* conta com um regulador de tensão MC33269 para suprir a alimentação do XBee, feita em 3,3 V. Na Figura 21 é mostrado o XBee já conectado ao Arduino utilizando o *shield*.

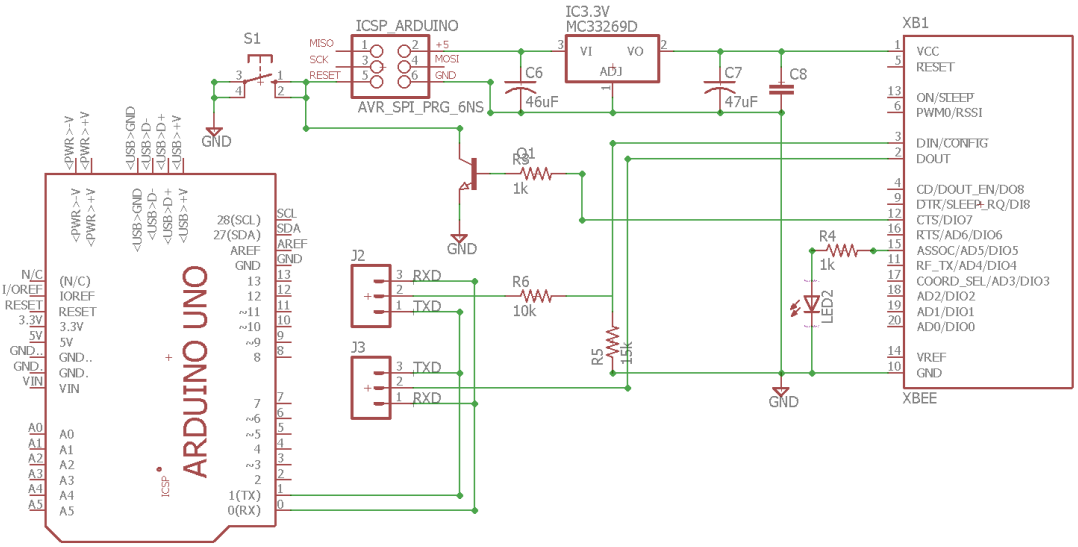


Figura 20 – Esquemático do *shield* do Arduino para XBee.

É importante ressaltar que o uso deste *shield* é por motivos de facilidade de implementação e de robustez dos protótipos. Seria possível desenvolver uma placa desta e fabricar na LPKF disponível da FEEC, ou mesmo mandar fabricar fora da universidade, como no OSH Park[30], mas esta não seria tão robusta ou teria um custo maior. Em um futuro produto, o XBee seria conectado na mesma placa onde estaria o microcontrolador e os sensores do nó.

Na Figura 22 é mostrado o XBee Explorer Stick v1.0, dispositivo requerido para conexão do XBee em um computador ou Raspberry Pi via porta USB. Este dispositivo utiliza um chip FT231XS, que converte a interface UART para USB, criando uma porta

[30] <https://oshpark.com/>.

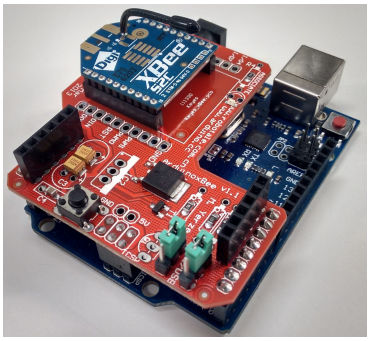


Figura 21 – Arduino utilizando o *shield* para conexão ao XBee.

serial virtual no sistema operacional para que aplicativos possam se comunicar com o XBee. Na Figura 23 é mostrado seu esquemático.

Figura 22 – Placa utilizada para conexão do XBee a computadores e ao Raspberry Pi.

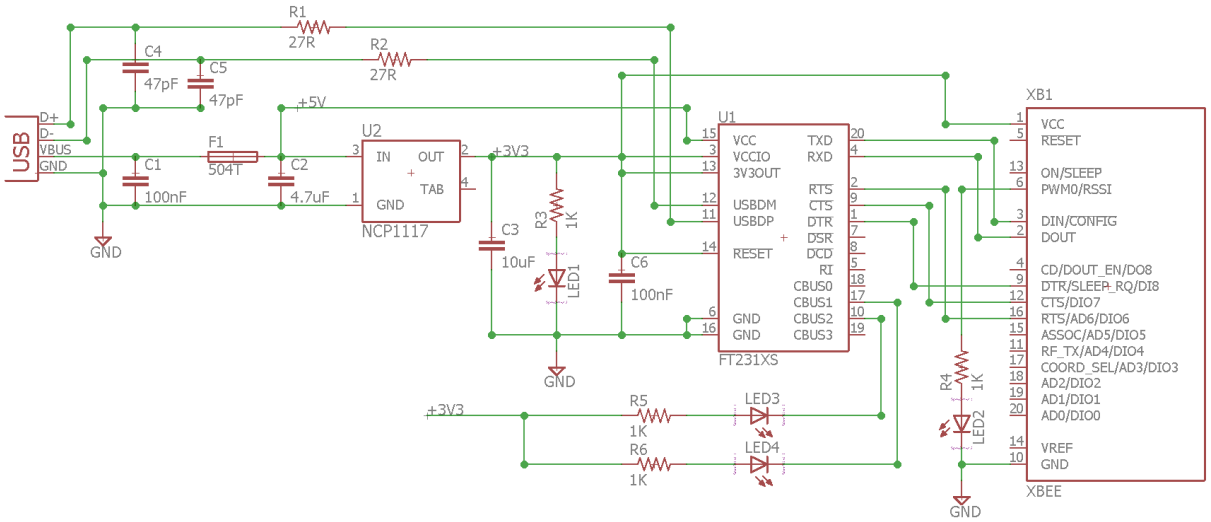


Figura 23 – Esquemático do XBee Explorer Stick v1.0.

3.5 Sistema Operacional

O sistema operacional é um programa, geralmente escrito em linguagens de baixo nível como Assembly e linguagem C, responsável por gerenciar os recursos de *hardware*. É responsável pela abstração do *hardware*, gerenciamento de memória, dos sistemas de arquivos, divisão do tempo de CPU entre processos. Sistemas operacionais usados em

computadores pessoais são geralmente o Microsoft Windows, Apple macOS e Linux e em servidores e computação de alto desempenho Linux, FreeBSD, entre outros.

Em nosso uso, necessitaremos de um sistema operacional para a base, que deverá ter um *hardware* mais poderoso. Avaliaremos a necessidade de um sistema para os nós, ou se rodará um programa mais simples, desenvolvido aqui na plataforma da placa de desenvolvimento escolhida.

3.5.1 Para o concentrador

3.5.1.1 Linux

É um *Kernel* (núcleo) utilizado como base para diversos sistemas operacionais, criado em 1991 por Linus Torvalds. É muito versátil, sendo utilizado em sistemas móveis tais como Android, Bada, Tizen, também para servidores, supercomputadores, desktop, terminais de atendimento, sistemas embarcados como televisões, geladeiras, etc.

O *Kernel* Linux é a base de diversas distribuições, tais quais Debian, Ubuntu, Fedora, entre muitas outras. A possibilidade de criar distribuições, com as necessidades requeridas, como a instalação apenas dos aplicativos necessários, baixo consumo de memória RAM, uso ou não de GUI, tornam o uso de Linux muito atrativo.

O Raspbian, por exemplo, é uma distribuição especifica para o Raspberry Pi, apesar disso faz uso de bibliotecas comuns do Linux. Logo um aplicativo desenvolvido para ele pode ser portado[31] sem grandes dificuldades para outras distribuições ou mesmo outra arquitetura de processador.

3.5.1.2 Windows 10 IoT

O Microsoft Windows é o sistema operacional para computadores mais utilizado no mundo. Historicamente seus programas eram desenvolvidos para uso exclusivo no próprio sistema para computadores pessoas. Atualmente sua desenvolvedora vem criando uma plataforma única, para todos os dispositivos que executam Windows, chamado de UWP (*Universal Windows Platform*), o aplicativo desenvolvido poderá ser executado em qualquer dispositivo Windows, como: computadores, *Smartphones*, dispositivos de IoT, console de *videogame*, etc[32].

Apenas recentemente foi lançado o Windows IoT, porém por ser mais recente e menos utilizado em sistemas embarcados, a plataforma ainda não é madura e nem possui muita documentação. Ela possui suporte a apenas 3 placas de desenvolvimento: Raspberry

[31] Convertido para outra distribuição ou arquitetura fazendo algumas pequenas mudanças no código fonte

[32] <https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide>.

Pi 2 e 3, Qualcomm Dragonboard 410c e Minnowboard Max. Mesmo nestas placas ainda existem problemas de compatibilidade, como o Bluetooth que ainda não era funcional no Raspberry Pi 3 na época da definição do sistema.

3.5.1.3 Android Things (Google Brillo)

O Android Things é um sistema operacional anunciado em 2015, para dispositivos IoT, que utiliza o *Kernel* Linux, pretendendo ser totalmente de código aberto após o lançamento. Diversas placas de desenvolvimento são capazes de executar o sistema em sua versão preliminar, como o Raspberry Pi 3 e a Intel Edison. Ainda é pouco documentado, porém no futuro pode ser um bom ponto de partida para aplicações IoT, já que conta com suporte do Google para o seu desenvolvimento.

No início do projeto era conhecido apenas como Google Brillo e estava disponível apenas para parceiros do Google, razão pela qual ainda não tínhamos acesso.

3.5.1.4 Comparação

A escolha foi pelo Raspbian devido ao suporte existente ao *hardware* do Raspberry Pi. O Windows 10 IoT ainda está em desenvolvimento e podem ocorrer problemas neste sentido. Além disso, a possibilidade de portar aplicativos escritos para outras placas de desenvolvimento no futuro, devido ao uso das mesmas bibliotecas no Linux, é fundamental para futuras revisões do projeto. O Android Things seria a melhor opção se já estivesse disponível publicamente no início do projeto.

3.5.2 Para os nós

A placa de desenvolvimento escolhida, o Arduino Uno R3, permite o uso de um sistema operacional, como o FreeRTOS, mas devido ao seu *hardware* limitado deve ser estudado se o seu uso trará ganhos. Como já explicitado, a escolha da necessidade ou não de um sistema operacional é acoplada a escolha da placa de desenvolvimento, sendo citados os sistemas avaliados. O operacional consome memória e tempo de processamento, desta maneira é necessário entender anteriormente se a aplicação necessita de um sistema e se o microcontrolador escolhido tem esta capacidade de memória.

Um sistema embarcado que é executado sem um sistema operacional consiste basicamente de uma rotina de inicialização, onde uma porção de código é executada apenas uma vez para colocar todos os dispositivos e sensores em um estado conhecido, depois o sistema entra em um *loop* de execução.

Existem duas opções no desenvolvimento deste *loop*, a primeira é utilizar uma instrução de *delay*, onde o *firmware* irá aguardar uma certa quantidade de tempo para executar novamente a ação. Por exemplo, é requerido a leitura de um sensor e a verificação

de uma condição a cada segundo, portanto é utilizado uma instrução para aguardar 1 segundo. A segunda opção pode fazer controlar diversos eventos, suponha que existam mais de uma ação a ser executada, por exemplo, a cada 5 segundos um sensor deve ser lido, a cada 7 outro sensor, a cada 30 executar uma ação de controle, a depender das leituras passadas dos sensores, e a cada minuto enviar dados para um concentrador. É possível utilizar um método onde é medido o tempo passado entre a execução passada e a atual, desta maneira o *firmware* sempre estará verificando se é hora de uma das ações ser executada.

Também é possível utilizar um sistema operacional, conhecido como RTOS (*Real-time operating system*) para implementar rotinas de controle mais complexas. A principal desvantagem é o maior consumo de recursos, mas, a depender da aplicação e da complexidade, pode ser necessário. É sempre necessário conhecer bem sua aplicação, o código pode demorar mais a executar, fazendo com que as outras rotinas do *firmware* atrasem, ou até mesmo causar um *stack buffer overflow*[33] caso o programa possua recursão. No caso de um RTOS, o sistema pode não fazer uma boa alocação do tempo para os diversos processos.

Aplicações cuja falha pode ser catastrófica, tais quais automotivas, detecção de incêndios e vazamentos de gazes tóxicos, por exemplo, devem seguir normas de programação que regem o desenvolvimento de *software*, como evitar o uso de variáveis globais, recursão no código, *looping* de espera, etc. Um padrão de regras de programação é a da MISRA (Motor Industry Software Reliability Association) (60), para uso em sistemas embarcados em carros. Outras áreas utilizam este padrão com poucas modificações. Um exemplo de falha em situações críticas é o caso da Toyota, onde o *firmware* de um sistema do carro pode ter causado aceleração não intencional de veículos da marca (61).

3.5.2.1 Zephyr

É um sistema recente, desenvolvido pela Wind River Systems, uma subsidiária da Intel especialista em sistemas embarcados. Desenvolveu o sistema operacional e protocolos do Curiosity, sonda da NASA *Jet Propulsion Laboratory* enviada para Marte em 2012.

Em fevereiro de 2016 seu código fonte foi liberado com a licença Apache 2.0 para a Linux Foundation. É capaz de ser executado em sistemas com 8 KB de RAM, suporte a Bluetooth, 802.15.4. Dentre suas principais características estão[34] [35]:

- Suporte às arquiteturas ARC, ARM e x86;
- Sistema de arquivos *Flash Filesystem*;

[33] Quando o número de variáveis é maior que o espaço em memória alocado para elas. Um motivo para a ocorrência deste erro, que pode travar a execução do programa, é o uso de recursão.

[34] <https://www.zephyrproject.org/content/zephyr-kernel-v100-release-notes>.

[35] <https://www.zephyrproject.org/content/zephyr-kernel-v150-release-notes>.

- Único espaço de endereçamento para todo o sistema, permite que o código de aplicativo e de *kernel* sejam executados no mesmo espaço de endereçamento.

Dentre as placas suportas estão[36]:

- Arduino 101;
- Galileo Gen1 e Gen2;
- Minnowboard Max;
- Arduino Due;
- NXP FRDM-K64F.

3.5.2.2 Contiki

É um sistema com foco em dispositivos sem fio de baixo consumo energético. Inicialmente desenvolvido por Adam Dunkels em 2003, enquanto aluno de ciências da computação da Universidade de Mälardalen na Suécia[37], e desde então por grandes empresais, como Texas Instruments, Atmel, Cisco, SAP, ST Microeletronics, entre outras[38].

Tem suporte a *multitasking*[39], ao protocolo TCP/IP, necessita de 10 KB de memória RAM e 30 KB de memória ROM, onde utilizando sua interface gráfica a necessidade de RAM sobe para 30 KB. Dentre os SoC suportados pelo sistema estão[40]:

- Atmel AVR;
- TI MSP430;
- Atmel Atmega128;
- Microchip pic32mx795f512l.

3.5.2.3 FreeRTOS

Dentre os sistemas apresentados, o FreeRTOS é o único capaz de ser executado no Arduino Uno[41], já que necessita de apenas 236 bytes de RAM para o escalonador e entre 5 e 10 KB de ROM. O FreeRTOS é um sistema de código aberto desenvolvido por Richard

[36] <https://wiki.zephyrproject.org/view/Supported_Boards>.
[37] <https://www.wired.com/2014/06/contiki/>.
[38] <http://www.contiki-os.org/>.
[39] Capacidade de execução de mais de uma aplicação ao mesmo tempo e o chaveamento da CPU para atender a todas
[40] <http://www.contiki-os.org/hardware.html>.
[41] <https://github.com/feilipu/Arduino_FreeRTOS_Library>.

Barry que utiliza uma licença GPL (a mesma do Linux) com algumas modificações. Já foi portado para muitos microcontroladores, como Atmel, Fujitsu, Intel 8052, PIC, Renesas, Texas Instruments, dentre outros.

Em seu site está disponível um guia de uso, explicando como o FreeRTOS funciona, sua arquitetura[42] e um manual de referência, com as configurações e funções disponíveis por meio de uma API[43].

3.6 Alimentação dos nós sensores

Como já discutido anteriormente, a tensão de saída do painel solar varia conforme o número de células em série, de modo que é muito variável em relação a marcas e modelos. Tomando como base os modelos analisados durante a revisão bibliográfica, consideramos que a tensão mínima que a solução deve oferecer suporte é de 30 V e a ideal de 60 V. Em busca de desenvolver um dispositivo versátil é importante que sua alimentação seja flexível para ser conectada a uma variada gama de tensões, desta maneira respeitando os limites mínimos.

Para isso foram analisadas diversas opções de conversores para transformar o alto valor de tensão da saída, como já discutido, se situando entre 10 e 40 V, do painel para a pequena tensão requerida pelos componentes eletrônicos do nó, entre 3,3 e 5 V. Existem dois tipos principais de conversores, os lineares e os chaveados, popularmente chamados de *Buck* ou de *Step Down Converter* quando a aplicação é diminuir a tensão.

Os regulares lineares operam em uma faixa menor de tensão. Por exemplo, o muito usado regulador de 5 V LM7805 (62) é capaz de operar com uma faixa de entrada entre 6,5 e 25 V (a depender da versão utilizada).

$$P_{Dissipação} = (V_{Entrada} - V_{Saída}) \times I_{Saída} \tag{3.1}$$

A potência dissipada é proporcional a diferença das tensões de entrada e saída, logo aplicar um regulador linear para uma grande diferença de tensões, como no caso dos painéis solares, acarreta uma grande dissipação térmica e baixa eficiência (63).

Um conversor chaveado tipicamente tem a eficiência de 90%, muito superior aos lineares (64). As três principais desvantagens são a maior complexidade, resultando no uso maior de área de PCB, maior ruído na saída e as possíveis interferências eletromagnéticas nos circuitos próximos, as duas últimas devido ao chaveamento.

[42] <http://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf>.

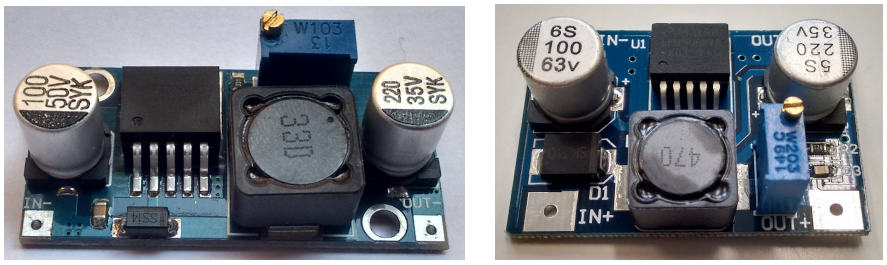
[43] <http://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V9.0.0.pdf>.

Existem circuitos integrados que possibilitam o uso de fontes chaveadas utilizando um baixo número de componentes, como o LM2576 e o LM2576HV da Texas Instruments, que utilizam apenas mais 4 componentes passivos (na versão com saída fixa), permitem entradas de até 40/60 V e saídas entre 1,23 V até 37/57 V (65), respectivamente. O LM2576 é facilmente encontrado no Brasil, enquanto o LM2576HV pode ser importando de sites chineses já montado em uma placa por R$ 10,00.

Outros componentes também poderiam ser utilizados, como a série R-78HBxx da RECOM[44], que possui apenas 3 pinos, não necessitando de nenhum componente externo, alimentação até 72 V, porém esbarrando no alto custo do componente, mais de US$ 11,00 em grandes quantidades na Digi-Key[45]. Outra opção seria o LM3485[46], também da Texas Instruments, sendo descartado devido ao limite de até 35 V da entrada

Foi utilizado o circuito integrado LM2596S, que permite entradas até 40 V em uma versão já integrada a uma placa com os demais componentes necessários. A versão do CI utilizada permite o ajuste da tensão de saída, desta maneira são adicionados mais componentes, como um potenciômetro, para efetuar esta regulagem. Na Figura 24a é mostrada a placa, seu esquemático, com exceção de valores de componentes é igual ao da versão de 60 V, mostrado na Figura 25.

Para a versão que permite maior tensão de entrada, foi importado o LM2576HV, que, como já citado, permite uma tensão de entrada de até 60 V. Ambas versões devem poder ser utilizadas em qualquer um dos protótipos. Uma foto da placa com o CI é mostrada na Figura 24b e o esquemático na Figura 25.



(a) Foto do regulador de tensão utilizado para tensões até 40 V.

(b) Foto do regulador de tensão utilizado para tensões até 60 V.

Figura 24 – Reguladores chaveados utilizados no projeto.

O Arduino tem duas entradas de alimentação, uma USB tipo B de 5 V e outra de 2,1 mm para tensões entre 7 e 12 V. Quando utilizando a entrada de 2,1 mm ele tem um regulador de tensão LM1117 para diminuir para 5 V utilizados pelo microcontrolador. Quando alimentado pelo USB existe um fusível para evitar sobrecorrentes.

[44] <https://www.recom-power.com/pdf/Innoline/R-78HBxx-0.5_L.pdf>.
[45] <https://www.digikey.com/products/en?mpart=R-78HB15-0.5&v=945>.
[46] <http://www.ti.com/product/LM3485>.

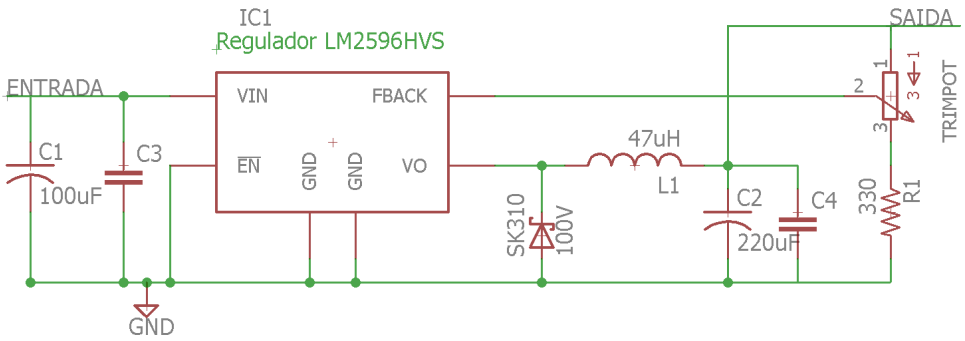


Figura 25 – Esquemático do regulador de tensão utilizado para tensões até 60 V.

Foi feita a escolha pelo regulador linear do Arduino, pois assim uma imprecisão na regulagem de tensão não ocasionaria danos ao microcontrolador e aos demais componentes alimentados por 5 V. Desta maneira também foi necessário utilizar um outro regulador linear para ligação do sensor de corrente e como alimentação do amplificador operacional. Foi utilizado o CI LM7805 (62), devido a disponibilidade e baixo custo, porém existem outras opções de reguladores chaveados, que possuem maior eficiência, dentre elas o OKI-78SR[47] e o 78xxSR da Murata[48], compatível com a pinagem do LM7805. Pode-se também utilizar o AP3211[49] da Diodes Incorporated, que utiliza componentes extras.

3.7 Medição de Tensões e Correntes

Existem dois tipos de sensores de corrente: os invasivos, cuja corrente passa pelo sensor, e os não invasivos, que ficam em volta do fio que conduz a corrente. Por uma questão de facilidade de instalação um sensor alicate seria a melhor escolha. No entanto surgem dois problemas: no geral possui uma baixa resolução e a dificuldade de encontrar uma opção que trabalhe com corrente contínua. O alicate YHDC 20A SCT-013 é mostrado na Figura 26.

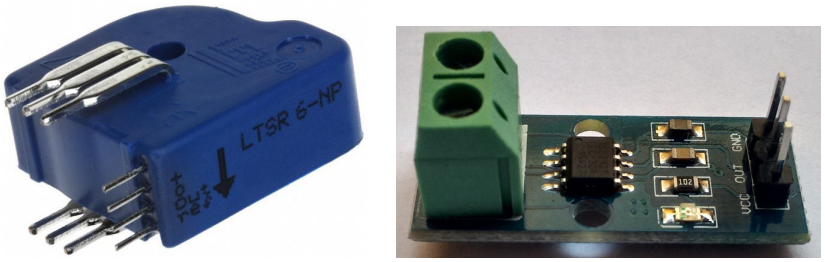
Figura 26 – Amperímetro Alicate YHDC 20A SCT-013.[50]

[47] <http://power.murata.com/data/power/oki-78sr.pdf>.
[48] <http://power.murata.com/data/power/dms-78xxsr.pdf>.
[49] <https://www.diodes.com/products/power-management/dc-dc-converters/integrated-power-stage/buck-converter/part/AP3211>.
[50] <http://s3.amazonaws.com/img.iluria.com/product/18B9DA/3A3A8A/850xN.jpg>.

A LEM é uma fabricante de transdutores de corrente. Seu modelo LTSR 6-NP[51], mostrado na Figura 27a, seria uma ótima opção para a aplicação. Utilizando um sensor hall com fundo de escala em corrente contínua até 19,2 A e é alimentado por 5 V. Seu problema é o alto custo unitário, de quase US$ 20,00 para compras de poucas unidades ou de US$ 12,00 para grandes quantidades.

A fabricante Allegro possui uma ampla gama de sensores hall para medida de corrente. Os modelos ACS722C e ACS725LC são sensores alimentados por 3,3 V e possuem versões unidirecionais, fundo de escala de 10 A e sensibilidade de 264 mV/A. A diferença entre ambos é que o ACS725 possui certificação automotiva. O ACS723LC e ACS724LC são alimentados por 5 V, possuem versões unidirecionais, fundo de escala de 10 A e sensibilidade de 400 mV/A, novamente a diferença é a certificação automotiva do ACS724LC.

No mercado nacional é possível encontrar o sensor Allegro ACS712[52] com fundo de escala de -30 A até 30 A, tanto para C.C. como C.A., com custo de aproximadamente R$ 10,00 do módulo já montado ou de importando apenas o componente de menos de US$ 2,00 em grandes quantidades. Sua resistência interna é de 1,2 mOhm (66), o que ocasiona uma baixa queda de tensão durante a medição. Ele é montado em uma placa discreta mostrada na Figura 27b e seu esquemático na Figura 28.



(a) Foto do sensor de corrente LEM LTSR 6-NP.[53]

(b) Foto do sensor de corrente utilizado.

Figura 27 – Sensores de corrente.

Sua sensibilidade é de aproximadamente 0,066 mV/A. Foi feita uma caracterização do sensor utilizando um amperímetro Agilent modelo 34401A, os dados são exibidos na Tabela 5 (nesta caracterização a sensibilidade foi de 0,0658 mV/A). Com o *software* SciDAVis[54] são traçados em um gráfico exibido na Figura 29 com uma reta criada utilizando regressão linear. Este resultado torna evidente a característica linear do sensor utilizado e sua confiabilidade para realizar a medição de corrente. A baixa sensibilidade será tratada utilizando um amplificador operacional.

[51] <http://www.lem.com/docs/products/ltsr%206-np.pdf>.
[52] <http://www.allegromicro.com/en/Products/Current-Sensor-ICs/Zero-To-Fifty-Amp-Integrated-Conductor-Sensor-ICs/ACS712.aspx>.
[53] <http://media.digikey.com/photos/LEM%20Photos/LTSR%206-NP.JPG>.
[54] <http://scidavis.sourceforge.net/>.

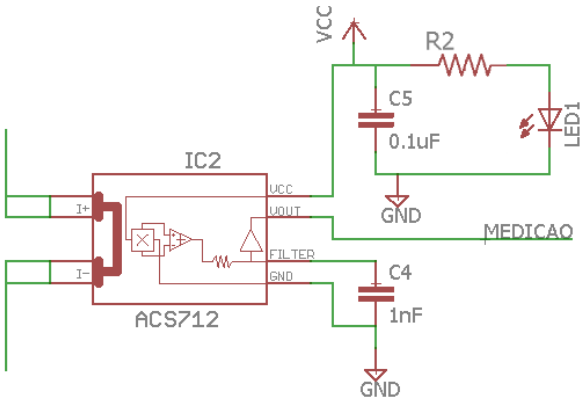


Figura 28 – Esquemático do sensor hall Allegro ACS712.

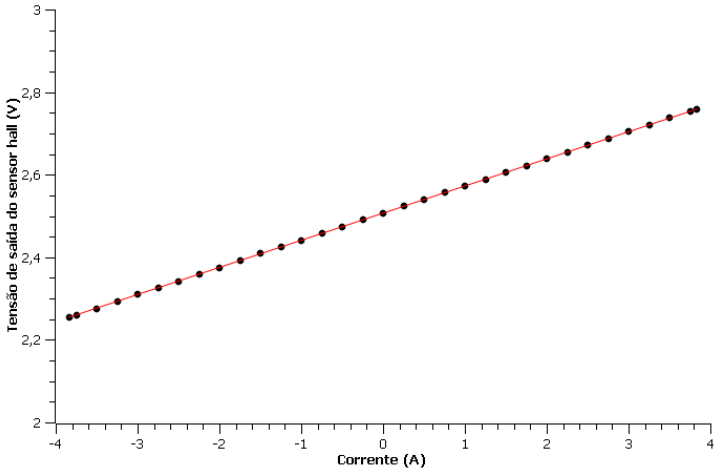


Figura 29 – Gráfico da corrente (A) pela tensão de saída do sensor (V).

3.8 Placa projetada

Diversos componentes utilizados, tais quais o conversor DC-DC, sensor hall de corrente, módulo de comunicação sem fio, Arduino, são componentes discretos que já possuem sua própria PCB. Portanto devem apenas fazer interface com a placa produzida, que é para fazer a leitura dos diversos sinais e tratá-los. É possível construir uma nova placa com os componentes selecionados, o que diminuiria a lista de materiais do sistema.

Existem vários programas para o projeto de placas de circuito impresso. Podemos destacar o Autodesk Eagle[55], um *software* muito utilizado para desenvolvimento amador de placas, já que possui uma versão gratuita com limitações de uso. Sua antiga desenvolvedora CadSoft foi comprada pela Autodesk em 2016. No geral, é um bom *software* para o layout de placas mais simples, mas que possui algumas limitações para o desenvolvimento de projetos mais complexos.

Um *software* gratuito e de código aberto que se destaca é o Kicad[56], atualmente

[55] <www.autodesk.com/products/eagle/overview>.
[56] <www.kicad-pcb.org>.

Tabela 5 – Resultados do ensaio do sensor de corrente.

Corrente (A)	Tensão (V)
-3,83	2,26
-3,75	2,26
-3,50	2,28
-3,25	2,29
-3,00	2,31
-2,75	2,33
-2,50	2,34
-2,25	2,36
-2,00	2,38
-1,75	2,39
-1,50	2,41
-1,25	2,43
-1,00	2,44
-0,75	2,46
-0,50	2,47
-0,25	2,49
0,00	2,51
0,25	2,53
0,50	2,54
0,75	2,56
1,00	2,57
1,25	2,59
1,50	2,61
1,75	2,62
2,00	2,64
2,25	2,66
2,50	2,67
2,75	2,69
3,00	2,71
3,25	2,72
3,50	2,74
3,75	2,75
3,83	2,76

é desenvolvido com apoio do CERN[57]. É uma ferramenta muito poderosa, que possibilita o desenvolvimento de placas complexas. Outro *software* muito utilizado é o Altium Designer[58], uma ferramenta profissional, com mais recursos que as outras duas citadas, mas com custo relativamente elevado. Então nossa escolha recai sobre o Kicad para o projeto de placas.

Esta placa é necessária para amplificar o sinal de tensão do sensor de corrente. O sensor de corrente utilizado possui uma sensibilidade de 66 mA/V, que é um valor muito baixo para ser posteriormente utilizado pelo conversor analógico digital do microcontro-

[57] <https://home.cern/about/updates/2015/02/kicad-software-gets-cern-treatment>.
[58] <http://www.altium.com/altium-designer/>.

lador ATmega. Isto ocorre devido ao fundo de escala do sensor ser de 30 A, muito maior que o necessário. Considerando que o conversor digital analógico do microcontrolador ATmega é de 10 bits e a alimentação padrão de 5 V, a resolução é aproximadamente 5 mV, consequentemente uma resolução de corrente de 75 mA, um valor que deve ser melhorado.

Existem outros componentes melhores ao projeto, como o Allegro ACS723LC, que possui alimentação em 5 V, fundo de escala de 10 A e sensibilidade de 400 mA/V. Utilizando o conversor digital analógico do ATmega, isto resultada em uma resolução de 12,5 mA, que seria um ótimo valor para o projeto. A possibilidade de outros fundos de escala viriam utilizando um componente da mesma série, com outra configuração.

Com o ACS712 é necessário utilizar um amplificador para melhorar a resolução das leituras, que não seria necessário utilizando os componentes mencionados acima. O fundo de escala deve diminuir o valor máximo de corrente a ser medido. Queremos aumentar a sensibilidade em 4 vezes, para 264 mA/V, o que resulta em uma resolução de corrente de 19 mA. Utilizamos uma configuração de amplificador inversor e o circuito é mostrado na Figura 30. Seu ganho dado pela relação de R4/R3, cujos valores utilizados foram de 3,9K e 1K, respectivamente.

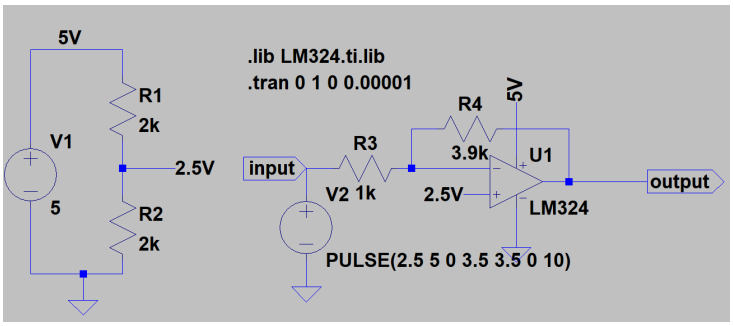


Figura 30 – Circuito amplificador utilizado para o sensor de corrente.

Fizemos uma simulação utilizando o *software* LTspiceXVII[59], com o modelo do amplificador LM324[60]. O resultado da simulação é mostrado na Figura 31. Como o amplificador não é *rail-to-rail* e ele não está sendo alimentado por uma fonte simétrica é de se esperar uma não linearidade maior quando próximo de zero. O circuito foi montado experimentalmente, resultando numa tensão de saída de aproximadamente 0,6 V. Desta maneira o circuito é capaz de operar entre 0 A e 7,2 A com resolução de 19 mA. Poderia ser obtido um melhor resultado utilizando um amplificador *rail-to-rail*.

Foi projetada uma placa para fazer este tratamento, cujo esquemático é mostrado na Figura 32, seu layout na Figura 33a e a placa já com os componentes soldados na Figura 33b.

[59] Disponível em: <http://www.linear.com/designtools/software/>.
[60] <https://github.com/pepaslabs/LTSpice-parts/blob/master/parts/op%20amp/LM324.ti.lib>.

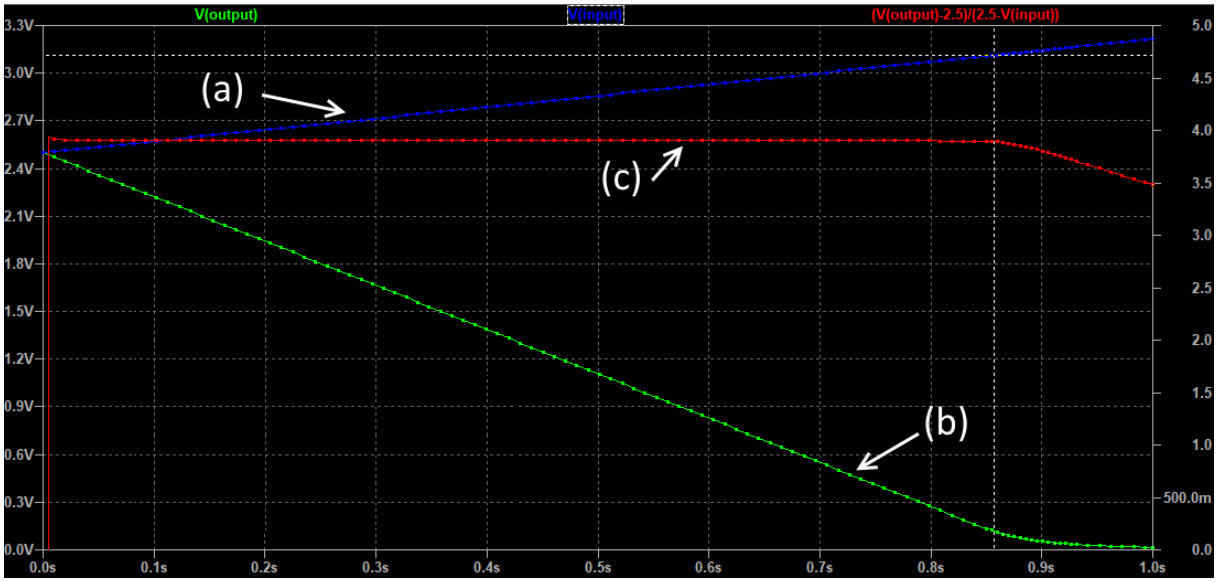


Figura 31 – Resultados da simulação do circuito amplificador do sensor de corrente. (a) Tensão de entrada do amplificador, (b) Tensão de saída do amplificador e (c) Ganho do amplificador.

3.8.1 Caixa fabricada utilizando uma impressora 3D

Para a instalação do sistema em ambientes externos é necessário que ele esteja devidamente acomodado em uma caixa fechada e vedada contra as intempéries do tempo. Como forma de estudo de uma eventual versão comercial foi projetada uma caixa para uso com o protótipo. Esta caixa não foi vedada, mas procurou-se evitar ao máximo aberturas, existindo apenas nos conectores para os cabos.

É muito comum na indústria e atualmente na academia o uso de impressoras 3D para a fabricação de protótipos, com a finalidade de validação do projeto e posterior fabricação. Seu custo diminuiu muito nos últimos anos com a entrada no mercado de diversos fabricantes, como a *startup* Makerbot[61], comprada em 2013 pela Stratasys[62]. O Brasil possui várias empresas que vendem impressoras 3D, como a Cliever[63], Metamáquina[64] e Sethi3D[65], uma empresa de Campinas. Esta última é a fornecedora do equipamento Sethi3D AiP A3 usado para impressão, mostrado na Figura 34.

Para fazer o projeto da caixa é necessário um *software* de CAD paramétrico. Entre os diversos *softwares* possíveis estão o FreeCAD[66], o único *software* gratuito e de código aberto disponível, possui uma interface mais complexa que as demais soluções, além de apresentar alguns travamentos nos testes de uso efetuados.

[61] <www.makerbot.com>.
[62] https://www.forbes.com/sites/kellyclay/2013/06/19/3d-printing-company-makerbot-acquired-in-604-million-deal/
[63] <www.cliever.com>.
[64] <www.metamaquina.com.br>.
[65] <www.sethi3d.com.br>.
[66] <www.freecadweb.org>.

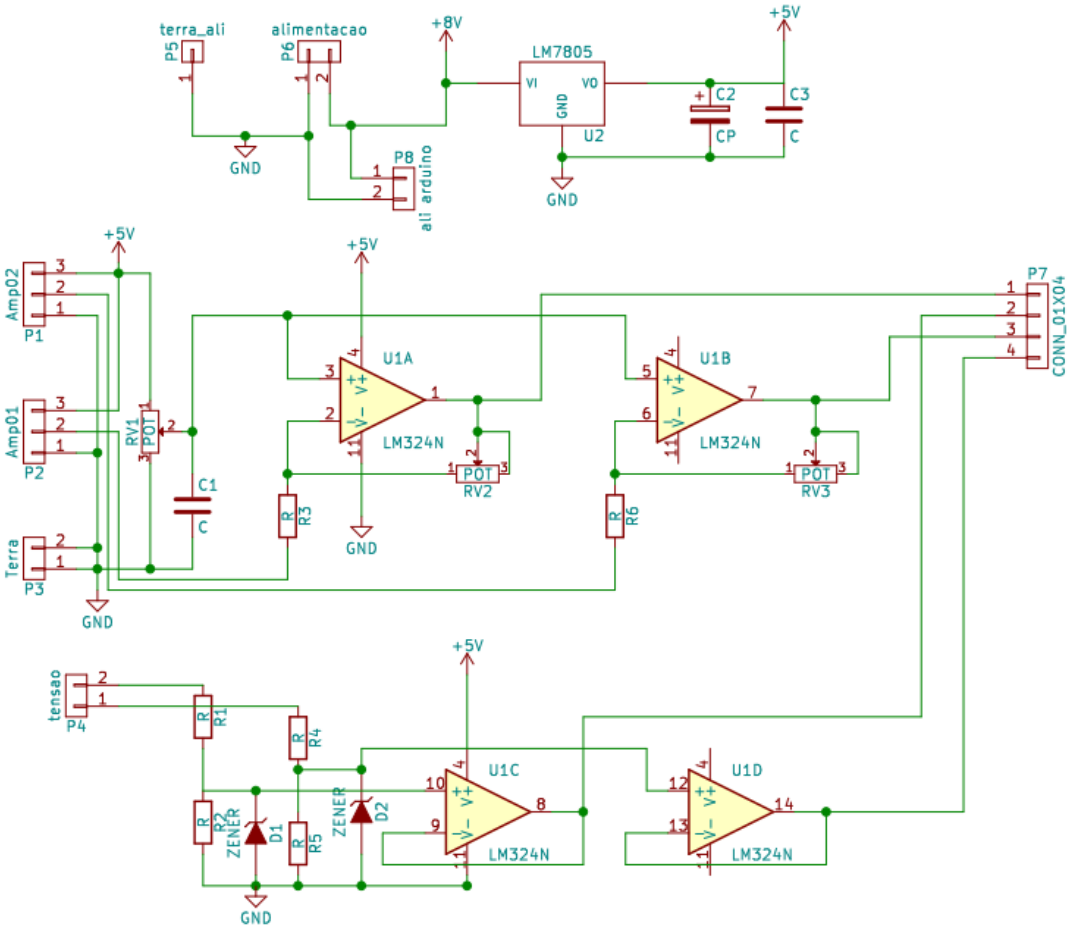
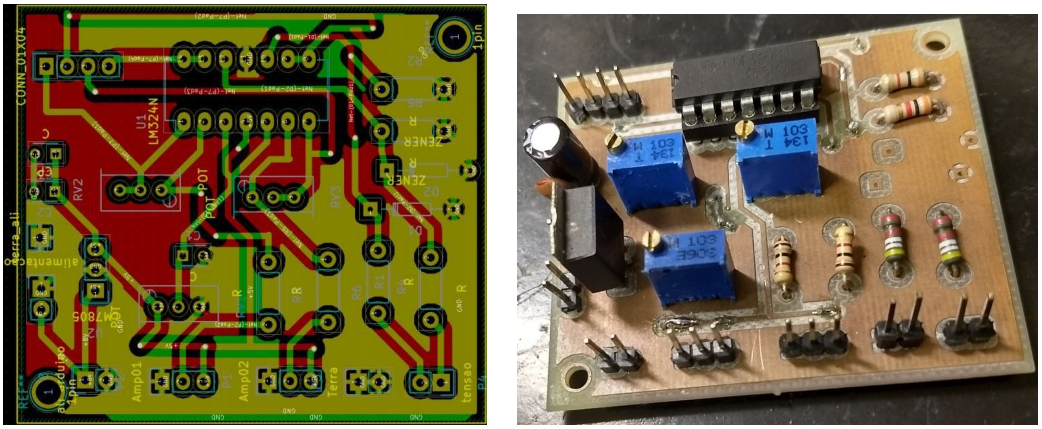


Figura 32 – Esquemático da placa de tratamento de sinais.



(a) Layout da placa no *software*. (b) Placa com componentes soldados.

Figura 33 – Projeto da placa e ela fabricada com componentes

O Autodesk Inventor[67] é uma ferramenta de fácil uso, muito utilizada para impressão 3D amadora, onde uma versão para estudantes está disponível[68]. E Esta é uma boa opção para o ambiente acadêmico no qual desenvolvemos este projeto.

[67] <https://www.autodesk.com/products/inventor/overview>.
[68] <https://www.autodesk.com/education/free-software/inventor-professional>.

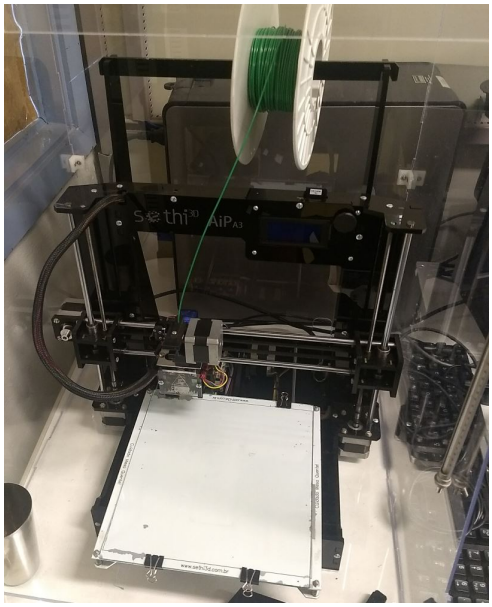


Figura 34 – Impressora 3D Sethi3D AiP A3.

Outras ferramentas são muito utilizadas na indústria, como o SolidWorks[69], desenvolvido pela francesa Dassault Systèmes, o Solid Edge[70], pela Siemens PLM, e o PTC Creo[71], são outras boas opções para o projeto de modelos para impressão e fabricação dos componentes.

O programa utilizado foi o Autodesk Inventor, pois em comparação ao FreeCAD é uma ferramenta superior e mais estável, além de possuir uma versão para estudantes.

Depois de feito o projeto, ele deve ser exportado no formato STL, para posteriormente com um outro *software* ser criado o padrão de impressão. Este padrão diz a espessura das camadas, definindo onde o bico extrusor irá depositar material em cada uma das linhas utilizadas na impressão. Este *software* também é responsável por criar eventuais estruturas de suporte[72] para a impressão na peça que posteriormente serão removidas. O *software* utilizado para este fim foi o *software* gratuito Repetier[73]. Na Figura 35 é mostrado o padrão de impressão utilizando camadas de 0,2 mm.

O projeto feito no *software* Autodesk Inventor é mostrado na Figura 36a. Posteriormente, ele já fabricado utilizando uma impressora Sethi3D, com o *software* Repetier para fazer o fatiamento da impressão, é mostrado na Figura 36 já com os componentes eletrônicos dentro.

[69] <www.solidworks.com>.
[70] <www.siemens.com/solidedge>.
[71] <www.ptc.com/products/creo>.
[72] Uma estrutura de suporte é necessária caso alguma parte da peça não possua apoio em baixo. Por exemplo, caso fosse ser impressa uma caixa com uma das laterais viradas para baixo, a lateral que ficou na parte de cima deve ser suportada por uma estrutura, se não ela irá simplesmente cair, já que o plástico no momento da injeção está muito maleável.
[73] Disponível em: <www.repetier.com>.

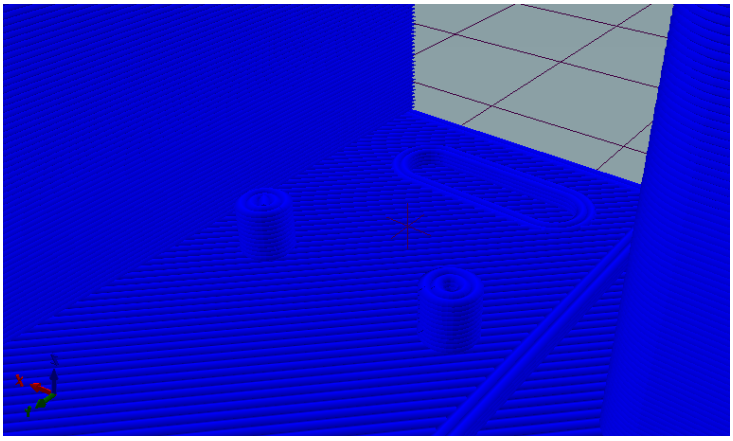
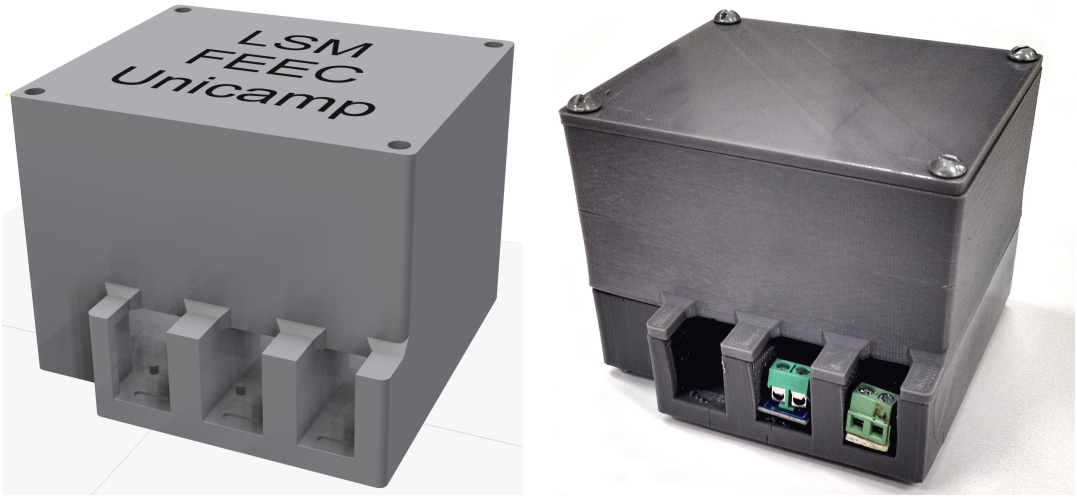


Figura 35 – Padrão de impressão 3D utilizando o *software* Repetier.



(a) Projeto renderizado no *software* Auto-desk Inventor.

(b) Caixa montada incluindo componentes.

Figura 36 – Projeto da caixa e caixa já montada com os componentes dentro.

Numa eventual versão comercial, a caixa deveria fazer uso de terminais tipo forquilha ou conectores MC4, geralmente utilizados em painéis solares, o que facilitaria a blindagem da caixa. Sua fabricação a nível de protótipo permitiu o conhecimento das ferramentas e a verificação positiva em utilizar a impressora 3D para validação dos primeiros protótipos. Nos testes da caixa, foi constatado que o uso de dois conectores MC4 um em cada lado da caixa, permitiria instalar o sistema já na saída dos painéis, antes de qualquer carga, facilitando a instalação.

3.9 Consumo de energia estimado do sistema

O componente com maior consumo de energia é o rádio. O ATmega328P é um microcontrolador de baixo consumo, no entanto a placa Arduino também possuí um conversor UART-USB para fazer a comunicação entre o computador e o Arduino, além de

LEDs que elevam o consumo de energia do circuito. Na análise de consumo consideraremos o pior caso, onde o rádio está transmitindo, além do Arduino sem os recursos de economia de energia ativados. Também foi totalizado o consumo do consumo do amplificador operacional, além de LEDs, resultando em 10 mA. Os consumos são mostrados na Tabela 6. A alimentação do rádio é 3,3 V, mas como ele utiliza um regulador linear, foi considerado 5 V para calcular o consumo.

Tabela 6 – Consumo de energia dos componentes utilizados na montagem dos protótipos.

Componente	Tensão (V)	Corrente (mA)	Potência (mW)
Placa Microcontrolador	5,0	50,0	250,0
Sensor Corrente	5,0	1,0	5,0
Rádio	5,0	45,0	225,0
Outros	5,0	10,0	50,0
Total			530,0

Uma medida para garantir a alimentação sempre em 5 V do protótipo, foi a utilização de um regulador linear após o regulador chaveado, que diminui a tensão de 6,5 V para 5 V, o que aumenta o consumo para 689 mW, já que a diferença entre as tensões é eliminada em calor. O regulador chaveado que diminui a tensão do painel para a alimentação do circuito tem uma eficiência típica de 80%, elevando o consumo estimado final para 861 mW.

A presença do regulador linear se justifica, pois, a tensão de saída do regulador chaveado utilizado pode ser alterada. Ele foi utilizado como uma medida de segurança, que garantiu que os componentes não tivessem uma tensão de alimentação superior ao permitido. Caso não fosse utilizado, o consumo estimado do protótipo seria de 662 mW.

Dos 50 mA utilizados pela placa de desenvolvimento, 35 mA são devidos ao conversor da UART/USB, LEDs e demais componentes da placa (67). Desta maneira, caso o microcontrolador seja implementado diretamente, sem o uso da placa de desenvolvimento, seu consumo seria reduzido para 15 mA. Assim, o nó sensor teria seu consumo estimado em 355 mW antes dos reguladores.

3.10 Custo

O custo inicial de protótipos é geralmente muito superior a fabricação em larga escala. Por este motivo, nas Tabelas 7 e 8 são apresentados os custos dos componentes comprados no Brasil em reais, para a montagem do protótipo, e o preço em dólar na hipotética compra de grandes quantidades. No caso dos nós, é considerado que será montada uma placa englobando todos os componentes no lugar da compra de placas isolas.

Tabela 7 – Custo dos componentes contidos em cada nó.

Componente	Custo em Reais	Custo em dólar
Arduino	R$ 59,90	US$ 1,96
XBee (XB24CAWIT-001)	R$ 130,00	US$ 17,50
XBee *Shield*	R$ 49,90	N/A
Sensor de corrente (ACS712)	R$ 26,90	US$ 1,80
Regulador Step Down (LM2576HV)	R$ 10,00	US$ 3,41
Demais componentes (placas, resistores, diodos, etc)	R$ 20,00	US$ 7,00
Caixa	R$ 10,00	US$ 2,00
Total	R$ 306,70	US$ 33,67

Tabela 8 – Custo dos componentes contidos no concentrador.

Componente	Custo em reais	Custo em dólar
Raspberry Pi 3	R$ 200,00	US$ 35,00
XBee Explorer	R$ 51,31	US$ 4,00
XBee XB24CAWIT-001	R$ 130,00	US$ 17,50
Fonte Raspberry Pi	R$ 30,00	US$ 10,00
Total	R$ 411,31	US$ 66,50

3.11 Conclusões do capítulo

Este capítulo fez um estudo de componentes utilizados no desenvolvimento de dispositivos de internet das coisas e embarcados, que podem ser úteis para projetos além do desenvolvido. A existência de *hardware* de baixo custo, fruto dos investimentos no desenvolvimento dos *smartphones*, possibilitou o desenvolvimento de produtos que no passado teriam custo muito elevado. Além disso, projetos que se iniciaram com objetivo de uso na universidade, como o Raspberry Pi ou Beaglebone, facilitaram muito o desenvolvimento de protótipos de dispositivos embarcados, que no futuro podem virar produtos comerciais. As placas de desenvolvimento de grandes fabricantes, como a Qualcomm Dragonboard, possuem um custo superior. Uma diferença de aproximadamente 3 vezes o valor, com o Raspberry Pi custando R$ 200,00 e a Dragonboard R$ 600,00.

O uso de uma rede *mesh* é importante para elevar a área de cobertura, mas as soluções existentes são consideravelmente mais caras que uma rede estrela. Desta maneira, caso se queira atender apenas uma pequena área, uma solução estrela seria mais apropriada, pois diminuiria consideravelmente o custo do projeto.

O uso de uma impressora 3D se mostrou interessante. Seu custo inicial é alto, a utilizada na fabricação da caixa está na ordem de R$ 4.000,00, mas o material é relativamente barato, competitivo com uma caixa de plástico comum. Isto permitiu a acomodação

dos componentes sem ser necessário recorrer a adaptações e cortes de uma caixa genérica. Como a caixa foi projetada exclusivamente para o primeiro protótipo, isto a deixou com aparência bastante profissional. Além disto, sua fabricação e teste, permitiu encontrar limitações, que serão melhoradas numa próxima versão. O principal problema é a dificuldade na instalação que poderia ser resolvida utilizando conectores MC4 ou terminais tipo forquilha, além disso, a falta de vedação, que poderia facilmente ser sanada utilizando conectores MC4 e vedação de borracha, mesmo utilizando a impressora 3D.

4 Integração do software com o hardware

Para o desenvolvimento dos *softwares* são necessários ao menos dois ambientes de programação. Para os nós foi utilizado o próprio programa fornecido com o Arduino[1], enquanto no concentrador foi escolhida a linguagem Python[2], devido a disponibilidade desta no Raspbian, sistema operacional utilizado no Raspberry Pi, e no Windows, ambiente de desenvolvimento utilizado para a escrita do programa.

Esta linguagem permite a comunicação serial com o XBee, utilizando a biblioteca Pyserial[3] para acesso a porta serial (emulada via USB), outra biblioteca mais especifica para gerenciar o modo API do XBee. Também permite hospedar um site e gerar páginas HTML, como no PHP, utilizando o *framework* Flask[4], além de acesso a um banco de dados, como SQLite[5], que poderia facilmente ser atualizado para um banco de dados mais robusto conforme a necessidade, bastando escrever uma classe abstraindo o acesso ao banco de dados.

O programa desenvolvido deve ser executado como um *daemon* no Linux, ou seja, um serviço que é iniciado com o sistema operacional, sempre em segundo plano. Este *daemon* também é responsável por iniciar um servidor HTTP usando a biblioteca Flask. Este servidor Web pode ser acessado em qualquer lugar do mundo, bastando apenas ter acesso ao endereço IP atribuído ao Raspberry Pi. Além disso, os dados devem ser enviados para um serviço em nuvem.

Primeiramente será introduzido o funcionamento da rede Zigbee, posteriormente analisado o desenvolvimento do *firmware* utilizado no Arduino e, por fim, a implementação do programa responsável por gerenciar todo funcionamento do sistema.

4.1 Configuração da rede sem fio Zigbee

O XBee S2 e S2C utilizados implementam o padrão Zigbee e permitem o uso de duas maneiras distintas de comunicação. A maneira de fácil implementação, é chamada de modo transparente ou modo AT. Neste modo, qualquer dado enviado ao XBee é enviado pela rede, como em uma conexão serial. Para alterar alguma configuração, tal qual o destino do dado, é necessário entrar no modo de configuração e enviar os comandos AT[6].

[1] <https://www.arduino.cc/en/Main/Software>.
[2] <https://www.python.org/>.
[3] <https://github.com/pyserial/pyserial>.
[4] <http://flask.pocoo.org/>.
[5] <https://docs.python.org/2/library/sqlite3.html>.
[6] Conjunto de comandos desenvolvidos por Dennis Hayes que permitem, através de uma série de pequenos textos, ligar, desligar e alterar parâmetros de conexão. Foi muito utilizado em modems de conexão

A entrada neste modo é feita não enviando dados por no mínimo um segundo, depois enviando o texto +++ (repare que não é necessário usar o retorno de cursor após este comando) e posteriormente aguardar novamente um segundo.

O segundo modo, chamado de modo API, trabalha com *frames* de dados. Cada um destes tem predefinida a sua configuração, o tipo do pacote, o destinatário dos dados, a paridade, a necessidade ou não de resposta, etc. Os dados são colocados no pacote, permitindo uma maior agilidade do programa, já que não é necessário toda vez esperar ao menos dois segundos para entrar no modo de comandos AT (68). Utilizando o XBee S2C é possível entrar no modo de comandos AT, utilizando os passos acima, mesmo quando em modo API. Essa possibilidade foi introduzida devido a maior capacidade de processamento do microcontrolador EM357 (69).

Geralmente os programas são inicialmente desenvolvidos utilizando o modo transparente e, conforme ocorre a evolução do código, são passados para o modo API. Mais informações sobre os *firmwares* utilizados para cada modo e as opções de configuração utilizando o *software* do fornecedor podem ser encontradas no Apêndice A, seção A.1.

Existem duas maneiras de se enviar dados em uma rede Zigbee. A primeira, direta entre dois nós chamada de *unicast*, onde usa-se o endereço de 16 bits ou 64 bits e apenas o nó destino receberá a mensagem, evidentemente, como é uma rede *mesh*, possivelmente utilizando outros nós para redirecionar a mensagem, conforme a Figura 37. Pode-se enviar mensagens para o coordenador utilizando o endereço DH = 0 e DL = 0, valido para o modo transparente e API, o que é útil quando o coordenador é o concentrador de dados da rede. A segunda é a transmissão *broadcast*, onde todos os nós da *Personal Area Network* devem receber a mensagem enviada. Esta é repetida pelos roteadores vizinhos um certo número de vezes, limitando o trafego da rede e seu alcance (70). Para enviar dados utilizando *broadcast* basta endereçá-los para DH = 0 e DL = 0xFFFF (71).

4.1.1 Canais de rede sem fio

Assim como o padrão IEEE 802.11, o XBee Zigbee opera na frequência de 2,4 GHz, com 16 canais possíveis (nomeados de 11 a 26). A existência de suporte para estes canais depende da versão de *hardware* utilizada[7], conforme a Tabela 9.

A escolha destes canais é feita pela configuração "SC" (*Scan Channels*) com comando AT ou pelo *software* de configuração XCTU. Cada bit da máscara representa um canal, desta maneira a configuração utiliza 2 bytes, o canal 11 é representado pelo bit menos significativo (ou seja, pelo dígito mais à direta) e o 26 pelo mais significativo. A configuração para poder utilizar receptores de diferentes versões depende de limitar o

à internet discada.

[7] <https://www.digi.com/resources/documentation/digidocs/90001537/references/r_channels_zigbee.htm>.

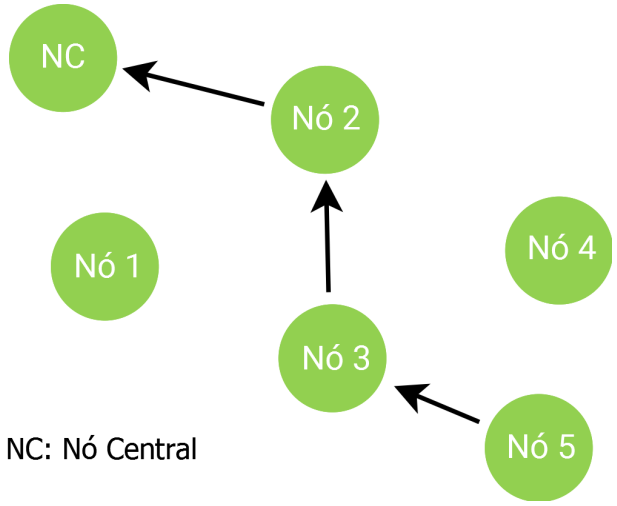


Figura 37 – Nó 5 enviando dados via *unicast* para o no central utilizando a topologia *Mesh*.

Tabela 9 – Canais disponíveis por versão de rádio XBee.

Versão	Canais
XBees antigos	12 até 23
Não Pro versões S2/S2B/S2C	Todos os canais
Pro S2	11 até 24
Pro S2B/S2C	11 até 25

concentrador a trabalhar apenas em canais suportados por todos os nós da rede, evitando assim a possível escolha de um canal no qual alguns nós não possam trabalhar, por não suportarem tal frequência de operação.

4.1.2 Frame de dados no modo API do XBee

Todos os comandos e modos de envio de dados podem ser utilizados no modo API, que é uma maneira mais prática e segura de transmitir os dados, porém aumentando a dificuldade da implementação do *software*. No modo transparente os dados são enviados como em uma conexão serial, ou seja, tomando uma transmissão com o seguinte texto "Olá mundo!", não se tem garantia que todo o texto será enviado junto, ou se será dividido em mais de um pacote. Sendo assim, o receptor poderia receber apenas "mundo!", o que, a depender do protocolo de comunicação utilizado, poderia ou não ter sentido, não sendo possível saber qual parte da mensagem foi perdida.

Enviando esta mesma mensagem utilizando o modo API, quem faz a divisão dos pacotes é o próprio usuário, sendo assim, é possível dividir o conteúdo a ser enviado de uma maneira que o aplicativo seja capaz de detectar alguma falha de transmissão e pedir o reenvio. Outro ponto a ser considerado é a presença no modo API de um mecanismo de paridade na criação dos pacotes, de modo a garantia que o dado enviado não possui erros

simples de troca de bits.

O modo API funciona com a criação de pacotes (*Frames*) de dados, seguindo um formato pré-definido para cada tipo de pacote. Este formato, que é mostrado na Figura 38, se inicia sempre no primeiro byte com o número em hexadecimal 7E. A distância é dada pela contagem do número de bytes incluindo a distância e a soma de verificação. Esta por sua vez é calculada desconsiderando o demarcador de frame e comprimento, somando todos os bytes restantes e subtraindo os 8 bits menos significativos (ou seja, mais a direita) do número em hexadecimal FF (68, 72).

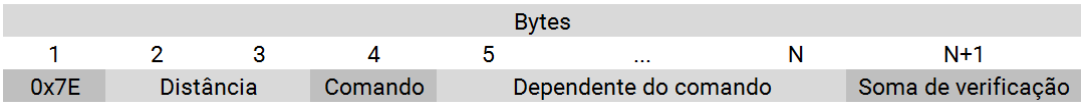


Figura 38 – Estrutura genérica de um *frame* de dados utilizados no XBee.

O conteúdo a partir do quinto byte é dependente do comando utilizado, podendo ser, por exemplo, um endereço e a mensagem a ser enviada. Os principais tipos de *Frames* são mostrados na Tabela 10.

Tabela 10 – Principais tipos de comandos AT:

Código	Descrição
0x8A	Status do Modem
0x10	Envio de dados
0x8B	Resultado do envio de dados
0x08	Comando AT
0x88	Resultado do comando AT
0x90	Pacote Recebido
0x17	Comando AT remoto
0x97	Reposta do comando AT remoto

Para mais informações sobre *Frames* de dados leia o Apêndice A.2.

4.2 Firmware desenvolvido

O desenvolvimento do *firmware* foi feito utilizando o ambiente IDE Arduino. Este *firmware* deve receber e tratar os dados do sensor de corrente e tensão e posteriormente enviando ao concentrador. Deve-se escolher uma taxa de amostragem, além de uma taxa de envio. Considerando-se que a rede pode ter dezenas, ou mesmo centenas, de nós, deve-se utilizar o mínimo possível de banda.

O sistema amostra um número configurável de vezes por segundo a tensão e a corrente na saída do painel solar. Para o cálculo da mediana foi utilizada a biblioteca de

código aberto QuickStats [8]. É criado então um vetor com a leitura da tensão e corrente feita a cada segundo. Este vetor, a cada tempo também pré-definido, é totalizado sendo calculado sua média e então enviado ao concentrador. Por padrão, o número de medições por segundo é 3, com um intervalo de 10 segundos entre os envios. Na Figura 39 é mostrado este procedimento de amostragem e o dado final de tensão que é enviado ao concentrador. Este procedimento é efetuado tanto para a tensão quanto para a corrente.

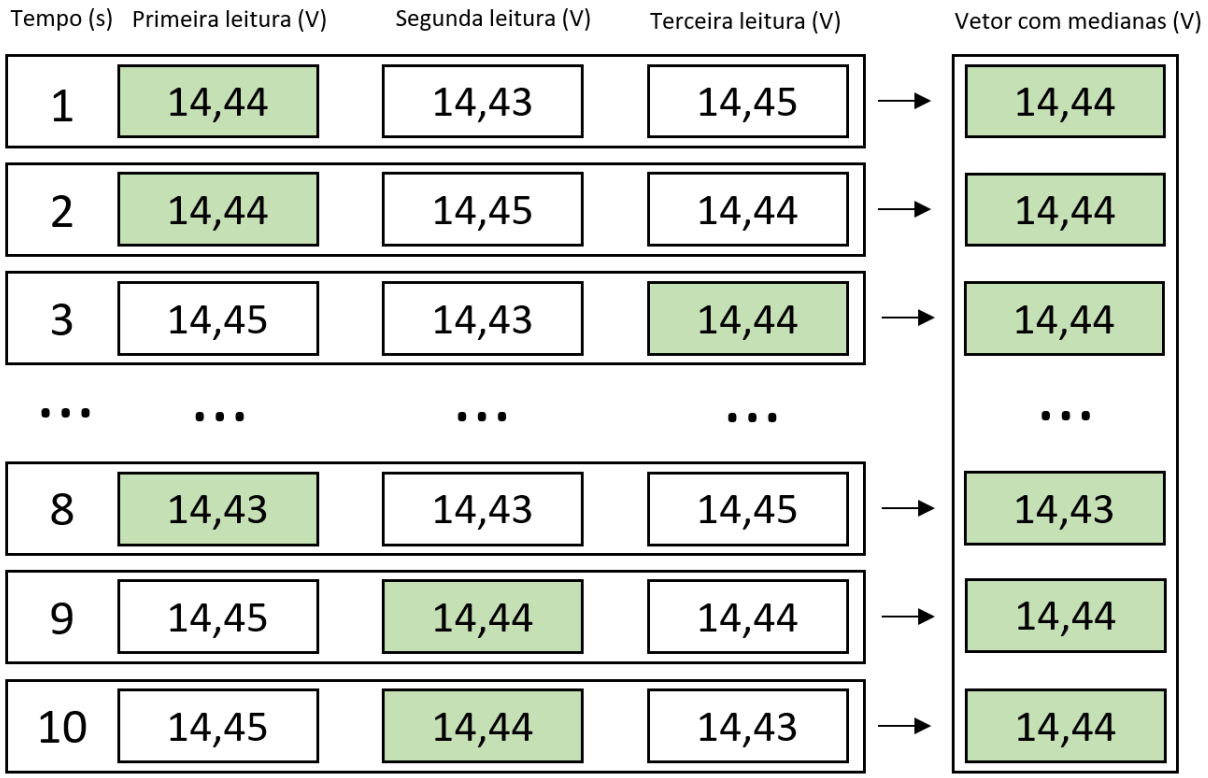


Figura 39 – Procedimento de amostragem dos dados.

Este envio a cada tempo pré-definido pode ser atrasado caso o nó envie a informação ao concentrador e o XBee acuse um problema com a entrega. Se isto ocorrer, o tamanho do vetor que armazena as leituras a cada segundo será incrementado. O limite do tamanho é definido no *firmware* em tempo de compilação para evitar que o programa trave caso ele fique sem conexão por muito tempo.

O formato de envio para o concentrador é o JSON, que possui *parsers* [9] para várias linguagens de programação. No caso do *firmware*, inicialmente foi utilizada uma biblioteca chamada de ArduinoJson[10], porém, devido a limitações de memória, seu uso não se mostrou consistente. Os primeiros textos eram gerados, mas posteriormente a biblioteca parava de funcionar. Como as strings a serem geradas eram pequenas e fáceis

[8] Disponível em <https://github.com/dndubins/QuickStats>.

[9] Processo de análise de texto, seja em linguagem natural ou linguagem de computador, conforme as regras de uma gramática. Na computação, uma biblioteca *parser* permite a análise de uma *string* resultado apenas nos itens de informação desejados dela.

[10] <https://bblanchon.github.io/ArduinoJson/>.

se serem criadas, o problema não foi investigado, sendo desenvolvida uma função própria para criação das *strings*. Um exemplo de *string* JSON utilizado é mostrado a seguir:

$$\{'t' : 23441,' r' : 10329,' v' : 11.592,' c' : 0.724\} \tag{4.1}$$

Onde t significa o tempo que o nó está ligado em segundos, r o tempo em milissegundos entre o envio atual e o anterior, v a tensão em volts e c corrente em amperes. É importante manter este formato consistente e a tensão e corrente sempre nas mesmas unidades, pois assim é possível criar um nó e ele será compatível com o concentrador ou alterar o *software* do concentrador mantendo a compatibilidade com os nós.

A importância de se conhecer diferença de tempo entre os envios está em que este tempo pode variar se o XBee perder a conexão, conforme já explicado, mas, além disso, o XBee demora algumas dezenas ou centenas de milissegundos para retornar o resultado do envio. Desta maneira, é necessário adicionar esta diferença na hora de totalizar a geração, ou ela será subestimada.

Existe uma opção de escolha para usar ou não o modo API do XBee explicado anteriormente. A implementação do modo API faz uso da biblioteca e esta biblioteca permite o envio de mensagens utilizando *frames*, calculando automaticamente o tamanho a mensagem e soma de checagem, além de analisar a resposta do XBee para a transmissão, permitindo o tratamento de exceções, como já explicado acima. Com a utilização do modo API e a confirmação do recebimento da mensagem, foi implementado um padrão de LEDs conforme o resultado do envio. Um fluxograma do funcionamento do *firmware* é exibido na Figura 40.

4.3 *Software* desenvolvido para o concentrador

O *software* do concentrador foi desenvolvido utilizando a linguagem Python. Foi utilizado o ambiente de desenvolvimento Windows com a distribuição Anaconda[11] visando a utilização posterior em ambiente Linux. Desta maneira, todo o programa foi escrito evitando o uso de recursos exclusivos de um sistema operacional e utilizando bibliotecas disponíveis em ambos os sistemas. Quando impossível utilizar a mesma solução, foi empregado um algoritmo para detectar o sistema e tomar as ações necessárias visando o sistema em específico.

Como o programa deve estar em execução durante vários meses, ou mesmo anos, qualquer ponto de falha deve ser tratado para evitar a interrupção de sua execução. Desta maneira focou-se em buscar possíveis pontos de falha no código e tratá-los. Foi adotado um diário de suas operações utilizando a biblioteca de *logging* padrão do Python. Estão

[11] Disponível em: <https://www.continuum.io/downloads>.

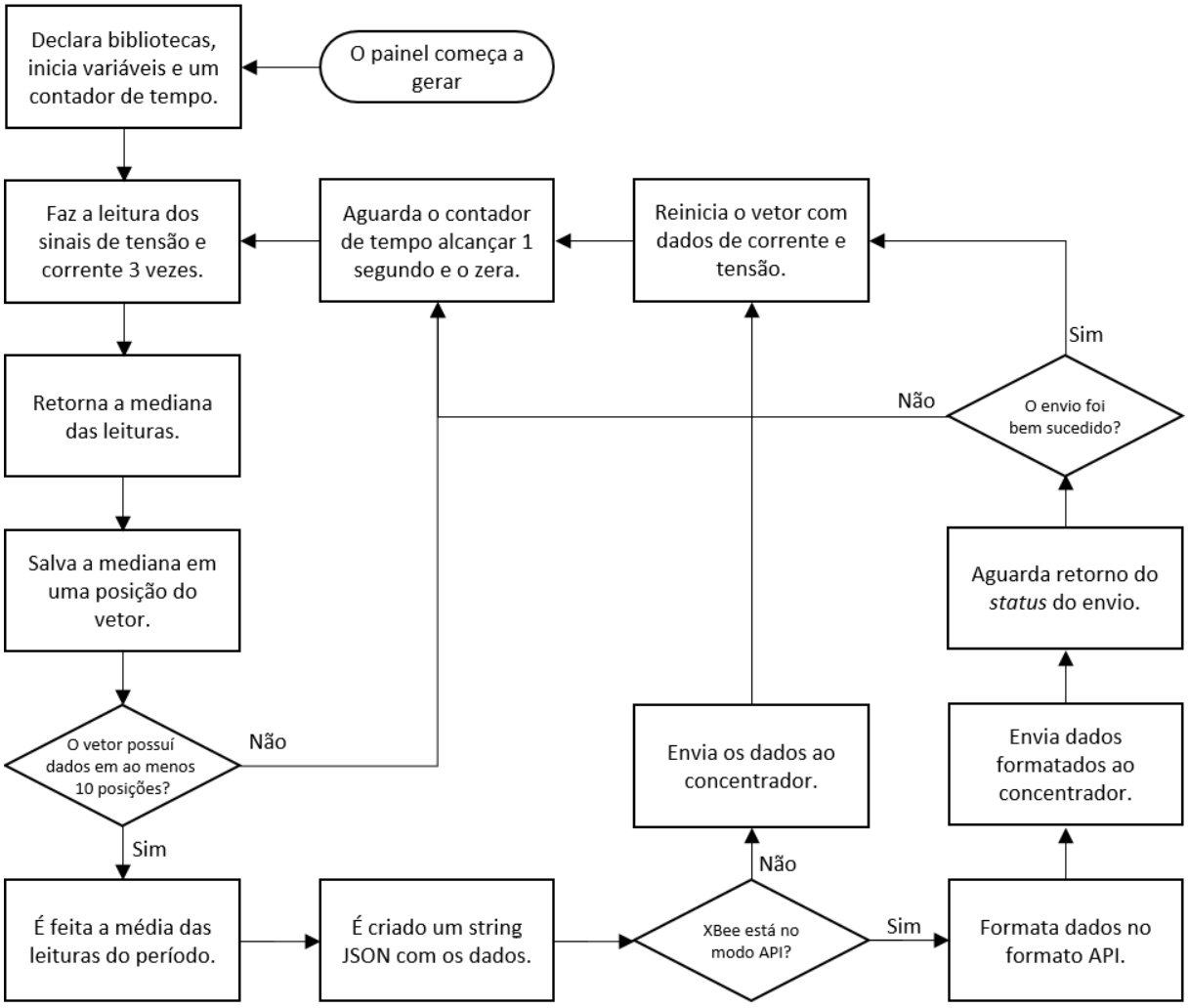


Figura 40 – Fluxograma do funcionamento do *firmware*.

disponíveis 3 níveis de *logging*, o mínimo onde apenas as mensagens de erro são salvas, o médio, onde as mensagens de erro e informativas são preservadas e o *debug*, onde todas as mensagens são salvas. Por padrão é utilizada a opção intermediária.

O programa é executado em dois processos separados. O processo principal é responsável por receber as leituras do XBee, tratá-las, enviá-las para banco de dados e posteriormente enviar estes dados a um serviço em nuvem. Um fluxograma do funcionamento em alto nível é mostrado na Figura 41. O segundo processo é responsável por hospedar uma página web em um servidor HTTP utilizando a biblioteca Flask.

O sistema é capaz de detectar a desconexão do XBee e sua posterior conexão. Isto pode ocorrer por um mal contato na ligação entre o Raspberry Pi e o XBee. Esta característica é especialmente útil durante os testes de campo do dispositivo, com o *software* sendo executado em modo *daemon*, onde não se possui acesso à interface gráfica, com exceção de acesso pela rede sem fio.

Os dados são armazenados em um banco de dados SQLite utilizando uma classe

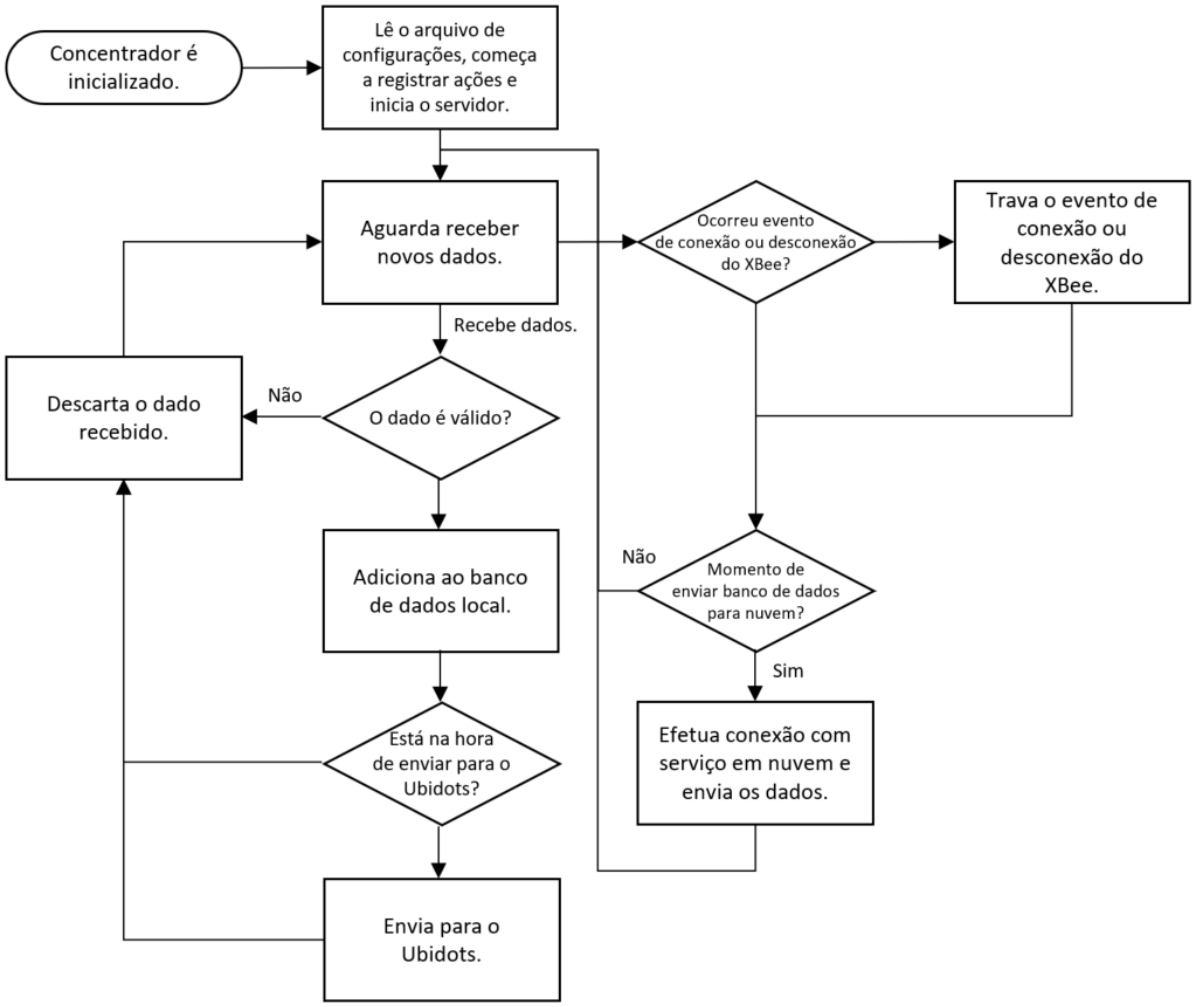


Figura 41 – Fluxograma do funcionamento do *software* do concentrador.

Python especialmente escrita. Este banco de dados armazena as estradas em arquivos binários criados a cada mês e os acessos são realizados pelo próprio Python, sem a necessidade de um servidor como o MySQL. A principal desvantagem é que o acesso é mais lento que em um servidor, mas nos testes realizados este não foi um fator limitante.

Devido a maneira com que a classe foi escrita, é possível substituir este banco de dados por um servidor mais robusto ou mesmo um servidor em nuvem. Os campos armazenados no banco de dados são mostrados na Tabela 11.

Depois de um tempo determinado no arquivo de configuração, o arquivo com o banco de dados é enviado a um serviço de armazenamento em nuvem. O escolhido foi o Dropbox[12], devido a facilidade de enviar arquivos utilizando um Shell Script. Para sua execução foi utilizada a biblioteca subprocess com o comando POpen, que permite a execução de um comando no terminal do Linux sem aguardar o retorno deste.

O concentrador possui um arquivo de configuração chamado "config.ini". Este ar-

[12] Disponível em: <www.dropbox.com>.

Tabela 11 – Campos presentes no banco de dados.

Nome do campo	O que é?
node_name	Nome do nó ao qual o dado pertence
zigbee_address	Endereço do qual o concentrador recebeu os dados, é baseado nele e no arquivo de configuração que o node_name é utilizado
corrente	Corrente medida
tensao	Tensão medida
datetime	Data e hora da medição
tempo_amostragem	Quantos segundos foram utilizados como amostra para tomar as medições de tensão e corrente

quivo define as configurações básicas do programa, como a porta serial utilizada pelo XBee e *baudrate*, além das configurações de cada um dos nós, como o nome e endereço físico de rede. Também foi criada uma configuração chamada de "send_to_cloud", que habilita ou não o envio para a nuvem das informações daquele painel. As informações de inclinação do painel, orientação em relação ao norte, posição geografia (latitude e longitude), ano de instalação, também podem ser salvas neste arquivo para serem utilizados nos relatórios gerados. Um exemplo de arquivo de configuração é mostrado na Figura 42.

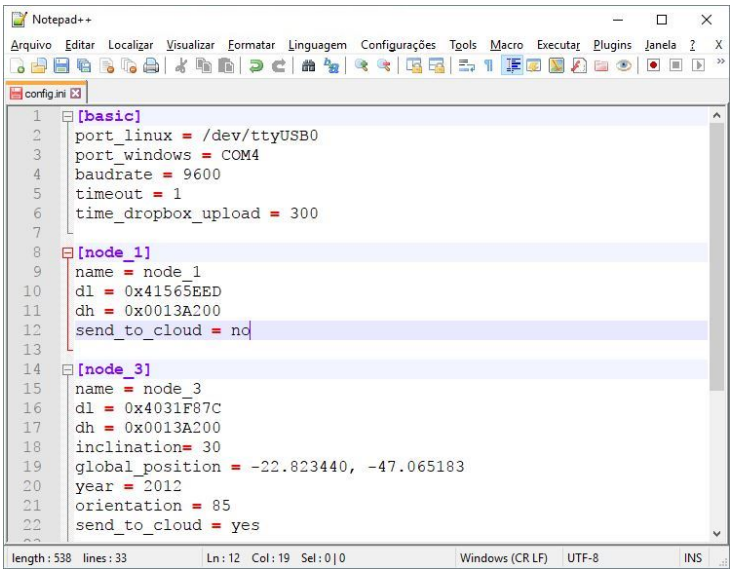


Figura 42 – Arquivo de configurações do concentrador.

Dentre os serviços em nuvem disponíveis, foi escolhido o Ubidots. Foi então criado um dispositivo no Ubidots para cada painel solar. Cada dispositivo possui uma localização geográfica armazenada e os dados de tensão e corrente, enquanto a potência é inferida. Um arquivo de configuração chamado "cloud.ini" armazena as chaves dos campos no Ubidots utilizadas para enviar os dados de cada um dos nós, sendo necessário, portanto, criar os canais manualmente e copiar estas chaves para o arquivo de configuração. O site também possui um painel onde é possível exibir informações já consolidadas apenas dos nós de

interesse. Para evitar muito tráfego de rede, os dados são enviados a cada 2 minutos.

Desta maneira o concentrador possui os seguintes módulos:

- **main:** responsável por inicializar o programa, ativar o servidor, receber os dados dos nós e enviar para nuvem;
- **server:** hospeda o servidor para acesso via rede local e internet;
- **access_database:** funções para consultas e gravação ao banco de dados. Repare que apenas alterando este arquivo é possível trocar qual o banco de dados utilizando, passando para um mais competente se a quantidade de dados aumentar;
- **plotador:** parte do programa que gera os gráficos utilizados para exibir os dados ao usuário utilizando a biblioteca Matplotlib. Este arquivo é utilizado tanto pelo servidor, quanto pelo aplicativo desktop. Desta maneira qualquer mudança na aparência do gráfico é espelhada em ambos;
- **cloud:** implementa o envio das informações para algum serviço em nuvem. Possui a função "enviar_dados_nuvem", que recebe os dados de corrente, tensão e tempo de amostragem, enviando ao serviço na nuvem. O controle entre os tempos de chamada desta função é feito pelo arquivo "main";
- **main_interface_db:** implementa uma interface de usuário para análise dos bancos de dados e geração de relatórios, será explicado a seguir;
- **aux_functions:** funções utilizadas por mais de um dos módulos acima, que evitam código duplicado.

4.4 Interface para o usuário analisar dados

As leituras são armazenadas em bancos de dados SQLite dividas mês a mês no concentrador. Caso o usuário queria analisar estes dados de uma maneira mais fácil em seu computador pessoal, foi desenvolvido também uma interface gráfica que permite a ele abrir cada um destes arquivos e analisar as informações. Uma imagem desta interface é mostrada na Figura 43.

Esta interface permite exportar os dados de geração em arquivo CSV ou XLS para utilização posterior em outro *software*, como o Excel, para uma análise mais detalhada. É possível salvar o gráfico criado no formato de imagem PNG. Caso exista um período de interesse é possível exportar os gráficos do mês em formato PDF.

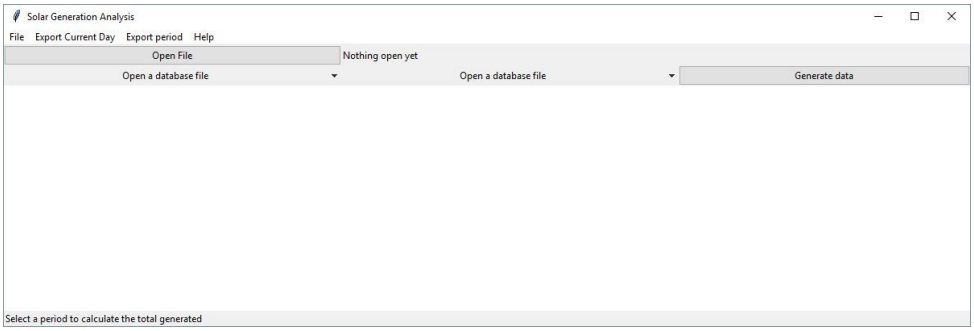


Figura 43 – Interface de usuário para visualização de dados e geração de relatórios.

4.5 Segurança

Com a conexão de dispositivos à internet, se faz cada vez mais importante a segurança destes equipamentos. No geral, dispositivos IoT contam com um processador de desempenho equivalente a um computador de 10 anos atrás e um sistema operacional completo. Isso abre portas para graves falhas de segurança, ameaçando toda rede em que estão conectados, inclusive causado problemas em escala global na internet, como o que aconteceu em 2016 (73), que utilizou câmeras de segurança para um ataque de negação de serviço[13].

Sistemas operacionais possuem diversos erros em sua programação, que todos os meses são encontrados por pesquisadores, sendo lançadas atualizações de segurança que corrigem estes problemas. Por uma questão mercadológica, dispositivos embarcados ou mesmo *smartphones* muitas vezes não são atualizados, permanecendo com falhas de segurança já publicamente conhecidas, de fácil exploração. Isto ocorre devido às atualizações possuírem um custo de desenvolvimento e validação, além de um possível risco de danificar o dispositivo durante o processo de atualização[14], não sendo interessante para a empresa utilizar recursos para dar suporte a um produto que não trará mais nenhum lucro.

Esta pouca preocupação com segurança é um dos fatores que podem dificultar a ampla adoção de dispositivos de internet das coisas, já que possivelmente você estaria colocando uma porta aberta na questão de segurança na sua casa. Deve-se adotar escolhas de *software* e rede sem fios que aumentem a segurança e que possibilitem manter o dispositivo atualizado.

O sistema operacional Raspbian é baseado na distribuição Debian do sistema operacional Linux. Ela é facilmente atualizável, com um simples comando de terminal e posterior reinicio do sistema. Também deve existir a preocupação de desligar os possíveis

[13] Uma grande quantidade de computadores ou dispositivos acessam um site ao mesmo tempo, causando com que o servidor não consiga responder todas as requisições e tirando o serviço do ar.

[14] Durante a atualização, no geral o *firmware* anterior é apagado ou sobrescrito bloco a bloco pela nova versão, desta maneira se ocorrer um problema durante o processo de atualização o dispositivo poderá não funcionar. Existem maneiras de evitar que isso ocorra, como colocar um *bootloader* que não é apagado durante atualização ou duas partições de *firmware*.

métodos de comunicação pela internet, como SSH e VPN. Geralmente distribuições Linux para sistemas embarcados ou mesmo produtos para o consumidor, possuem senhas de acesso padrão, permitindo, ao mesmo tempo, fácil acesso aos usuários e aos possíveis atacantes pela internet. Por este motivo é importante trocar as senhas padrão da distribuição Linux utilizada, que no caso do Raspbian é o usuário "pi" e a senha "raspberry".

O serviço deve ter acesso ao mínimo necessário dos recursos do sistema operacional. Por exemplo, o usuário administrador (*root*) pode acessar todas as pastas e editar qualquer arquivo no sistema. Caso o serviço que controla as funções de internet das coisas do dispositivo esteja sendo executado como usuário administrador e seja comprometido por um atacante, este ganha acesso a todas as funções do sistema. Por outro lado, se ele estiver sendo executado como usuário comum, ele não poderá comprometer outras áreas do sistema operacional.

4.5.1 Segurança em redes sem fio

Outro ponto importante é a segurança nas comunicações sem fio. Vários tipos de ataque são possíveis. Um exemplo seria um atacante gravar conteúdo dos pacotes trocados entre as comunicações dos nós. Este também pode adicionar pacotes na comunicação da rede, fazendo com que a rede os detecte como autênticos. Desta maneira é importante que as comunicações sem fio possuam criptografia, para que mesmo se o atacante tiver acesso aos pacotes, eles estarão criptografados e ele não terá acesso ao seu conteúdo. O Zigbee oferece suporte à criptografia AES de 128 bits, um padrão que é o estado da arte na indústria.

Com a rede utilizando criptografia, o atacante, a não ser que conheça a chave criptográfica, não será capaz de gerar pacotes específicos para a rede. Outro ponto importante é que mesmo que o protocolo de rede seja conhecido, ou venha a ser público no futuro, ele deve ser manter seguro. A segurança não deve estar no desconhecimento de seu funcionamento, mas sim em criptografia, onde mesmo o atacante conhecendo os detalhes de funcionamento do sistema, sem acesso as chaves de criptografia, o sistema se mantém seguro.

Mesmo com criptografia ainda existe a possibilidade de o atacante gravar pacotes e posteriormente reproduzir estes pacotes na rede, que deve reconhecer estes pacotes como inválidos. Por exemplo, caso a rede seja utilizada para controlar um portão, um controle remoto emite um comando para abrir o portão e alguém com más intensões grava este comando com o intuito de posteriormente reproduzir o comando. A rede deve saber que este comando gravado não é válido no momento, caso contrário o portão irá abrir e teremos um grande problema.

4.5.2 Segurança no padrão Zigbee

Para o curso de segurança de redes e computadores de 2017 do Massachusetts Institute of Technology, foi escrito um artigo por Fan et al.(74) onde compraram um produto comercial da Samsung e tentaram atacar. Ele mostra de maneira mais prática os problemas e pontos positivos do padrão Zigbee.

O Zigbee pode ou não utilizar criptografia para comunicação e isto é configurado pelo desenvolvedor da solução. Existem basicamente duas chaves de segurança: a chave da rede, que possui 128 bits, definida pelo coordenador e transmitida para os nós quando eles se conectam à rede, e a chave de ligação global (*Global link key*), que todos os nós devem ter acesso antes da rede ser iniciada. A chave de ligação global é utilizada apenas para transmitir a chave de rede a cada um dos nós durante sua conexão inicial à rede ou quando a chave global é alterada.

Desta maneira se o atacante possuir a chave de ligação global, ele deve esperar algum nó se conectar à rede para obter a chave de segurança da rede, podendo então se comunicar com os demais nós. No geral, produtos comerciais utilizam sempre a mesma chave de ligação global, que é "ZigbeeAlliance09". Isto é uma grande falha de segurança, existente com o intuito de dar conveniência aos usuários, que podem conectar qualquer novo dispositivo à sua rede, e ao fabricante, que pode utilizar sempre o mesmo *firmware*, não tendo que fazer nenhuma gravação personalizada na fabricação com uma chave única para cada dispositivo. Dentre as soluções possíveis, ainda não ideais, seria a fabricante utilizar uma chave própria para seus produtos na mesma linha, que seria escondida do usuário, ou uma chave diferente para cada dispositivo vendido.

A solução prática seria utilizar uma criptografia assimétrica, onde existe uma chave pública, que encriptaria a chave da rede, que poderia ser transmitida sem fio para os nós que por sua vez a descriptografariam utilizando sua chave privada. Esta combinação de chave pública e privada seria única de cada nó e poderia ser gerada aleatoriamente a cada inicialização. O problema é ela necessita de mais capacidade de processamento e memória do que uma criptografia simétrica, como AES, o que era um grande problema quando o padrão Zigbee foi criado. Este conceito é utilizado no Google Thread, tornando-o mais seguro (75).

Como utilizamos o Zigbee, devido ao Thread ainda não ser um produto comercial no início do projeto, devemos mitigar esta falha do padrão. Desta forma iremos programar a mesma chave de ligação global em todos os nós. Esta chave não será a padrão utilizada, mas sim uma exclusiva do nosso projeto. Assim os dados enviados pelos nós ao concentrador estarão seguros.

Os nós do nosso sistema fazem as leituras necessárias e enviam estas informações ao concentrador. Agora voltamos ao caso destes pacotes serem gravados e depois reproduzidos

por um atacante. O padrão Zigbee emprega um contador de pacotes (*frame counter*) em cada um dos nós (76). O valor deste contador é enviado no cabeçalho dos pacotes. Caso a rede receba um pacote que possua um número menor do que o esperado, este pacote é descartado. Como a chave da rede, por segurança, deve ser gerada novamente a cada inicialização do coordenador, os pacotes eventualmente gravados pelo atacante ou serão inválidos devido a chave de segurança ou devido ao contador de pacotes.

Desta maneira, alterando a chave padrão da rede sem fio Zigbee, em junção com os outros mecanismos de proteção da rede, é possível obter maior segurança de falhas.

4.5.3 Processo de fabricação de *hardware*

O processo de desenvolvimento e fabricação de produtos com componentes eletrônicos vai muito além do que em geral se imagina. A preocupação com a facilidade de fabricação é muito importante, pois boa parte do custo do produto advém desta etapa. Outro ponto importante são os testes pós montagem para evitar enviar para clientes produtos com defeito, que elevariam o custo de suporte. A etapa de desenvolvimento de equipamentos de teste pode ser tão, ou mais, trabalhosa que o desenvolvimento do próprio produto e geralmente é negligenciada no planejamento.

Como um exemplo, em The Hardware Hacker, Huang(77) exemplifica o desenvolvimento de dos produtos da linha Chibitronics[15]. O processo de desenvolvimento do produto em questão demorou aproximadamente 2 semanas, divididas em desenho em um *software* vetorial da arte, projeto da placa no Altium e programação na IDE do Arduino. Este produto contava com diversos sensores, que processavam som, luz e toque, portanto a etapa de teste deve ser capaz de detectar problemas em qualquer um destes componentes. O desenvolvimento do sistema de testes para todos estes sensores levou 2 meses. Desta maneira, o desenvolvimento do aparato de testes demorou 4 vezes mais do que o do produto. É evidente que este tempo varia conforme o produto, no caso do exemplo, ele era muito simples, mas possuía vários sensores, que dificultavam os testes.

No projeto desenvolvido, caso fosse fabricado industrialmente, seria necessário o desenvolvimento dos aparatos de testes e calibração do dispositivo. Pode ser utilizado um dispositivo que teste individualmente o sensor de tensão e depois o de corrente, em conjunto com a rede sem fio. Quando problemas em um destes componentes for encontrado, a placa deve ser segregada e enviada para um técnico corrigir o problema. Existem dois pinos para acesso a comunicação serial do XBee, necessários para programá-lo na linha de montagem durante a produção dos dispositivos com as configurações de rede, além da chave de segurança.

A montagem também deve ser simplificada, o que foi levado em conta na segunda

[15] <https://chibitronics.com/>.

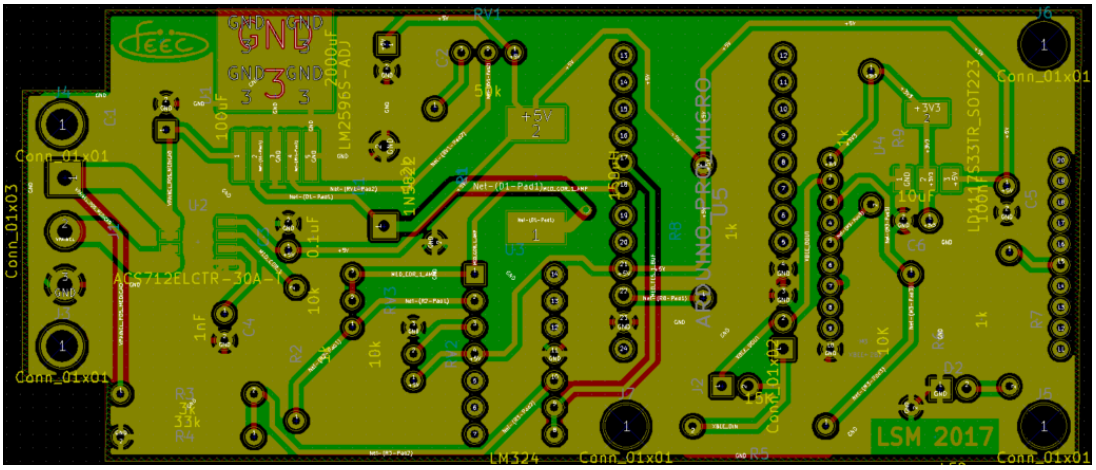


Figura 44 – Layout da segunda versão da placa no Kicad.

versão do protótipo. A placa deve ser colocada na parte de baixo da caixa. Então, a parte de cima pode ser encaixada terminando a montagem.

4.5.4 Projeto de versão atualizada

Devido a restrições de orçamento, na primeira versão foram feitas diversas escolhas de projeto que beneficiam a redução de custo para a montagem do protótipo. Desta maneira, foi feito o projeto de uma versão sem estes compromissos. As principais mudanças foram colocar todos os componentes em apenas uma placa, diminuindo o tamanho dos nós, removendo componentes não utilizados da placa Arduino, o que diminuiria o consumo em 175 mW (67), além do regulador linear que era utilizado após o chaveado. Isto diminui o consumo de energia do nó em aproximadamente 49%, passando de 861 mW para 444 mW, extrapolando os dados anteriormente calculados para o primeiro protótipo.

O microcontrolador foi alterado do ATmega328P, com empacotamento DIP, disponível no Arduino Uno R3, para um ATmega32U4, empacotamento SMD, utilizando uma placa de desenvolvimento baseada no projeto da SparkFun Pro Micro. O microcontrolador ATmega32U4 já possui suporte a USB nativo, não necessitando de um outro circuito integrado para tal. O XBee também foi colocado na mesma placa. Os demais componentes se mantiveram do projeto anterior. O esquemático é mostrado na Figura 45. A placa projetada possui dimensões de 43,0 mm por 101,1 mm, mostrada na Figura 44 e ela já montada com os componentes na Figura 46.

Uma nova caixa foi projetada utilizando o Autodesk Inventor, que possui as dimensões de 10,7 mm comprimento, 48 mm de largura e 50 mm de altura. Uma imagem do projeto da caixa é mostrada na Figura 47a e a caixa fabricada utilizando novamente a impressora 3D da Sethi3D é mostrado na Figura 47b.

O projeto da placa não foi mandado para fabricação. Deixamos o projeto da placa e da caixa para uma futura continuidade deste trabalho e como uma forma mais perto de

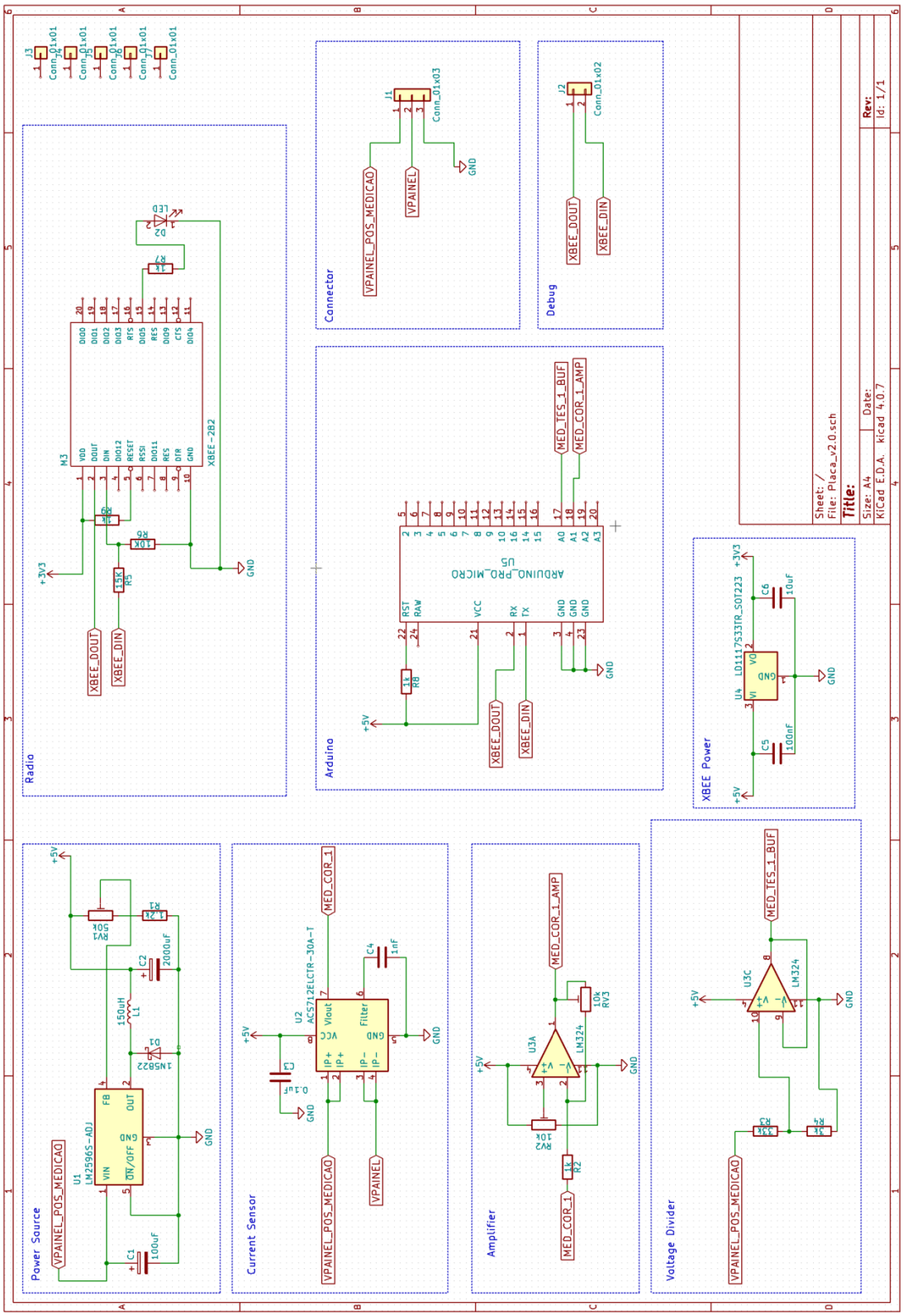


Figura 45 – Esquemático da segunda versão da placa no Kicad.

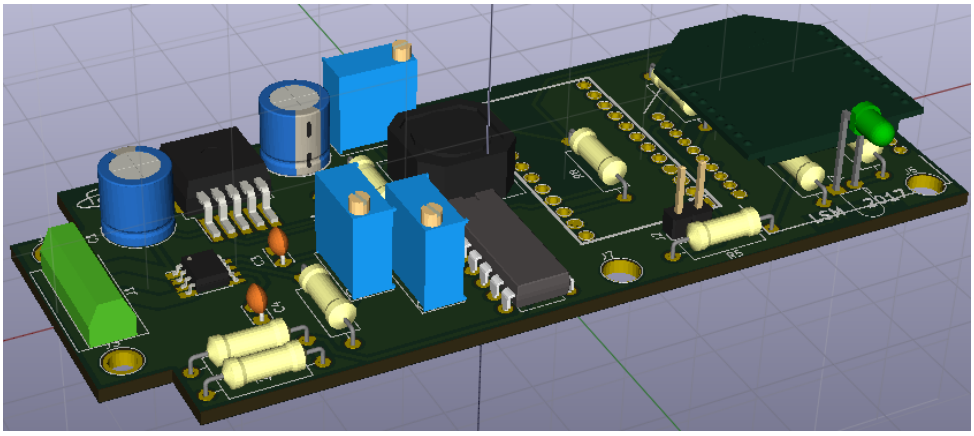
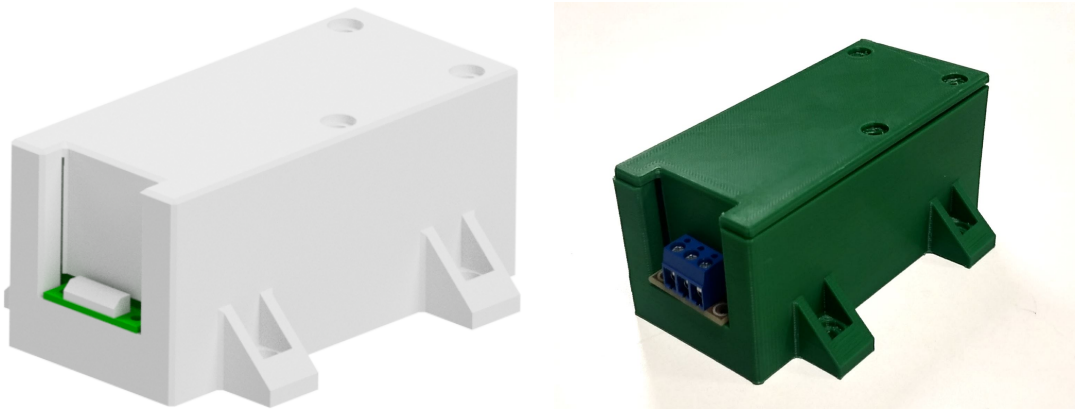


Figura 46 – Visão 3D da segunda versão da placa.



(a) Projeto da caixa no Inventor.

(b) Caixa fabricada com placa, mas sem os componentes internos.

Figura 47 – Projeto e caixa já fabricada.

um produto ao consumidor.

4.6 Conclusões do capítulo

O *software* desenvolvido para o protótipo do sistema consistiu no programa em Python que é carregado no concentrador, que recebe os dados da rede de sensores sem fio, armazena localmente, hospedando uma página HTML, além de enviar os dados para a nuvem. A escolha da linguagem Python permitiu portabilidade entre as plataformas. O sistema do concentrador pode ser executado tanto em Windows quanto em Linux. Além de flexibilidade, a linguagem permite fácil comunicação pela porta serial, ao mesmo tempo permite conexões a internet, além de bibliotecas de análise de dados como o Pandas ou para geração de gráficos como o Matplotlib. Desta maneira, o uso da linguagem Python foi muito acertada e é recomendada para novos projetos de concentradores.

O *firmware* dos nós foi desenvolvido utilizando a IDE Arduino, que emprega a linguagem C++ com algumas adições. Este ambiente de desenvolvimento foi muito útil

para o projeto, sendo de fácil e rápido uso. Em comparação ao ambiente CodeWarrior, nota-se a falta de *breaking points*, o que foi uma desvantagem, que trouxe um pouco mais de complexidade ao desenvolvido do *firmware*. Este, por sua vez, amostra a tensão e corrente gerada pelo painel, armazenando estes dados a cada segundo e depois de alguns segundos totalizando, enviando-os pela rede sem fio ao concentrador. Ele se mostrou estável durante as leituras, possuindo funções de proteção. Caso, por exemplo, o nó deixe de detectar o concentrador, ele irá aumentar o tamanho do vetor de dados até retomar a comunicação com o concentrador ou atingir um limite previamente definido, não perdendo assim leituras.

Tanto o concentrador quanto os nós sensores, possuem um padrão de comunicação que foi projetado para ser expansível. Desta maneira, com mínimas alterações, o sistema pode ser expandido para outros tipos de dados. Talvez a característica mais importante do *software* seja a modularidade e facilidade de expansão. É evidente que as necessidades de um usuário do sistema serão diferentes de outros usuários: alguns podem necessitar de uma quantidade maior de dados, cujo servidor SQLite, não seja capaz de gerir, outros ainda podem precisar de mais capacidade de processamento/armazenamento do que um Raspberry Pi pode fornecer, o *software* do concentrador pode ser utilizado em um computador x86, dentre muitas outras opções que podem ser feitas sem grandes alterações.

A segurança na rede sem fio no projeto do *software* foi assegurada, já que o padrão Zigbee é fraco em relação a sua implementação. É interessante notar o descaso que muitos fabricantes fazem com a segurança dos usuários em nome da facilidade de fabricação e da comodidade do usuário. Elas utilizam senhas iguais em todos os produtos que fazem uso do padrão Zigbee, o que torna fácil conectar e enviar comandos em uma rede sem fio utilizando este padrão em um produto comercial. Além disso, quando falhas em seus produtos são descobertas, elas não oferecem suporte, deixando um produto que pode ser altamente vulnerável à ataques cibernéticos com o usuário, às vezes pouco tempo depois da compra.

Foi realizado o projeto de uma segunda versão do *hardware* dos nós, que diminui seu tamanho e o consumo de energia em 48%, sua fabricação será feita em um trabalho futuro.

5 Resultados

Neste capitulo são detalhados e discutidos testes que foram feitos com a rede e os protótipos. Inicialmente um teste de conexão, para mostrar os rádios XBee sendo conectados um a um e a consequente formação da rede *mesh*. Posteriormente uma comparação na medição de um nó com um osciloscópio comercial. Um teste de campo, instalando o nó sensor em um painel solar ligado a uma bomba d'água de corrente contínua, demonstrando as possíveis interfaces de visualização de dados. Por fim um teste de consumo de energia do nó, comparando o consumo estimado com o real, além da eficiência da fonte em diferentes níveis de tensão de alimentação.

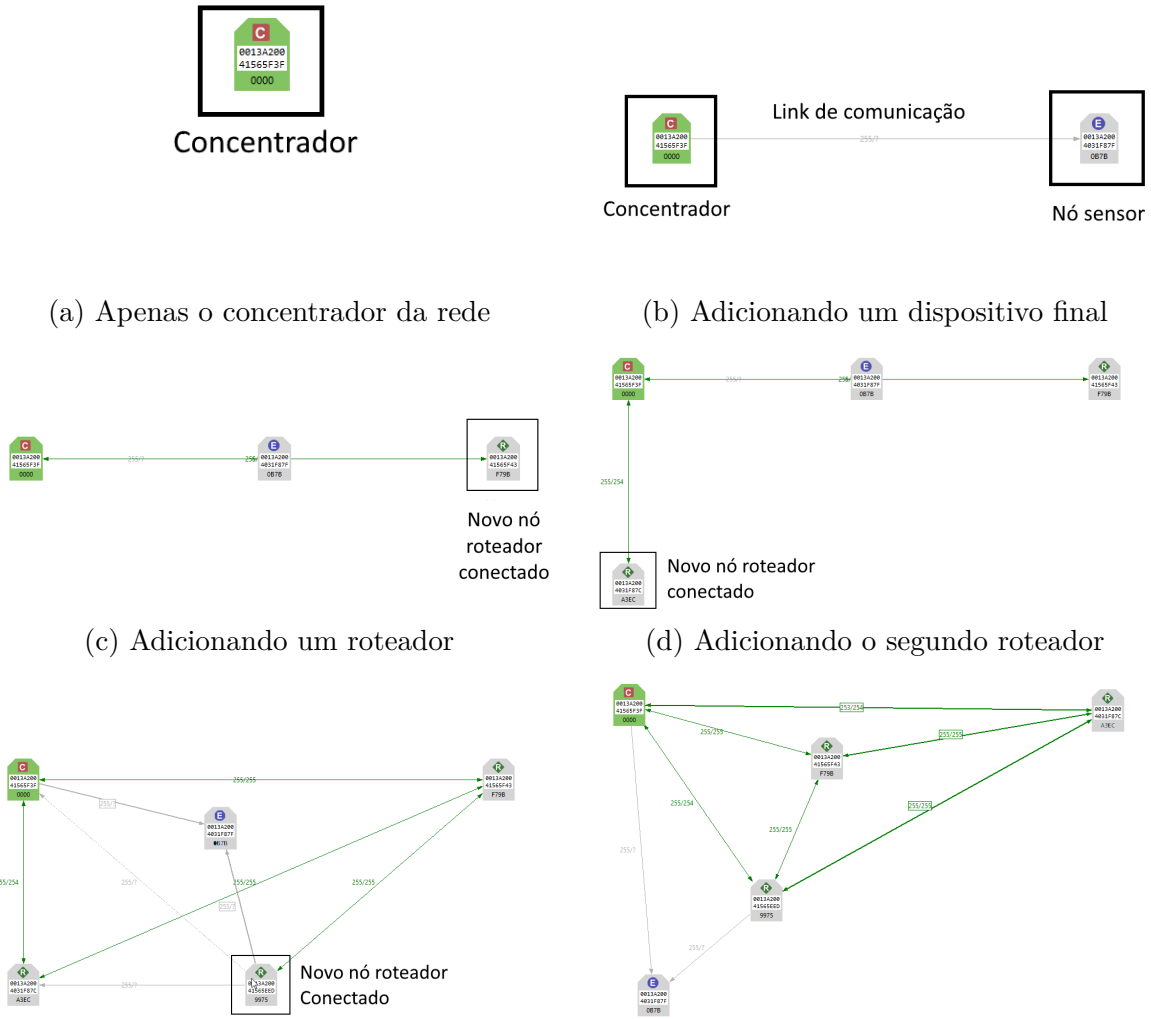
5.1 Conexão dos nós

A rede é criada pelo concentrador. Assim cada vez que um nó é ligado ele procura nos canais configurados por um concentrador na mesma rede. Depois de encontrá-lo trocam chaves de segurança AES para efetuar futuras comunicações.

Utilizando o *software* XCTU, fornecido pela empresa Digi, é possível visualizar a conexão de cada um dos nós no sistema e a formação da rede *mesh*. Desta maneira, na Figura 48, cada nó foi sendo adicionado um a um e uma imagem do estado atual da rede salva, sendo possível, desta maneira, visualizar a construção da rede *mesh* com a formação das ligações de comunicação entre os diversos nós.

Inicialmente na Figura 48a apenas o coordenador na rede está ligado. Na Figura 48b é adicionado um dispositivo final, que não pode rotear pacotes, repare que a seta que representa a comunicação entre os dispositivos aponta exclusivamente para o dispositivo final, indicando que ele não pode rotear pacotes. Nas Figuras 48c, 48d e 48e são adicionados dispositivos configuradores como roteadores. Repare que as setas utilizadas apontam para ambos os lados, indicando a função de roteador de pacotes. Já na Figura 48f os 4 nós e o concentrador são rearranjados, ficando claras as ligações da comunicação em malha, onde, por eles estarem perto uns dos outros, possuem ligações com os demais roteadores e com o concentrador, com exceção do dispositivo final, que apenas faz duas ligações.

Desta maneira, visualizamos que a rede *mesh* pode se montar automaticamente, com as conexões entre os nós possibilitando a retransmissão de pacotes entre eles.



(a) Apenas o concentrador da rede

(b) Adicionando um dispositivo final

(c) Adicionando um roteador

(d) Adicionando o segundo roteador

(e) Adicionando o terceiro roteador

(f) Rede *mesh* completa, com 3 roteadores e um dispositivo final

Figura 48 – Formação da rede sem fio Zigbee visualizada pelo *software* XCTU.

5.2 Validação das medições do sistema

Para validar a acurácia das medidas de tensão e corrente dos nós, foi feito um teste com o emulador de painel solar Ametek TerraSAS ETS 600/25, exibido na Figura 49, com as especificações detalhadas na Tabela 12. Para as medidas foi utilizado o osciloscópio LeCroy MSO 44MXs-B, mostrado na Figura 50, com uma ponteira de corrente AP15 que possui uma sensibilidade de 10 mA por divisão e acurácia de 1%. Um inversor, com método de MPPT de perturbação e observação desenvolvido por Reis(78), foi utilizado para enviar a energia gerada pelo painel para a rede elétrica. Os testes foram efetuados em 3 diferentes situações simuladas, uma no meio da manhã, outra logo antes do máximo de incidência solar e, por último, no fim da tarde.

Figura 49 – Emulador de painel solar Ametek TerraSAS ETS 600/25.

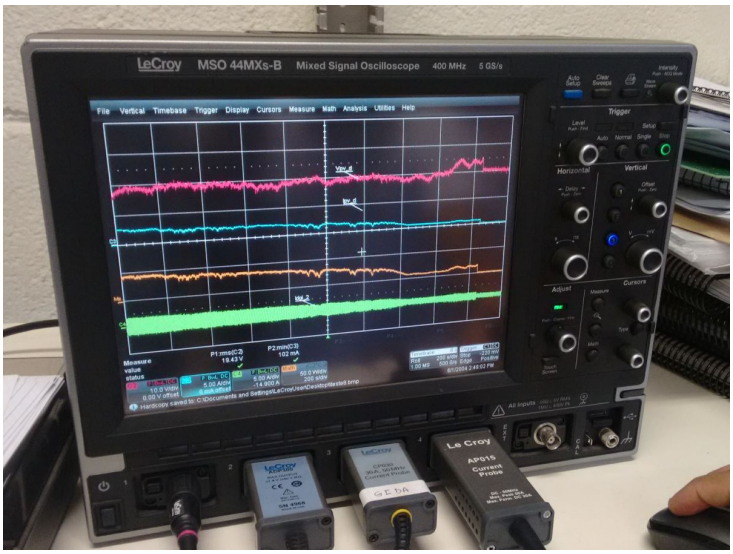


Figura 50 – Osciloscópio LeCroy MSO 44MXs-B.

Tabela 12 – Características utilizadas no emulador de painel solar:

Parâmetro	Valor
Tensão de circuito aberto	18.93 V
Corrente de curto circuito	8.57 A
Tensão de operação ótima	16.30 V
Corrente de operação ótima	8.12 A

Os resultados de tensão e corrente são exibidos na Figura 51 e da potência no Figura 52. As diferenças entre os sinais podem ser explicadas pela maior taxa de amostragem do osciloscópio e maior ruído nas leituras. A oscilação na tensão é devido ao algoritmo de MPPT, que procura pelo melhor ponto de operação. A energia total gerada medida pelo sistema foi de 70,7 Wh e pelo osciloscópio foi de 69,8 Wh, uma diferença de 1,3%.

A importância deste teste é demonstrar que utilizando componentes de baixíssimo

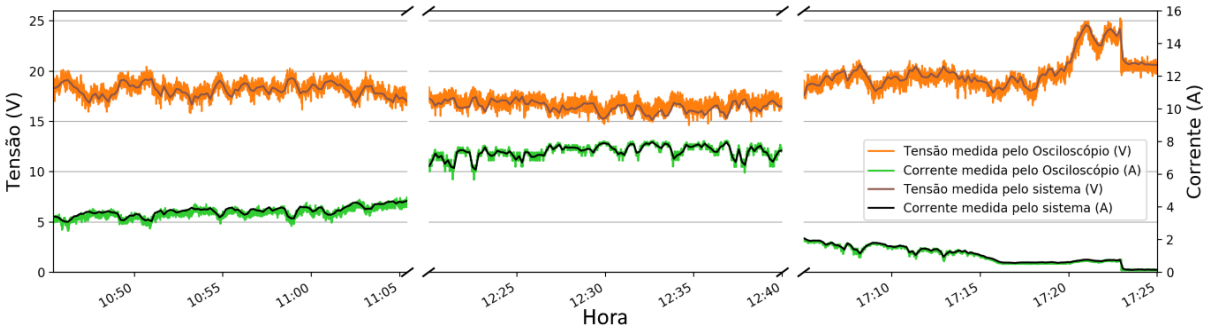


Figura 51 – Tensão e corrente medidas pelo nó do sistema e utilizando o osciloscópio.

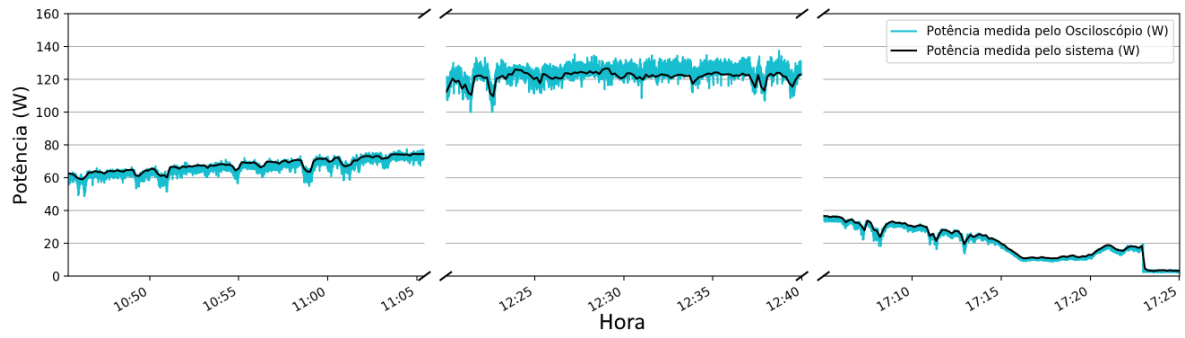


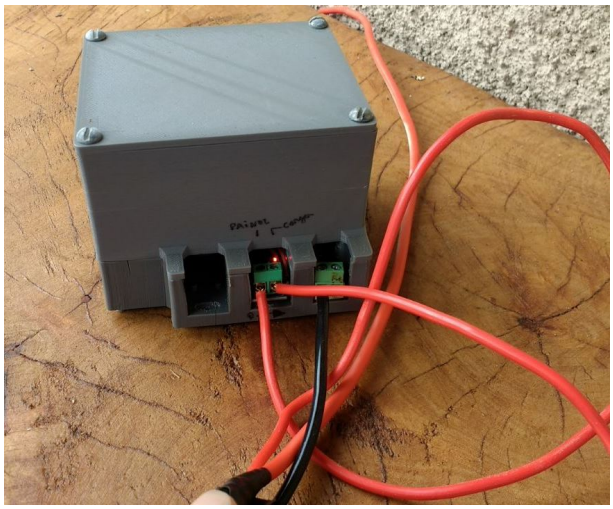
Figura 52 – Potência medida pelo nó do sistema e utilizando o osciloscópio.

custo é possível replicar experimentos e medidas de equipamentos de custo muito mais elevado.

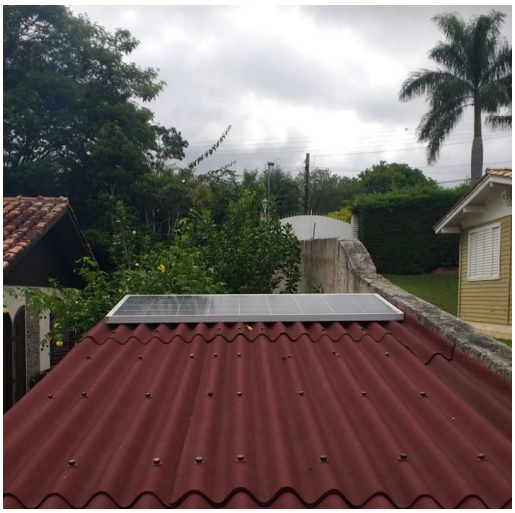
5.3 Estudo de campo

O sistema foi instalado em uma bomba d'água Shurflo 8000 de 12V alimentada por um painel solar Yingli de 150 W, modelo YL150P-17B. Ele foi mantido ligado por vários dias e os dados salvos localmente para visualização pela interface web e envio dos bancos de dados ao serviço em nuvem. O nó de medição ligado entre a saída do painel e a bomba d'água é mostrado na Figura 53a e o painel instalado no telhado na Figura 53b.

Os dados de 3 dias seguidos com bastante sol são mostrados nas Figuras 54, que faz uso da interface web do sistema, e nas Figuras 55 e 56, que foram geradas através da aplicação desktop, salvando apenas o gráfico de geração.



(a) Nó instalado entre a saída do painel solar e a bomba d'água.



(b) Painel solar instalado no telhado.

Figura 53 – Sistema instalado em campo.

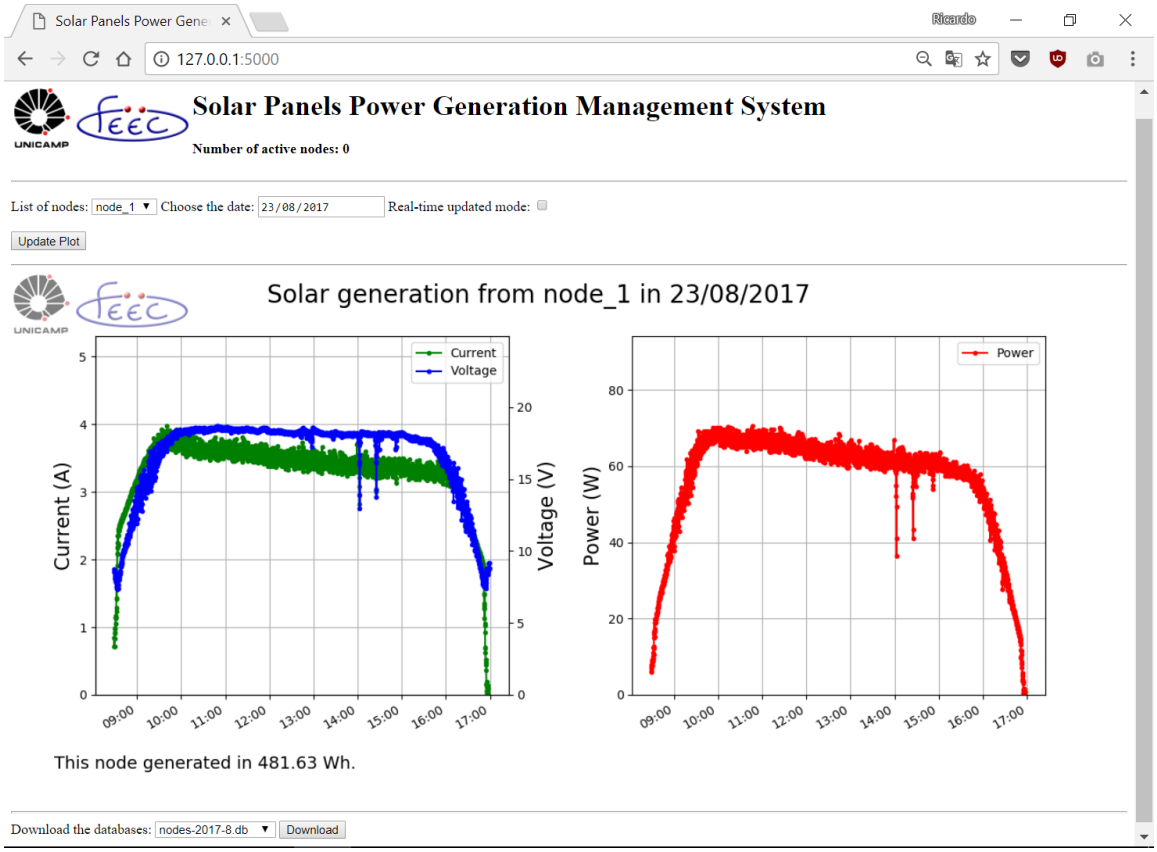


Figura 54 – Geração medida pelo sistema no dia 23/08/2017.

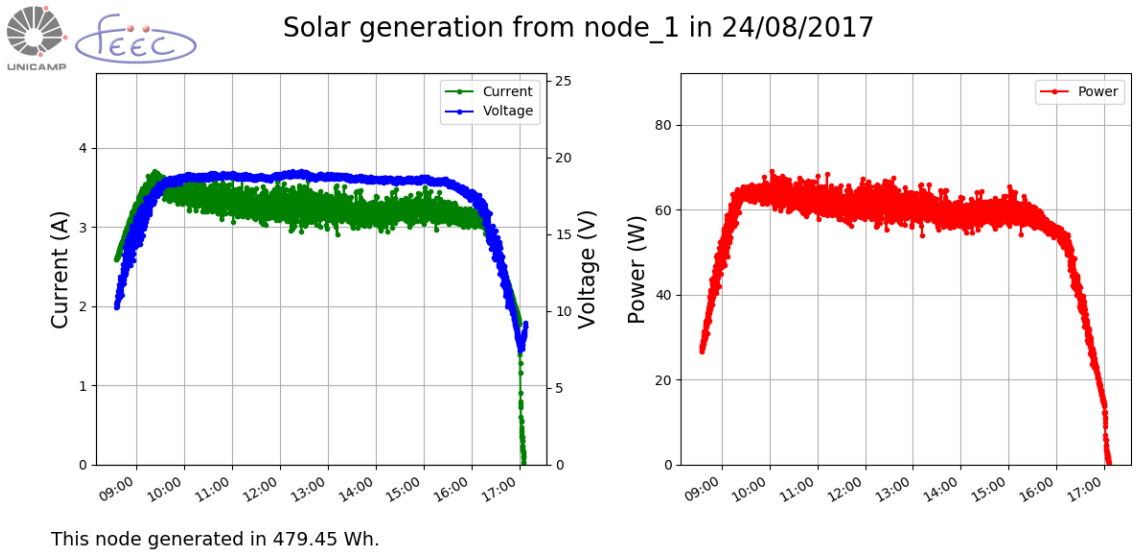


Figura 55 – Geração medida pelo sistema no dia 24/08/2017.

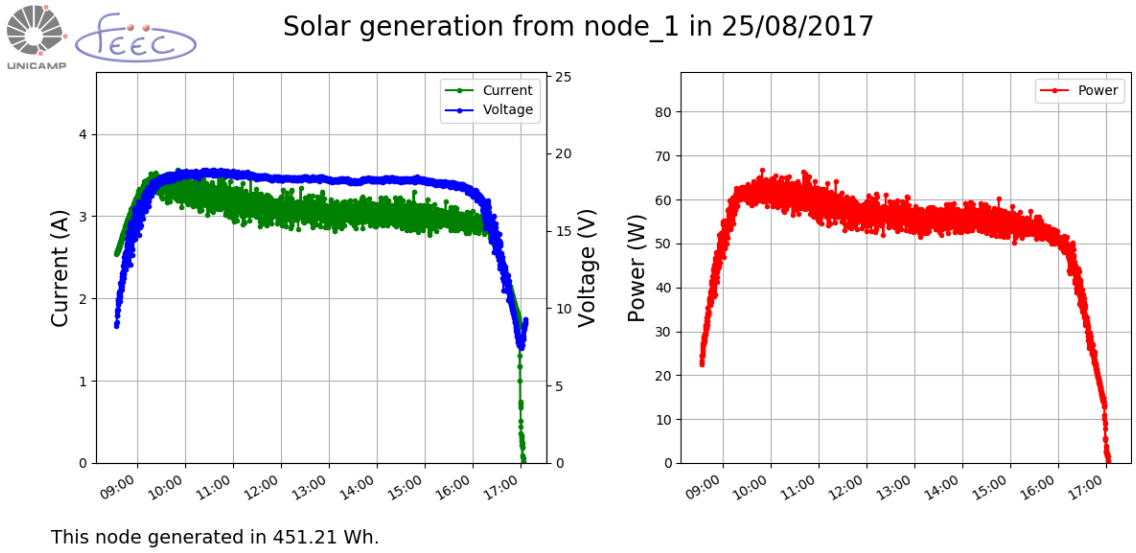


Figura 56 – Geração medida pelo sistema no dia 25/08/2017.

O sistema também envia informações para a nuvem, utilizando o serviço Ubidots. Os dados do dia 25/08/2015 são exibidos nas Figuras 57 e 58.

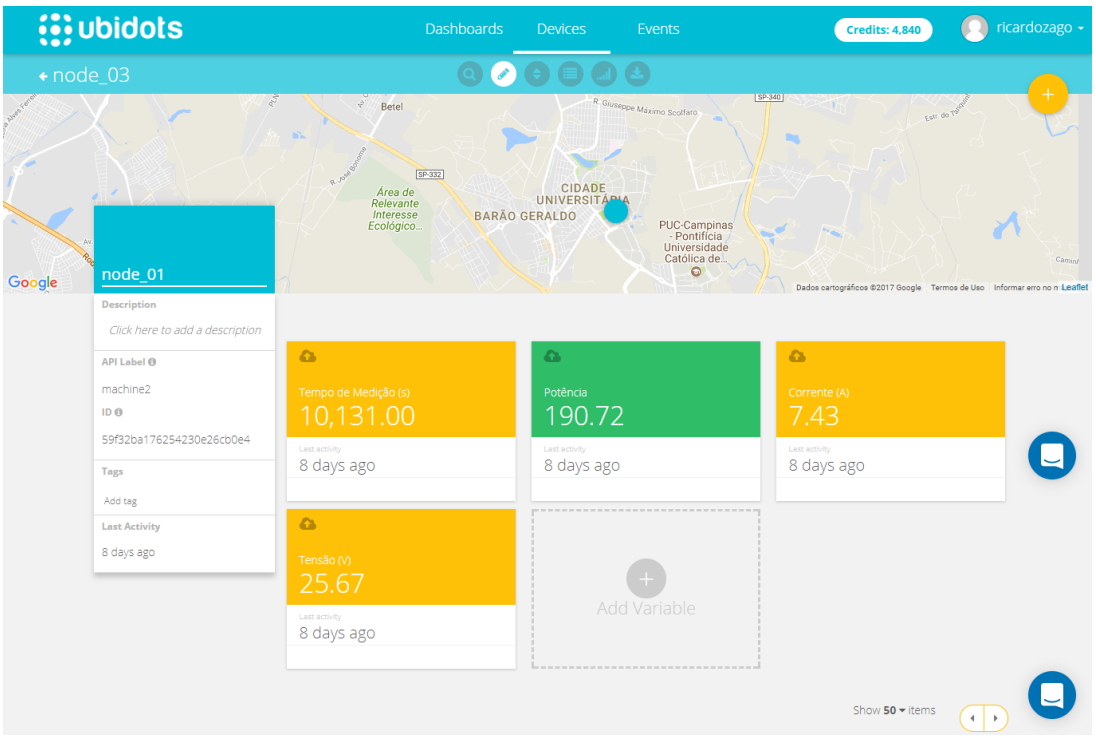


Figura 57 – Página Web do Ubidots com dados enviados por um nó da rede.

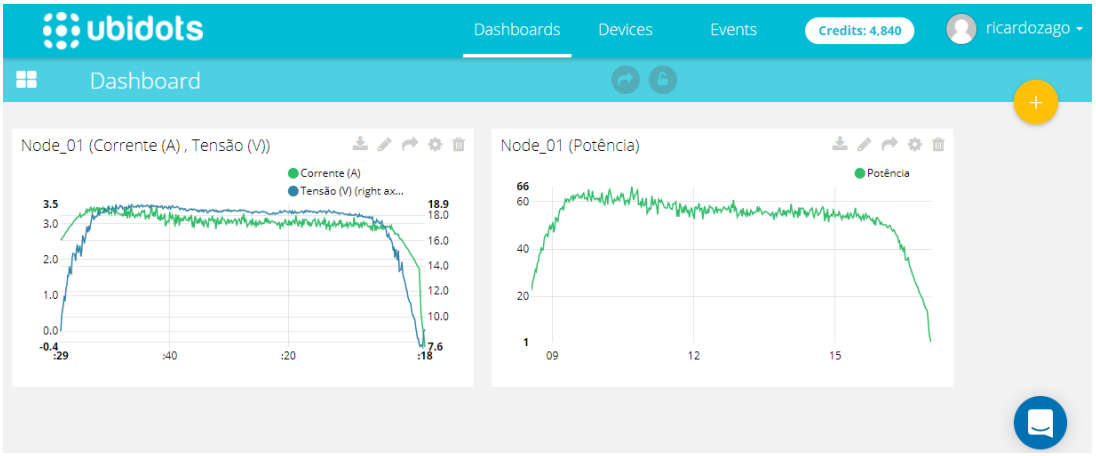
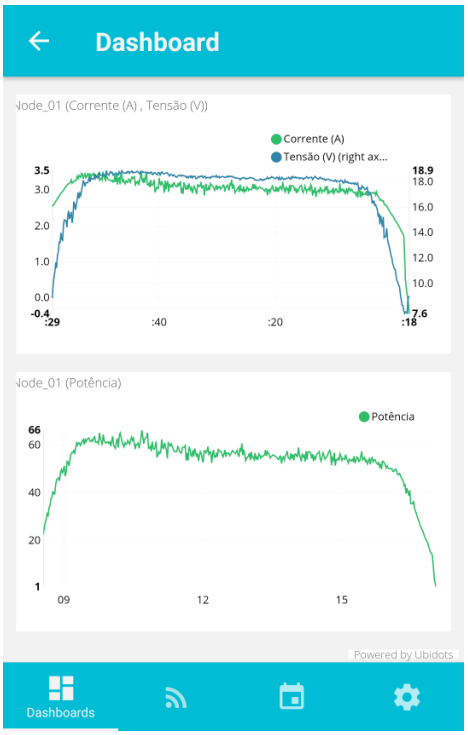


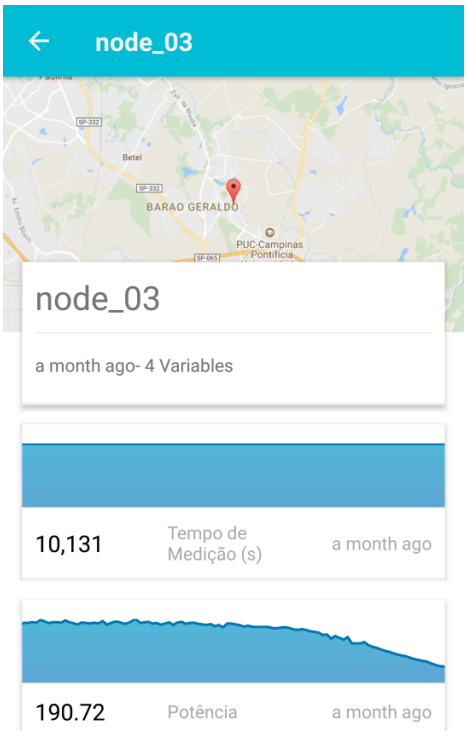
Figura 58 – Página Web do Ubidots com painel de informações exibindo informações de tensão, corrente e potência de um painel.

O Ubidots possui um aplicativo para Smartphones[1], os dados de geração exibidos pelo aplicativo são mostrados nas Figuras 59a, 59b e 59c.

[1] Disponível na Google Playstore em <https://play.google.com/store/apps/details?id=com.ubidots.ubiapp>.



(a) Dashboard do Ubidots com a geração do dia do painel.



(b) Posição geográfica do painel e suas 500 últimas leituras.

(c) Resto das 500 últimas leituras do painel.

Figura 59 – Aplicativo Ubidots para *Smartphones* Android.

5.4 Consumo de energia medido do nó

Anteriormente o consumo do nó com base nas especificações dos componentes foi estimado e com o protótipo montado este consumo pode ser medido. Foram feitos ensaios utilizando uma fonte no laboratório para simular o painel solar com as tensões de 9 V, 14 V, 20 V e 28 V. O concentrador foi ativado, permitindo com que ele recebesse os dados de medição, desta maneira simulando o nó em operação.

O método para efetuar as medições de consumo individuais de cada componente é medir o consumo total do sistema, sendo alimentado sem o conversor DC-DC, e remover um a um cada componente, desta maneira é possível aferir seu consumo tomando o consumo com o item e subtraindo o consumo com o item removido. Este estudo é demonstrado na Tabela 13.

Tabela 13 – Consumo do nó removendo componentes um a um.

Item removido	Tensão (V)	Corrente (mA)	Potência (mW)	Potência em 5V sem perda do regulador linear (mW)	Consumo do componente (mW)
Total	6,5	90,6 mA	588,9	453 mW	
Rádio	6,5	56,8 mA	369,2	284 mW	169 mW
Arduino	6,5	17,4 mA	113,1	87 mW	197 mW
Sensor Corrente	6,5	5,1 mA	33,2	25,5 mW	61,5 mW
Amp. Operacional	6,5	3,5 mA	22,8	17,5 mW	8,0 mW

O consumo total medido dos componentes foi de 453 mW, enquanto o estimado foi de 530 mW. Esta diferença ocorre devido a termos considerado sempre o pior caso na estimativa de consumo, com o microcontrolador em seu maior estado de consumo, o rádio sempre transmitindo, LEDs acesos, etc. O rádio é alimentado em 3,3 V, sendo utilizado um regulador linear para alimentá-lo a partir dos 5 V utilizados pelos demais componentes. Na Tabela 14 é mostrado o consumo individual de cada componente.

Tabela 14 – Consumo individual dos principais componentes dos nós

Componentes	Tensão (V)	Corrente (mA)	Potência (mW)
Arduino	5,0	39,4	197,0
Rádio	5,0	33,8	169,0
Sensor de corrente	5,0	12,3	61,5
Amplificado Operacional	5,0	1,6	8,0

O consumo do Arduino foi de 39,4 mA, enquanto o estimado foi de 50 mA. Segundo estudos do consumo do Arduino (67), 35 mA corresponde a gastos fixos ligados ao con-

versor UART-USB e LEDs, que não podem ser desativados na placa de desenvolvimento, mas que são desnecessários quando utilizados em um sistema apenas com o microcontrolador. Desta maneira, o consumo do sistema poderia ser diminuído destes 35 mW, que representam 228 mW quando o Arduino é alimentado com 6,5 V. O consumo do rádio XBee foi estimado em 225 mW e o real de medido foi de 169,0 mW. Provavelmente esta diferença ocorre devido ao transmissor não estar sempre transmitindo informações e a potência estimada fazia esta consideração. O rádio é alimentado em 3,3 V, é utilizado um regulador linear para sua alimentação a partir dos 5 V fornecidos, desta maneira, seu consumo, desconsiderando as perdas no regulador, é de 111,5 mW em 3,3 V.

Foram feitos ensaios para medir a eficiência do conversor DC-DC com a tensão de alimentação do sistema em 9 V, 14 V, 20 V e 28 V. Os dados são mostrados na Tabela 15. A eficiência ficou aquém no esperado, que era 80%, estando entre 63% e 70%. Como utilizamos um regulador numa placa já montada, com resistências e indutâncias calculadas para correntes maiores, provavelmente este fato diminuiu a eficiência do sistema.

Tabela 15 – Variação da eficiência do conversor DC-DC alterando a tensão de alimentação.

Tensão (V)	Corrente (mA)	Consumo (mW)	Eficiência
Pós DC-DC (6.5)	90,6	588,9	
9	93,8	844,2	0,70
14	60,7	849,8	0,69
20	44,2	884,0	0,67
28	33,3	932,4	0,63

5.5 Conclusões do capítulo

A solução de rede sem fio se mostrou adequada, enviando as informações dos nós sensores ao concentrador com confiabilidade e sendo capaz de criar e gerenciar a rede *mesh* automaticamente.

Em relação aos nós sensores, foi verificado sua acurácia na medição. Em comparação a um osciloscópio comercial, demonstrando uma diferença de apenas 1,3% no total gerado durante 3 situações diferentes: no início da manhã, pouco antes da máxima incidência solar e no final da tarde.

No estudo de consumo de energia do sistema e da eficiência do conversor DC-DC, foram analisados o consumo individual de cada um dos componentes, como o microcontrolador, rádio e sensor de corrente. Também foi medida a eficiência do conversor DC-DC, que se mostrou abaixo do esperado. Isso se deve a ter sido utilizada uma versão de prateleira, cujas resistências e indutâncias foram calculadas para a fonte fornecer uma maior

potência para a carga, não sendo adequados para a pequena corrente de alimentação dos componentes dos nós.

6 Conclusões finais

Este trabalho apresenta o desenvolvimento de uma rede de sensores sem fio (RSSF) para medição de geração de energia na saída do painel solar, antes de eventuais reguladores de carga ou inversores, onde o nó sensor é alimentado pela energia do próprio painel. O projeto abrange uma revisão bibliográfica de rede de sensores sem fio, de sistemas de geração solar fotovoltaicos, seus componentes como o controlador de carga, inversor e métodos de MPPT, soluções comerciais de medição, resolução normativa da ANEEL e computação na nuvem. A Resolução Normativa Nº 482 de 2012 foi primordial para o crescimento da instalação de sistemas fotovoltaicos no Brasil.

Uma discussão dos passos que devem ser seguidos no desenvolvimento de um dispositivo baseado em um sistema embarcado ou dispositivo de internet das coisas foi feita, focando nos componentes essenciais. Desta maneira, espera-se que este estudo possa servir como base para outros projetos de sistemas embarcados, mesmo que não relacionados a medição de energia. Foram discutidos motivos de escolha de componentes como processadores, microcontroladores, suas interfaces de comunicação com periféricos UART, I^2C e SPI. Também foi feito um estudo dos sistemas operacionais para o concentrador e os nós sensores.

O desenvolvimento do *hardware* utilizou somente componentes de baixo custo disponíveis no Brasil. Esta escolha se mostrou acertada, pois a compra destes componentes foi facilitada e eles supriram bem as necessidades. O protótipo do nó sensor possui um consumo de energia de 850 mW quando alimentado com 14 V. Este valor está dentro dos objetivos do projeto, que era de no máximo 1,0% da potência nominal do painel, considerando um momento com bastante sol.

Ensaios comparando a medição do nó sensor com um osciloscópio LeCroy MSO 44MXs-B, utilizando uma ponteira de medição de corrente, demonstraram uma diferença do total de energia gerada mensurada entre os sistemas de apenas 1,3%, o que demonstra a capacidade do sistema de medir com acurácia a geração solar do painel. O sistema também foi instalado em um painel solar que alimentava uma bomba d'água de corrente contínua, sem uso de baterias ou de inversor. O sistema foi capaz de medir a geração e totalizar os dados durante todo o dia.

Na questão custo, o objetivo inicial era que o protótipo tivesse custo de no máximo 10% do painel solar. Este objetivo considerou o preço do painel no final do ano de 2015, como foi demonstrado com dados da Bloomberg na introdução deste trabalho, o custo dos módulos solares caiu aproximadamente 40% neste no período entre a criação dos objetivos e a conclusão do trabalho, portanto o objetivo de custo não foi atingido, como mostrado

na Tabela 16.

Tabela 16 – Custo dos componentes dos nós em dólares.

Componente	Custo em dólar
Microcontrolador (ATmega328P)	US$ 1,96
XBee (XB24CAWIT-001)	US$ 17,50
Sensor de corrente (ACS712)	US$ 1,80
Regulador Step Down (LM2576HV)	US$ 3,41
Demais componentes (placas, resistores, diodos, etc)	US$ 7,00
Caixa	US$ 2,00
Total	US$ 33,67
Painel solar 330W Canadian Solar CS6U-330P	US$ 226,56 [1]

No entanto, considerando que mais da metade do custo do nó é relacionado a solução de rádio utilizada, uma mais barata, mesmo perdendo um pouco o alcance, poderia ser uma solução interessante para estar dentro do objetivo de preço.

O *software* desenvolvido para o protótipo do sistema consistiu no programa em Python que é carregado no concentrador, recebendo os dados da rede de sensores sem fio e armazenando localmente e num serviço em nuvem, como o Ubidots. A escolha da linguagem Python para este servidor se mostrou acertada, pois ela permitiu portabilidade entre a plataformas, o sistema do concentrador pode ser executado tanto em Windows quanto em Linux.

O desenvolvimento do *firmware* utilizou a IDE Arduino, que emprega a linguagem C++ com algumas adições. O *firmware* possui funções de proteção. Caso, por exemplo, o nó deixe de detectar o concentrador, ele irá aumentar o tamanho do vetor de dados até retomar a comunicação com o concentrador, não perdendo assim leituras. Tanto o concentrador, quanto os nós sensores, possuem um padrão de comunicação que foi projetado para ser expansível. Desta maneira, com mínimas alterações o sistema pode ser expandido para outros tipos de dados. Talvez a parte mais importante do *software* seja esta, a modularidade e facilidade de expansão.

Um ponto a se destacar na questão do *software*, é o estado da segurança dos produtos de Internet das Coisas já vendidos aos usuários, que no geral possuem inúmeras vulnerabilidades de segurança não corrigidas. Além disso, protocolos como o Zigbee necessitam de uma chave de segurança para garantir o sigilo da comunicação. Esta chave em produtos comerciais, no geral, é a mesma, recomendada pelo consórcio criador do padrão. Isto ocorre para facilitar a fabricação do dispositivo, já que não é necessário personalizar um item, o que dificulta a produção, e comodidade para o usuário, já que basta ligar o dispositivo e ele irá se conectar a rede Zigbee doméstica, sem uma etapa a mais na configuração. Estas falhas foram mitigadas, trocando a senha padrão, desta maneira apenas

com a nova senha é possível se conectar ao concentrador e enviar dados.

O sistema é útil a seu usuário, pois permite visualizar tendências ao longo do tempo e a real geração do painel. Por exemplo, o usuário compra painéis de 150 W, mesmo no horário de maior geração do dia, provavelmente este valor não será atingido, além disso a geração varia muito durante o dia. Com o sistema, seu usuário pode sabe realmente quanto seus painéis estão gerando, tomando como base esta informação para futuras expansões. Também é possível, utilizando o sistema por um período prolongado, verificar o envelhecimento do painel, se sua geração decaiu, ou não, durante o tempo.

A instalação de painéis sem nenhum tipo de medição é uma maneira extremamente passiva de uso. Por exemplo, o relógio instalado pela concessionária não mede o total gerado pelo painel, apenas a quantidade de energia enviada para a rede e a recebida. Caso o painel passe a gerar menos, seja por um sombreamento parcial, seja por um defeito ou problema na instalação, o usuário pode relacionar o aumento na conta de luz não a este problema, mas a uma mudança de hábito da família. Mesmo um inversor inteligente apenas verificaria uma diminuição na geração do sistema, sem identificar qual painel está com defeito, nosso sistema possui inteligência para detectar o local do problema, já que faz a medição individual dos painéis. Mesmo um problema simples como sombreamento parcial, que pode ser detectado por inspeção visual, já haveria vantagem, pois, o usuário seria notificado, percebendo o problema rapidamente.

O projeto pode trazer benefícios que se conhecem como empoderamento da sociedade, que deixa de ter uma atitude exclusivamente passiva e passa a ter um papel ativo. O usuário deixa de depender exclusivamente da concessionária de energia para todos os seus gastos e passando a gerar, se não tudo, uma parte da energia que consome e, além disso, ter como mensurar esta energia gerada.

6.1 Trabalhos Futuros

A seguir sugerimos algumas ações que possibilitam expandir o atual trabalho e seu escopo de funcionamento:

- Metade do custo do projeto está na solução de rádio utilizada, logo, propomos alterar a solução de rádio para uma de menor custo;
- Instalar o sistema em uma maior quantidade de painéis e armazenar os dados durante o período. Procurar tendências, possivelmente aplicando algoritmos de aprendizado de máquina para melhorar a eficiência dos painéis quando se encontrar problemas como sombreamento parcial, ângulo ruim, dentre outros. Também é possível tomar como foi o dia no lugar que o painel está instalado (utilizando serviços online) e comparando com dias anteriores com o mesmo padrão climático, detectando assim

problemas com o painel. Esta solução seria mais rápida do que a anterior, pois os dados necessários seriam menores;

- Conforme se encontre problemas, avisar ao usuário, seja por e-mail, seja por SMS, o que o serviço Ubidots já permite, faltando apenas o algoritmo que detecte falhas.

- Adaptar o sistema com sensores de temperatura, vazão e nível d'água e utilizá-lo para análise de aquecimento d'água em sistemas que utilizam o sol para aquecimento.

- O sistema desenvolvido mensura a geração DC do painel. A medição AC, pós inversor, traria a possibilidade de, além de monitorar a geração do painel, também a eficiência do inversor e o consumo das cargas. Para adicionar esta característica ao projeto existem circuitos integrados especializados como os da série ADE da Analog Devices[2] e a série MCP da Microchip[3]. O microcontrolador possui portas I^2C e SPI que poderiam ser utilizadas para fazer a interface de comunicação.

- O sensor de corrente utilizado nos nós pode medir a geração de energia em ambos os sentidos. Desta maneira, além de medir a o consumo, ele poderia também poderia medir a potência enviada. O sistema desenvolvido poderia ser modificado para ser uma rede de sensores sem fio para medição de fluxo de corrente CC (ou CA, fazendo a modificação sugerida anteriormente).

- Utilizando o conceito de Blockchain (79), seria possível utilizar nós espalhados por diversas residências para implementar um sistema de troca de energia. Sistemas com este conceito já foram implementados em diversas cidades do mundo, como em Nova Iorque(80) e na Austrália (81).

[2] <http://www.analog.com/en/products/analog-to-digital-converters/integrated-special-purpose-converters/energy-metering-ics.html>.

[3] <http://www.microchip.com/wwwproducts/en/MCP3905A>.

Publicações

Artigos publicados

- **ZAGO, R. M.**; FRUETT, F. A low-cost solar generation monitoring system suitable for internet of things. In: 2017 2nd International Symposium on Instrumentation Systems, Circuits and Transducers (INSCIT). [S.l.]: IEEE, 2017.

Referências

1 FRERIS, L.; INFIELD, D. *Renewable Energy in Power Systems*. [S.l.]: John Wiley & Sons, 2008. Citado 2 vezes nas páginas 18 e 23.

2 INTERNATIONAL ENERGY AGENCY. *Key World Energy Statistics 2017*. 2017. Citado 2 vezes nas páginas 18 e 25.

3 INTERNATIONAL ENERGY AGENCY. *Snapshot of global photovoltaic markets*. 2015. Disponível em: <http://www.iea-pvps.org/fileadmin/dam/public/report/PICS/IEA-PVPS_-___A_Snapshot_of_Global_PV_-_1992-2015_-_Final_2_02.pdf>. Citado na página 18.

4 AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA. *Capacidade de Geração do Brasil*. 2016. Disponível em: <http://www2.aneel.gov.br/aplicacoes/capacidadebrasil/capacidadebrasil.cfm>. Citado 2 vezes nas páginas 18 e 25.

5 FU, R. et al. *U.S. Solar Photovoltaic System Cost Benchmark: Q1 2017*. 2017. NREL Publications Database. Disponível em: <https://www.nrel.gov/docs/fy17osti/68925.pdf>. Citado na página 18.

6 PROJETOS solares surpreendem em leilão de energia. 2014. Site da Folha de São Paulo. Disponível em: <http://www1.folha.uol.com.br/mercado/2014/10/1541707-projetos-solares-surpreendem-em-leilao-de-energia.shtml>. Citado na página 18.

7 BRASIL, C. I. do. *Energia solar fotovoltaica pode crescer mais de 300% até o fim do ano, diz setor*. 2017. Agência Brasil. Disponível em: <http://agenciabrasil.ebc.com.br/economia/noticia/2017-07/energia-solar-fotovoltaica-pode-crescer-mais-de-300-ate-o-fim-do-ano-diz>. Citado na página 18.

8 AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA. *RESOLUÇÃO NORMATIVA Nº 482, DE 17 DE ABRIL DE 2012*. [S.l.], 2012. Citado 2 vezes nas páginas 19 e 36.

9 MEARES, L. *Product How-To: Solar power anti-islanding and control*. 2012. Website EDN Network. Disponível em: <http://www.edn.com/design/systems-design/4391907/Product-How-To--Solar-power-anti-islanding-and-control>. Citado na página 19.

10 BASSI, S. *IBM transforma Internet das Coisas em investimento estratégico bilionário*. 2015. Computer World. Disponível em: <http://computerworld.com.br/ibm-transforma-internet-das-coisas-em-investimento-estrategico-bilionario>. Citado na página 19.

11 FINGAS, J. *Samsung pours $1.2 billion into the Internet of Things*. 2016. Site Engadget. Disponível em: <https://www.engadget.com/2016/06/21/samsung-invests-in-internet-of-things/>. Citado na página 19.

12 BOHN, D. *Google Home: a speaker to finally take on the Amazon Echo*. 2016. Website The Verge. Disponível em: <http://www.theverge.com/2016/5/18/11688376/google-home-speaker-announced-virtual-assistant-io-2016>. Citado na página 19.

13 RAWAT, P. et al. Wireless sensor networks: a survey on recent developments and potential synergies. *The Journal of Supercomputing*, Springer Nature, v. 68, n. 1, p. 1–48, oct 2013. Disponível em: <https://doi.org/10.1007/s11227-013-1021-9>. Citado na página 21.

14 RUIZ-GARCIA, L. et al. Testing ZigBee motes for monitoring refrigerated vegetable transportation under real conditions. *Sensors*, MDPI AG, v. 10, n. 5, p. 4968–4982, Maio 2010. Disponível em: <https://doi.org/10.3390/s100504968>. Citado na página 22.

15 CASEY, K.; LIM, A.; DOZIER, G. A sensor network architecture for tsunami detection and response. *International Journal of Distributed Sensor Networks*, SAGE Publications, v. 4, n. 1, p. 27–42, jan 2008. Disponível em: <https://doi.org/10.1080/15501320701774675>. Citado na página 22.

16 LI, X. et al. Design and implementation of a wireless sensor network-based remote water-level monitoring system. *Sensors*, MDPI AG, v. 11, n. 12, p. 1706–1720, jan 2011. Disponível em: <https://doi.org/10.3390/s110201706>. Citado na página 22.

17 MUDUMBE, M. J.; ABU-MAHFOUZ, A. M. Smart water meter system for user-centric consumption measurement. In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. IEEE, 2015. Disponível em: <https://doi.org/10.1109/indin.2015.7281870>. Citado na página 22.

18 JALALI, R.; EL-KHATIB, K.; MCGREGOR, C. Smart city architecture for community level services through the internet of things. In: *2015 18th International Conference on Intelligence in Next Generation Networks*. IEEE, 2015. Disponível em: <https://doi.org/10.1109/icin.2015.7073815>. Citado na página 22.

19 SMETS, A. et al. *Solar Energy - The Physics and Engineering of Photovoltaic Conversion, Technologies and Systems*. [S.l.]: UIT Cambridge, 2015. Citado 4 vezes nas páginas 23, 25, 29 e 30.

20 U.S. ENERGY INFORMATION ADMINISTRATION. *Fossil fuels still dominate U.S. energy consumption despite recent market share decline*. 2016. Disponível em: <http://www.eia.gov/todayinenergy/detail.cfm?id=26912>. Citado na página 24.

21 INTERNATIONAL ENERGY AGENCY. *World Energy Outlook 2015*. 2015. Disponível em: <http://www.worldenergyoutlook.org/weo2015/>. Citado na página 24.

22 U.S. ENERGY INFORMATION ADMINISTRATION. *What is U.S. electricity generation by energy source?* 2015. Disponível em: <https://www.eia.gov/tools/faqs/faq.cfm?id=427&t=3>. Citado na página 25.

23 TSAO, J.; LEWIS, N.; CRABTREE, G. Solar faqs. *US department of Energy*, 04 2006. Citado na página 25.

24 FRAAS, L.; PARTAIN, L. *Solar Cells and Their Applications*. 2ª. ed. [S.l.]: Wikey, 2010. Citado 2 vezes nas páginas 26 e 27.

25 HALLIDAY; RESNICK; WALKER, J. *Fundamentos de Física, volume 4: óptica moderna*. 8. ed. [S.l.]: LTC, 2008. v. 4. Citado 2 vezes nas páginas 26 e 27.

26 AMERICAN SOCIETY FOR TESTING AND MATERIALS (ASTM) TERRESTRIAL REFERENCE SPECTRA FOR PHOTOVOLTAIC PERFORMANCE EVALUATION. *Reference Solar Spectral Irradiance: Air Mass 1.5*. [S.l.]. Citado na página 27.

27 RAZAVI, B. *Fundamentals of Microelectronics*. 2. ed. [S.l.]: Wiley, 2013. Citado na página 28.

28 SPECIFICATION for Solar Simulation for Photovoltaic Testing. [S.l.]: ASTM International. Citado na página 29.

29 REZK, H.; ELTAMALY, A. M. A comprehensive comparison of different MPPT techniques for photovoltaic systems. *Solar Energy*, Elsevier BV, v. 112, p. 1–11, feb 2015. Disponível em: <https://doi.org/10.1016/j.solener.2014.11.010>. Citado na página 31.

30 SEYEDMAHMOUDIAN, M. et al. A comparative study on procedure and state of the art of conventional maximum power point tracking techniques for photovoltaic system. *International Journal of Computer and Electrical Engineering*, International Academy Publishing (IAP), v. 6, n. 5, p. 402–414, 2014. ISSN 1793-8163. Disponível em: <https://doi.org/10.17706/ijcee.2014.v6.859>. Citado na página 32.

31 SUNNY WebBox with Bluetooth® Wireless Technology. 2011. Disponível em: <http://files.sma.de/dl/11567/WEBBOXBT-DAU111912W.pdf>. Citado 2 vezes nas páginas 32 e 33.

32 SMA Sunny WebBox Now Boasts Bluetooth® Technology. 2011. Disponível em: <http://www.sma-america.com/newsroom/current-news/news-details/news/10812-sma-sunny-webbox-now-boasts-bluetoothR-technology.html>. Citado na página 32.

33 PINTO, P. *Painéis solares podem ser alvo de ataques*. 2017. Disponível em: <https://pplware.sapo.pt/gadgets/hardware/paineis-solares-podem-alvo-ataques/>. Citado na página 33.

34 MOSTAFA, G.; KHAN, F. An efficient method of solar panel energy measurement system. *1st International Conference on the Developements in Renewable Energy Technology*, 2009. ISSN 978-984-33-0616-6. Citado na página 34.

35 KABALCI, E.; GORGUN, A.; KABALCI, Y. Design and implementation of a renewable energy monitoring system. In: *4th International Conference on Power Engineering, Energy and Electrical Drives*. [S.l.]: IEEE, 2013. Citado na página 35.

36 SIREGAR, S.; SOEGIARTO, D. Solar panel and battery street light monitoring system using GSM wireless communication system. In: *2014 2nd International Conference on Information and Communication Technology (ICoICT)*. [S.l.]: IEEE, 2014. Citado na página 35.

37 ERASMUS, Z.; BAGULA, A. Remote sensor network for off-grid renewable energy monitoring. In: *2016 IST-Africa Week Conference*. [S.l.]: IEEE, 2016. Citado na página 35.

38 DHOLE, S. V. et al. Review of solar energy measurement system. *International Journal on Recent and Innovation Trends in Computing and Communication (IJRITCC)*, v. 5, n. 1, p. 272 – 275, 2017. ISSN 2321-8169. Citado na página 35.

39 PATIL, S.; VIJAYALASHMI, M.; TAPASKAR, R. Solar energy monitoring system using iot. *Indian journal of scientific research*, v. 15, n. 2, p. 149–155, 2017. ISSN 2250-0138. Citado na página 35.

40 AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA. *RESOLUÇÃO NORMATIVA Nº 687, DE 24 DE NOVEMBRO DE 2015*. [S.l.], 2015. Citado na página 36.

41 CONVÊNIO ICMS 16, DE 22 DE ABRIL DE 2015. 2015. Disponível em: <https://www.confaz.fazenda.gov.br/legislacao/convenios/2015/cv016_15>. Citado na página 37.

42 KUMAR, S. P. et al. Smart health monitoring system of patient through IoT. In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. [S.l.]: IEEE, 2017. Citado na página 37.

43 CHANDRA, A. A. et al. Cloud based real-time monitoring and control of diesel generator using the IoT technology. In: *2017 20th International Conference on Electrical Machines and Systems (ICEMS)*. [S.l.]: IEEE, 2017. Citado na página 38.

44 RIOS, J.; ROMERO, C. A.; MOLINA, D. Instrumentation and control of a DC motor through the ubidots platform. In: *2015 Workshop on Engineering Applications - International Congress on Engineering (WEA)*. [S.l.]: IEEE, 2015. Citado na página 38.

45 ESCOBAR, L. J. V.; SALINAS, S. A. e-health prototype system for cardiac telemonitoring. In: *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. [S.l.]: IEEE, 2016. Citado na página 38.

46 BOLIVAR, L. E. P.; SILVA, G. A. da. Solar radiation monitoring using electronic embedded system raspberry pi database connection MySQL, ubidots and TCS-230 sensor. In: *2015 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*. [S.l.]: IEEE, 2015. Citado na página 38.

47 ANDERSON, C. *Drones go to work*. 2017. Harvard Business Review. Disponível em: <https://hbr.org/cover-story/2017/05/drones-go-to-work>. Citado na página 40.

48 CATSOULIS, J. *Designing Embedded Hardware*. 2. ed. [S.l.]: O'Reilly Media, 2009. Citado 2 vezes nas páginas 41 e 42.

49 OVERVIEW and Use of the PICmicro Serial Peripheral Interface. [S.l.]. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/spi.pdf>. Citado na página 42.

50 IRAZABAL, J.-M.; BLOZIS, S. *Application note I2C bus AN10216-01 I2C manual*. [S.l.], 2003. Disponível em: <http://www.nxp.com/documents/application_note/AN10216.pdf>. Citado na página 43.

51 CELLAN-JONES, R. *A 15 pound computer to inspire young programmers*. 2011. Website BBC. Disponível em: <http://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html>. Citado na página 44.

52 ATMEL. *8-bit AVR Microcontrollers - ATmega328/P - Datasheet Complete*. [S.l.], 2016. Citado 2 vezes nas páginas 47 e 48.

53 NXP. *Kinetis KL25 Sub-Family*. [S.l.], 2014. Disponível em: <http://www.nxp.com/assets/documents/data/en/data-sheets/KL25P80M48SF0.pdf>. Citado na página 47.

54 SMITH-ROSE, R. L. *Guglielmo Marconi Italian physicist*. 2016. Encyclopædia Britannica. Disponível em: <https://global.britannica.com/biography/Guglielmo-Marconi>. Citado na página 48.

55 OLIVE-DRAB. *SCR-536 Walkie Talkie*. 2016. Disponível em: <http://olive-drab.com/od_electronics_scr536.php>. Citado na página 48.

56 MOTOROLA MOBILITY. *What better way to discover Motorola's heritage than by exploring the stories behind some of our biggest innovations?* 2010. Disponível em: <https://www.motorola.com/us/consumers/about-motorola-us/About_Motorola-History-Timeline/About_Motorola-History-Timeline.html#1980>. Citado na página 48.

57 TONDARE, S. M.; PANCHAL, S. D.; KUSHNURE, D. T. Evolutionary steps from 1g to 4.5g. *International Journal of Advanced Research in Computer and Communication Engineering*, v. 3, n. 4, apr 2014. Citado na página 48.

58 DIGI INTERNATIONAL. *How to set up a ZigBee mesh network*. [S.l.], 2016. Disponível em: <https://docs.digi.com/display/XBeeZigBeeMeshKit/How+to+set+up+a+ZigBee+mesh+network>. Citado na página 52.

59 DIGI XBee® Zigbee Embedded Zigbee and Thread-ready RF modules provide OEMs with a simple way to integrate mesh technology into their application. Disponível em: <https://www.digi.com/products/xbee-rf-solutions/2-4-ghz-modules/xbee-zigbee>. Citado na página 55.

60 MIRA Ltd. *MISRA-C:2004 Guidelines for the use of the C language in Critical Systems*. 2004. Disponível em: <www.misra.org.uk>. Citado na página 60.

61 KOOPMAN, P. *A Case Study of Toyota Unintended Acceleration and Software Safety*. 2014. Disponível em: <https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf>. Citado na página 60.

62 FAIRCHIELD. *LM78XX / LM78XXA - 3-Terminal 1 A Positive Voltage Regulator*. [S.l.], 2014. Citado 2 vezes nas páginas 62 e 64.

63 LEE, B. S. *Understanding the Terms and Definitions of LDO Voltage Regulators*. [S.l.], 1999. Disponível em: <http://www.ti.com/lit/an/slva079/slva079.pdf>. Citado na página 62.

64 ROHM SEMICONDUCTOR. *Switching Regulator Basics Advantages vs Disadvantages in Comparison with Linear Regulator*. 2015. Citado na página 62.

65 LM2576/LM2576HV Series SIMPLE SWITCHER® 3A Step-Down Voltage Regulator. [S.l.], 2013. Disponível em: <http://www.ti.com/lit/ds/symlink/lm2576.pdf>. Citado na página 63.

66 ALLEGRO MICROSYSTEMS, LLC. *ACS712 - Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 2.1 kVRMS Isolation and a Low-Resistance Current Conductor*. 2013. ed. [S.l.]. Disponível em: <https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf>. Citado na página 65.

67 GAMMON, N. *Power saving techniques for microprocessors*. 2012. Disponível em: <http://www.gammon.com.au/forum/?id=11497>. Citado 3 vezes nas páginas 73, 90 e 102.

68 FALUDI, R. *Building Wireless Sensors Networks*. O'Reilly Media, 2010. ISBN 978-0-596-80773-3. Disponível em: <http://shop.oreilly.com/product/9780596807740.do>. Citado 3 vezes nas páginas 77, 79 e 120.

69 SILICON LABS. *EM351/EM357 High-Performance, Integrated ZigBee/802.15.4 System-on-Chip*. [S.l.], 2013. Disponível em: <https://www.silabs.com/Support%20Documents/TechnicalDocs/EM35x.pdf>. Citado na página 77.

70 DIGI. *XBee ZigBee Addressing*. [S.l.], 2015. Article Number 000001692. Disponível em: <http://knowledge.digi.com/articles/Knowledge_Base_Article/XBee-ZigBee-Addressing>. Citado na página 77.

71 TEXAS INSTRUMENTS. *Broadcasts and group addressing (Z-Stack, ZigBee)*. [S.l.]. Disponível em: <https://e2e.ti.com/support/wireless_connectivity/w/design_notes/broadcasts-and-group-addressing-z-stack-zigbee>. Citado na página 77.

72 JR., S. E. *XBee API Mode Tutorial Using Python and Arduino*. 2014. Website. Disponível em: <http://serdmanczyk.github.io/XBeeAPI-PythonArduino-Tutorial/>. Citado na página 79.

73 KREBS, B. *Hacked Cameras, DVRs Powered Today's Massive Internet Outage*. 2016. Disponível em: <https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/>. Citado na página 86.

74 FAN, X. et al. *Security Analysis of Zigbee*. 2017. Publicado na página do curso 6.857: Computer and Network Security do Massachusetts Institute of Technology. Disponível em: <https://courses.csail.mit.edu/6.857/2017/project/17.pdf>. Citado na página 88.

75 CRAGIE, R.; CURTIS, B. *PART TWO: Securing The Connected Home From Outside Threats*. 2016. Disponível em: <http://threadgroup.org/news-events/blog/ID/109/PART-TWO-Securing-the-Connected-Home-From-Outside-Threats>. Citado na página 88.

76 DIGI INTERNATIONAL INC. *XBee®/XBee-PRO S2C ZigBee® RF Module User Guide*. [S.l.], 2016. Citado na página 89.

77 HUANG, A. B. *The Hardware Hacker*. Random House LCC US, 2017. ISBN 159327758X. Disponível em: <http://www.ebook.de/de/product/26465850/andrew_bunnie_huang_the_hardware_hacker.html>. Citado na página 89.

78 REIS, M. V. G. dos. *Estudo e implementação de estrategias de detecção de ilhamento em inversores para sistemas fotovoltaicos de geração distribuída*. Dissertação (Mestrado) — Universidade Estadual de Campinas, 2016. Citado na página 95.

79 CHRISTIDIS, K.; DEVETSIKIOTIS, M. Blockchains and smart contracts for the internet of things. *IEEE Access*, Institute of Electrical and Electronics Engineers (IEEE), v. 4, p. 2292–2303, 2016. Citado na página 108.

80 RUTKIN, A. *Blockchain-based microgrid gives power to consumers in New York*. 2016. Disponível em: <https://www.newscientist.com/article/2079334-blockchain-based-microgrid-gives-power-to-consumers-in-new-york/>. Citado na página 108.

81 RUTKIN, A. Blockchain aids solar sales. *New Scientist*, Elsevier BV, v. 231, n. 3088, p. 22, aug 2016. Citado na página 108.

82 PHILLIPS, S. *Getting a Python script to run in the background (as a service) on boot*. 2013. Disponível em: <http://blog.scphillips.com/posts/2013/07/getting-a-python-script-to-run-in-the-background-as-a-service-on-boot/>. Citado na página 122.

Anexos

ANEXO A – Configurações do rádio XBee

Neste apêndice são explicitados as configurações do rádio XBee e o processo e campo de um *frame* de dados da rede.

A.1 Configurações do XBee

A versão do XBee S2 tem *firmwares* específicos para cada uma das funções (coordenador, roteador e dispositivo final) e dos modos (transparente ou API). Enquanto no S2C existe apenas um *firmware* e a configuração pode ser alterada pelo *software* XCTU[1].

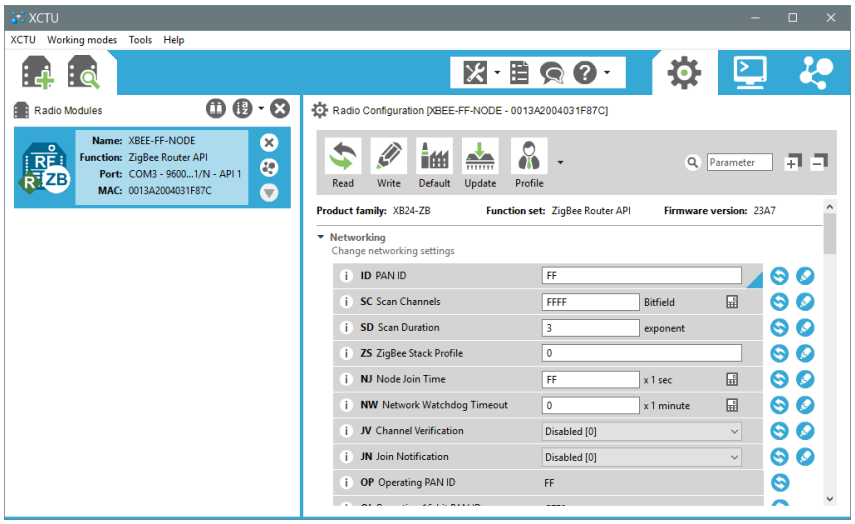


Figura 60 – Interface gráfica do programa XCTU utilizado para alterar configurações e atualizar o *firmware* do XBee.

Neste *software*, exibido na Figura 60, é possível atualizar o *firmware* dos dispositivos XBee, ou alterar a versão instalada, como no caso do S2, passando do modo transparentes para o API, ou vice-versa. Também é possível editar as configurações dos módulos, dentre elas:

- **PAN ID (registrador ID):** cada rede Zigbee tem um endereço de 16 bit criando uma *Personal Area Network* (PAN). Quando um XBee é ligado, automaticamente procura um coordenador com o mesmo endereço e se conecta a ele, portanto todos os dispositivos da rede devem ter o mesmo PAN ID;
- **Serial Number (SH/SL):** Cada XBee tem um número serial único de 64 bit. Geralmente é dividido em duas partes de 32 bit, chamadas de *Serial High* (**SH**), a

[1] <https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu>

parte mais significativa, e *Serial Low* (S**L**), a parte menos significativa. Este número pode ser utilizado para endereçar outros dispositivos na rede, é muito parecido com o endereço MAC em redes LAN;

- **Destinatário (DH/DL):** quando o XBee está no modo transparente os dados são enviados para o destinatário presente neste registrador;
- **16-bit Network Address (MY):** um endereço de 16 bit é atribuído pelo coordenador a cada um dos nós da rede. Este endereço é único apenas na própria rede, ao contrário do endereço de 64 bit. Ele permite que a limitada memória do microcontrolador presente do XBee seja capaz de armazenar um número maior de nós;
- **Node Identifier (NI):** Cada nó pode ter um nome que pode ser atribuído pelo usuário. Quando um nó está no modo API e recebe um pacote de dados, ele é capaz de saber o nome de quem enviou o dado, característica que pode ser útil dependendo da aplicação;
- **Baud Rate (BD):** é a taxa de transmissão de dados (desconsiderando dados de controle) entre o nó e o transceptor. Varia entre 1200 e 115200 kbps, sendo 9600 kbps o padrão;
- **Criptografia:** o XBee suporte criptografia AES, que vem desabilita por padrão. É necessário alterar os registradores *Encryption Enable* (EE) para 1 (*enable*) e colocar um texto de 32 caracteres no Encryption Key (KY)[2].

Para alterar algumas destas configurações, pode-se fazer uso do aplicativo XCTU, além disso, também é possível utilizando comandos AT, os principais são citados a seguir:

- **AT:** responde "OK", é útil para verificar se está realmente no modo de comando, já que depois de 10 segundos de inatividade volta para o modo normal;
- **ATID:** retorna qual é o ID da rede, um parâmetro configurado pelo usuário que está entre 0x0 e 0xFFFF. Caso seja adicionado um endereço depois do comando, este será o novo endereço do XBee;
- **ATSH/ATSL:** Cada rádio XBee tem um endereço de 64 bit único, como o *MAC Address* de uma placa de rede. Este endereço é dividido entre duas partes de 32 bit, chamadas de *high part* (ATS**H**), que seriam os bits mais significativos, e *low part* (ATS**L**), os menos;

[2] <http://www.digi.com/resources/documentation/Digidocs/90001110-88/tasks/t_set_up_basic_encryption_for_your_xbee_network.htm>

- **ATDH/ATDL:** Estes comandos exibe as partes altas e baixas do endereço destino. Adicionando um novo endereço na frente, pode-se trocar o destinatário;
- **ATCN:** para sair do modo de comandos;
- **ATWR:** todos os dados alterados ficam salvos em uma memória volátil, depois de reiniciado o XBee são perdidos, este comando salva na memória permanente.

A.2 Detalhamento de um *frame* XBee de dados

Para melhor detalhamento é possível utilizar o *software* XCTU, que tem um gerador de Frames mostrado na Figura 61, ou o livro de Faludi(68), que exemplifica os principais modos com muitos exemplos.

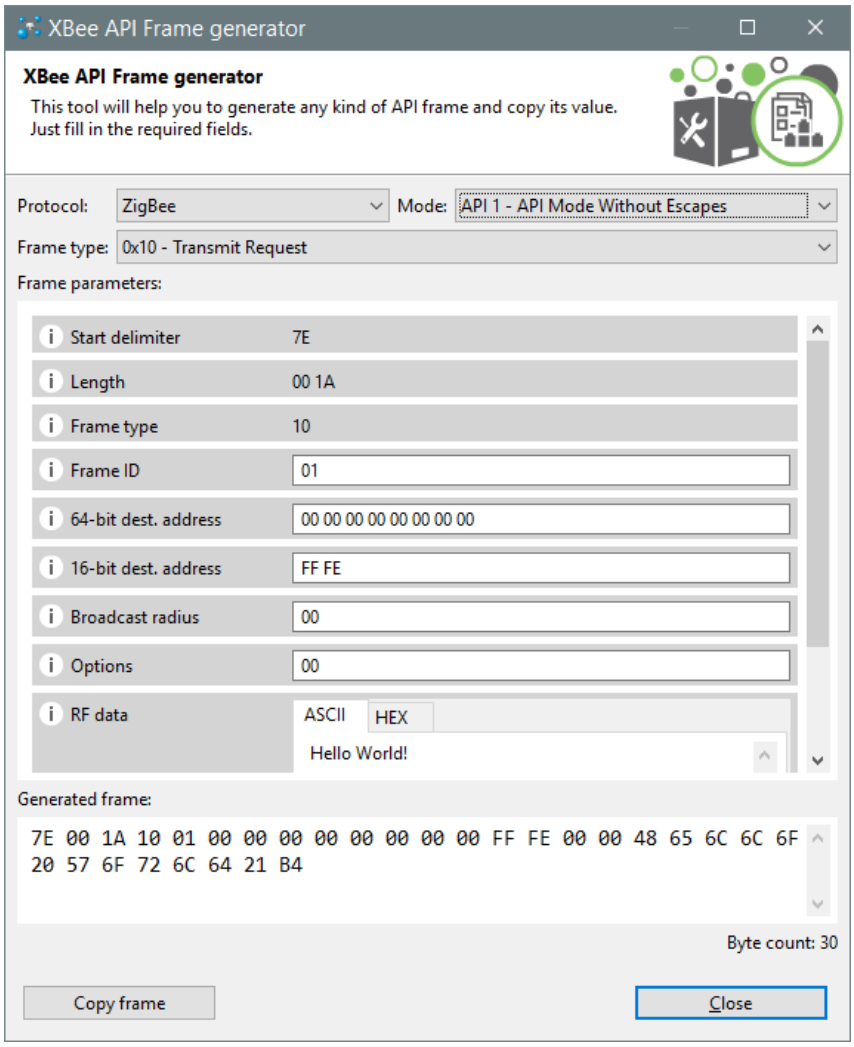


Figura 61 – Gerador de *frames* do *software* XCTU.

Na Figura 62 é exibido o *frame* correspondente a mensagem "Hello world" e são explicados cada byte da mensagem. Após este dado ser enviado o XBee recebe um outro

frame, caso a opção por desabilitar esta resposta não tenha sido ativada, que retorna o sucesso ou não do envio da mensagem.

Byte	Função
7E	Identificador Frame
00	Comprimento, byte mais significativo
1A	Comprimento, byte menos significativo
10	Tipo de frame para transmissão de dados
01	Requer uma resposta para transmissão
00	Endereço de 64 para transmissão
...	
00	
FF	Endereço de 16 bits, caso desconhecido pode-se usar FFFE
FE	
00	Raio do broadcast
00	Opções: 0x01 - Desativa ACK (resposta ao envio) 0x20 - Envia dado com criptografia (se criptografia ativada) 0x40 - Usa um tempo maior de espera para o envio
48	H
65	e
...	...
64	d
21	!
B4	Soma de verificação

Figura 62 – *Frame* gerado para enviar a mensagem "Hello World!".

Pode-se criar uma classe para gerir cada uma das particularidades do padrão Zigbee, no entanto o trabalho de apoiou em bibliotecas já consolidadas no desenvolvimento IoT. A utilizada no Python foi a XBee, disponível em seu repositório no Github[3], pode-se também instalar utilizando o gerenciador nativo de pacotes do Python, o Pip. Outra opção seria o Pyxb[4], que foi preterida devido a menor disponibilidade de documentação. No caso dos nós, foi utilizada a biblioteca xbee-arduino[5], que gera automaticamente os frames para envio, recebe o frame com o resultado do envio, o que permite descobrir se ocorreu algum problema com a transmissão, possibilitando implementar um reenvio via *software*, o que não seria possível utilizando o modo transparente.

[3] <https://github.com/nioinnovation/python-xbee>
[4] <https://github.com/alflanagan/pyxbee>
[5] <https://github.com/andrewrapp/xbee-arduino>

ANEXO B – Transformando um programa em serviço no Linux

Um sistema operacional possui vários processos sendo executados em segundo plano, como gerenciamento de bateria, conexão sem fio, protocolos de rede, dentre outros. Enquanto o sistema operacional estiver sendo executados estes programas também estão, sem intervenção do usuário, em segundo plano. São iniciados e fechados automaticamente pelo sistema. É interessante transformar programas que devem ser executados sem a intervenção do usuário em serviços.

Stephen Phillips (82) apresenta um *Shell Script* para sistemas baseados na distribuição Debian para criar um serviço a partir de um programa escrito em Python.

O script utilizado em nossa aplicação segue abaixo. Ele é utilizado na pasta /etc/init.d do sistema e é necessário conceder permissão de execução.

```
#!/bin/sh

### BEGIN INIT INFO
# Provides: SolarMonitor
# Required-Start: $remote_fs $syslog
# Required-Stop: $remote_fs $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Monitor Solar Generation in a wireless sensor
   network
# Description: Service that monitor de solar generation from a solar
   plant using a zigbee wireless sensor network.
### END INIT INFO

# Change the next 3 lines to suit where you install your script and what
    you want to call it
DIR=/usr/local/bin/SolarMonitor
DAEMON=/home/pi/Server/main.py
DAEMON_NAME=Solar_Monitor

# Add any command line options for your daemon here
DAEMON_OPTS=""
```

```
# This next line determines what user the script runs as.
DAEMON_USER=pi

# The process ID of the script when it runs is stored here:
PIDFILE=/var/run/$DAEMON_NAME.pid

. /lib/lsb/init-functions

do_start () {
    log_daemon_msg "Starting␣system␣$DAEMON_NAME␣daemon"
    start-stop-daemon --start --background --pidfile $PIDFILE --make-
       pidfile --user $DAEMON_USER --chuid $DAEMON_USER --startas $DAEMON
        -- $DAEMON_OPTS
    log_end_msg $?
}
do_stop () {
    log_daemon_msg "Stopping␣system␣$DAEMON_NAME␣daemon"
    cat $PIDFILE | xargs pgrep -P | xargs kill -9
    start-stop-daemon --stop --pidfile $PIDFILE --retry 10
    log_end_msg $?
}

case "$1" in

    start|stop)
        do_${1}
        ;;

    restart|reload|force-reload)
        do_stop
        do_start
        ;;

    status)
        status_of_proc "$DAEMON_NAME" "$DAEMON" && exit 0 || exit $?
        ;;

    *)
```

```
        echo "Usage: /etc/init.d/$DAEMON_NAME {start|stop|restart|status}"
        exit 1
        ;;

esac
exit 0
```

ANEXO C – Códigos fonte

C.1 Concentrador

C.1.1 main.py

```
#!/usr/bin/env python3

##########################################
# LSM - DSIF - FEEC - Unicamp
# Autor: Ricardo Mazza Zago
# Dezembro de 2017
##########################################

import multiprocessing
import time
import configparser
import json
import binascii
import string
import platform
import subprocess
import datetime
import logging
import socket
import serial
import serial.tools.list_ports
from xbee import ZigBee
import server
import access_database
import aux_functions
import cloud


def init_server():
    server.Server()

# Variáveis globais
```

```
global received_data
received_data = []


def data_received(data): # Ler dados assincronamente do Xbee
    received_data.append(data)


def create_nodelist(parser):
    # Criar lista de nos lendo arquivo de configuracao
    nodes = dict()
    try:
        for i in range(0, aux_functions.number_sensors(parser)):
            nodes_address = parser.get(
                'node_' + str(i + 1), 'DH')[2:] + parser.get('node_' + str(
                    i + 1), 'DL')[2:]
            nodes_name = parser.get('node_' + str(i + 1), 'name')
            nodes[nodes_address] = nodes_name

    except configparser.NoOptionError:
        logger.error('[MAIN] O número de nós não é igual ao número de
            endereços colocados no arquivo de configuração. Erro ao ler:
            sensor' +
                     str(i + 1) + '_DH ou sensor' + str(i + 1) + '_DL')
    return nodes


def add_node(parser, address):
    node = dict()

    node_count = str(aux_functions.number_sensors(parser) + 1)

    address_dh = "0x" + address[:8]
    address_dl = "0x" + address[8:]

    node["dh"] = address_dh
    node["dl"] = address_dl

    node["name"] = "node_" + node_count
```

```
    parser['node_' + node_count] = node

    with open(aux_functions.get_path("config.ini"), 'w') as configfile:
        parser.write(configfile)

    logger.error(
        "[MAIN] Não foi encontrado tal nó no arquivo de configurações, ele
            foi adicionado com nome de: node_" + node_count)


def is_serial_still_connected(porta):
    res = serial.tools.list_ports.comports()
    for item in res:
        if item.device.upper() == porta.upper():
            return True
    return False


def is_connected():
    try:
        REMOTE_SERVER = "www.google.com"
        host = socket.gethostbyname(REMOTE_SERVER)
        # Verifica se existe conexao a internet
        s = socket.create_connection((host, 80), 2)
        return True
    except:
        pass
    return False


if __name__ == "__main__":
    # Inicia o sistema de log
    logger = logging.getLogger("Main")
    logger.setLevel(logging.DEBUG)

    # Cria o logging file handler
    fh = logging.FileHandler(aux_functions.get_path("logs/main.log"))
    logFormatter = logging.Formatter(
```

```
    '%(asctime)s - %(name)s - %(levelname)s - %(message)s')
fh.setFormatter(logFormatter)
logger.addHandler(fh)
# Para tambem imprimir os logs no console
consoleHandler = logging.StreamHandler()
consoleHandler.setFormatter(logFormatter)
logger.addHandler(consoleHandler)

# Espera conexao a internet no Linux, Raspberry nao possui RTC e
    depende da internet para horario
if platform.system() == "Linux":
    while is_connected() == False:
        logger.error("Não conectado a internet, aguardando 5 segundos.
            ")
        time.sleep(5)
    logger.info("Conectado a internet.")

# Inicia o servidor em outro processo
ctx = multiprocessing.get_context('spawn')
server = ctx.Process(target=init_server, daemon=True)
server.start()

try:
    # Comeca o inicio do loop do programa
    logger.info('Starting main loop.Terminate with Control-C')
    # Variaveis que controlam a proxima execucao do item
    last_update_upload_dropbox = time.time()
    last_update_is_xbee_connected = time.time()

    # Dicionario em python para controle do ultimo envio dos dados ao
        servico em nuvem
    cloud_control = dict()

    # Outras variaveis
    node_list_control = 0
    xbee_desconectado = False

    # Le arquivo de configuracoes
    parser = configparser.SafeConfigParser()
```

```
parser.read(aux_functions.get_path('config.ini'))

# Inicia a comunicacao serial
# Le o arquivo de configuracao, considerando Windows e Linux
    diferentes
if platform.system() == "Linux":
    porta = parser.get('basic', 'port_linux')
elif platform.system() == "Windows":
    porta = parser.get('basic', 'port_windows')

# Tenta se conectar a porta serial, lembrando que o radio XBee
    pode nao estar conectado
try:
    serial_port = serial.Serial(porta, parser.get(
        'basic', 'baudrate'), timeout=int(parser.get('basic', '
            timeout')))
    xbee = ZigBee(serial_port, callback=data_received)
except serial.SerialException:
    # Nao pode se conectar, ira tentar novamente no loop de
        execucao
    xbee_desconectado = True
    logger.error('[MAIN] Não foi possivel conectar a porta ' +
                 porta + ', conecte o XBee, apenas o servidor será
                     ativado.')
nodes = create_nodelist(parser)

while True:
    ##################################################
    # Verifica se recebeu novos dados do XBee
    while len(received_data) > 0:
        if 'source_addr_long' in received_data[0]:
            # address = (received_data[0]['source_addr_long']).hex
                ().upper()
            # Nao compativel com Python 3.4
            address = str(binascii.hexlify(received_data[0][
                        'source_addr_long']), 'ascii').upper()
            try:
                node_name = nodes[address]
            except:
```

```
            add_node(parser, address)
            nodes = create_nodelist(parser)
            node_name = nodes[address]

        try:
            xbee_data_packet = received_data.pop(
                0)['rf_data'].decode("ascii")
            # Para garantir que todos os caracteres sao
                legiveis no JSON
            dados = json.loads(
                ''.join(x for x in xbee_data_packet if x in
                    string.printable))
            logger.debug('[MAIN] Recebido de {} com nome {} os
                seguintes dados {} em {}'.format(
                address, node_name, dados, time.asctime()))

            if 'c' in dados and 'v' in dados and 'r' in dados:
                access_database.add_insert_data(node_name,
                    address, dados[
                                                'c'], dados['v'],
                                                    dados['r'])

                if aux_functions.is_time_to_cloud(cloud_control,
                     node_name, tempo=60):
                    if cloud.enviar_dados_nuvem(node_name, dados
                        ['c'], dados['v'], dados['r']):
                        logger.debug('[Cloud] Reading sent to
                            cloud.')
                    else:
                        logger.error('[Cloud] Error in sending to
                            Cloud')

            else:
                logger.error('[MAIN] Dado faltando no pacote
                    recebido.')

        except ValueError: # Inclui simplejson.decoder.
            JSONDecodeError
            logger.error('[MAIN] Decoding JSON has failed')
```

```
                logger.error(xbee_data_packet)
            else:
                received_data.pop(0)

            # Eventos que ocorrem de tempo em tempo
            now = time.time()

            # Verifica conexao do XBee (conecta ou desconecta se
               necessario)
            if now - last_update_is_xbee_connected >= 1:
                if is_serial_still_connected(porta):
                    if xbee_desconectado:
                        serial_port = serial.Serial(porta, 9600, timeout=1)
                        xbee = ZigBee(serial_port, callback=data_received)
                        xbee_desconectado = False
                        logger.error("Xbee␣foi␣reconectado!")
                elif xbee_desconectado == False:
                    logger.error("Xbee␣foi␣desconectado")
                    xbee.halt()
                    serial_port.close()
                    xbee_desconectado = True
                last_update_is_xbee_connected = now

            # Envia banco de dados ao Dropbox
            if platform.system() == "Linux":
                if now - last_update_upload_dropbox > int(parser.get('basic
                    ', 'time_dropbox_upload')):
                    database_file = aux_functions.get_path() + 'database/
                       nodes-' + str(datetime.datetime.now().year) + '-' +
                       str(datetime.datetime.now().month) + '.db'
                    subprocess.Popen(
                        [aux_functions.get_path() + 'Dropbox-Uploader/
                           dropbox_uploader.sh', 'upload', database_file, '
                           /SolarPanel'])
                    last_update_upload_dropbox = now
            ##########################################

    except KeyboardInterrupt:
        # Fecha programa em caso do usuario teclara Control + C
```

```
        logger.info('[MAIN] Closing Xbee port...')
        serial_port.close()
        logger.info('[MAIN] Closed... Stopping server...')
        server.terminate()
        server.join()
        logger.info('[MAIN] Server stopped')
```

C.1.2 server.py

```
##########################################
# LSM - DSIF - FEEC - Unicamp
# Autor: Ricardo Mazza Zago
# Dezembro de 2017
##########################################

import datetime
import logging
from io import BytesIO
from matplotlib.backends.backend_agg import FigureCanvasAgg
from matplotlib import figure
import access_database
import flask
import plotador
import aux_functions

# Inicia o logs
logger = logging.getLogger("Server")
logger.setLevel(logging.DEBUG)
fh = logging.FileHandler(aux_functions.get_path("logs/server.log"))
logFormatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
    - %(message)s')
fh.setFormatter(logFormatter)
logger.addHandler(fh)
# Imprime os logs tambem
consoleHandler = logging.StreamHandler()
consoleHandler.setFormatter(logFormatter)
logger.addHandler(consoleHandler)

class Server:
```

```
def __init__(self):
    logger.info("Servidor␣Iniciado")

    # Inicia o Flask
    self.app = flask.Flask(__name__, static_folder='static')
    self.app.config.update(DEBUG=False, SECRET_KEY='secret!')

    # Adiciona as paginas
    self.app.add_url_rule('/', 'index', self.index)
    self.app.add_url_rule('/plot.png', 'plot', self.plot)
    self.app.add_url_rule('/database/<string:name>', 'get_database',
       self.get_database)
    self.app.add_url_rule('/number_nodes.json', 'number_nodes', self.
       number_nodes)

    # Start server
    self.app.run(host='0.0.0.0')

def index(self):

    lista_dos_nos = aux_functions.create_node_list()

    lista_banco_de_dados = aux_functions.list_databases()

    return flask.render_template('index.html', lista_nos=lista_dos_nos
       , lista_banco_de_dados=lista_banco_de_dados)

def get_database(self, name):
    return flask.send_from_directory(aux_functions.get_path("/database
       "), name)

def number_nodes(self):
    node_count=len(access_database.get_actives_nodes(seconds=180))
    return '{"node_count":' + str(node_count) + '}'

def plot(self):
    node = flask.request.args.get('node')
    data = flask.request.args.get('data')
```

```
        if data is None:
            fig = None
        else:
            fig = figure.Figure(figsize=(12, 6), facecolor='white')

            database_name=aux_functions.get_path() + 'database/nodes-' +
               str(datetime.datetime.strptime(data, "%Y-%m-%d").year) + '-
               ' + str(datetime.datetime.strptime(data, "%Y-%m-%d").month)
                + '.db'

            fig, _ = plotador.gerar_grafico_node_data(fig, node, data,
               database_name)

        if fig is None:
            logger.error('[WEB] Não foi encontrado registros para esta
               configuração')
            return flask.send_from_directory(self.app.static_folder, "
               nothing.png")
        else:
            # Transforma a imagem em PNG
            canvas = FigureCanvasAgg(fig)
            output = BytesIO()
            canvas.print_png(output, dpi=80)
            response = flask.make_response(output.getvalue())
            response.mimetype = 'image/png'
            return response

if __name__ == "__main__":
    print('[WEB] The server should be started from the main contoller!')
```

C.1.3 access_database.py

```
##########################################
# LSM - DSIF - FEEC - Unicamp
# Autor: Ricardo Mazza Zago
# Dezembro de 2017
##########################################

import sqlite3
import datetime
```

```
import numpy as np
import aux_functions


def connect_to_db(file=None):
    if file is None:
        conn = sqlite3.connect(aux_functions.get_path() + 'database/nodes-
            ' + str(datetime.datetime.now().year) + '-' + str(datetime.
            datetime.now().month) + '.db')
    else:
        conn = sqlite3.connect(file)
    create_table(conn)
    return conn


def close_db(conn):
    conn.close()


def create_table(conn):
    cursor = conn.cursor()
    q = '''CREATE TABLE IF NOT EXISTS Leituras (
        [Index] INTEGER PRIMARY KEY AUTOINCREMENT,
        node_name TEXT NOT NULL,
        zigbee_address TEXT NOT NULL,
        corrente DOUBLE NOT NULL,
        tensao DOUBLE NOT NULL,
        datetime DATETIME NOT NULL,
        tempo_amostragem INTEGER NOT NULL
    );'''
    cursor.execute(q)
    conn.commit()


def retrive_data(col, file = None):
    conn = connect_to_db(file)
    # Faz retornar cada resultado único em uma linha do vetor, não uma
        lista de listas (o que faria sentido se a consulta pudesse
        retornar mais de uma coluna por linha)
```

```
    conn.row_factory = lambda cursor, row: row[0]
    cursor = conn.cursor()
    retorno = cursor.execute(
        'SELECT␣DISTINCT␣{}␣FROM␣Leituras'.format(col)).fetchall()
    close_db(conn)
    return retorno


def get_corrente_tensao_tempo_amostragem(node_name=0, data=0,
    zigbee_address=0, file = None):
    conn = connect_to_db(file)
    cursor = conn.cursor()
    comando = 'select␣corrente,␣tensao,␣datetime,␣tempo_amostragem␣from␣
       Leituras␣where␣node_name="{0}"␣and␣strftime("%s",␣datetime)>=
       strftime("%s",␣"{1}")␣and␣strftime("%s",␣datetime)<=strftime("%s",
       ␣"{1}","+1␣day")'.format(
           node_name, data)
    retorno = cursor.execute(comando).fetchall()
    close_db(conn)
    return np.asarray(retorno)


def add_insert_data(node_name, zigbee_address, corrente, tensao,
    tempo_amostragem, file=None):
    conn = connect_to_db(file)
    cursor = conn.cursor()
    cursor.execute('insert␣into␣Leituras␣(node_name,␣zigbee_address,␣
       corrente,␣tensao,␣datetime,␣tempo_amostragem)␣values␣(?,?,?,?,?,?)
       ',
                   (node_name, zigbee_address, corrente, tensao, datetime.
                      datetime.now(),tempo_amostragem))
    conn.commit()
    close_db(conn)


def get_unique_dates(file = None):
    conn = connect_to_db(file)
    conn.row_factory = lambda cursor, row: row[0]
    cursor = conn.cursor()
```

```
    retorno = cursor.execute('SELECT␣DISTINCT␣DATE(datetime)␣FROM␣Leituras
        ␣order␣by␣DATE(datetime)').fetchall()
    close_db(conn)
    return retorno


def get_actives_nodes(file = None, seconds = 180):
    conn = connect_to_db(file)
    conn.row_factory = lambda cursor, row: row[0]
    cursor = conn.cursor()
    atual = str(datetime.datetime.now()-datetime.timedelta(seconds=seconds
        ))
    retorno = cursor.execute('SELECT␣DISTINCT␣node_name␣FROM␣Leituras␣
        where␣strftime("%s",␣datetime)>=strftime("%s",␣"{}")'.format(atual
        )).fetchall()
    close_db(conn)
    return retorno
```

C.1.4 cloud.py

```
##########################################
# LSM - DSIF - FEEC - Unicamp
# Autor: Ricardo Mazza Zago
# Dezembro de 2017
##########################################

import configparser
from ubidots import ApiClient
import aux_functions

parser_cloud = configparser.SafeConfigParser()
parser_cloud.read(aux_functions.get_path('cloud.ini'))

def enviar_dados_nuvem(node, corrente, tensao, tempo_leitura):
    api = ApiClient(token=parser_cloud.get('basic', 'ubidots_token'))
    corrente_ubi = api.get_variable(parser_cloud.get(node, 'corrente_ubi')
        )
    tensao_ubi = api.get_variable(parser_cloud.get(node, 'tensao_ubi'))
    tempo_ubi = api.get_variable(parser_cloud.get(node, 'tempo_ubi'))
    try:
```

```
        response = corrente_ubi.save_value({'value': corrente})
        print(response)
        response = tensao_ubi.save_value({'value': tensao})
        print(response)
        response = tempo_ubi.save_value({'value': tempo_leitura})
        print(response)
        return True
    except:
        return False
```

C.1.5 plotador.py

```
##########################################
# LSM - DSIF - FEEC - Unicamp
# Autor: Ricardo Mazza Zago
# Dezembro de 2017
##########################################

import datetime
import configparser
import numpy as np
from matplotlib import dates
import matplotlib.image as image
import aux_functions
import access_database

def gerar_grafico_node_data(fig, node, data, file=None):

    leituras = access_database.get_corrente_tensao_tempo_amostragem(
        node_name=node, data=data, file=file)
    # Verifica se o acesso ao bd retornou resultados
    if len(leituras) == 0:
        print("Não achou nenhum registro")
        return None, None
    else:
        data_e_hora = []
        # Converte os dados lidos do banco de dados para o formato
            correto
        for data in leituras[:, 2]:
            data_e_hora.append(datetime.datetime.strptime(
```

```
                data, "%Y-%m-%d␣%H:%M:%S.%f"))

        # Define o conteúdo de cada um dos eixos
        x = np.array([data_e_hora[i] for i in range(len(data_e_hora))])
        Corrente = leituras[:, 0].astype(np.float)
        Tensao = leituras[:, 1].astype(np.float)

        tempo_amostragem = leituras[:, 3].astype(np.int)
        Potencia = Corrente * Tensao
        Potencia_geracao = Corrente * Tensao * tempo_amostragem / 1000
        total_gerado = np.sum(Potencia_geracao) / 3600

        amax_Corrente = np.amax(Corrente)
        amax_Tensao = np.amax(Tensao)
        amax_pot = np.amax(Potencia)

        # Para remover os eventuais pontos de descontinuidade
        discontinuous = list(np.where(np.diff(x) >= datetime.timedelta(
            minutes=3)))

        for pos in discontinuous:
            Corrente[pos]=np.nan
            Tensao[pos]=np.nan
            Potencia[pos]=np.nan

        fig = plotar(fig, x, Corrente, Tensao, Potencia, total_gerado,
            node, data, amax_Corrente, amax_Tensao, amax_pot)

    return (fig, total_gerado)

def plotar(fig, x, Corrente, Tensao, Potencia, total_gerado, node, data,
    amax_Corrente, amax_Tensao, amax_pot):
    # Criar uma figura para plotar os dados
    im = image.imread(aux_functions.get_path('static/logo.png'))

    ########################################################
    # Adiciona uma janela de plot para a tensão (dois eixos x em um
        único gráfico)
    axis = fig.add_subplot(1, 2, 1)
```

```
# Define um eixo secundário
axis2 = axis.twinx()

# Plota e toma o resultado para posteriormente definir as legendas
lns1 = axis.plot(x, Corrente, marker='.', color='g', label='Current')
lns2 = axis2.plot(x, Tensao, marker='.', color='b', label='Voltage')

# Define as legendas
lns = lns1 + lns2
labs = [l.get_label() for l in lns]
axis.legend(lns, labs, loc=1)
fig.suptitle('Solar␣generation␣from␣{}␣in␣{}'.format(
    node, datetime.datetime.strptime(data, "%Y-%m-%d␣%H:%M:%S.%f").
       strftime('%d/%m/%Y')), fontsize=20)

# Define os limites dos eixos
axis.set_ylim((0, amax_Corrente + amax_Corrente / 3))
axis2.set_ylim((0, amax_Tensao + amax_Tensao / 3))

# Define os labels dos eixos X
axis.set_ylabel("Current␣(A)", fontsize=16)
axis2.set_ylabel("Voltage␣(V)", fontsize=16)
# Define o label do eixo Y
axis.xaxis.set_major_formatter(dates.DateFormatter('%H:%M'))
axis.grid()
fig.autofmt_xdate(bottom=0.15, rotation=30, ha='right')

#########################################################
# Segundo plot para a Potência
axis = fig.add_subplot(1, 2, 2)
axis.plot(x, Potencia, marker='.', color='r', label='Power')
axis.legend(loc=1)
axis.set_ylim((0, amax_pot + amax_pot / 3))
axis.set_ylabel("Power␣(W)", fontsize=16)
axis.xaxis.set_major_formatter(dates.DateFormatter('%H:%M'))
axis.grid()
fig.autofmt_xdate(bottom=0.15, rotation=30, ha='right')

#########################################################
```

```
    axis.figure.figimage(im, 2, 530, alpha=.5, zorder=1)
    texto_rodape = 'This␣node␣generated␣in␣{:.2f}␣Wh'.format(total_gerado
        )
    parser = configparser.SafeConfigParser()
    parser.read(aux_functions.get_path('config.ini'))
    try:
        global_position = parser.get(node, 'global_position')
        texto_rodape = texto_rodape + ",␣in␣" + global_position
    except:
        pass
    try:
        inclination = parser.get(node, 'inclination')
        texto_rodape = texto_rodape + ",␣with␣inclination␣of␣" +
            inclination + "␣degree(s)"
    except:
        pass
    try:
        orientation = parser.get(node, 'orientation')
        texto_rodape = texto_rodape + ",␣\nwith␣orientation␣" +
            inclination + "␣degree(s)␣clockwise␣relative␣to␣north"
    except:
        pass
    try:
        year = parser.get(node, 'year')
        texto_rodape = texto_rodape + ",␣installed␣in␣the␣year␣of␣" + year
    except:
        pass
    texto_rodape = texto_rodape + "."
    axis.annotate(texto_rodape, (-1.4, 0), (0, -50), fontsize=14, xycoords
        ='axes␣fraction', textcoords='offset␣points', va='top')
    fig.subplots_adjust(left=0.08, bottom=0.2, right=0.98, top=None,
        wspace=0.3, hspace=None)

    return fig
```

C.1.6 main_interface_db.py

```
##########################################
# LSM - DSIF - FEEC - Unicamp
```

```
# Autor: Ricardo Mazza Zago
# Dezembro de 2017
########################################

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.backends.backend_pdf import PdfPages, FigureCanvasPdf
from matplotlib.figure import Figure

import datetime
import access_database
import numpy as np
import plotador
import pandas as pd

from tkinter import *
from tkinter.ttk import *
from tkinter.filedialog import askopenfilename, asksaveasfilename
from tkinter import messagebox

import matplotlib
matplotlib.use('TkAgg')

class Janela:

    def __init__(self, raiz):
        self.raiz = raiz
        self.raiz.title('Solar Generation Analysis')

        #Criação da interface Gráfica
        menu = Menu(raiz)
        raiz.config(menu=menu)
        filemenu = Menu(menu)
        menu.add_cascade(label="File", menu=filemenu)
        filemenu.add_command(label="Open", command=self.onOpen)
        filemenu.add_separator()
        filemenu.add_command(label="Exit", command=self.quit)

        exportmenu_today = Menu(menu)
        menu.add_cascade(label="Export Current Day", menu=exportmenu_today
```

```
        )
    exportmenu_today.add_command(label="To␣CSV", command=self.
        export_to_csv)
    exportmenu_today.add_command(label="To␣Excel␣(.xls)", command=self
        .export_to_xls)
    exportmenu_today.add_command(label="Save␣to␣PNG", command=self.
        save_image)

    exportmenu_month_year = Menu(menu)
    menu.add_cascade(label="Export␣period", menu=exportmenu_month_year
        )
    exportmenu_month_year.add_command(label="Save␣Monthly␣Plots␣in␣PDF
        ", command=self.export_to_pdf)

    helpmenu = Menu(menu)
    menu.add_cascade(label="Help", menu=helpmenu)
    helpmenu.add_command(label="About", command=self.about)

    # Criando Interface utilizando layout grid
    # Primeira Linha
    self.open_file_button = Button(self.raiz, text='Open␣File')
    self.open_file_button.bind("<Button-1>", lambda event: self.onOpen
        ())
    self.open_file_button.grid(row=0, column=0, sticky='NSEW')
    self.texto = Label(self.raiz, text='Nothing␣open␣yet')
    self.texto.grid(row=0, column=1, columnspan=1, sticky='NSEW')

    # Segunda Linha
    self.variables_dropdown_nodes = StringVar(self.raiz)
    OPTIONS = ["Open␣a␣database␣file"]
    self.variables_dropdown_nodes.set(OPTIONS[0])
    self.dropdown_nodes = OptionMenu(
        self.raiz, self.variables_dropdown_nodes, *OPTIONS)
    self.dropdown_nodes.grid(row=1, column=0, sticky='NSEW')

    self.variables_dropdown_data1 = StringVar(self.raiz)
    OPTIONS = ["Open␣a␣database␣file"]
    self.variables_dropdown_data1.set(OPTIONS[0])
    self.dropdown_node_data1 = OptionMenu(
```

```
            self.raiz, self.variables_dropdown_data1, *OPTIONS)
        self.dropdown_node_data1.grid(row=1, column=1, sticky='NSEW')

        self.outrobotao = Button(self.raiz, text='Generate data')
        self.outrobotao.bind("<Button-1>", self.plota)
        self.outrobotao.grid(row=1, column=2, sticky='NSEW')

        # Terceira linha (Canvas)
        self.fig = Figure(figsize=(12, 6), facecolor='white')
        self.canvas = FigureCanvasTkAgg(self.fig, master=raiz)
        self.canvas.get_tk_widget().grid(row=2, column=0, columnspan=3,
            sticky='NSEW')

        # Quarta linha
        self.texto_total_gerado = Label(
            self.raiz, text='Select a period to calculate the total
                generated')
        self.texto_total_gerado.grid(
            row=3, column=0, columnspan=3, sticky='NSEW')

    #Funções da interface Gráfica
    def onOpen(self):
        ftypes = [('Database', '*.db'), ('All files', '*')]
        file_name = askopenfilename(filetypes=ftypes)

        if file_name != "":

            self.texto['text'] = file_name
            self.raiz.title('Solar Generation Analysis - File: ' +
                file_name)

            # Primeiro menu com a escolha dos nos
            self.variables_dropdown_nodes.set('')
            self.dropdown_nodes['menu'].delete(0, 'end')
            if self.texto['text'] == 'Nothing open yet':
                return
            else:
                new_choices = access_database.retrive_data(
                    "node_name", self.texto['text'])
```

```
            if len(new_choices) == 0:
                new_choices = ['Nothing found!']

        for choice in new_choices:
            self.dropdown_nodes['menu'].add_command(
                label=choice, command=lambda temp=choice: self.
                    variables_dropdown_nodes.set(temp))
        self.variables_dropdown_nodes.set(new_choices[0])

        # Segundo menu com a data
        self.variables_dropdown_data1.set('')
        self.dropdown_node_data1['menu'].delete(0, 'end')

        dates = access_database.get_unique_dates(self.texto['text'])

        new_choices.clear()

        for date in dates:
            new_choices.append(datetime.datetime.strptime(date, "%Y-%m
                -%d").strftime('%d/%m/%Y'))
        if not dates:
            new_choices = ['Nothing found!']
        for choice in new_choices:
            self.dropdown_node_data1['menu'].add_command(
                label=choice, command=lambda temp=choice: self.
                    variables_dropdown_data1.set(temp))

        self.variables_dropdown_data1.set(new_choices[len(new_choices)
            -1])


# Gera o gráfico
def plota(self, _):
    node = self.variables_dropdown_nodes.get()
    data = datetime.datetime.strptime(self.variables_dropdown_data1.
        get(), "%d/%m/%Y").strftime('%Y-%m-%d')

    self.fig.clear()
```

```
        fig, total_gerado = plotador.gerar_grafico_node_data(
            self.fig, node, data, self.texto['text'])

        if fig is None:
            print("No records found")
            messagebox.showwarning(
                "Warn", "No records found")
        else:
            self.canvas.draw()

            self.texto_total_gerado[
                'text'] = "A total of: {:.2f} Wh was generated".format(
                    total_gerado)

    def quit(self):
        self.raiz.destroy()

    def about(self):
        messagebox.showinfo(
            "Info", "################################\n# DSIF - FEEC - 
                Unicamp\n# Autor: Ricardo Mazza Zago\n
                ################################")

    # Exporta para CSV
    def export_to_csv(self):
        ftypes = [('.csv (Comma-separated values)', '*.csv')]
        f = asksaveasfilename(filetypes=ftypes, defaultextension=".csv")
        if f is None:
            return
        df = create_pandas_dataframe(self)
        df.to_csv(f)

    def export_to_xls(self):
        ftypes = [('.xls (Excel)', '*.xls')]
        f = asksaveasfilename(filetypes=ftypes, defaultextension=".xls")
        if f is None:
            return
        df = create_pandas_dataframe(self)
        df.to_excel(f)
```

```
    def save_image(self):
        ftypes = [('.png␣(PNG)', '*.png')]
        f = asksaveasfilename(filetypes=ftypes, defaultextension=".png")
        if f is None:
            return
        self.fig.savefig(f)

    def export_to_pdf(self):
        node = self.variables_dropdown_nodes.get()
        data = datetime.datetime.strptime(self.variables_dropdown_data1.
           get(), "%d/%m/%Y")
        ftypes = [('.pdf␣(PDF)', '*.pdf')]
        f = asksaveasfilename(filetypes=ftypes, defaultextension=".pdf")
        if f is None:
            return
        with PdfPages(f) as pages:
            for i in range(1, 32):
                fig = Figure(figsize=(12, 6), facecolor='white')
                if i <= 9:
                    data_texto = data.strftime('%Y-%m-0') + str(i)
                else:
                    data_texto = data.strftime('%Y-%m-') + str(i)
                (fig, _) = plotador.gerar_grafico_node_data(fig, node,
                   data_texto, self.texto['text'])
                if fig is not None:
                    canvas = FigureCanvasPdf(fig)
                    canvas.print_figure(pages)

def create_pandas_dataframe(self):
    node = self.variables_dropdown_nodes.get()
    data = datetime.datetime.strptime(self.variables_dropdown_data1.get(),
        "%d/%m/%Y").strftime('%Y-%m-%d')
    leituras = access_database.get_corrente_tensao_tempo_amostragem(
        node_name=node, data=data, file=self.texto['text'])
    temp = np.zeros(len(leituras[:,0]))
    for i in range(0,len(leituras[:,0])):
        temp[i] = float(leituras[i, 0]) * float(leituras[i, 1])
        leituras[i,3]=float(leituras[i,3])/1000
```

```
    leituras = np.column_stack((leituras,temp))
    df = pd.DataFrame(leituras)
    df.columns = ['Corrente␣(A)', 'Tensão␣(V)', 'Data␣e␣Hora', 'Tempo␣de␣
       medição␣(s)', 'Potência␣(W)']
    return df

raiz = Tk()
Janela(raiz)
raiz.mainloop()
```

C.1.7 aux_functions.py

```
##########################################
# LSM - DSIF - FEEC - Unicamp
# Autor: Ricardo Mazza Zago
# Dezembro de 2017
##########################################

import configparser
import os
import platform
import time

def number_sensors(parser):
    counter = 0
    while True:
        try:
            parser.get('node_' + str(counter + 1), 'name')
            counter = counter + 1
        except:
            return counter

def create_node_list():
    temp = list()
    for i in range(number_sensors(get_parser())):
        temp.append("node_" + str(i+1))
    return temp

def get_parser():
    # Lendo arquivo de configuracoes
```

```
    parser = configparser.SafeConfigParser()
    parser.read(get_path('config.ini'))
    return parser

def list_databases():
    f = []
    for (_, _, filenames) in os.walk(get_path("database")):
        f.extend(filenames)
        break
    return f

def get_path(file = None):
    if platform.system() == "Linux":
        slash='/'
    else:
        slash="\\"
    if file is None:
        path = os.path.dirname(os.path.realpath(__file__)) + slash
    else:
        path = os.path.dirname(os.path.realpath(__file__)) + slash + file
    return(path)

def is_time_to_cloud(cloud_control, node, tempo=60):
    if not node in cloud_control:
        cloud_control[node] = time.time()
        return True
    else:
        if time.time()-cloud_control[node]>=tempo:
            return True
        else:
            return False
```

C.2 Firmware

```
/*****************************************
  LSM - DSIF - FEEC - Unicamp
  Autor: Ricardo Mazza Zago
  Dezembro de 2017
*****************************************/
```

```
//http://playground.arduino.cc/Main/QuickStats
//https://github.com/dndubins/QuickStats
//Put files in user: C:\Users\user_name\Documents\Arduino\libraries\
   QuickStats or in the same folder
#include "QuickStats.h"
//To suport Xbee API Mode
#include "XBee.h"

//Some definitions
//Number medians to take every second
const int median_samples = 3;
//Number of seconds to measure before send data to central node
const int seconds_to_made_avg = 10;
//Use Xbee API Mode
const bool API_Mode = true;
//Max seconds to sample
const int max_seconds = 120;


int sensorPin0 = A0; // Select the input pin for the tensao
int sensorPin1 = A1; // Select the input pin for the corrente
float tensao[median_samples]; // Voltage samples
float corrente[median_samples]; // Current samples
unsigned long pastMillis = 0; //To control time
float array_tensao[max_seconds]; //Every second sample
float array_corrente[max_seconds]; //Every second sample
unsigned long pastMillis_envio = 0; //To control time between readings
unsigned long actualMillis_envio = 0; //To control time between readings

int leituras_atual = 0;
float media_tensao;
float media_corrente;

String text;
XBee xbee = XBee();
uint8_t data[45];

// SH + SL Address of receiving XBee
XBeeAddress64 addr64 = XBeeAddress64(0x00000000, 0x00000000);
```

```
ZBTxRequest zbTx = ZBTxRequest(addr64, data, sizeof(data));
ZBTxStatusResponse txStatus = ZBTxStatusResponse();

QuickStats stats;

int statusLed = 13;
int errorLed = 13;

void setup() {
  pinMode(statusLed, OUTPUT);
  pinMode(errorLed, OUTPUT);
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port
        only
  }
  xbee.setSerial(Serial);
}

void loop() {
  //Time control
  while ((unsigned long)(millis() - pastMillis) >= 1000)
  {
    //Get median_samples measures to remove outliars
    for (int i = 0; i < median_samples; i++) {
      tensao[i] = 5 * (float)analogRead(sensorPin0) / 1023;
      delay(1);
      tensao[i] = 18.10 * tensao[i];
      corrente[i] = 5 * (float)analogRead(sensorPin1) / 1023;
      corrente[i] = 6.88 * (2.5 - corrente[i]);
      delay(1);
    }

    //Find median
    array_tensao[leituras_atual] = stats.median(tensao, median_samples);
    array_corrente[leituras_atual] = stats.median(corrente, median_samples
        );

    leituras_atual = leituras_atual + 1;
```

```
if (leituras_atual >= seconds_to_made_avg) {
  media_tensao = 0;
  media_corrente = 0;
  for (int i = 0; i < leituras_atual; i++) {
    media_tensao = media_tensao + array_tensao[i];
    media_corrente = media_corrente + array_corrente[i];
  }
  media_tensao = media_tensao / leituras_atual;
  media_corrente = media_corrente / leituras_atual;

  actualMillis_envio=millis();

  //Send data via serial
  text = String("{\"t\":") + String(millis() / 1000, DEC) + String("
      ,\"r\":") + String(actualMillis_envio-pastMillis_envio, DEC) +
      String(",\"v\":") + String(media_tensao, 3) + String(",\"c\":") +
       String(media_corrente, 3) + String("}");

  if (API_Mode) {
    text.toCharArray(data, 45);
    xbee.send(zbTx);

    if (xbee.readPacket(300)) {
      if (xbee.getResponse().getApiId() == ZB_TX_STATUS_RESPONSE) {
        xbee.getResponse().getZBTxStatusResponse(txStatus);
        if (txStatus.getDeliveryStatus() == SUCCESS) {
          //flashLed(statusLed, 4, 50);
          pastMillis_envio=actualMillis_envio;
          leituras_atual = 0;
        } else {
          flashLed(errorLed, 3, 250);
        }
      }
    }
    else
    {
      flashLed(errorLed, 3, 250);
    }
```

```
      }
      else
      {
        Serial.print(text);
        pastMillis_envio=actualMillis_envio;
        leituras_atual = 0;
      }
    }
    pastMillis = millis();

    if(leituras_atual>=max_seconds-1){
      leituras_atual=0;
      pastMillis_envio=actualMillis_envio;
    }
  }
}

void flashLed(int pin, int times, int wait) {

  for (int i = 0; i < times; i++) {
    digitalWrite(pin, HIGH);
    delay(wait);
    digitalWrite(pin, LOW);

    if (i + 1 < times) {
      delay(wait);
    }
  }
}
```