

Gabriel Favalesso Fraga

**EstGeiger: Proposta de solução IoT para
medição de níveis de radiação com transmissão
LoRa**

Vitória, ES

2018

Gabriel Favalesso Fraga

EstGeiger: Proposta de solução IoT para medição de níveis de radiação com transmissão LoRa

Monografia apresentada ao Curso de Engenharia de Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Engenharia de Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Departamento de Informática

Orientador: Prof. Dr. Roberto Colistete Jr.

Coorientador: Prof. Dr. Moisés Renato Nunes Ribeiro

Vitória, ES

2018

Gabriel Favalesso Fraga

EstGeiger: Proposta de solução IoT para medição de níveis de radiação com transmissão LoRa/ Gabriel Favalesso Fraga. – Vitória, ES, 2018
98 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Roberto Colistete Jr.

Projeto de Graduação – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Departamento de Informática, 2018.

1. IoT. 2. LoRa. 3. Radiação. I. Colistete Jr., Roberto. II. Universidade Federal do Espírito Santo. IV. EstGeiger: Proposta de solução IoT para medição de níveis de radiação com transmissão LoRa

CDU 02:141:005.7

Gabriel Favalesso Fraga

EstGeiger: Proposta de solução IoT para medição de níveis de radiação com transmissão LoRa

Monografia apresentada ao Curso de Engenharia de Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Engenharia de Computação.

Vitória, ES, 07 de dezembro de 2018:

Prof. Dr. Roberto Colistete Jr.
Orientador

**Prof. Dr. Moisés Renato Nunes
Ribeiro**
Coorientador

Prof. Dr. José Luis Passamai Junior
Convidado 1

Prof. Dr. Rodolfo da Silva Villaça
Convidado 2

Vitória, ES
2018

Agradecimentos

Gostaria de agradecer aos meus pais, Adilson e Helena, e minha irmã Elizangela, por terem me dado todo o suporte que eles foram capazes, por sempre apoiarem minhas escolhas e por acreditarem e me fazerem acreditar que o acesso a educação é o melhor presente que alguém pode ganhar. Agradecer a Mayene, por ter me ajudado a evoluir como pessoa, me apoiado, sempre ter estado comigo enfrentando todas as dificuldades, não só da vida acadêmica, como em todas situações da vida.

A todas as amizades feitas na universidade que foram essenciais para que eu tenha atingido essa conquista, sem elas o trajeto teria sido extremamente mais complicado. Em especial aos amigos, Enéias, Fricks, Gustavo, Marco, Marcos e Raffael, por termos enfrentado todos os desafios da faculdade e com a ajuda de um dos outros conseguido superar, assim como a ajuda de alguns deles no desenvolvimento deste projeto. Também ao Alex, por ter contribuído de várias formas para o desenvolvimento deste projeto.

Ao meu orientador, Roberto, por sempre dedicar seu tempo para me auxiliar em diversas questões durante o desenvolver do projeto e principalmente me incentivado de várias formas possíveis. Também aos professores do Grupo de Física Aplicada da UFES, principalmente ao Prof. Marcos Tadeu, por oferecerem a parceria e conseguir toda uma estrutura para instalação dos dispositivos desenvolvidos neste projeto e também ao Prof. Passamai, que será o usuário deste projeto, contribuindo também com a sua instalação. Aos demais professores da UFES, por de alguma forma, mesmo que indiretamente, terem contribuído para minha formação, é uma profissão de muito respeito e merece ser mais valorizada.

Ao LabNerds e ao projeto FUTEBOL, por incentivarem pesquisas do tipo e por cederem parte dos equipamentos utilizados no projeto. Também ao coorientador Moisés, por auxiliar no que foi preciso e ter articulado o contato inicial com o Prof. Marcos Tadeu, que por consequência acabou transformando o projeto em algo maior.

A todo o time de Arquitetura de Tecnologia Industrial da Vale, por toda a ajuda e aprendizado que me passaram, por toda atenção e compreensão neste tempo que estive com eles. Mas principalmente por terem me incentivado na área e consequentemente na escolha do tema deste projeto.

Gostaria também de agradecer a todas as pessoas envolvidas na instalação dos sensores, que oferecerem todo tipo de ajuda possível. Em especial ao Hotel da Léa, Restaurante Gaeta e Restaurante Curuca por disponibilizarem espaço e infraestrutura para o projeto.

“Daria tudo o que sei pela metade do que ignoro.”

René Descartes

Resumo

O projeto desenvolvido neste trabalho tem como proposta a criação e instalação de uma infraestrutura de dispositivos de Internet das Coisas (ou IoT - *Internet of Things*) para monitoramento remoto da radiação beta e gama na Praia de Meaípe, localizada no bairro Meaípe, Guarapari, no estado do Espírito Santo. Tal infraestrutura consiste, inicialmente, de dois dispositivos nós, apelidados de EstGeiger (Estação Geiger) que fazem a coleta dos dados de níveis de radiação e transmitem através do protocolo LoRaWAN para um concentrador também instalado na região, que por sua vez envia tais informações para plataformas IoT na *internet*, onde as informações ficarão disponíveis para os usuários finais.

Este monitoramento é de grande importância para o Grupo de Física Aplicada da Universidade Federal do Espírito Santo, que desenvolvem vários estudos que visam o bem estar da sociedade. A utilização de dispositivos que fazem a coleta e transmissão dos dados remotamente será de grande utilidade para o Grupo de Física Aplicada, pois além de ter os dados disponíveis em tempo real, economiza muito esforço para realizar as medições, que antes eram feitas manualmente e necessitavam de integrantes irem ao local.

Palavras-chaves: IoT; LoRa; LoRaWAN; Radioatividade;

Abstract

The project developed in this work proposes the creation and installation of an Internet of Things (IoT) infrastructure for remote monitoring of radiation beta and gamma in Meaípe's beach, located in the Meaípe neighborhood, Guarapari, in Espírito Santo. This infrastructure initially consists of two nodes, known as EstGeiger (Geiger Station), which collects data from radiation levels and transmits over the LoRaWAN protocol to a gateway also installed in the region, which in turn sends such information to IoT platforms on the *Internet*, where information will be available to end users.

This monitoring have a great importance for the Applied Physics Group of the Federal University of Espírito Santo, which develop several studies aimed at the welfare of society. The use of devices that collect and transmit the data remotely will be of great use for the Applied Physics Group, since in addition to having the data available in real time, it saves a lot of effort to carry out the measurements, which were previously done manually and needed members to go to the place.

Keywords: IoT; LoRa; LoRaWAN; Radioactivity;

Lista de ilustrações

Figura 1 – Penetração dos tipos de radiação	21
Figura 2 – Gráfico do tempo de vida útil da bateria e alcance.	26
Figura 3 – Arquitetura de uma rede LoRaWAN	28
Figura 4 – Topologia básica	28
Figura 5 – Diagrama de camadas LoRaWAN	29
Figura 6 – Frequência dos canais - AU915-928	30
Figura 7 – Dispositivo Safecast bGeigie Nano	32
Figura 8 – Linha de dispositivos uRADMonitor	33
Figura 9 – Dashboard de monitoramento dos dispositivos uRADMonitor	33
Figura 10 – Versão final da EstAcqua em funcionamento na balsa no Lago Terra Alta	34
Figura 11 – LoPy 4	35
Figura 12 – Expansion Board V2.0	36
Figura 13 – Sensor de radiação CAJOE - modelo NGMC-V1	37
Figura 14 – Esquema de funcionamento de um tubo Geiger-Müller	37
Figura 15 – Saída do sensor medida no osciloscópio ao detectar radiação	38
Figura 16 – Sensor de umidade, pressão atmosférica e temperatura, BME280	38
Figura 17 – Página inicial de uma aplicação na TTN	39
Figura 18 – Integrações disponíveis na TTN	40
Figura 19 – Diagrama lógico de funcionamento do nó	43
Figura 20 – Conexão entre sensor de radiação e LoPy 4	44
Figura 21 – Conexão entre sensor BME280 e LoPy 4	44
Figura 22 – Criação da estrutura do dispositivo	45
Figura 23 – Versão final do nó EstGeiger	46
Figura 24 – Arquitetura da EstGeiger	48
Figura 25 – Arquitetura baseada no protocolo LoRaWAN	49
Figura 26 – Distância de transmissão alcançada em ambiente urbano	50
Figura 27 – Distância de transmissão alcançada em ambiente aberto	51
Figura 28 – Esquema de testes com fonte radioativa	52
Figura 29 – Dashboard Cayenne de um dispositivo	53
Figura 30 – Gráfico de radiação ao longo de um dia na plataforma Cayenne	54
Figura 31 – Gráfico de temperatura em um período selecionado	54
Figura 32 – Gráfico de umidade no período de uma semana	54
Figura 33 – Gráfico de pressão atmosférica no período de uma semana	55
Figura 34 – Gráfico do consumo de bateria a longo de uma semana	55
Figura 35 – Dashboard da página de dispositivos do Ubidots	56
Figura 36 – Gráfico de radiação na plataforma Ubidots	56

Figura 37 – Exemplo de histograma	57
Figura 38 – Localização dos dispositivos instalados	58
Figura 39 – Instalação do nó EstGeiger-01	59
Figura 40 – Instalação do nó EstGeiger-02	60
Figura 41 – Gráfico de radiação medida em Meaípe, pelo nó EstGeiger-01	61
Figura 42 – Gráfico de temperatura medida em Meaípe, pelo nó EstGeiger-01	61
Figura 43 – Gráfico de umidade medida em Meaípe, pelo nó EstGeiger-01	62
Figura 44 – Gráfico de pressão medida em Meaípe, pelo nó EstGeiger-01	62
Figura 45 – Gráfico de radiação medida em Meaípe, pelo nó EstGeiger-02	62
Figura 46 – Gráfico de temperatura medida em Meaípe, pelo nó EstGeiger-02	62
Figura 47 – Gráfico de umidade medida em Meaípe, pelo nó EstGeiger-02	63
Figura 48 – Gráfico de pressão medida em Meaípe, pelo nó EstGeiger-02	63
Figura 49 – Distância de transmissão alcançada no teste em Meaípe	63
Figura 50 – Mini solar Lipo Charger v1.0	69
Figura 51 – SparkFun MOSFET Power Controller	69

Lista de tabelas

Tabela 1 – Estimativa de dispositivos conectados	24
Tabela 2 – Taxa de dados para o padrão LoRaWAN AU915-928 (ALLIANCE, 2015)	31
Tabela 3 – Tamanho máximo de <i>payload</i> por DR (ALLIANCE, 2015)	31
Tabela 4 – Custo do projeto - Nό EstGeiger	47
Tabela 5 – Custo do projeto - NanoGateway	47
Tabela 6 – Comparação de medições com fonte radioativa à 15 cm	52
Tabela 7 – Comparação de medições com fonte radioativa à 10 cm	52
Tabela 8 – Comparação de medições com fonte radioativa à 5 cm	52
Tabela 9 – Comparação de medições com fonte radioativa à 0 cm	53
Tabela 10 – Consumo da EstGeiger	57
Tabela 11 – Distância entre nós e NanoGateway	59
Tabela 12 – <i>Airtime</i> calculado	64
Tabela 13 – Taxa de perda de pacotes	65

Listas de abreviaturas e siglas

ADC	Analog to Digital Converter
ANATEL	Agência Nacional de Telecomunicações
API	Application Programming Interface
BW	Bandwidth
Chirp	Compressed High Intensity Radar Pulse
CPM	Counts Per Minute
CPS	Counts Per Second
CR	Code Rate
CSS	Chirp Spread Spectrum
CSV	Comma-Separated Values
GPIO	General Purpose Input/Output
GPS	Global Positioning System
FTP	File Transfer Protocol
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
IoT	Internet of Things
LPP	Low Power Packet
LPWAN	Low Power Wide Area Network
NERDS	Núcleo de Estudos em Redes Definidas por Software
RSSI	Received Signal Strength Indicator
SF	Spreading Factor
SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface

TTN The Things Network
UFES Universidade Federal do Espírito Santo
USB Universal Serial Bus

Sumário

1	INTRODUÇÃO	16
1.1	Motivação	16
1.2	Objetivos	17
1.2.1	Objetivo Geral	17
1.2.2	Objetivos Específicos	17
1.3	Metodologia	18
1.4	Organização do Texto	18
2	FUNDAMENTAÇÃO TEÓRICA	20
2.1	Radiação	20
2.1.1	Conceitos	20
2.1.1.1	Radiação Alfa, Beta e Gama	20
2.1.1.2	Efeitos Biológicos da Radiação	22
2.1.2	Aplicação	22
2.2	Internet das Coisas	23
2.2.1	Definição	23
2.2.2	Impacto Socioeconômico	23
2.2.3	Desafios de Implementação	24
2.2.4	Aplicação	25
2.3	Tecnologia LoRa	25
2.3.1	Camada Física	26
2.3.2	Protocolo LoRaWAN	27
2.3.2.1	Topologia	27
2.3.2.2	Classes de Dispositivos	28
2.3.2.3	Segurança	29
2.3.3	Regulamentação	29
2.3.3.1	Frequência dos Canais	30
2.3.3.2	Taxa de Dados	30
3	TRABALHOS RELACIONADOS	32
3.1	Produtos de Mercado	32
3.1.1	Safecast bGeigie Nano - Mobile Radiation Monitoring Device	32
3.1.2	uRADMonitor	33
3.2	EstAcqua	34
4	ARQUITETURA E IMPLEMENTAÇÃO	35

4.1	Hardware Utilizado	35
4.1.1	LoPy 4	35
4.1.2	Expansion Board	36
4.1.3	Sensor de Radiação	36
4.1.4	BME280	38
4.2	Plataformas IoT	39
4.2.1	The Things Network	39
4.2.2	Cayenne (CAYENNE, 2018)	40
4.2.3	Ubidots (UBIDOTS, 2018)	40
4.3	Implementação de Software	41
4.3.1	Características da EstGeiger	41
4.3.2	Diagrama de Funcionamento	42
4.4	Construção	42
4.4.1	Conexão dos Sensores	42
4.4.2	Desenvolvimento e Montagem da Estrutura	43
4.4.3	Custo do Projeto	47
4.5	Arquitetura do Projeto	47
5	AVALIAÇÃO E RESULTADOS	50
5.1	Distância de Transmissão	50
5.1.1	Ambiente Urbano (sem linha de visada)	50
5.1.2	Ambiente Aberto (com linha de visada)	51
5.2	Validação dos Dados	51
5.3	Testes de Funcionamento	53
5.4	Consumo de Energia	57
5.5	Instalação Inicial em Meaípe	58
5.5.1	Instalação dos Dispositivos	58
5.5.2	Dados Medidos	61
5.5.3	Testes de Distância	63
5.6	Airtime, Colisão e Taxa de Perda de Pacotes	64
5.6.1	Airtime	64
5.6.2	Colisão	64
5.6.3	Taxa de Perda de Pacotes	65
5.7	Avaliação	65
5.7.1	Pontos Positivos	65
5.7.2	Pontos a Melhorar	66
6	CONCLUSÕES E TRABALHOS FUTUROS	67
6.1	Perspectivas e Trabalhos Futuros	67
6.1.1	Controle via IoT Cloud de configurações da EstGeiger	68

6.1.2	Reenvio mais robusto de pacotes pelo NanoGateway	68
6.1.3	Otimização de consumo de energia	69
6.1.4	Servidor LoRaWan e IoT Cloud local na UFES	70
6.1.5	Envio de dados estatísticos de radiação	70
6.1.6	Uso de outros medidores Geiger	70
6.1.7	Versão portátil da EstGeiger	71
6.1.8	Uso de Gateway LoRaWan	71
 REFERÊNCIAS		 72
 APÊNDICES		 75
 APÊNDICE A – SOFTWARE TTN PAYLOAD DECODER		 76
 APÊNDICE B – SOFTWARE DO NANOGATEWAY		 77
 APÊNDICE C – SOFTWARE DO ESTGEIGER NÓ		 89

1 Introdução

1.1 Motivação

Com volumes gigantescos de dados e em ambientes cada vez mais conectados e repletos de métricas, a Internet das Coisas (ou IoT – *Internet of Things*) vem conquistando cada vez mais espaço e em uma velocidade assustadora.

A Internet das Coisas é um termo que vem sendo dito desde meados de 2009 e uma das definições mais bem aceitas é dada por Kevin Ashton, num artigo publicado em RFID Journal (2009), que diz o seguinte: “... se tivéssemos computadores que soubessem de tudo o que há para saber sobre coisas, usando dados que foram colhidos, sem qualquer interação humana, seríamos capazes de monitorar e mensurar tudo, reduzindo o desperdício, as perdas e o custo. Gostaríamos de saber quando as coisas precisarão de substituição, reparação ou atualização, e se eles estão na vanguarda ou se tornaram obsoletos.”.

Desde então IoT vem se desenvolvendo com o uso de diversas tecnologias de transmissão (principalmente wireless), sistemas microeletromecânicos (Micro-Electro-Mechanical Systems) e a *Internet*.

IoT é tido com uma das molas propulsoras da transformação digital, por impulsionar e amparar processos de inovação em todas as áreas.

Dito isso, uma excelente aplicação é a criação de uma infraestrutura de dispositivos para medir o nível de radiação ao longo de uma praia a fim de analisar o impacto na saúde da população ao redor. Essa tarefa já é feita por um Grupo de Física Aplicada da UFES, porém o monitoramento da radiação das praias é feito presencialmente e depende de integrantes irem pessoalmente ao local e realizarem a medição manualmente.

Estudos de níveis de radiação nas praias de Guarapari e arredores vem sendo feito durante alguns anos por pesquisadores da UFES. As praias em questão também foram alvo de pesquisa da comunidade internacional como é o caso de FUJINAMI N.; T. KOGA (1999) que mostra um estudo feito nas praias de Guarapari e Meaípe.

O uso de novas tecnologias de transmissão e de dispositivos capazes de realizarem medições remotamente em tempo integral, enviando os dados medidos para *Internet* onde poderão ser acessados e analisados pelos pesquisadores, trará uma enorme contribuição para esse tipo de pesquisa.

Além de todo o desenvolvimento tecnológico feito no trabalho, também visamos um objetivo de cunho mais social, de forma a levar o conhecimento para a sociedade, isso é feito disponibilizando os dados medidos publicamente na Internet. Tais dados poderão

ser utilizados em um momento futuro por escolas, a fim de abordar o tema mais a fundo e trazer mais incentivo para os estudantes. Como Pereira (2014) mostra em seu trabalho existe um certo déficit nos currículos de física do ensino médio a respeito de tal assunto e iniciativas como a proposta deste trabalho contribuem para uma melhora neste ponto.

1.2 Objetivos

Confira nesta seção os objetivos gerais e específicos deste projeto.

1.2.1 Objetivo Geral

O projeto em questão irá abordar uma aplicação de IoT de baixo custo, que consiste no desenvolvimento de uma estação Geiger para medição de níveis de radiação. Tal dispositivo deve medir o nível de radiação no local, enviar os dados para um gateway que por sua vez irá enviar esses dados para uma plataforma IoT na nuvem, onde ficará disponível para análise.

1.2.2 Objetivos Específicos

Através do desenvolvimento deste projeto, deseja-se alcançar os seguintes objetivos específicos:

- Utilizar componentes de baixo custo e softwares livres, de forma que seja um projeto acessível a qualquer empresa ou pessoa;
- Uso da tecnologia de transmissão LoRa, implementando o protocolo LoRaWAN;
- Uso de plataformas IoT gratuitas, que permitam o armazenamento de dados por um dado tempo e que os usuários possam visualizar e adquirir tais dados;
- Validar dados do sensor Geiger utilizado, de forma que os valores medidos não destoem dos valores medidos pelo dispositivo Geiger já utilizado pelo Grupo de Física Aplicada;
- Possuir uma precisão que atenda as necessidades de medição, definidas pelos usuários;
- Usar microcontroladores que permitam o uso da linguagem de programação MicroPython;
- Fazer datalogger local nas estações para guardar informações mais detalhadas;
- Utilizar sensores de monitoramento, como temperatura, umidade, pressão e nível de bateria, para que facilite o monitoramento da EstGeiger, visto que estará instalada em outra cidade e em local de difícil acesso;

- Utilizar uma estrutura que facilite a instalação da EstGeiger nos locais disponibilizados;

1.3 Metodologia

Com base nos objetivos e nos requisitos apresentados, o projeto deve funcionar baseado nas seguintes etapas:

- (a) **Aquisição dos dados:** consiste na primeira etapa de todo o processo, onde os níveis de radiação do ambiente são medidos pelo sensor Geiger, que será conectado a um microcontrolador que é responsável por processar os dados;
- (b) **Transmissão dos dados:** uma vez tendo os dados de radiação devidamente processados, o microcontrolador deve empacotá-los para enviar via LoRa para o gateway, que por sua vez irá enviar esses dados via WiFi para plataformas IoT na Internet;
- (c) **Visualização dos dados:** a visualização dos dados será através das plataformas IoT na *Internet*. Deve conter não só o último valor medido, mas também um histórico em modo gráfico dos valores e datas correspondentes. Deve ainda ser capaz de exportar tais dados (em arquivo csv, por exemplo) para que o usuário possa tratar da maneira que quiser;
- (d) **Criação da estrutura:** a estrutura desenvolvida deve encapsular os sensores e o microcontrolador, de forma a deixar a antena bem posicionada para facilitar a transmissão dos dados. Inicialmente, a aplicação conta com duas EstGeiger, responsáveis por fazer a medição dos níveis de radiação e um gateway, apelidado de NanoGateway, responsável por enviar os dados colhidos para *Internet*.
- (e) **Instalação:** a instalação será feita em torres previamente instaladas pelo Grupo de Física Aplicada da UFES. Tais torres possuem caixas a prova de água e poeira onde a EstGeiger ficará acoplada. O gateway por sua vez deverá ser instalado em um ponto que tenha visibilidade para todas as EstGeiger, com o objetivo de receber os dados da melhor forma possível, ou seja, sem perdas freqüentes de pacotes.

1.4 Organização do Texto

O trabalho está estruturado em 6 capítulos.

Após a Introdução apresentada, o Capítulo 2 mostra os fundamentos teóricos abordados no projeto, como conceitos dos fenômenos medidos e explicações das tecnologias utilizadas.

O Capítulo 3 vai mostrar alguns outros trabalhos que abordam o mesmo tema e foram utilizados como base para este.

O Capítulo 4 vai detalhar a arquitetura do projeto, mostrando o hardware utilizado, falando sobre as plataformas IoT, a implementação do Software embarcado e também como foi feita a construção para chegar no resultado final da EstGeiger.

O Capítulo 5 analisará os resultados encontrados e vai avaliar o que atendeu os requisitos e o que pode ser melhorado.

No Capítulo 6 são apresentadas as conclusões finais e propostas de trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo será abordado os principais conceitos envolvidos no projeto. Como trata-se de uma estação para medição de níveis de radiação com características de um projeto de IoT, será falado a respeito dos três principais fatores envolvidos, que é o evento a ser monitorado, ou seja, a radiação, a tecnologia de transmissão utilizada e uma breve contextualização a respeito de Internet das Coisas, que é a área deste projeto.

2.1 Radiação

Nesta seção será discutido de forma sucinta os conceitos de radiação, objeto de estudo da EstGeiger. Por se tratar de um tema presente em diversas áreas, será dado apenas foco nos pontos fundamentais a respeito da radiação com o objetivo principal de compreender a atuação do dispositivo desenvolvido.

2.1.1 Conceitos

Radiação é energia que viaja sob a forma de partículas de alta velocidade(radiação de partículas) ou ondas(radiação eletromagnética).

A radiação de partículas ocorre quando um átomo instável (ou radioativo) se desintegra. A radiação eletromagnética (EM), por outro lado, não tem massa e viaja pelas ondas. A radiação EM pode variar de energia muito baixa para energia muito alta, e chamamos esse espaço de espectro eletromagnético. Dentro do espectro EM, existem dois tipos de radiação – ionizantes e não ionizantes (SANTIAGO, 2017).

Os materiais radioativos e a sua decorrente radioatividade produzida existem no espaço sideral desde a origem do universo. Vários materiais radioativos fizeram parte da formação do planeta Terra e estão aqui até hoje.

2.1.1.1 Radiação Alfa, Beta e Gama

Existem três modalidades de radiações denominadas alfa, beta e gama que podem ser separadas por um campo magnético ou por um campo elétrico.

- **Radiação Alfa (α):** também chamada de partículas alfa ou raios alfa, nada mais são do que partículas carregadas por dois prótons e dois nêutrons, sendo, portanto, núcleos de hélio. As partículas alfa são bastante energéticas, mas são facilmente barradas por uma folha de papel;

- **Radiação Beta (β):** raios beta ou partículas beta, têm em sua composição elétrons com carga negativa, correspondente a um elétron; ou positiva, correspondente a um pósitron (anti elétron ou anti matéria). As partículas beta são mais penetrantes e menos energéticas que as partículas alfa, conseguem atravessar lâminas de chumbo de até 2 mm ou de alumínio de até 5 mm no ar, mas podem ser barradas até por uma placa de madeira com pelo menos 2,5 cm de espessura;
- **Radiação Gama (γ):** o comprimento de onda deste tipo de radiação varia de 0,5 Å a 0,005 Å(unidade de medida: ångström). Os raios gama são ondas eletromagnéticas, e possuem carga e massa nulas, emitem continuamente calor e têm a capacidade de ionizar o ar e torná-lo condutor de corrente elétrica. As partículas gama percorrem milhares de metros no ar, são mais perigosas, quando emitidas por muito tempo podem causar má formação nas células. Os raios gama conseguem atravessar chapas de aço de até 15 cm de espessura, mas são barradas por grossas placas de chumbo ou paredes de concreto (SANTIAGO, 2017).

A Figura 1 exemplifica como é a penetração da radiação alfa, beta e gama.

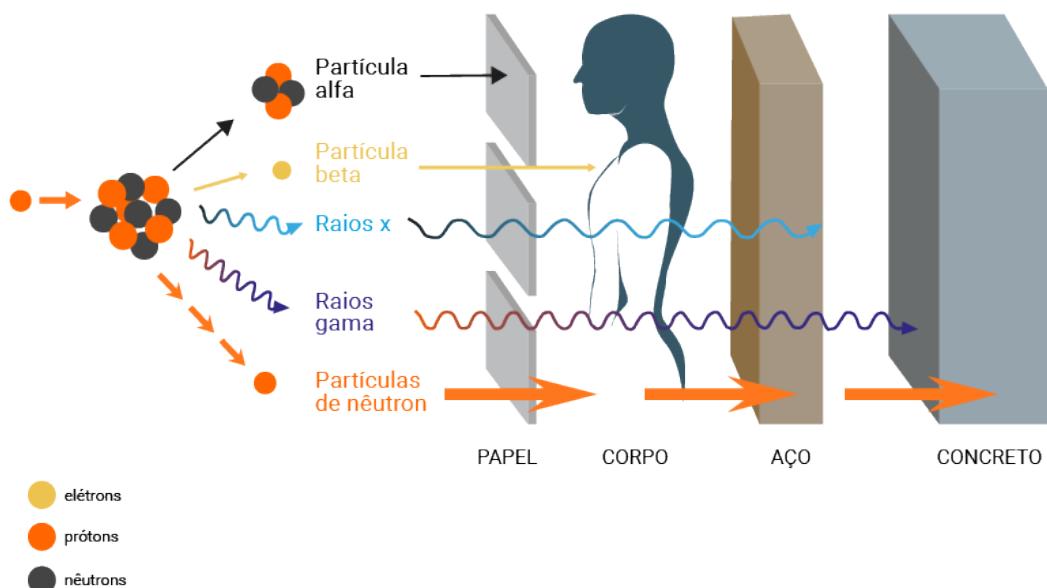


Figura 1 – Penetração dos tipos de radiação
Fonte: Blog Radioproteção na Prática

O sensor utilizado no projeto da EstGeiger para medir radiação, mede apenas radiação beta e gama, por conta de seu tipo. O sensor será abordado mais detalhadamente no Capítulo 4.1.3.

2.1.1.2 Efeitos Biológicos da Radiação

Desde a descoberta da radiação mais de um século de pesquisa tem fornecido grande conhecimento acerca dos mecanismos biológicos pelos quais esta pode afetar a saúde.

Sabe-se que a radiação pode produzir efeitos em nível celular, causando sua morte ou modificação. Quando o número de células afetadas ou até mesmo mortas for grande o suficiente, a radiação poderá resultar na disfunção e morte do órgão. A probabilidade de ocorrência de alguns efeitos é proporcional à dose de radiação recebida, sem a existência de limiar. Isto significa que doses pequenas, abaixo dos limites estabelecidos por normas e recomendações de proteção radiológica, podem induzir tais efeitos. Entre estes efeitos, destaca-se o câncer (SANTIAGO, 2017).

Em contrapartida, também existem várias pesquisas que mostram que a exposição ou uso moderado da radiação pode ser benéfico a saúde, como redução dos casos de câncer por exemplo. Por conta disso o estudo da radiação é muito importante, para que possa ser utilizada com mais clareza para o bem estar da sociedade.

2.1.2 Aplicação

Muitos estudos são feitos atualmente envolvendo temas ligados a radiação. Neste projeto vamos focar no trabalho feito pelos pesquisadores do departamento de Física da UFES, que desenvolvem pesquisas feitas a partir do monitoramento da areia monazítica nas praias de Meaípe e arredores.

A escolha do local se deu devido ao elevado índice de radiação medidos, por conta de ser uma cidade construída próxima a região com areia monazítica como cita FUJINAMI N.; T. KOGA (1999) VASCONCELOS (2013) em seu trabalho.

Depois de alguns anos de estudo, descobriram que as praias da Areia Preta e Meaípe, podem contribuir para a redução dos casos de câncer de mama. A atmosfera dessas duas regiões tem um índice elevado de radiação presente. A pesquisa extensa que foi feita, mostra, estatisticamente, que o nível de radiação no local gera essa atividade biopositiva (CBN, 2017).

Em outras pesquisas que foram feitas medições da radioatividade na faixa de areia utilizada pelos banhistas das praias de Meaípe mostram que o material radioativo não é fixo, ou seja, houve movimentação entre alguns pontos. Assim o nível de radiação em determinados pontos são alterados dependendo dessa movimentação (CALHEIRO D. S.; PASSAMAI JR., 2016b) (CALHEIRO D. S.; PASSAMAI JR., 2016a).

O projeto da EstGeiger busca trazer mais dados que possam ajudar nas pesquisas. A idéia é instalar os dispositivos em diversos pontos de interesse, para que sejam monitorados em tempo integral e remotamente, assim gerando informações que contribuam ainda mais

para obtenção de resultados nesta área.

2.2 Internet das Coisas

2.2.1 Definição

A Internet é uma das maiores revoluções tecnológicas criadas pela humanidade, ela se tornou parte integrante do dia a dia de empresas, setor público e indivíduos, ao ponto que a tecnologia passou a ser indispensável. Uma grande mudança tecnológica está acontecendo mundialmente e está centrada em torno da Internet das Coisas, principalmente devido aos avanços tecnológicos no desenvolvimento de hardware mais acessíveis, que tornou os custos de produção mais baratos, além da redução de peso, tamanho e baixo consumo energético; e consequentemente alavancou a crescente criação de dispositivos inteligentes como sensores vestíveis (wearables) (PEREIRA ; CARVALHO, 2017).

A Internet das Coisas advém da evolução da tecnologia de comunicação Máquina-a-Máquina (M2M) que através da interconexão das “coisas” ou objetos inteligentes, visa enfatizar, além da monitoração e controle, os processos de otimização e autonomia.

No cenário atual, os objetos inteligentes vêm ocupando cada vez mais um espaço permanente na vida das pessoas. A IoT permite que esses objetos se conectem à Internet, e essa conexão se torna essencial para que haja a coleta de informações, interação desses objetos entre si e com pessoas, podendo controlá-los remotamente com bases nas informações recebidas, criando novos padrões no âmbito social e organizacional (EVANS, 2016).

2.2.2 Impacto Socioeconômico

Em relação ao impacto socioeconômico, há uma discussão crescente sobre as perspectivas e previsões da IoT no Brasil e no mundo. Segundo a Cisco 2014, aproximadamente 50 bilhões de dispositivos estarão conectados em 2020, conforme mostra a Tabela 1. Como o Brasil é uma das maiores economias em crescimento no âmbito mundial e considerado o quarto mercado em M2M, segundo os dados do Ministério das Comunicações, desempenhará uma função relevante no mercado de IoT.

A International Data Corporation (IDC) prevê que o mercado de IoT chegará a US\$ 7,1 trilhões em 2020. A Cisco estima que a IoT pode acrescentar US\$ 352 bilhões na economia brasileira até a mesma data, sendo a maioria dos projetos relacionados ao setor de iniciativa privada que deve aumentar pelo menos em 20% no volume de recurso (TELECO, 2016). Adicionalmente, a Agência Brasileira de Desenvolvimento Industrial (ABDI) estima que nos próximos 25 anos haverá investimento crescente para modernização de infraestruturas das cidades. É esperado que a implementação da IoT no cotidiano irá

mudar pessoas e empresas e a forma como interagem com o mundo virtual, pelo aumento do número de informações disponíveis, aumento da produtividade e redução de custos global.

Tabela 1 – Estimativa de dispositivos conectados

Fonte: (PEREIRA ; CARVALHO, 2017)

Autor	Estimativa
Forbes 2014	> 40 bilhões de dispositivos conectados até 2020.
Tera 2013	50 bilhões de equipamentos conectados até 2020.
Gentili 2015	13,4 bilhões de dispositivos conectados, 130 milhões no Brasil
Teleco	30 milhões de medidores inteligentes em 2025 no Brasil 9 milhões de objetos conectados na indústria no Brasil 2 milhões de objetos conectados nas cidades inteligentes em 2025 no Brasil
Gartner 2013	incremento de receita > 300 bilhões de dólares até 2020 1,9 trilhão de dólares na economia global segmentos na liderança da IoT serão a manufatura (15%), a saúde (15%) e os seguros(11%).
IdC Brasil	130 milhões de dispositivos conectados em 2015 o mercado de IoT crescerá para cerca de US\$ 15,6 bilhões em 2020.

2.2.3 Desafios de Implementação

Como toda tecnologia inovadora, IoT encontrará desafios no que diz respeito à implementação. As principais barreiras a serem superadas tem relação com a falta de ambiente favorável (com dispositivos e equipamentos adequados) e a questão da segurança (privacidade e confiabilidade de dados).

Segundo Pereira ; Carvalho (2017), dentre os principais problemas e desafios encontrados atualmente, podemos citar:

- **Segurança:** em um mundo cada vez mais conectado, onde “coisas” se conectam com “coisas” e pessoas, a segurança é uma questão cada vez mais complexa de se conseguir e assim é uma das principais barreiras que impedem a efetiva adoção da IoT. Devido a imensa quantidade de dispositivos, as ameaças também estão amplamente distribuídas, assim é necessário a implementação de mecanismos de segurança para evitar ataques massivos.

- **Grandes volumes de dados:** com perspectivas de bilhões de dispositivos conectados, esses irão provocar um grande fluxo de dados que deverão ser tratados. Um dos desafios é analisar elevadas quantidades de dados provenientes de diferentes fontes em vários formatos e realizá-lo em tempo real.
- **Interoperabilidade:** como IoT ainda é muito recente, não existe uma padronização de protocolos e arquitetura, dessa forma são utilizados vários padrões diferentes baseados em elementos proprietários e abertos. Assim, como cada um utiliza protocolos, tecnologias de comunicação e padrões diferentes, isso pode gerar a ausência de interoperabilidade e tornar mais difícil o avanço do IoT.
- **Privacidade:** muitos dos dispositivos de IoT vão gerar dados específicos para indivíduos e suas atividades. Estes dados podem variar desde informações de saúde até padrões de compras, assim possuindo um grande valor econômico para empresas. Muito se é discutido de quem será o dono desses dados e como poderão controlar o compartilhamento dos mesmos.
- **Escala:** uma vez que blocos de endereçamento IPv4 estão escassos, foi criado o novo padrão de endereçamento IPv6, para cobrir esse déficit. Apesar disso, grande parte das organizações vai na direção oposta e continuam utilizando o padrão IPv4, o que impacta na difusão do IoT.

2.2.4 Aplicação

Internet da coisas pode ser utilizada em uma diversidade enorme de aplicações, os principais projeto estão nas áreas de saúde, gestão, cidades inteligentes, automação e segurança residencial, no setor automobilístico e de automação industrial.

Este projeto vai mostra a aplicação de IoT na criação de uma estação Geiger, tentando demonstrar a viabilidade do uso de vários dispositivos de medição do nível de radiação. A idéia também é mostrar que os desafios da área podem ser superados, como questões de segurança e gestão de grande quantidade de dados e dispositivos.

2.3 Tecnologia LoRa

LoRa é uma tecnologia de transmissão de dados sem fio que utiliza a técnica de modulação proprietária da Semtech™. LoRa oferece soluções para problemas reais em comunicação sem fio na forma de transmissões de longo alcance e baixo consumo de energia em sistemas de borda. Ela pode operar sob a rede LoRaWAN, que constitui e define o protocolo MAC para a rede.

LoRa faz parte de uma especificação chamada Low-Power Wide-Area Network (LPWAN), que define os tipos de redes de telecomunicação wireless que cobrem uma

área grande com baixo consumo de energia (ALLIANCE, 2015). A Figura 2 mostra um comparativo entre os tipos de tecnologias disponíveis em relação a cobertura do sinal e consumo de energia.

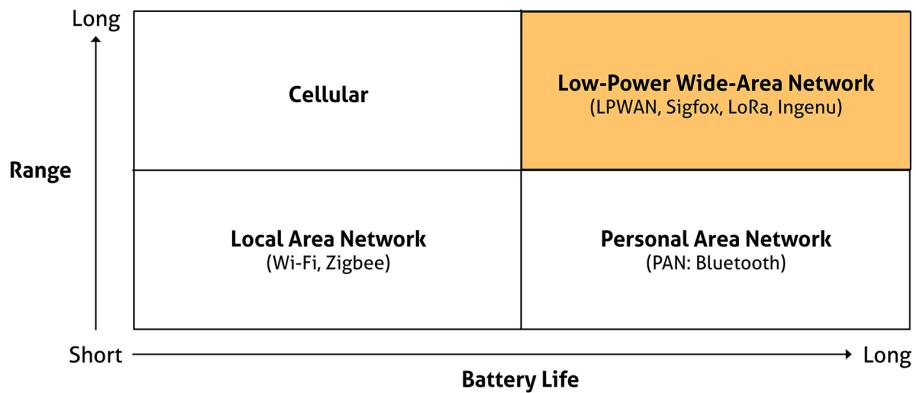


Figura 2 – Gráfico do tempo de vida útil da bateria e alcance.

Fonte: Beta Solutions

Nas subseções seguintes serão explicados mais detalhadamente a respeito da camada física e do protocolo LoRaWAN.

2.3.1 Camada Física

LoRa usa a tecnologia de modulação CSS (Chirp Spread Spectrum), desenvolvida originalmente para aplicações de radar e utilizada em aplicações militares. Os sinais Chirp (Compressed High Intensity Radar Pulse) possuem amplitude constante e varrem toda a largura de banda, variando a frequência de maneira linear em um determinado espaço de tempo. (MARQUES; BOCHIE, 2018)

Vários parâmetros estão disponíveis para a customização da modulação LoRa, entre eles a Largura de Banda (BW), o Fator de Espalhamento (SF) e Taxa de Código (CR). Esses parâmetros influenciam a taxa de bits efetiva da modulação, sua resistência ao ruído e sua facilidade de decodificação.

Dentre as vantagens do CSS podemos citar: baixos requisitos de potência de transmissão; robustez inerente a degradações do canal; resistência contra multipercorso; resistência contra efeito Doppler. (MARQUES; BOCHIE, 2018)

Algumas relações importantes podem ser citadas, como a relação entre a taxa de bits de dados desejada para a modulação LoRa, que pode ser expressa através da seguinte equação¹:

$$Rb = SF * \frac{1}{\frac{2^SF}{BW}} \quad (2.1)$$

¹ Disponível em: <<http://www.semtech.com/images/datasheet/an1200.22.pdf>>.

Onde:

Rb = Taxa de bits

SF = Fator de Espalhamento (7..12)

BW = Largura de Banda (Hz)

A modulação do LoRa também inclui um código correção de erros de tamanho variável, melhorando a robustez do sinal transmitido em detrimento da redundância e fornecendo recursos de recuperação contra a corrupção de bits. Isso é implementado por meio de diferentes taxas de codificação (CRs).

Desta forma definimos a taxa nominal de bits Rb como:

$$Rb = SF * \frac{\frac{4}{2^{SF}}}{BW} \quad (2.2)$$

Tais equações mostram que variando o fator de espalhamento (SF), a largura de banda (BW) e a taxa de codificação (CR), sendo CR uma grandeza adimensional, assim podemos optar por aumentar a taxa de bits ao custo de alcançar uma menor distância de transmissão ou diminuir a taxa de transmissão e transferir pacotes a uma distância maior.

2.3.2 Protocolo LoRaWAN

Com o crescimento da tecnologia LoRa, tornou-se interessante criar uma padronização para otimizar o uso da tecnologia. Por conta disso foi criado o LoRa Alliance, uma associação sem fins lucrativos cuja missão é padronizar a tecnologia LoRa. LoRa Alliance é patrocinado por empresas como IBM, Cisco, Orange, Flashnet, entre outras. (COSSINI, 2016)

O LoRa Alliance tenta padronizar a tecnologia LoRa por meio da especificação LoRaWAN. Este, como um protocolo padrão de comunicação, cobre requisitos fundamentais de IoT, tais como comunicação bidirecional segura, mobilidade, qualidade de serviço, ajustes de potência visando maximizar a duração da bateria dos módulos e serviços de localização. Com esse padrão, há a diminuição da complexidade de implementação de uma rede e das soluções executadas sobre ela.

2.3.2.1 Topologia

O LoRa, especificamente, é baseado em uma rede de topologia estrela, similar a uma rede de telefonia sem fio. Cada módulo LoRa envia e recebe dados a partir de Gateways (receptores de sinais enviados pelos módulos), que repassam os dados via conexão IP para os servidores adequados. (LORA ALLIANCE, 2017)

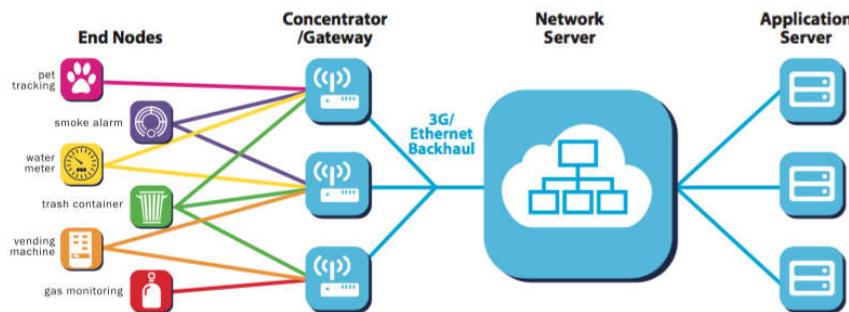


Figura 3 – Arquitetura de uma rede LoRaWAN

Fonte: LoRa Alliance

Na Figura 3, cada nó é um dispositivo dotado de um módulo LoRa que transmite e recebe sinais dos gateways. Estes, por sua vez, recebem as informações dos dispositivos e transmitem para um servidor local ou remoto.

A Figura 4 mostra o modelo da rede, explicitando até onde vai a camada física e onde começa o protocolo LoRaWAN e suas características.

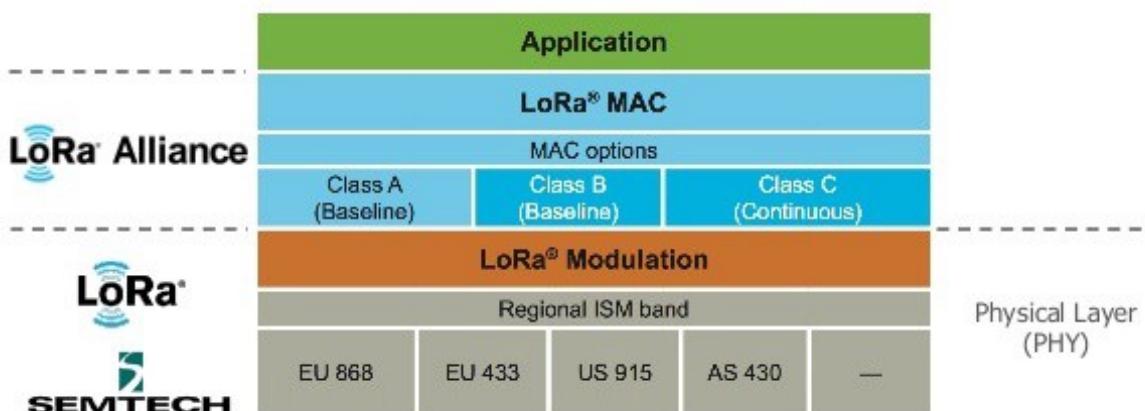


Figura 4 – Topologia básica

Fonte: IOP Institute of Physics

2.3.2.2 Classes de Dispositivos

Existem diversas aplicações diferentes para os end-devices LoRa. Por conta disso LoRaWAN foi feito a suportar 3 tipos de classes diferentes de end-devices:

- **Classe A:** end-devices da Classe A fornecem comunicação bidirecional. Para que isso seja possível cada transmissão de um end-device dessa classe é seguido de uma janela para recebimento de downlink curto. O momento que o end-device irá transmitir é configurado em cada end-device específico e é baseado em sua aplicação.
- **Classe B:** os dispositivos Classe B fornecem a funcionalidade Classe A e, além disso, eles abrem janelas de recebimento (downlink) em horários programados. Para obter

a sincronização necessária da rede, o end-device recebe um Beacon sincronizado no horário do gateway. Isso permite que o servidor saiba quando o dispositivo está escutando.

- **Classe C:** os dispositivos da Classe C possuem janelas de recepção que ficam abertas continuamente, fechadas apenas durante a transmissão.

2.3.2.3 Segurança

Segurança também é outra característica abordada pelo protocolo LoRaWAN. O protocolo define uma criptografia AES128 integrada de ponta a ponta.

Isso significa que cada end-device possui uma Network Session Key (NwkSKey) que é único por dispositivo compartilhado entre end-device e Network Server e possui criptografia AES 128bits. Além disso cada end-device também possui um Application Session Key (AppSKey) que, semelhante ao Network Session Key, é único por dispositivo compartilhado entre end-device e Application Server e possui criptografia AES 128bits. (LORA ALLIANCE, 2017)

A Figura 5 mostra de maneira mais detalhada como ocorre a comunicação entre sensor, gateway e network server, explicitando as diversas camadas.

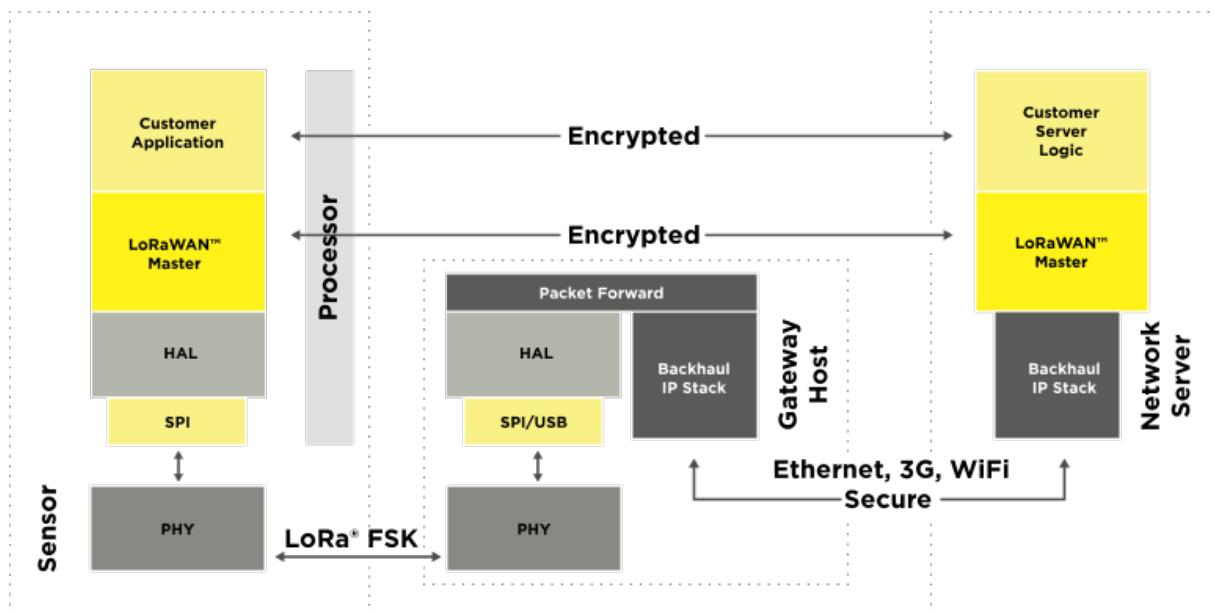


Figura 5 – Diagrama de camadas LoRaWAN
Fonte: LoRa Alliance

Observe ainda na Figura 5 que a criptografia está presente de ponta a ponta.

2.3.3 Regulamentação

Em dezembro de 2017 a ANATEL publicou o ato número 14.448, regulamentando a tecnologia LoRa no Brasil, definindo o plano de frequência para América Latina como

sendo o padrão australiano (ANATEL, 2017). O padrão australiano compreende o espectro de 915MHz a 928MHz.

A associação LoRa Alliance tem o objetivo de promover a adoção global do padrão LoRaWAN, garantindo a interoperabilidade de todos os produtos e tecnologias LoRaWAN. Para isso ela define uma série de padrões a serem utilizados. Nos tópicos a seguir serão mostradas algumas características que devem ser seguidas para o uso da tecnologia.

2.3.3.1 Frequência dos Canais

A banda australiana (AU915-928) deve ser dividida da seguinte forma:

- Upstream: 64 canais numerados de 0 a 63 utilizando LoRa 125 kHz BW variando de DR0 até DR5, usando taxa de codificação 4/5, começando em 915.2MHz e incrementando linearmente em 200 kHz até 927.8MHz;
- Upstream: 8 canais numerados de 64 a 71 usando o BW LoRa 500 kHz no DR6, começando em 915.9MHz e incrementando linearmente em 1.6 MHz a 927.1MHz;
- Downstream: 8 canais numerados de 0 a 7 utilizando o LoRa 500 kHz BW no DR8 para DR13, começando em 923,3 MHz e aumentando linearmente em 600 kHz até 927,5 MHz.

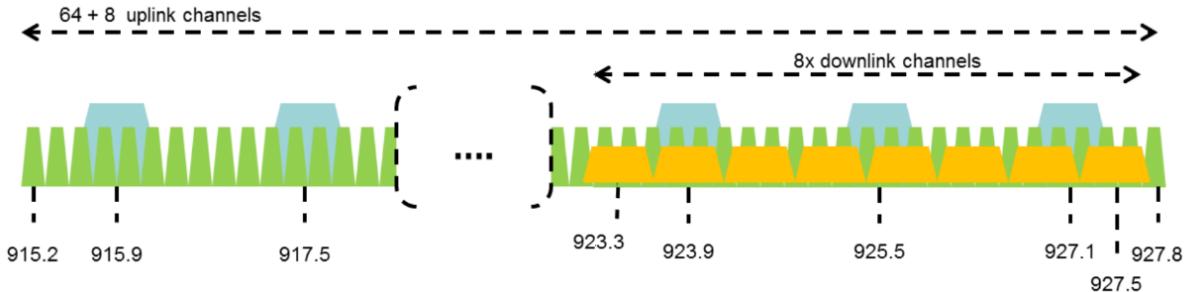


Figura 6 – Frequência dos canais - AU915-928

Fonte: LoRa Alliance

2.3.3.2 Taxa de Dados

Como mostrado na Seção 2.3.1, dependendo dos parâmetros configurados, pode-se variar a taxa de dados para atender melhor os requisitos de sua aplicação. A Tabela 2 mostra as configurações de parâmetros e taxa de dados alcançadas.

A Tabela 3 mostra o tamanho máximo de *payload* que pode ser utilizado por configuração de DR. No projeto da EstGeiger os parâmetros adotados foram: DR=4, SF=8 e BW=125kHz. Tais parâmetros foram escolhidos para atender especificações de tempo no ar (de 400ms definido pela ANATEL) e também as restrições da TTN (*The Things*

Tabela 2 – Taxa de dados para o padrão LoRaWAN AU915-928 (ALLIANCE, 2015)

Taxa de dados (DR)	Configuração	Taxa de bits [bit/seg]
0	LoRa: SF12/125kHz	250
1	LoRa: SF11/125kHz	440
2	LoRa: SF10/125kHz	980
3	LoRa: SF9/125kHz	1760
4	LoRa: SF8/125kHz	3125
5	LoRa: SF7/125kHz	5470
6	LoRa: SF8/500kHz	12500
7	LoRa: RFU	
8	LoRa: SF12/500kHz	980
9	LoRa: SF11/500kHz	1760
10	LoRa: SF10/500kHz	3900
11	LoRa: SF9/500kHz	7000
12	LoRa: SF8/500kHz	12500
13	LoRa: SF7/500kHz	21900
14	LoRa: RFU	
15	LoRa: Defined in LoRaWAN	

Network) de no máximo 30 segundos de tempo no ar por dia. Dessa forma, o *payload* disponível seria de 133 Bytes, porém nesse valor não está incluso os bytes necessários para operar no protocolo LoRaWAN, assim sendo o *payload* útil final de 120 Bytes.

Atualmente a EstGeiger usa apenas 29 bytes de payload, tendo a possibilidade de enviar mais dados futuramente.

Tabela 3 – Tamanho máximo de *payload* por DR (ALLIANCE, 2015)

Taxa de dados (DR)	Payload (bytes)
0	N/A
1	N/A
2	19
3	61
4	133
5	250
6	250
7	Não definido
8	41
9	117
10	230
11	230
12	230
13	230
14	Não definido
15	Não definido

3 Trabalhos Relacionados

Neste Capítulo serão mostrados trabalhos com ideias semelhantes ao que foi feito na EstGeiger e que serviram de inspiração, como produtos disponíveis no mercado e outros projetos acadêmicos.

3.1 Produtos de Mercado

3.1.1 Safecast bGeigie Nano - Mobile Radiation Monitoring Device

O dispositivo bGeigie Nano, desenvolvido pela Safecast, tem o objetivo de medir radiação ionizante, dos tipos alfa, beta e gama. A ideia do produto é ser um sensor móvel para ser instalado na parte exterior de veículos, por isso conta com GPS para geolocalização.

O equipamento possui um display que fornece informações do nível de radiação. Também pode ser feito datalog (em um cartão de memória MicroSD) com o nível de radiação medido, data e hora da mediação e geolocalização (SAFECAST, 2018).



Figura 7 – Dispositivo Safecast bGeigie Nano
Fonte: Safecast

O bGeigie Nano possuí uma versão com bluetooth e disponibiliza um aplicativo para iOS e Android para comunicação com o dispositivo, podendo visualizar os dados diretamente no smartphone.

A Safecast também disponibiliza uma API para que usuários possam postar os dados medidos e fazer a integração com outras aplicações. O upload dos dados para a API pode ser feita através do próprio aplicativo mobile ou através da coleta dos dados no

cartão MicroSD e transferidos via algum computador com acesso a internet.

Diferente da EstGeiger, abordado neste projeto, o bGeigie Nano é um equipamento com a finalidade de medição presencial e móvel.

3.1.2 uRADMonitor

Os dispositivos desenvolvidos pela uRADMonitor possuem a característica de monitorar níveis de radiação e também a qualidade do ar, disponibilizando os dados gerados online.



Figura 8 – Linha de dispositivos uRADMonitor

Fonte: uRADMonitor

Atualmente a empresa conta com cinco dispositivos, mostrados na Figura 8, da esquerda para direita os modelos são: model A, model KIT1, model D, model A3 e model INDUSTRIAL. Todos os modelos possuem conexão com a internet para enviar os dados coletados, o que pode mudar é a tecnologia de transmissão utilizada, podendo ser Ethernet, WiFi, GSM ou até mesmo LoRaWAN (URADMONITOR, 2018).

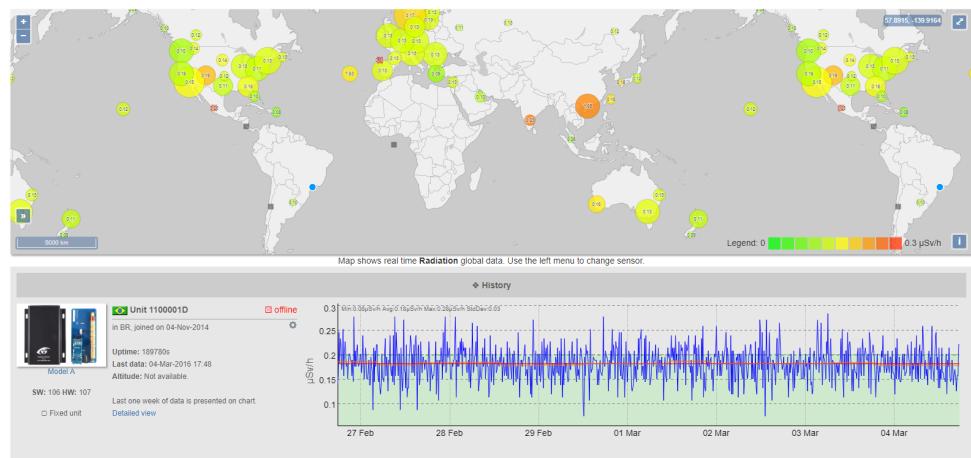


Figura 9 – Dashboard de monitoramento dos dispositivos uRADMonitor

Fonte: uRADMonitor

Os dados são transmitidos para o site da empresa, que monitora centenas de dispositivos no mundo inteiro. Além disso também é oferecida uma API para integração dos dados com outras aplicações.

3.2 EstAcqua

A EstAcqua é um projeto de mestrado defendido por Alan A. Helal pela UFES, que possui uma proposta próxima ao projeto desde trabalho. Tem como seu principal objetivo a implementação de uma solução de baixo custo que integra hardware e software para monitoramento ambiental (HELAL, 2018).

A EstAcqua possui sensores de superfície que coletam dados de pressão atmosférica, umidade, temperatura e iluminância, além de sensores submersos para coletar temperaturas em diferentes profundidades do lago.



Figura 10 – Versão final da EstAcqua em funcionamento na balsa no Lago Terra Alta

Fonte: (HELAL, 2018)

O projeto da EstAcqua foi utilizado em vários aspectos como base para o desenvolvimento da EstGeiger. Dentre suas características podemos citar: uso de tecnologia LoRa para transmissão, utilização de microcontrolador MicroPython, utilização de sensores de baixo custo para monitoramento ambiental e uso de plataformas IoT na nuvem para envio e apresentação dos dados coletados.

Tais características fazem do projeto uma excelente referência para o desenvolvimento da EstGeiger, que visa muitos objetivos em comum.

4 Arquitetura e Implementação

Neste Capítulo será mostrado como foi o desenvolvimento do projeto, desde o hardware escolhido e software embarcado até a construção física e arquitetura.

4.1 Hardware Utilizado

4.1.1 LoPy 4

O LoPy 4 é um microcontrolador desenvolvido pela Pycom. Ele possui MicroPython nativo e possui quatro opções de conexão sem fio: LoRa, WiFi, Bluetooth e Sigfox. A Figura 11 mostra uma imagem do LoPy 4, destacando algumas características.

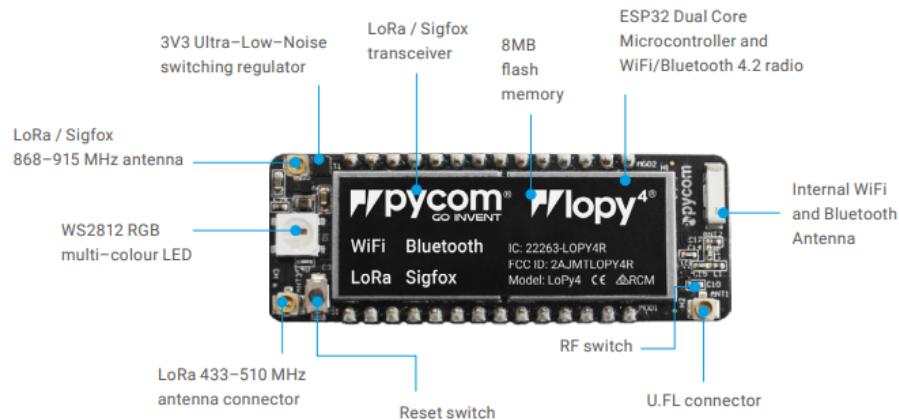


Figura 11 – LoPy 4

Fonte: Pycom

Dentre as principais características também podemos citar seu processador, um chipset ESP32 da Espressif, o baixo consumo de energia, vários tipos de interface disponíveis (UART, SPI, I2C, I2S e ADCs). Além de ser compacto, medindo 55 mm x 20 mm, ele também conta com 4 MB de memória RAM (*Randon Access Memory*) com capacidade multi-threading, hardware de ponto flutuante e memória interna flash de 8 MB (PYCOM, 2018b).

Em relação a tecnologia LoRa, o LoPy 4 pode operar nas bandas: 868 MHz (Europa), 915 MHz (Américas, Austrália e Nova Zelândia), 433 MHz(Europa) e 470-510 MHz (China).

Por conta de todas essas características o LoPy 4 se torna uma excelente ferramenta de desenvolvimento e é nele que roda o código da EstGeiger, desenvolvido em MicroPython.

4.1.2 Expansion Board

A placa de expansão da Pycom, representada na Figura 12, tem como objetivo facilitar o interfaceamento com o LoPy 4 (ou outros microcontroladores da Pycom, como WiPy, GPy, FiPy e SiPy) (PYCOM, 2018d).

Neste projeto é utilizada a versão 2.0. Ela possui uma entrada Micro USB, para alimentação e comunicação serial, também para alimentação há um conector para baterias de polímero de lítio (LiPo) e íon de lítio (Li-Ion) (PYCOM, 2018a).

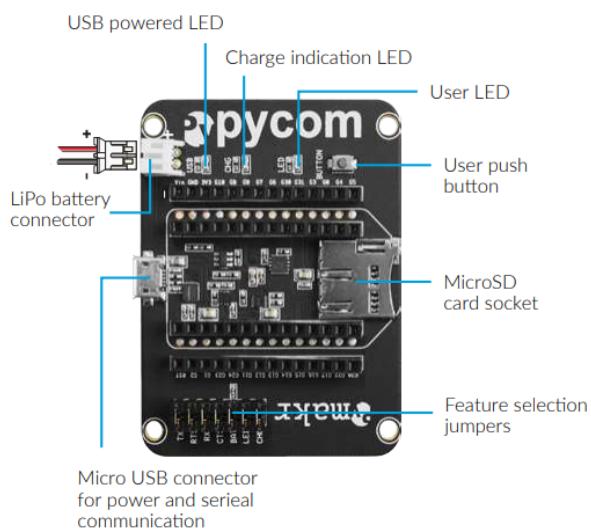


Figura 12 – Expansion Board V2.0

Fonte: Pycom

Dentre as características importantes para o projeto podemos citar ainda a existência de um slot para cartão MicroSD (até 32GB) e conectores fêmeas que extenderam os pinos do microcontrolador utilizado, para facilitar a prototipagem de soluções com sensores.

4.1.3 Sensor de Radiação

A placa Geiger (RHELETRONICS, 2018) original motivou clones, é um deles que usamos, da marca CAJOE, modelo NGMC-V1. Ela é compatível com alguns tubos Geiger (ou Geiger-Müller) do mercado, como o M4011, STS-5, SBM20 ou J305.

Podemos dizer que o tubo Geiger-Müller é a alma do sensor, ele consiste em uma câmara metálica cilíndrica em cujo eixo é estendido um fino fio metálico, é preenchido por um gás a baixa pressão. Uma tensão elétrica da ordem de centenas de volts é estabelecida entre o cilindro (que tem papel de cátodo) e o fio (ânodo).

Quando uma radiação ionizante penetra no contador, o gás é ionizado, ou seja, faz com que elétrons sejam liberados. Esses elétrons se multiplicam rapidamente pelo que é chamado de avalanche eletrônica, tornando o gás condutor durante um curto tempo

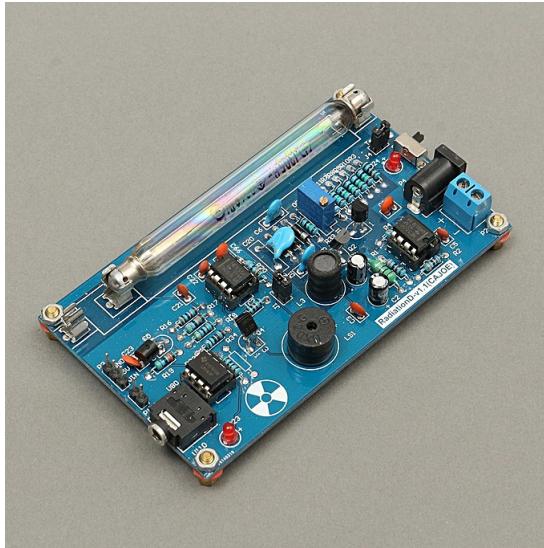


Figura 13 – Sensor de radiação CAJOE - modelo NGMC-V1
Fonte: AliExpress

(fenômeno de descarga elétrica). Após amplificação, o sinal elétrico assim produzido pode ser transformado em uma indicação visual ou sonora (TECINMED, 2014). A Figura 14 demonstra esse processo.

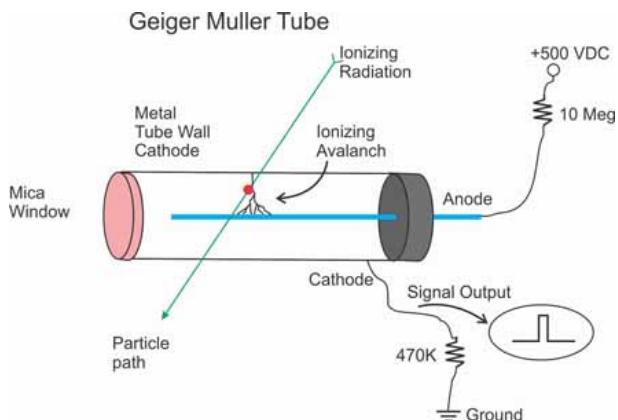


Figura 14 – Esquema de funcionamento de um tubo Geiger-Müller
Fonte: Images Scientifics Instruments

Este sensor é uma boa solução de baixo custo e oferece compatibilidade com microcontroladores disponíveis no mercado, como é o caso do LoPy 4, utilizado neste projeto. Para conexão entre as placas o sensor de radiação disponibiliza três pinos: dois de alimentação (tensão e terra) e um pino (chamado *Vin*) que é a saída do sinal gerado ao ser detectada alguma radiação.

O pino *Vin* mantém uma tensão de aproximadamente 3,3V e ao detectar radiação a placa gera um sinal negativo, ou seja, a tensão cai (ver Figura 15). Cada detecção gera um sinal com duração de aproximadamente $200\mu\text{s}$.

Como a unidade de medida de radiação escolhida para EstGeiger é CPM (Counts

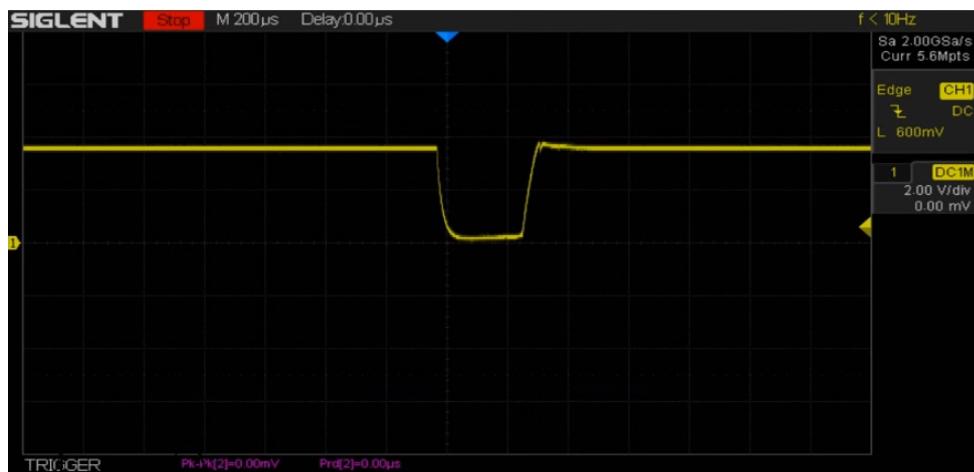


Figura 15 – Saída do sensor medida no osciloscópio ao detectar radiação

Per Minute), basta contarmos quantos sinais desse tipo ocorreram em um minuto que teremos o valor medido.

4.1.4 BME280

O BME280 é um sensor capaz de medir temperatura, umidade e pressão com uma boa precisão. É um sensor relativamente pequeno (2,5 mm x 2,5 mm x 0,93 mm) e que tem um baixo consumo de energia (BOSCH, 2018). Pode ser alimentado com tensão entre 1,7 V e 3,6 V e a comunicação pode ser feita através da interface I2C (*Inter-Integrated Circuit*) ou SPI (*Serial Peripheral Interface*).

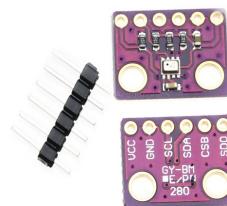


Figura 16 – Sensor de umidade, pressão atmosférica e temperatura, BME280
Fonte: Arduintronics

No projeto da EstGeiger ele é utilizado para monitorar a saúde do dispositivo e poder prever possíveis problemas, como exposição a altas temperaturas ou nível de umidade muito alto. É utilizado um driver desenvolvido por Colistete (2018b), o qual possui uma precisão maior.

4.2 Plataformas IoT

4.2.1 The Things Network

The Things Network é o servidor na nuvem utilizado neste projeto. É um dos servidores gratuitos para LoRaWAN mais populares, com uma comunidade de mais de 50 mil membros, mais de 5 mil gateways de usuários conectados à rede e mais de 26 mil aplicações em funcionamento distribuídas em 137 países (THE THINGS NETWORK, 2018).

The screenshot shows the TTN Console interface. At the top, there's a navigation bar with links for Applications, Gateways, Support, and a user account. Below the header, the application 'estacao_geiger' is selected. The main area is divided into several sections:

- APPLICATION OVERVIEW:** Displays basic information about the application, including Application ID (estacao_geiger), Description (Sensores de monitoramento de radiação), Created (2 months ago), and Handler (ttn-handler-eu). It also includes a documentation link.
- APPLICATION EUIS:** Shows a list of EUIS (EUI-64 identifiers) associated with the application.
- DEVICES:** Shows a list of registered devices, indicating 2 registered devices.
- COLLABORATORS:** Shows a list of collaborators, including 'estgeiger'. It includes buttons for managing collaborators, devices, and settings.

Figura 17 – Página inicial de uma aplicação na TTN

É um servidor bastante completo e com muitas características. Dentre as principais e mais utilizadas neste projeto, podemos citar:

- **Gerenciamento de dispositivos:** é possível criar aplicações e adicionar dispositivos nós, assim como excluí-los e ver os dados enviados por eles em tempo real. Além disso também é possível gerenciar os gateways adicionados.
- **Integrações com outras plataformas:** uma característica importante é a integração com outras plataformas, a TTN tem algumas muito interessantes, como é o caso do Cayenne e do Ubidots (CAYENNE, 2018) (UBIDOTS, 2018), utilizados neste projeto. A Figura 18 mostra as opções de integração da TTN.
- **Customização dos dados:** customização no sentido de ser possível fazer um pré tratamento dos dados recebidos, como por exemplo um decoder de payload para

enviar o dado já no formato ideal para outras plataformas. Um exemplo disso foi usado neste projeto, onde para encaminhar o dado para plataforma Ubidots foi necessário fazer um decoder, o código está disponível no Apêndice A.

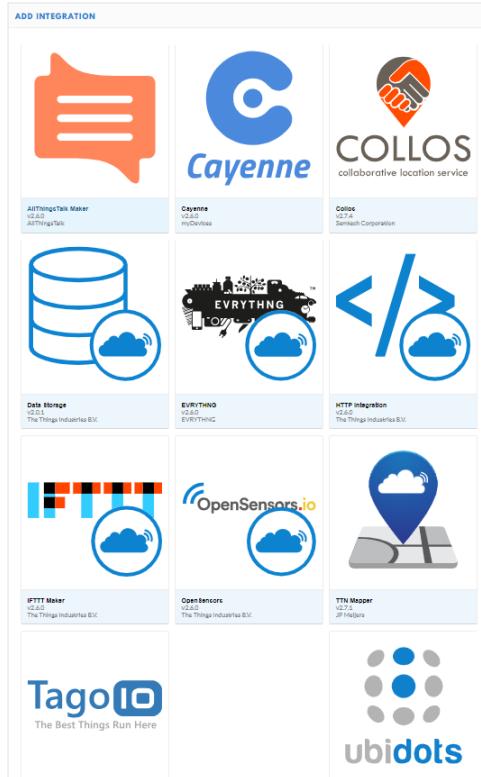


Figura 18 – Integrações disponíveis na TTN

4.2.2 Cayenne (CAYENNE, 2018)

Cayenne é uma solução IoT gratuita, que tem como objetivo acelerar o desenvolvimento de protótipos. Possui uma interface bem simples, facilitando o uso e é compatível com a TTN.

Tem como características várias opções gráficas para mostrar os dados dos sensores, além disso podem ser configurados alguns atuadores para comunicação com os dispositivos cadastrados. Ainda é possível visualizar dados antigos (até 1 ano) e fazer o download em um arquivo CSV. Por conta de todas essas características o Cayenne foi escolhido como a aplicação para exibição dos dados.

4.2.3 Ubidots (UBIDOTS, 2018)

Ubidots é uma plataforma especializada em conectar hardware e software. A idéia é bem próxima do Cayenne, porém implementada de uma maneira diferente, mas basicamente o que você faz é adicionar os sensores, escolher as métricas a serem visualizadas e definir algumas configurações de visualização.

A característica que fez o Ubidots também ser escolhido foi a capacidade de poder incorporar os gráficos gerados em outra página HTML, visto que o Cayenne não possui esse recurso, característica essa que foi solicitada pelos usuários finais da aplicação.

Porém, ele possui algumas restrições para uso de todo seu potencial, como por exemplo, para mais de um dispositivo cadastrado é necessário ter créditos, deixando de ser uma aplicação gratuita. No caso para utilização de até 3 dispositivos o custo seria de US\$5/mês, com 5 meses iniciais gratuitos e até 3 meses de histórico, também há a opção para 10 nós, 2 anos de histórico, por US\$20/mês .

4.3 Implementação de Software

4.3.1 Características da EstGeiger

Todo o código foi desenvolvido no firmware MicroPython da Pycom na versão 1.19.0.b4. O funcionamento da EstGeiger atende as seguintes características:

- **Parâmetros LoRa:** a frequência de transmissão utilizada é 915,2 MHz (dentro do padrão australiano). Fator de espalhamento 8 (SF=8), largura de banda de 125 kHz (BW=125 kHz) e taxa de dados igual a 4 (DR=4). Tais parâmetros foram escolhidos por conta dos obstáculos (falata de linha de visada) e distância entre nós e gateway e do tamanho do payload, que necessita de 14 a 29 Bytes (pois evita envio de dados repetidos de 4 sensores que variam lentamente, temperatura, umidade, pressão e bateria, ou seja, quando não há atualização dos valores) para enviar todas as métricas.
- **Consumo de energia:** o consumo de energia é um quesito muito importante para projetos de IoT. Neste projeto em particular, foram disponibilizadas tomadas no local onde serão instalados os dispositivos. Porém existe a ideia de uma versão futura que seja altamente econômica, não foi implementada neste projeto pois não fazia parte da especificação dos usuários, envolveu a importação de componentes que só chegaram ao final do projeto, e precisaria de longos testes de consumo, inclusive com uso de painel solar opcional. De qualquer forma a versão utilizada possui uma bateria de 2600mAh, para quedas de energia, tal bateria dura até 5 dias de uso na configuração atual.
- **Dados coletados:** além da radiação, que é a métrica principal do projeto, também são medidos dados de bateria, contador de pacote, RSSI, SNR, temperatura, umidade e pressão, com a finalidade de monitorar a saúde das estações e verificar a causa de possíveis falhas. Todos esses dados são levantados utilizando uma biblioteca estatística desenvolvida por Colistete (2018a) e depois enviados para as plataformas IoT na

nuvem. O pacote final de envio é codificado no padrão LPP (Low Power Packet), que é o padrão utilizado pela plataforma Cayenne.

- **Modo FTP:** o modo FTP pode ser habilitado com três finalidades, fazer download de arquivos de log e atualização OTA (Over The Air) do firmware MicroPython e do código MicroPython dos nós. O modo FTP pode ser ativado e desativado via dashboard do Cayenne. Ao ser ativado a estação se transforma em um ponto de acesso WiFi e é possível conectar-se a ela e fazer transferência de arquivos sem uma conexão física.
- **Arquivos de log:** além do próprio payload ser gravado em arquivo de logs no cartão microSD, permitindo armazenamento de dados por alguns anos ininterruptos, também são salvos um arquivo de intervalo entre pulsos das medições e outro arquivo com análise de histograma de tais tempos. O histograma é útil para a equipe de pesquisadores, porém como não são dados críticos que precisam estar disponíveis a todo momento, eles são disponibilizados via FTP, onde os pesquisadores poderão ir ao local de meses em meses e fazer download dos arquivos presencialmente. Como o histograma gera um conjunto de dados relativamente grande, que não são aceitos pelas plataformas IoT usadas, então fica disponível apenas localmente.

4.3.2 Diagrama de Funcionamento

Na Figura 19 é mostrado o modo como o software foi implementado, mostrando todos os passos em forma lógica de como o algoritmo roda.

4.4 Construção

A construção física do sensor foi feita em acrílico transparente com peças desenvolvidas e fabricadas sob medida. Os testes iniciais foram feitos em protoboard e posteriormente utilizamos os sensores conectados diretamente na expansion board. Confira as subseções a seguir para mais detalhes.

4.4.1 Conexão dos Sensores

- **Sensor de radiação**

O sensor de radiação possui três pinos: alimentação (5V), terra e sinal de saída. Os pinos de alimentação e terra são conectados aos respectivos pinos no LoPy 4. O pino de sinal de saída é conectado a um pino GPIO do LoPy 4, o qual é responsável por fazer a leitura do sinal do sensor. Veja a Figura 20.

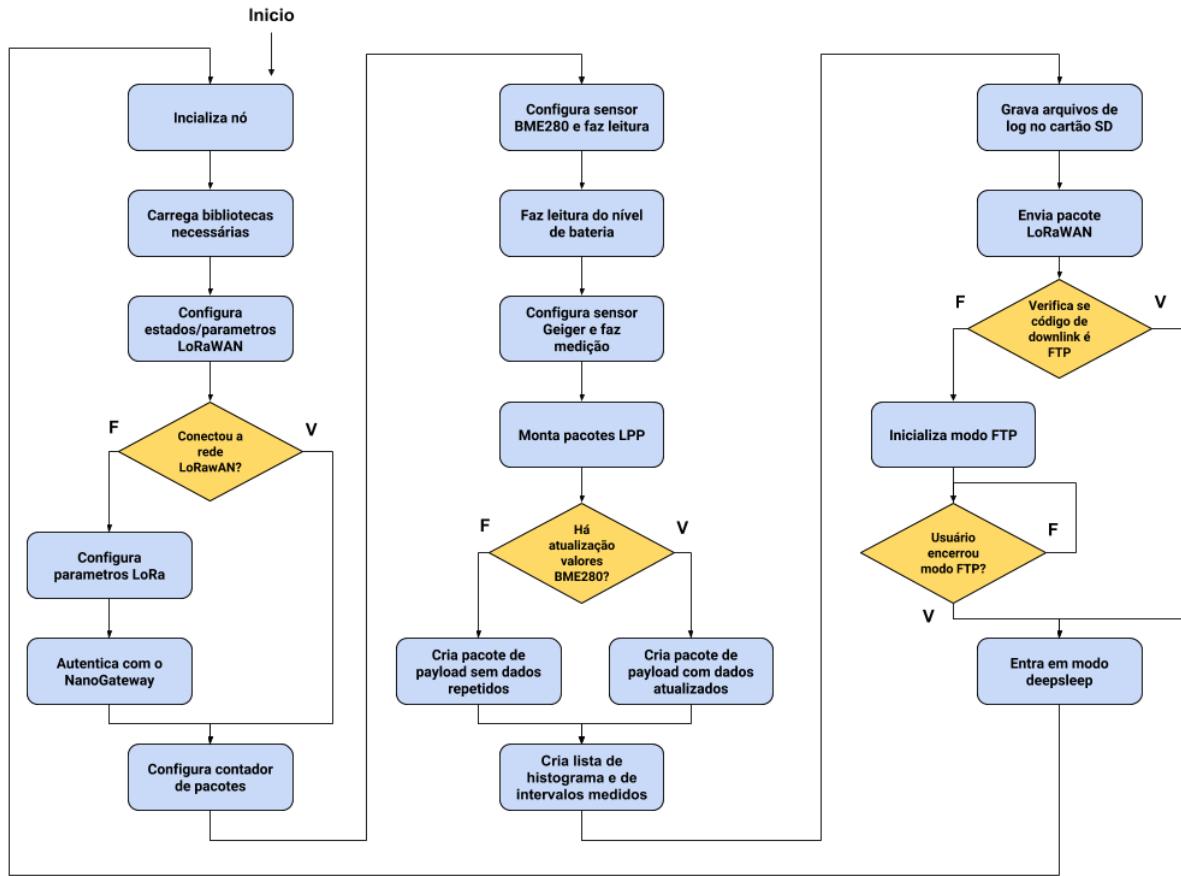


Figura 19 – Diagrama lógico de funcionamento do nó

• Sensor BME280

O sensor de temperatura, umidade e pressão, BME280 possui seis pinos: alimentação (V_{cc}), terra (GND) e comunicação (via pinos SCL, SDA, CSB e SDO). Os pinos de alimentação e terra são conectados aos respectivos pinos no LoPy 4 (lembre que a alimentação aqui é de 3,3V). A comunicação é feita através do barramento I2C, conectando os pinos SCL e SDA do sensor aos respectivos pinos do LoPy 4. Os outros pinos de comunicação (CSB e SDO) não são utilizados. Veja a Figura 21.

4.4.2 Desenvolvimento e Montagem da Estrutura

Como foi mencionado anteriormente, para encapsular as placas foi desenvolvida uma estrutura de acrílico. Para isso, foram medidas as dimensões e diâmetro dos furos de todas as placas para criar o modelo de acrílico.

A estrutura conta com dois "andares", onde no primeiro é parafusado o sensor de radiação e no segundo são parafusados o sensor BME280 e o Expansion Board com o LoPy 4. A bateria também vai presa no segundo andar.

Foram deixados furos extras para passagem dos fios de comunicação entre sensores.

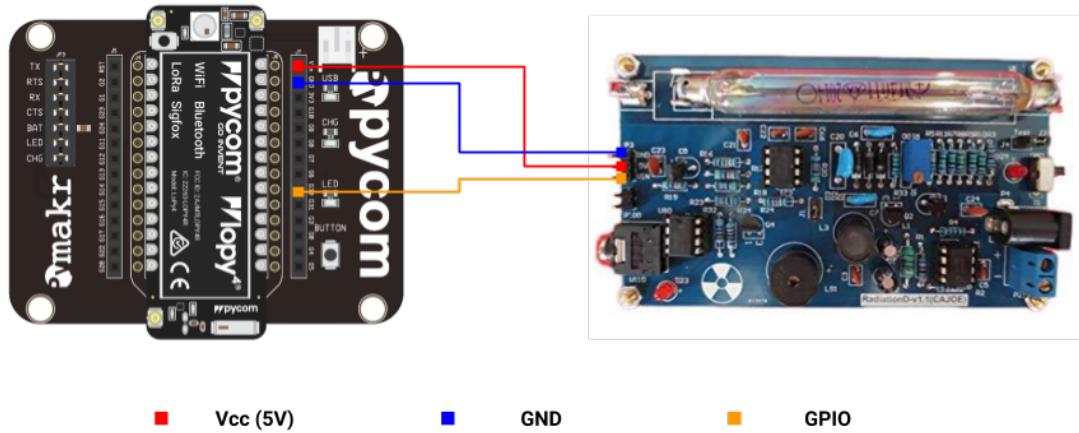


Figura 20 – Conexão entre sensor de radiação e LoPy 4

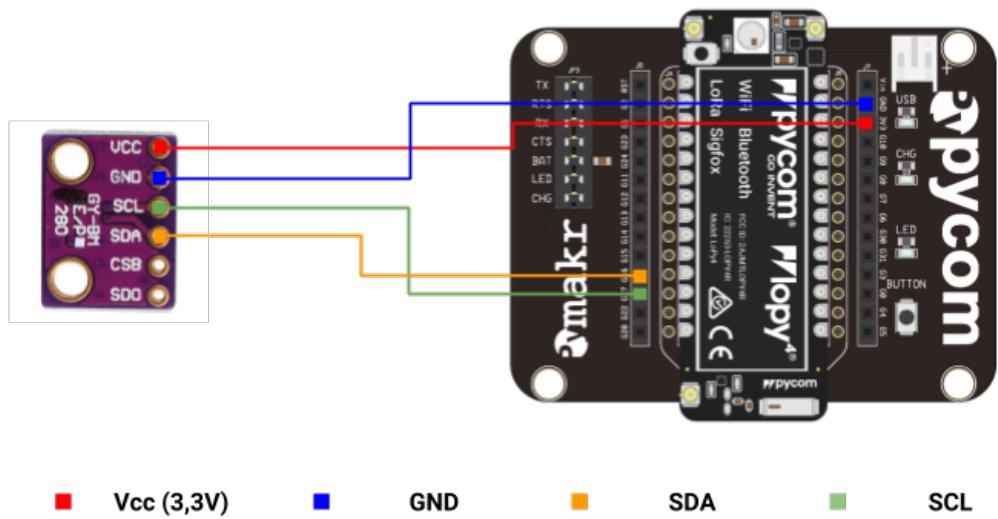


Figura 21 – Conexão entre sensor BME280 e LoPy 4

Veja a Figura 22 o desenvolvimento da estrutura.

A estrutura foi pensada em acrílico transparente para deixar o dispositivo bem visível, porém durante os testes notou-se que a luz externa afeta o sensor de radiação, então foi envelopado de preto a parte da estrutura que o sensor de radiação se encontra, resolvendo o problema.

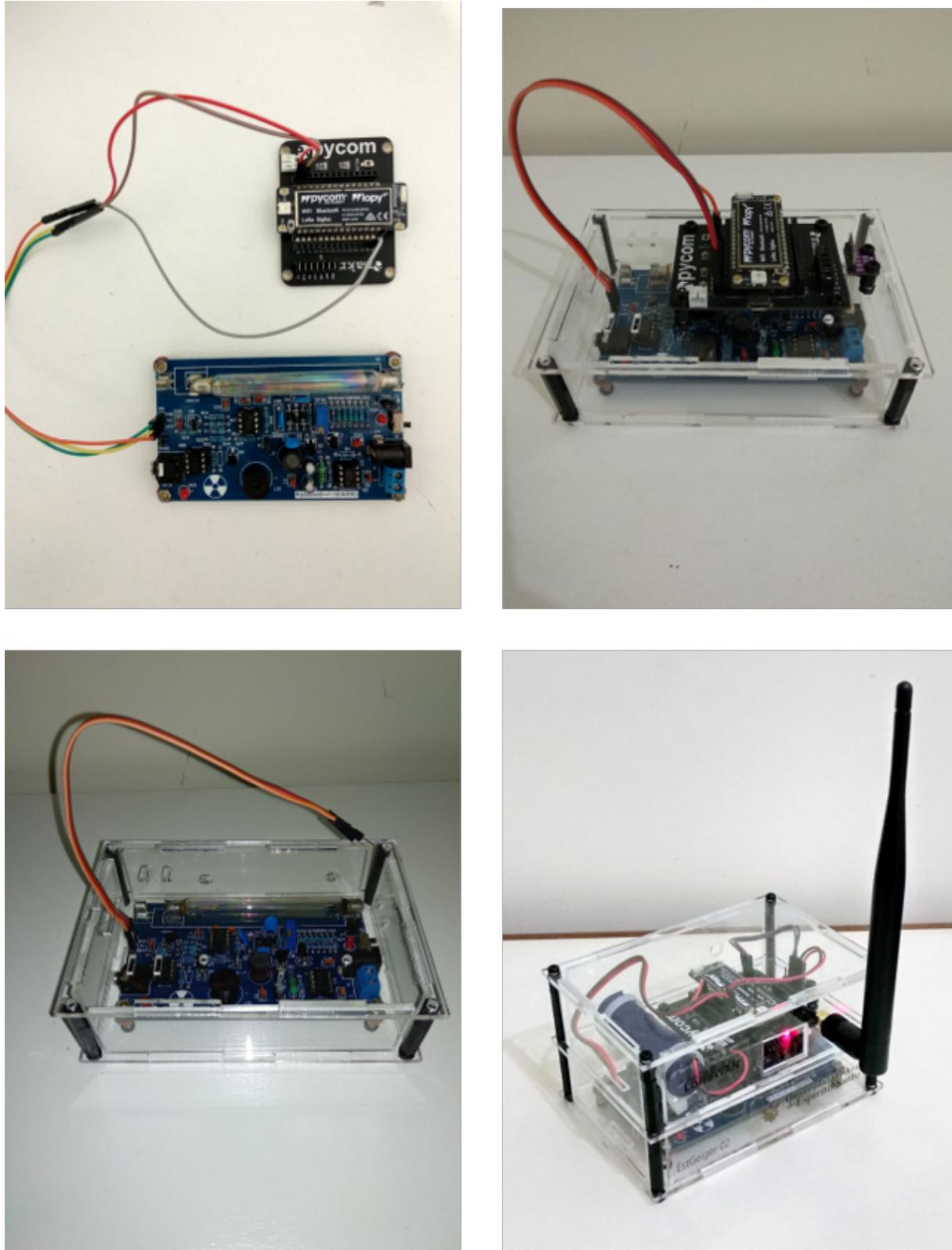


Figura 22 – Criação da estrutura do dispositivo

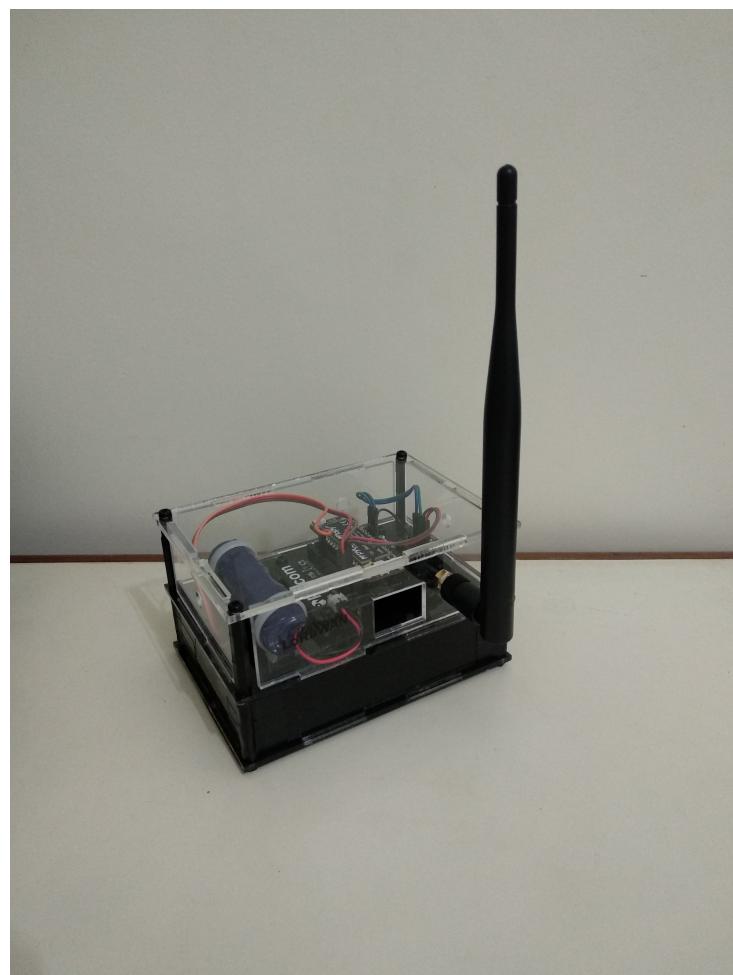


Figura 23 – Versão final do nó EstGeiger

4.4.3 Custo do Projeto

O custo do projeto é mostrado da Tabela 4.

Tabela 4 – Custo do projeto - Nô EstGeiger

Componente	Custo (R\$)	Referência
LoPy 4	R\$153	Pycom
Expansion Board	R\$70	Pycom
Sensor de Radiação	R\$150	AliExpress
Sensor BME280	R\$22	AleExpress
Bateria Li-ion 2600mAh	R\$28	MercadoLivre
Case de acrílico	R\$30	Copy Express
Fonte USB + Cabo	R\$19	MercadoLivre
Total	R\$472	

Tabela 5 – Custo do projeto - NanoGateway

Componente	Custo (R\$)	Referência
LoPy 4	R\$153	Pycom
Expansion Board	R\$70	Pycom
Pycase	R\$35	Pycom
Bateria Li-ion 2600mAh	R\$28	MercadoLivre
Carregador Portátil 10.000 mAh	R\$99	MercadoLivre
Fonte USB + Cabo	R\$19	MercadoLivre
Total	R\$404	

O valor de um nó é de R\$ 472. Lembrando que vários componentes são importados, assim este levantamento de custo foi feito com base na cotação do dólar no dia 29/11/2018 (onde 1,00 dólar estava valendo 3,85 reais).

O preço de um nó da EstGeiger sai aproximadamente 5 vezes mais barato que o sensor de medição utilizado atualmente pelos pesquisadores (Gamma Scout Alert Radiation Detector), que não tem transmissão das medidas sem fio, compensando assim seu investimento.

4.5 Arquitetura do Projeto

O projeto consiste em quatro principais componentes: o nó, que é a EstGeiger propriamente dita, o qual realiza a função de coleta de dados e transmissão LoRa para o gateway, o NanoGateway, que recebe as mensagens dos nós via LoRa e encaminha para o LoRaWAN server (TTN) via internet, a nuvem IoT, na qual podemos citar o network server que encaminha os dados recebidos para as devidas aplicações, as aplicações que geram a visualização do dado e a interage com o usuário que é feita através de dispositivos pessoais que podem acessar as aplicações na Internet.

A Figura 24 mostra de forma mais detalhada os componentes da arquitetura da EstGeiger.

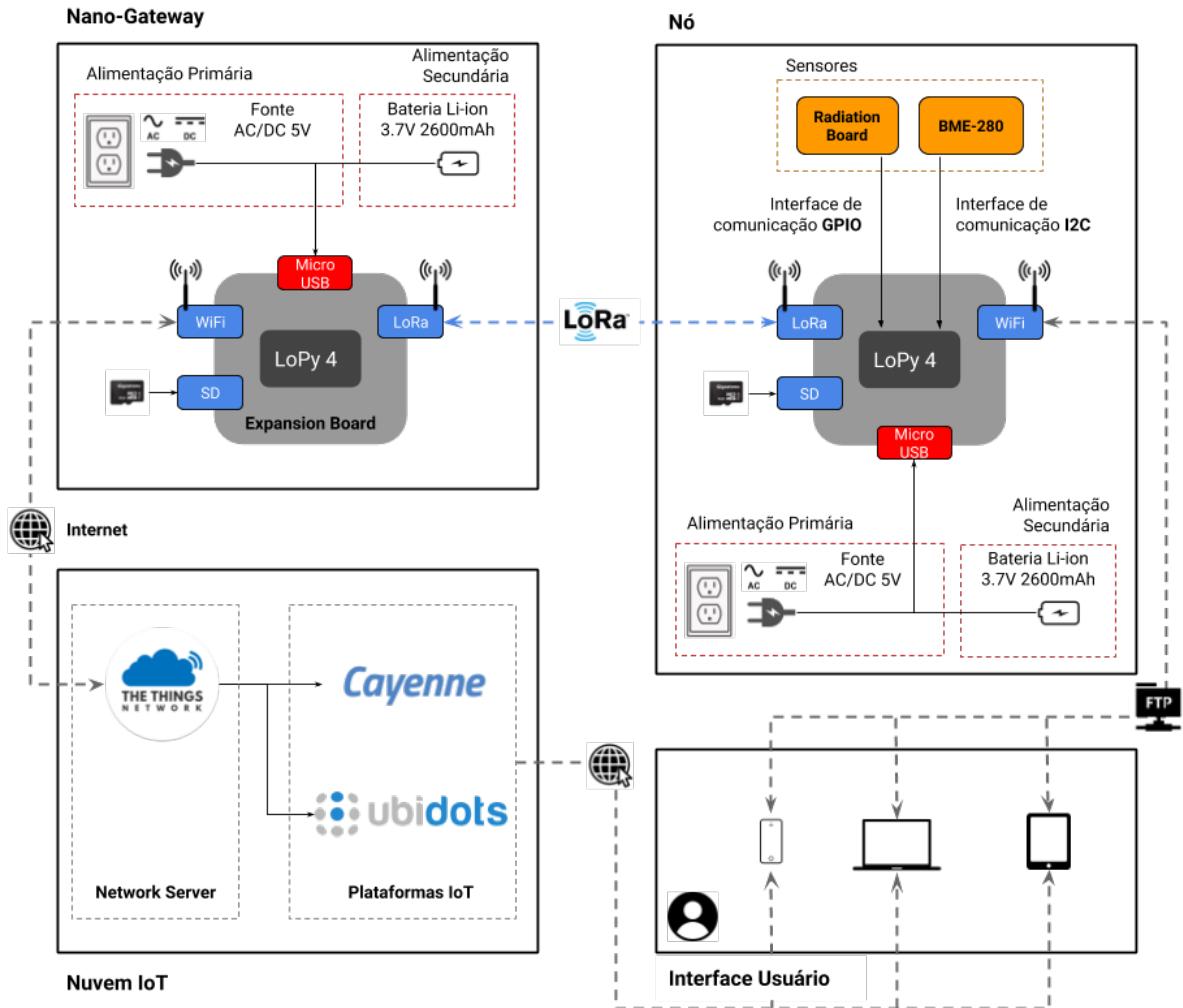


Figura 24 – Arquitetura da EstGeiger

A Figura 25 mostra quem desempenharia o papel dos componentes de rede de uma arquitetura LoRaWAN, seguindo o exemplo mostrado na Figura 3 no Capítulo 2.

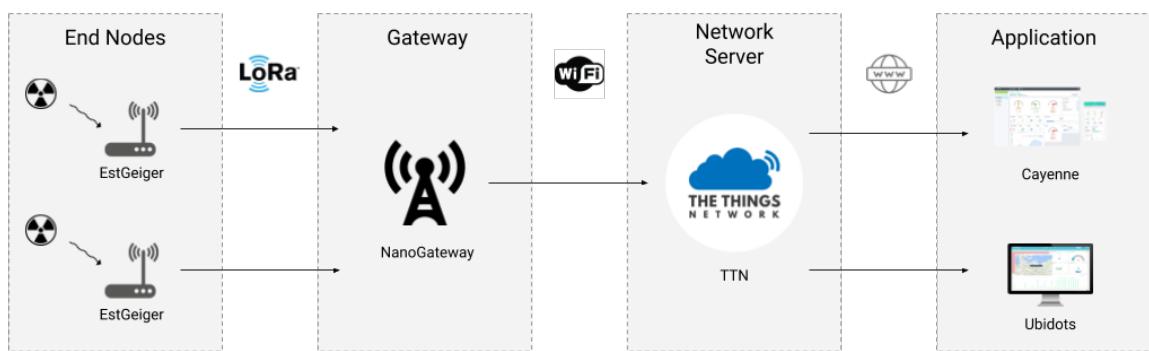


Figura 25 – Arquitetura baseada no protocolo LoRaWAN

5 Avaliação e Resultados

Neste Capítulo serão abordados e avaliados os resultados do testes feitos.

5.1 Distância de Transmissão

Os testes de distância de transmissão foram feitos em regiões com linha de visada entre gateway e nó, e sem linha de visada, em ambientes urbanos.

5.1.1 Ambiente Urbano (sem linha de visada)

Esses testes foram realizados com a intenção de simular dispositivos nós instalados em ambientes urbanos. Foi realizado em Bairro de Fátima, na Serra, ES, um bairro residencial.

Para tentar realizar o teste da forma mais correta possível, o NanoGateway foi posicionado no telhado de uma casa, acima do segundo andar, a cerca de 10 metros de altura. O nó foi sendo levado por dentro do bairro e foi sendo verificado se os pacotes chegavam.

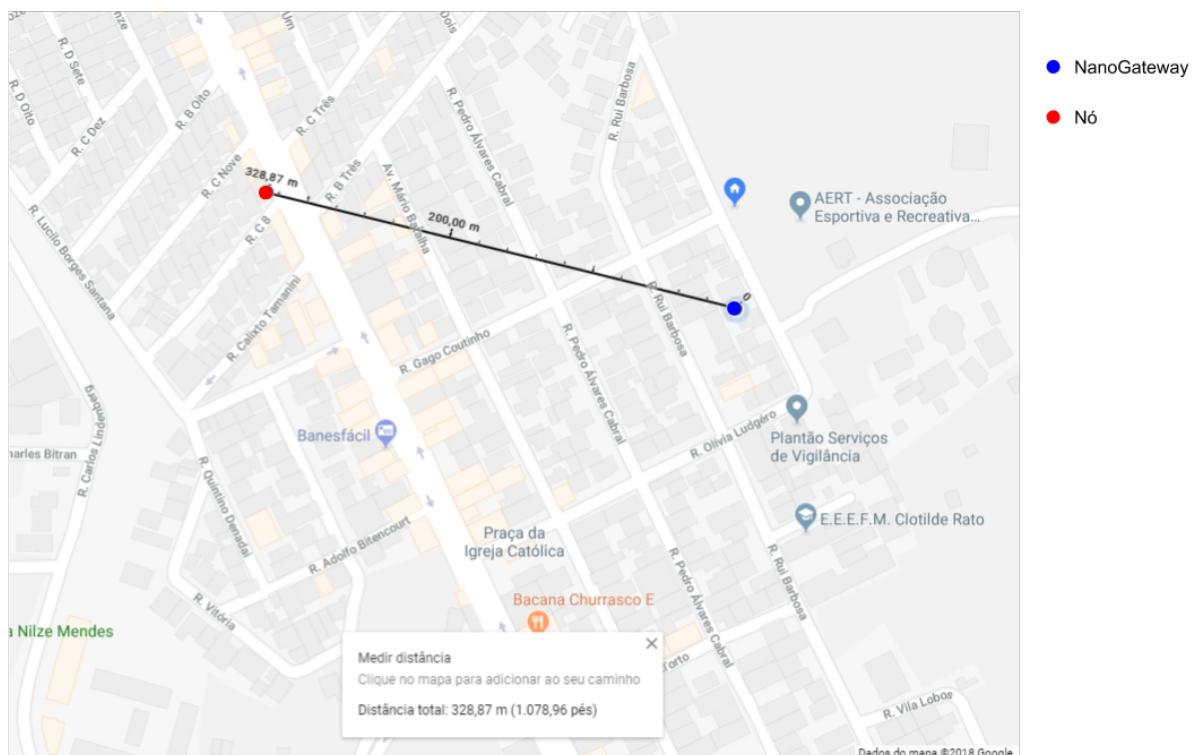


Figura 26 – Distância de transmissão alcançada em ambiente urbano

Fonte: Google Maps

O ponto mais distante no qual ainda era possível receber pacotes foi de 328,87 metros, conforme indica a Figura 26, com um valor de RSSI de -127.

5.1.2 Ambiente Aberto (com linha de visada)

Os testes de distância com linha de visada foram realizados na Praia de Camburi, em Vitória, ES.

O NanoGateway foi posicionado no Pier de Iemanja, no início da praia, enquanto o dispositivo nó era levado para a outra ponta da praia. Ao longo de todo o percurso feito na praia o nó conseguiu enviar os pacotes. Conforme esperado percorremos toda a distância possível da praia, chegando a outra ponta, conseguindo enviar pacotes.

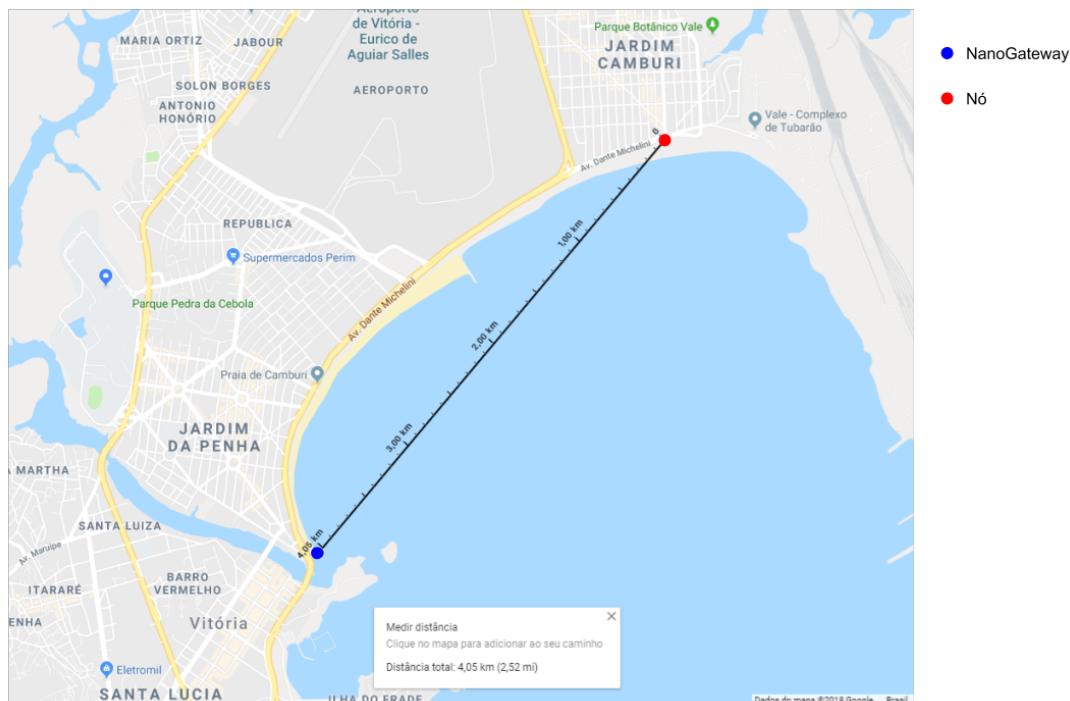


Figura 27 – Distância de transmissão alcançada em ambiente aberto

Fonte: Google Maps

A Figura 27 mostra a distância de transmissão alcançada, cerca de 4,05 km, com um valor de RSSI de -96. Contudo acreditamos que a distância de transmissão possa ser ainda maior, visto que neste teste não foi possível ir mais longe por conta de obstáculos físicos.

5.2 Validação dos Dados

Para validar os dados de radiação medidos pela EstGeiger utilizamos um outro sensor de mercado, chamado Gamma Scout Alert (GAMMA SCOUT, 2018), o mesmo

utilizado pelos pesquisadores do Grupo de Física Aplicada para realizarem suas medições atualmente.

O processo para validar os dados, consistiu em realizar com uma fonte de radiação (no caso, com amostras de areia monazítica das praias de Meaípe) em diferentes distâncias dos sensores. Ou seja, primeiro foi feita a medição com ambos os sensores a 15 cm da fonte de radiação, depois a 10 cm, depois a 5 cm e finalmente com a fonte encostada nos sensores. Foram feitas 3 medidas em cada posição. A Figura 28 exemplifica como foi feito o processo.

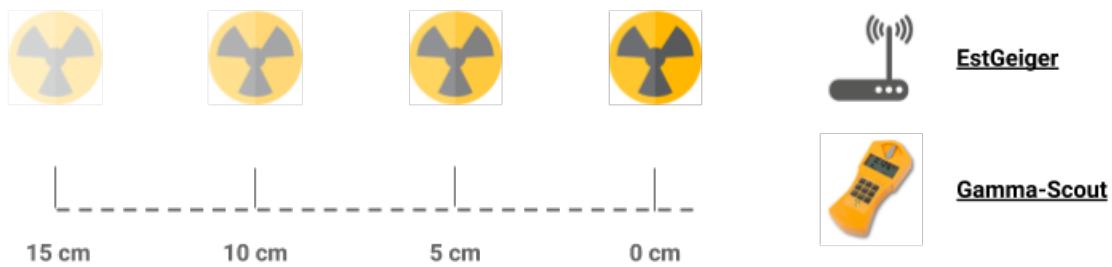


Figura 28 – Esquema de testes com fonte radioativa

O resultado é mostrado nas Tabelas 6, 7, 8 e 9.

Tabela 6 – Comparaçāo de medições com fonte radioativa à 15 cm

	Medição 1 (CPM)	Medição 2 (CPM)	Medição 3 (CPM)
EstGeiger	94	80	75
Gamma-Scout	86	81	80

Tabela 7 – Comparaçāo de medições com fonte radioativa à 10 cm

	Medição 1 (CPM)	Medição 2 (CPM)	Medição 3 (CPM)
EstGeiger	128	141	130
Gamma-Scout	143	118	122

Tabela 8 – Comparaçāo de medições com fonte radioativa à 5 cm

	Medição 1 (CPM)	Medição 2 (CPM)	Medição 3 (CPM)
EstGeiger	222	199	222
Gamma-Scout	218	215	207

Os resultados foram positivos, onde as medições de ambos os sensores ficaram bem próximas, variando entre 10% e 20%. Considerando a natureza estocástica dos eventos, em que devemos observar a faixa dos valores medidos e não apenas valores pontuais.

Tabela 9 – Comparaçāo de medições com fonte radioativa à 0 cm

	Medição 1 (CPM)	Medição 2 (CPM)	Medição 3 (CPM)
EstGeiger	762	797	781
Gamma-Scout	720	768	778

Lembrando que medições de radiação raramente vão dar valores idênticos, mesmo sendo feitas em ambientes com boas condições e sensores iguais. Por conta disso os valores são vistos em níveis ou faixas.

5.3 Testes de Funcionamento

Nos testes de funcionamento foram deixados dois nós enviando dados para um NanoGateway durante pouco mais de uma semana. Durante esse tempo foram sendo monitoradas as plataformas IoT (Cayenne e Ubidots) em relação aos dados recebidos.

- **Plataforma Cayenne**

Na plataforma Cayenne é possível ver todas as métricas enviadas pela EstGeiger, assim como oferece a possibilidade de acionar o modo FTP da estação diretamente por ela. Veja na Figura 29 como é o dashboard de um nó EstGeiger.

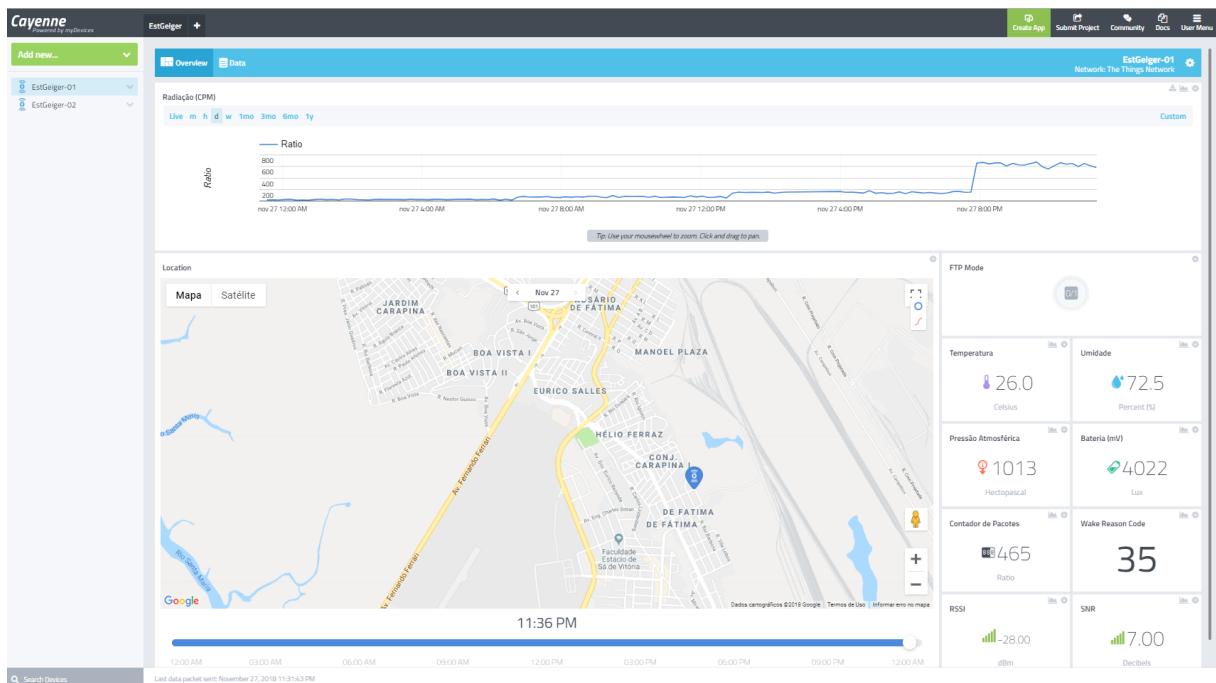


Figura 29 – Dashboard Cayenne de um dispositivo

O valor do nível de radiação medido é mostrado em um gráfico ao longo do tempo. Em um dos testes feitos, colocamos uma fonte de radiação cada vez mais perto do sensor

ao longo do dia, o gráfico visto no Cayenne é mostrado na Figura 30, sendo d a distância entre o nó e a fonte de radiação (em centímetros).



Figura 30 – Gráfico de radiação ao longo de um dia na plataforma Cayenne

Além do gráfico de radiação, também é possível visualizar os gráficos de outras métricas.

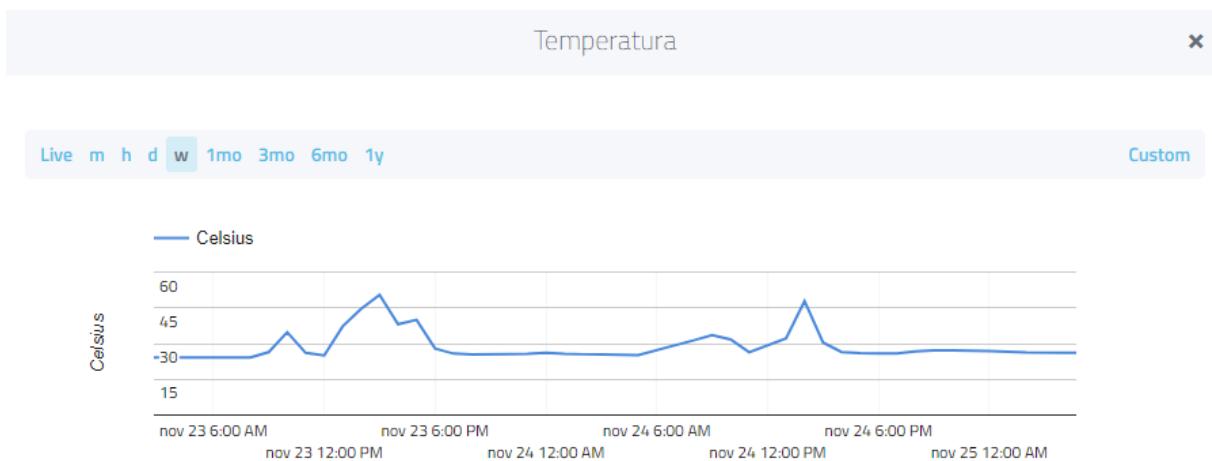


Figura 31 – Gráfico de temperatura em um período selecionado



Figura 32 – Gráfico de umidade no período de uma semana

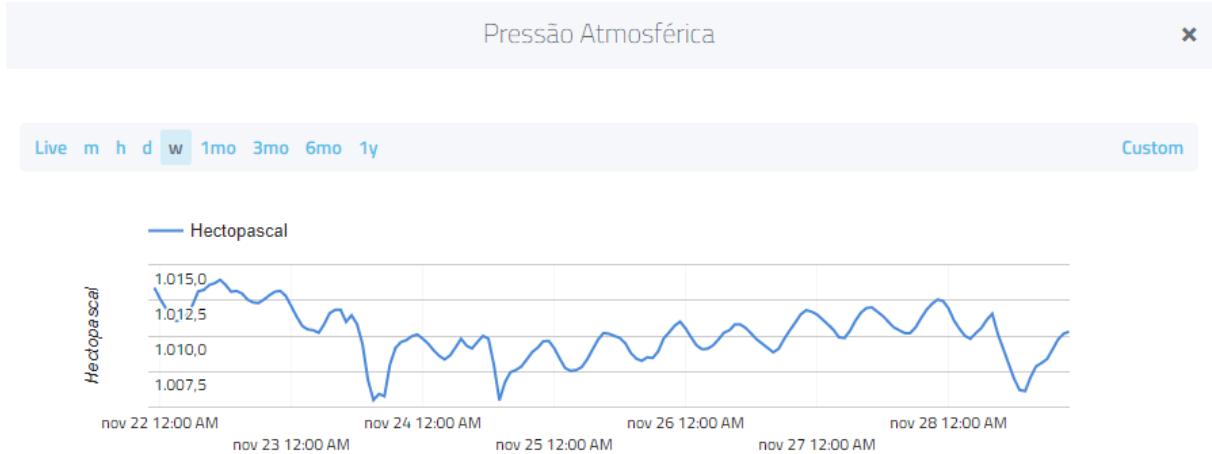


Figura 33 – Gráfico de pressão atmosférica no período de uma semana

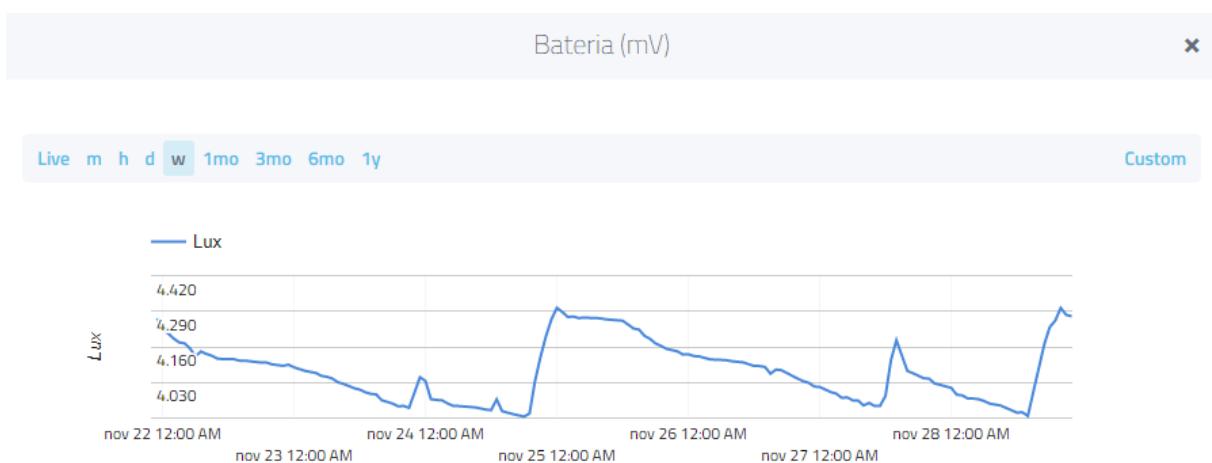


Figura 34 – Gráfico do consumo de bateria a longo de uma semana

A plataforma Cayenne só peca no quesito dados históricos, pois quando escolhemos o tempo de visualização maior que um dia, os dados ficam amostrados com uma frequência menor, ou seja, vários dados são perdidos, por exemplo, se o dispositivo envia dados a cada 10 minutos, no histórico da plataforma não conseguimos visualizar nessa frequência, os dados são mostrados a cada hora, havendo uma perda considerável de informação. Porém esse tipo de técnica é comum e esperado, principalmente em plataformas gratuitas. A solução é exportar os dados em arquivos .csv a cada 1 dia para ter os dados recebidos com frequência maior do que a cada hora. Lembrando que o arquivo de log local em cada nó tem também cópia em microSD de todos os dados enviados.

• Plataforma Ubidots

Na plataforma Ubidots só há a possibilidade de ver dados de radiação. Isso foi uma decisão de projeto, visto que o Ubidots teria uma razão de uso maior para exportar

visualização de dados para outras páginas HTML. A Figura 35 mostra a página principal de visualização dos dados dos dispositivos cadastrados.



Figura 35 – Dashboard da página de dispositivos do Ubidots

Assim como na plataforma Cayenne os dados de radiação são mostrados em forma de gráfico em linha temporal (ver Figura 36), e peca no mesmo ponto de restringir download de dados muito antigos, sendo possível (no máximo) fazer o download apenas dos últimos 500 dados recebidos pela plataforma, implicando em uns 3 dias com dados enviados a cada 10 minutos, ou 20 dias se enviados a cada 1 hora. Mas basta salvar os dados em arquivos .csv antes de findar tais períodos.

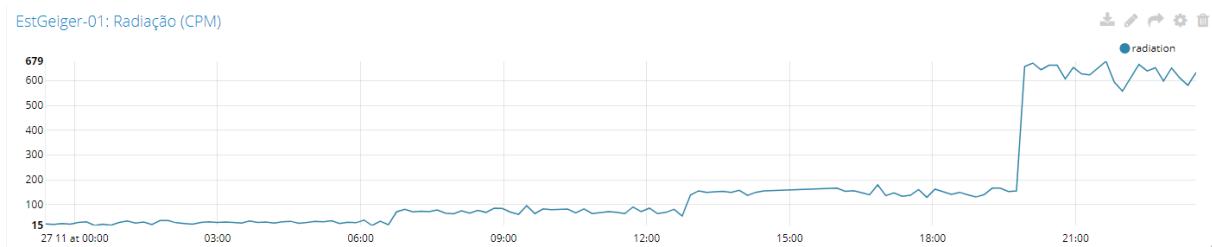


Figura 36 – Gráfico de radiação na plataforma Ubidots

• Arquivos locais - Histograma

Os histogramas gerados podem ser acessados somente no local, através de conexão física com o dispositivo ou por FTP (apenas próximo ao dispositivo). O modo FTP pode ser acionado pelo usuário diretamente pela internet, através do Cayenne ou pelo próprio TTN.

Os histogramas mostram a distribuição de frequência de intervalos de detecção de radiação. A Figura 37 mostra um histograma coletado para uma medição de radiação de 752 CPM.

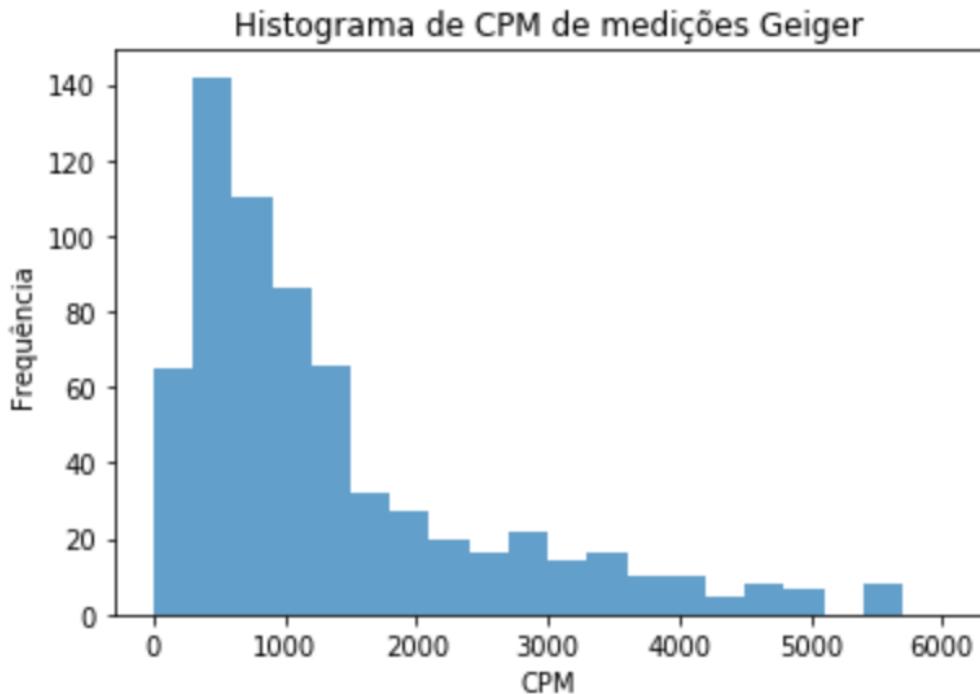


Figura 37 – Exemplo de histograma

5.4 Consumo de Energia

Para realizar os testes de consumo de energia, o dispositivo foi levado a um laboratório onde pode ser testado com fontes e multímetros.

Foram feitas medições de corrente elétrica nas etapas de inicialização do dispositivo, medição da radiação e modo deepsleep. O resultado é mostrado na Tabela 10.

Tabela 10 – Consumo da EstGeiger

Modo	Corrente média medida
Ativo/Medindo	99 mA
DeepSleep	22 mA*

*Valor referente ao consumo do sensor Geiger, visto que ele não desliga quando o dispositivo entra no modo deepsleep.

Como previsto, o dispositivo não apresentou um consumo de energia muito bom, onde esparave-se décimos de Miliampère no modo deepsleep para ser considerado econômico, isso se deve, principalmente, aos seguintes pontos: o sensor geiger fica ligado o tempo inteiro, inclusive no modo deepsleep e medições de CPM serem feitas em 1 minuto (reconfigurável),

ou seja, o nó EstGeiger fica em modo ativo durante uma fração pequena do período entre envio de dados.

5.5 Instalação Inicial em Meaípe

5.5.1 Instalação dos Dispositivos

As estações foram colocadas em torres previamente instaladas pelo Grupo de Física Aplicada, em parceria com o comércio local que cederam os locais.

A Figura 38 mostra os pontos de instalação dos dispositivos.

Inicialmente os dispositivos estão relativamente perto do gateway, pois foram os pontos disponibilizados (com segurança), mas com o desafio de não ter linha de visada devido ao NanoGateway e 2o nó da EstGeiger ficarem em recintos fechados, diminuindo em muito o alcance LoRaWan máximo que se teria com linha de visada. Mas futuramente podem ser instalados mais pontos ao longo da praia.

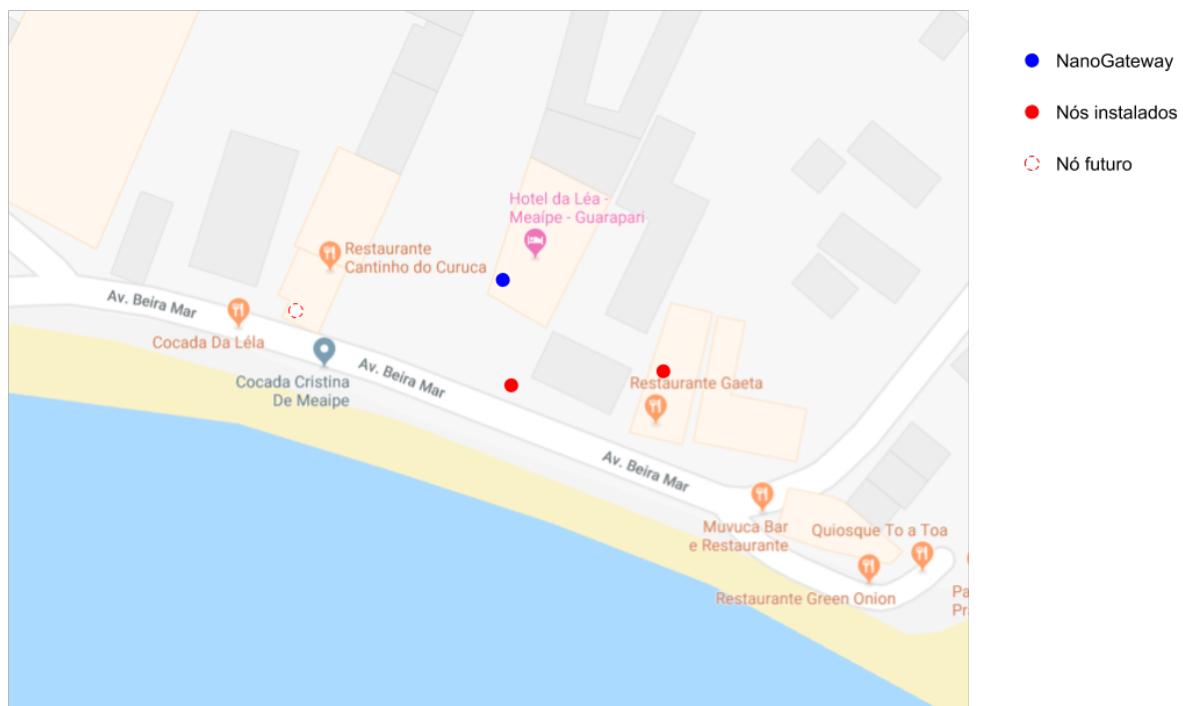


Figura 38 – Localização dos dispositivos instalados

As distâncias entre os dispositivos nós e o NanoGateway é mostrada na Tabela 11.

O primeiro dispositivo instalado (EstGeiger-01) foi em frente ao Hotel da Léa, em uma torre a aproximadamente 6 metros de altura. O segundo dispositivo (EstGeiger-02) foi instalado dentro de um estabelecimento, o Restaurante Gaeta, no andar térreo.

O NanoGateway foi instalado na recepção do Hotel da Léa, por questões de segu-

Tabela 11 – Distância entre nós e NanoGateway

Nós	Distância do NanoGateway (m)
EstGeiger-01	29
EstGeiger-02	45

rança, ficando em um local com pouca visada, mas mesmo assim conseguindo comunicação com os nós.

O software embarcado nos nós na instalação inicial está na versão 1.9.3 (disponível no Apêndice C), enquanto o NanoGateway está na versão 1.8.1 (disponível no Apêndice B), derivado da versão da Pycom Pycom (2018c), com novos recursos de registro em microSD (datalogging) dos dados recebidos e enviados pelo NanoGateway, luzes indicativas do status do NanoGateway usando o RGBLED, capacidade de reenviar pacotes ao servidor LoRaWan após a rede WiFi parar de funcionar por algumas horas e voltar. Tais desenvolvimentos foram feitos pelo autor e orientador, visando divulgação futura de tal código-fonte para a comunidade MicroPython/Pycom.



Figura 39 – Instalação do nó EstGeiger-01



Figura 40 – Instalação do nó EstGeiger-02

5.5.2 Dados Medidos

Após a instalação das duas estações (que ocorreu entre às 19:00h e 20:00h do dia 30/11/2018) os dados já estavam sendo enviados para a nuvem.

Os gráficos mostrados nas Figuras 41, 42, 43, 44, 45, 46, 47 e 48 mostram os dados de radiação, temperatura, umidade e pressão atmosférica medidos por cada estação até o momento da escrita deste documento. Os gráficos foram captados através da plataforma Cayenne, na nuvem, e valores anteriores às 19:00h do dia 30/11/2018 podem ser desconsiderados.

- **Nó EstGeiger-01 (Hotel da Léa)**



Figura 41 – Gráfico de radiação medida em Meaípe, pelo nó EstGeiger-01

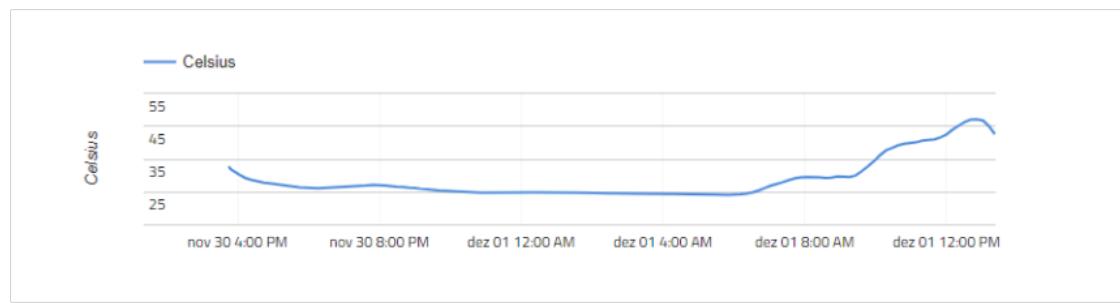


Figura 42 – Gráfico de temperatura medida em Meaípe, pelo nó EstGeiger-01

- **Nó EstGeiger-02 (interior do Restaurante Gaeta)**

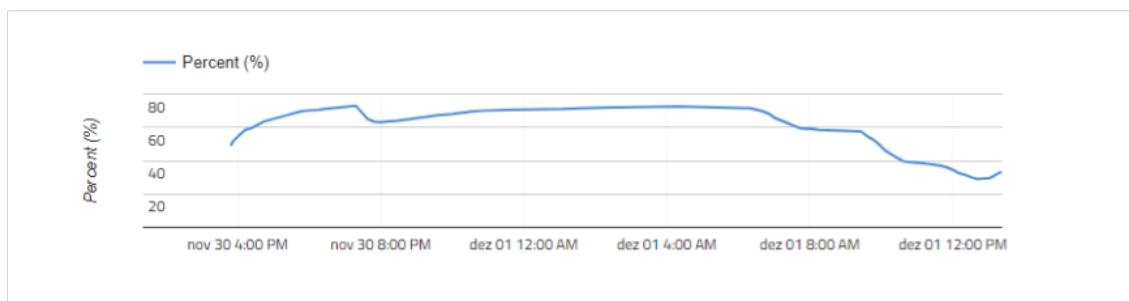


Figura 43 – Gráfico de umidade medida em Meaípe, pelo nó EstGeiger-01

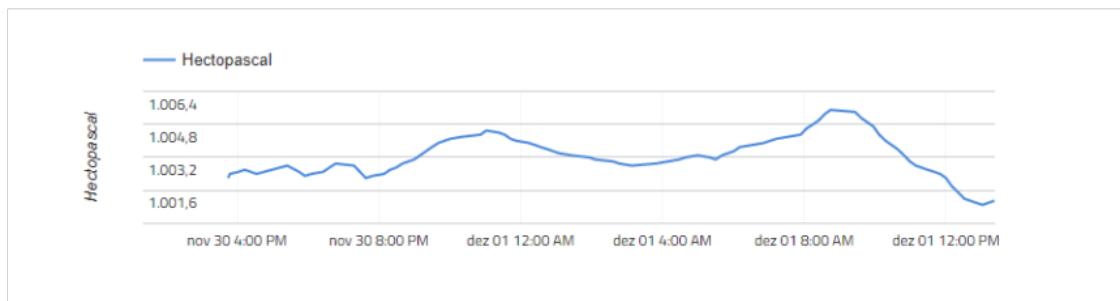


Figura 44 – Gráfico de pressão medida em Meaípe, pelo nó EstGeiger-01



Figura 45 – Gráfico de radiação medida em Meaípe, pelo nó EstGeiger-02

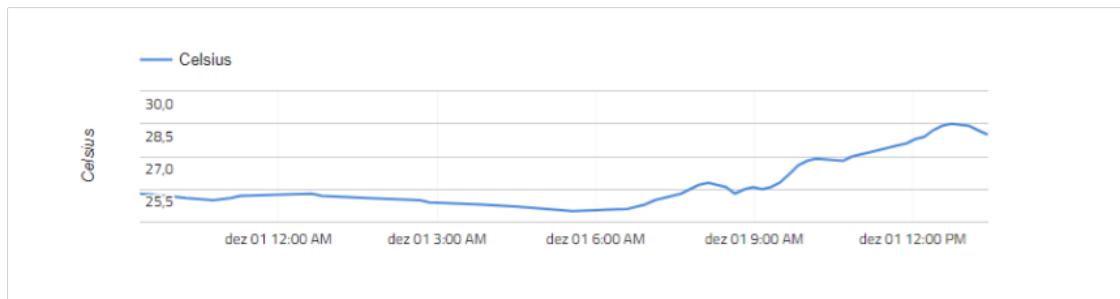


Figura 46 – Gráfico de temperatura medida em Meaípe, pelo nó EstGeiger-02

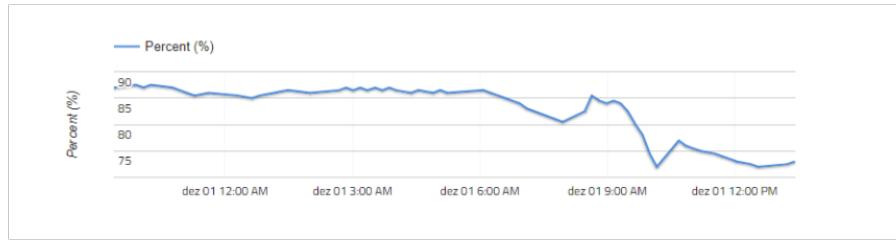


Figura 47 – Gráfico de umidade medida em Meaípe, pelo nó EstGeiger-02

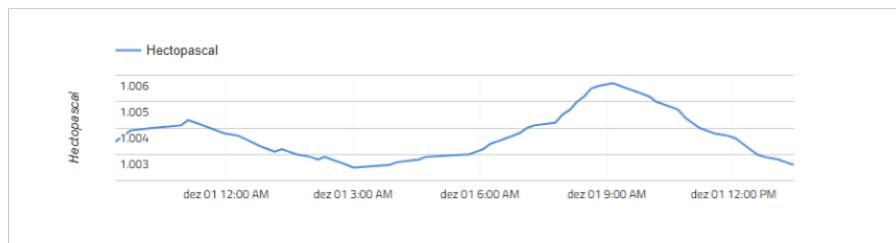


Figura 48 – Gráfico de pressão medida em Meaípe, pelo nó EstGeiger-02

5.5.3 Testes de Distância

Nos testes de distância levamos um nó ao longo de alguns pontos para verificar se os pacotes são recebidos pelo gateway e medir força do sinal (RSSI).

Os principais pontos testados foram os pontos que possuem a infraestrutura para instalação. Em todos os pontos o nó foi capaz de se comunicar com o gateway.

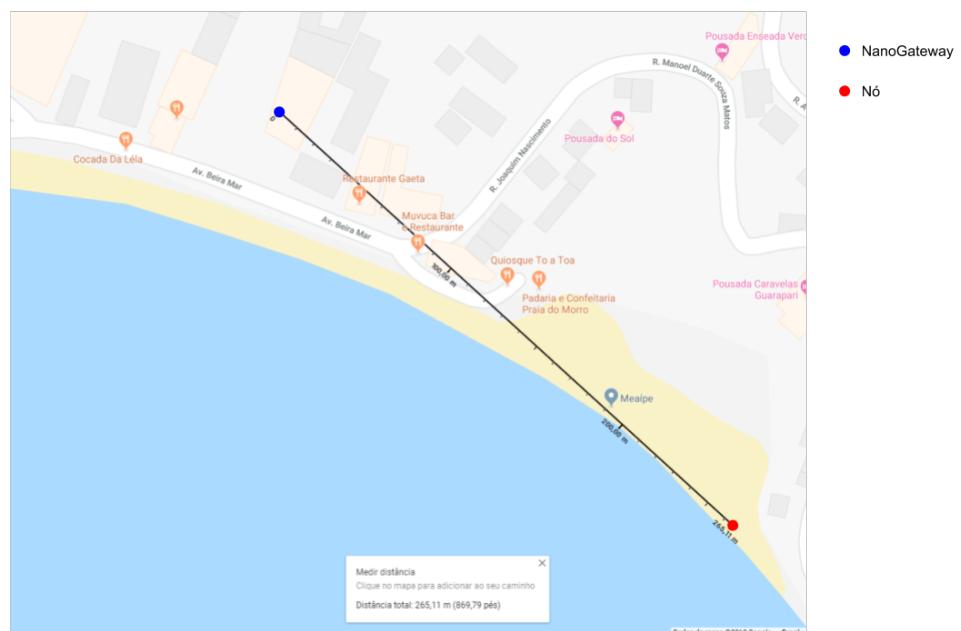


Figura 49 – Distância de transmissão alcançada no teste em Meaípe

Também testamos em distâncias maiores ao longo da praia. A Figura 49 mostra a distância alcançada, cerca de 265 metros, com valor de RSSI de -128. Essa foi a maior distância que pudemos chegar nos testes, ficaram faltando testes em distâncias ainda maiores.

5.6 *Airtime, Colisão e Taxa de Perda de Pacotes*

5.6.1 Airtime

Airtime pode ser definido como o tempo que o pacote fica no ar, ou o tempo de transmissão do pacote. Existem duas restrições que devem ser atendidas para que a transmissão esteja dentro dos padrões definidos pela legislação e pelo uso da plataforma TTN.

A primeira restrição é dada pela ANATEL, onde é definido que o tempo máximo de *airtime* de cada pacote é de 400 ms. A segunda restrição faz parte dos termos de uso da plataforma TTN, que diz que o limite de *airtime* diário é de 30 segundos para cada dispositivo.

Sendo assim as configurações dos parâmetros LoRa também foram pensadas de forma a atender tais requisitos. A Tabela 12 mostra o *airtime* calculado para o menor *payload* que pode ser enviado (14 Bytes) e para o *payload* completo (29 Bytes).

Tabela 12 – *Airtime* calculado

Tamanho do payload	Airtime
14 Bytes	82,432 ms
29 Bytes	123,395 ms

Dessa forma conseguimos ficar dentro dos 400 ms definidos pela ANATEL e também dentro dos 30 segundos diários definidos pala TTN, onde enviando 144 pacotes por dia teremos um total de aproximadamente 17,8 segundos.

5.6.2 Colisão

A probabilidade de colisão é definida pela razão entre *airtime* e intervalo entre cada transmissão. Assim sendo calculada da seguinte forma:

$$\text{Probabilidade de colisão máx.} = \frac{0,123s}{600s} = 0,021\%$$

Se houver colisão significa que os nós estão em sincronia, então: (5.1)

$$\text{Probabilidade de colisão sincronia} = \frac{0,123s}{15s} = 0,82\%$$

Para evitar colisão de pacotes e sincronia entre os nós foi utilizado o tempo de *deepsleep* variável pelo método ALOHA, onde um tempo aleatório (entre 0 e 15 segundos no caso deste projeto) é somado ao *deepsleep* de cada ciclo.

5.6.3 Taxa de Perda de Pacotes

A taxa de perda de pacotes é dada pela percentagem de pacotes perdidos. Onde detectar a perda de pacote é fácil no caso deste projeto, pois um dos dados enviados é justamente o contador de pacotes, sendo assim basta checar se houve um pulo no histórico do contador para detectar perda.

A Tabela 13 mostra as taxas de perdas para os dois nós instalados, sendo essas taxas avaliadas durante um período de 5 dias de funcionamento dos nós.

Tabela 13 – Taxa de perda de pacotes

Nó	PER (Packet Error Rate)
EstGeiger-01	3,58%
EstGeiger-02	2,37%

5.7 Avaliação

5.7.1 Pontos Positivos

As estações demonstraram-se muito consistentes nos testes realizados, cumprindo o que foi proposto, atendendo os requerimentos especificados pelos usuários finais, a saber.

Atingiram os resultados esperados da distância de transmissão, dadas as características da tecnologia empregada. Os dados medidos de radiação em CPM superaram as expectativas, ficando muito próximos dos valores de outros sensores de mercado já utilizados.

Além disso, a utilização da infraestrutura LoRaWAN foi excelente para o projeto, onde é possível a adição de nós extras de medição aproveitando o NanoGateway instalado, com capacidade de atender várias dezenas de nós LoRaWan.

As plataformas de visualização dos dados também cumpriram seu papel, permitindo o acesso remoto e em tempo real dos dados medidos, na forma de gráficos e arquivos com os dados exportados. Lembrando que por serem gratuitas possuem suas limitações.

Robustez face a quedas de energia, com baterias provendo autonomia de até 2 dias para o NanoGateway e até 5 dias para cada nó da EstGeiger.

Redundância e resiliência dos dados medidos são asseguradas via armazenamento local em cada nó e no NanoGateway da EstGeiger, e pelo reenvio de pacotes pelo NanoGa-

teway caso a rede Wifi fique inoperante no NanoGateway e volte a funcionar após algumas horas.

5.7.2 Pontos a Melhorar

Em relação à versão atual da EstGeiger, temos os seguintes pontos a melhorar em termos de testes, análises e procedimentos. Lembrando que a EstGeiger foi testada durante semanas em Vitória-ES, porém a instalação inicial na praia de Meaípe foi feita há pouquíssimo tempo, em 30/11/2018, o que possibilitou somente testes preliminares nesse local.

Tendo mais tempo disponível, está planejado um mapeamento mais detalhado do sinal LoRaWan ao redor do NanoGateway : nos 2 nós já instalados, no terceiro ponto futuro (restaurante Curuca) e pontos mais distantes ao longo da praia de Meaípe, com análise de sinal RSSI, SNR e estatística de perda de pacotes.

Após várias semanas de operação em Meaípe poderemos confirmar a robustez de operação aferida em semanas de testes em Vitória, porém dessa vez na praia de Meaípe, onde a temperatura e umidade nas caixas estanques variam bastante entre dia e noite.

Treinar os usuários finais do Grupo de Física Aplicada a acessar o modo FTP de cada nó EstGeiger para copiar os dados detalhados das medidas, úteis para análises estatísticas em artigos científicos.

Ter retorno dos usuários finais sobre a configuração desejada de certos parâmetros, como o tempo de medida Geiger (atualmente 1 minuto, mas pode ser definido à vontade, para 10 minutos, 1 hora, etc), o tempo entre envio de dados (atualmente 10 minutos, mas pode ser a cada 30 minutos, 1 hora, etc).

Melhorar o posicionamento do NanoGateway no Hotel da Léa, pois atualmente está na mesa da recepção do hotel devido à falta de tempo e recursos materiais no dia 30/11/2018 para instalação em lugar mais alto. Uma sugestão seria continuar perto da recepção, porém em posição elevada, ou no 1º andar do hotel, em área aberta e perto da piscina, fixado ao teto.

6 Conclusões e Trabalhos Futuros

Ao final do desenvolvimento deste projeto, o dispositivo desenvolvido foi capaz de atender os objetivos propostos baseados nos requerimentos definidos pelos usuários finais. Os nós EstGeiger conseguem medir o nível de radiação local e enviá-los para o NanoGateway também instalado próximo a localidade, sendo assim possível visualizar os dados na nuvem em qualquer lugar com acesso a internet.

Podemos dizer ainda que outro objetivo muito importante foi atingido, que é o de realizar um trabalho que tenha alguma utilidade após sua finalização, onde outras pessoas ou grupos possam usá-lo para desenvolver mais pesquisas em cima do que foi feito e do que ele gera, empregando conhecimento de novas tecnologias que a universidade desenvolve em algo que seja útil e traga retorno para a sociedade.

Até o momento da escrita deste documento, a instalação dos dispositivos no local utilizado pelos usuários finais foi muito recente, não sendo possível mostrar seu funcionamento final durante grandes períodos de tempo, apenas por cerca de um a dois dias na praia de Meaípe, porém de forma que mostrou a viabilidade do projeto funcionando em condições reais.

Para os usuários finais do Grupo de Física Aplicada da UFES, a operação da EstGeiger vai significar muito, com medidas ininterruptas (dia e noite, todos os dias) e acesso remoto das medições via Internet. Ou seja, comparadas às visitas in loco à praia de Meaípe a cada 1-2 semanas feitas pelos membros do Grupo de Física Aplicada da UFES, por algumas horas, os dados da EstGeiger serão muito mais completos temporalmente, além de gerar grande economia de tempo e gastos por parte dos usuários finais.

O presente projeto demonstra que soluções abertas de software e hardware, de baixo custo, podem ter um grande impacto para usuários finais, quer sejam cientistas ou sociedade como um todo. O planejado é que a EstGeiger fique operando durante anos de forma ininterrupta fornecendo dados aos usuários finais, com gráficos de radiação em CPM disponíveis em sites públicos como de associação de moradores, prefeitura e estabelecimentos comerciais da região (restaurantes, hotéis, etc).

Os dados medidos pelas estações são públicos e podem ser acessados nas plataformas Ubidots (UBIDOTS, 2018) e/ou Cayenne (CAYENNE, 2018).

6.1 Perspectivas e Trabalhos Futuros

A EstGeiger visa ser um projeto de longa duração, quer seja em termos de utilização por usuários finais como em desenvolvimento de novas versões por parte de diferentes

desenvolvedores, visando a produção de versão 2.x, etc, com qualidade suficiente para ser oferecida para fabricação por parte de empresas capixabas, realizando uma parceria entre UFES e mercado.

A demanda do Grupo de Física Aplicada da UFES é atualmente ter 3 nós EstGeiger em Meaípe (falta instalar o do restaurante Curuca), 1 nó na praia da Areia Preta no centro de Guarapari, outros 2 nós na Grande Vitória (Vila Velha e Vitória), porém tal demanda pode crescer face a disponibilidade de locais com segurança e aos baixos custos da EstGeiger.

Além disso, outros usuários finais poderão utilizar a EstGeiger ou modelo derivado, para aplicações diversas de monitoramento de radiação. Por exemplo, órgãos públicos instalando a EstGeiger em clínicas/laboratórios/hospitais que operam equipamento radioativo, para monitorar se os níveis de radiação em recintos habitados estão dentro dos níveis aceitáveis. Escolas instalando a EstGeiger para ensinar sobre radiação aos professores e alunos, praticando com amostras radioativas, medindo radiação variando com a distância da amostra, etc. Cursos de Geologia medindo radiação de rochas diversas, em laboratórios ou em visitas de campo. Como o granito e mármore do sul do Espírito Santo têm nível de radiação acima da média, então o monitoramento de radiação em locais com muito uso de tais materiais também pode ser importante para a saúde pública.

São diversas as melhorias técnicas previstas para a EstGeiger, visando v2.x, etc. Importante ressaltar que todas elas não faziam parte dos requisitos do Grupo de Física Aplicada, mas permitirão a EstGeiger ser um produto mais completo e útil para maior base de usuários.

6.1.1 Controle via IoT Cloud de configurações da EstGeiger

Via os controles no dashboard de IoT Cloud (Cayenne, etc) é possível enviar comandos e informações ao nó EstGeiger. Atualmente só o modo FTP é habilitado/desabilitado. Mas outras informações podem ser controladas, como o período de medida Geiger (default 60 s), período entre envio de dados (default 10 minutos), configuração do histograma, etc.

Esse recurso evitaria a necessidade de ir ao local do nó EstGeiger para mudar a configuração do mesmo acessando modo FTP.

6.1.2 Reenvio mais robusto de pacotes pelo NanoGateway

Aperfeiçoamento da capacidade de reenvio de pacotes por parte do NanoGateway, tal que se houver queda de acesso Internet mesmo durante dias (não somente horas), os pacotes seriam armazenados no NanoGateway e reenviados ao servidor LoRaWan após retorno da conexão WiFi. Tal versão do software MicroPython "nanogateway.py" será disponibilizada à Pycom e comunidade de usuários.

6.1.3 Otimização de consumo de energia

Menor consumo de energia de cada nó EstGeiger visando autonomia de alguns meses só com bateria, ou sem limite com bateria e painel solar. Tal economia de energia é feita via desligamento da placa Geiger fora do período de medidas, sendo que já foram adquiridos os componentes, porém devido à importação os mesmos só chegaram ao final do projeto atual. Além disso, são necessários longos testes de consumo medindo corrente elétrica x tempo das diversas etapas de funcionamento do nó (vide Figura 19), de implementação de soluções, caracterização de baterias, dimensionamento e caracterização de diferentes painéis solares (tamanho e potências diferentes) em condições reais externas. Para desligar a placa Geiger a priori será usado o "SparkFun MOSFET Power Controller"(SPARKFUN, 2018) que utiliza o MOSFET FDS6630A, de baixo consumo de energia (corrente de fuga de até 1,1 uA). Para lidar com alimentação externa via USB, bateria e painel solar, será usada a placa Elecrow Mini Solar Lipo Charger (ELECROW, 2018), com saída para a placa Geiger e o LoPy 4 na Expansion Board.

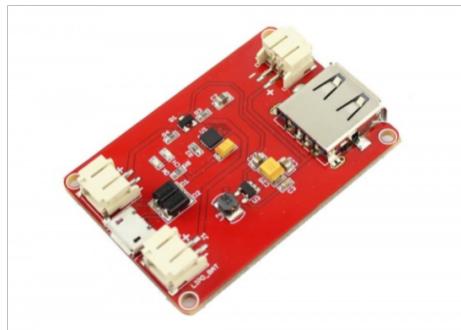


Figura 50 – Mini solar Lipo Charger v1.0

Fonte: Elecrow

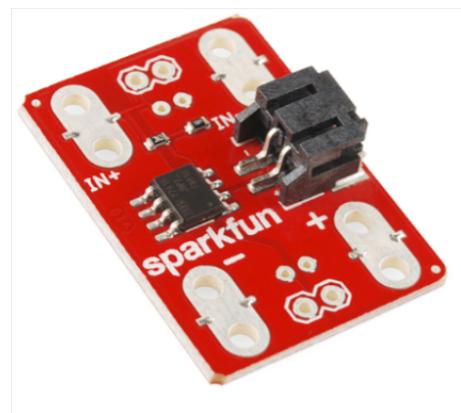


Figura 51 – SparkFun MOSFET Power Controller

Fonte: SparkFun

6.1.4 Servidor LoRaWan e IoT Cloud local na UFES

O servidor LoRaWan TTN tem limitações ao usar a versão pública na nuvem deles, limitando o airtime diário de cada nó a 30s por dia. Por exemplo, para envio de dados a cada 10 minutos, resulta em $30s/(6 \text{ pacotes por hora} \times 24 \text{ horas}) = 208,3 \text{ ms}$ de tempo máximo de cada pacote. Com SF = 8, BW = 125 kHz, CR = 4/5 (DR4 no LoRaWan AU915-928) isso limita o pacote a 62 bytes LoRa ou 49 bytes em LoRaWan, como usamos atualmente até 29 bytes, temos somente 20 bytes de capacidade de crescimento na payload na configuração atual. Mas com servidor LoRaWan TTN instalado no DataCenter do LabNerds da UFES, a limitação seria somente os 400 ms máximos limitados pela ANATEL, resultando em pacote de até 138 bytes LoRa (ou 125 bytes em LoRaWan), ou seja, teríamos 96 bytes de capacidade de crescimento na payload atual. Ou permitira usar configuração LoRaWan de maior alcance, SF = 9, BW = 125 kHz, CR = 4/5 (DR3 no LoRaWan AU915-928) com o pacote de até 66 bytes LoRa (ou 50 bytes em LoRaWan), ainda restando 21 bytes de capacidade de crescimento na payload.

Além disso, servidores IoT Cloud na nuvem pública têm várias limitações, como número máximo de dispositivos, tempo e frequência de armazenamento dos dados, etc. Instalando soluções de código-fonte aberto e gratuitas de IoT Cloud no DataCenter do LabNerds da UFES permitiria remover a maioria dessas limitações, maior controle dos dados, etc. A previsão é futuramente testarmos as soluções ThingsBoard (THINGSBOARD.IO, 2018) e Thinger.io (TINGER.IO, 2018), atualmente já instaladas experimentalmente no DataCenter do LabNerds da UFES, com a EstGeiger.

6.1.5 Envio de dados estatísticos de radiação

Utilizando um *payload* máximo maior, permitiria uma significativa capacidade de crescimento na quantidade de informações enviadas por cada nó EstGeiger. Por exemplo, podem ser enviados dados estatísticos diversos dos intervalos entre detecções Geiger (média, mediana, percentis, etc), ou mesmo histograma. A dificuldade é ter widgets nos IoT Cloud que consigam exibir dados com muitas componentes como histograma, mas algumas soluções com código-fonte aberto e gratuitas de IoT Cloud permitem a programação de widgets personalizados.

6.1.6 Uso de outros medidores Geiger

Outras placas com sensor Geiger podem ser usadas, como o RADMonitor model, com vantagens técnicas de medição de maior CPM ou mesmo radiação alfa (o que não foi requisitado pelos usuários finais), porém com maior custo. Por exemplo, o tubo Geiger LND712 (LND, INC, 2018) que mede radiação alfa, beta, gama, raios-X, é compacto e leve (49mm x 15mm, 8g), com boa sensibilidade, custa entre US\$70-100 (só o tubo).

O empecilho para medir radiação alfa é a necessidade de abertura até o tubo Geiger, não podendo ter parede de caixa, etc, bloqueando radiação alfa, e ao mesmo tempo a EstGeiger suportar maresia durante vários meses.

6.1.7 Versão portátil da EstGeiger

Uma versão derivada da EstGeiger, portátil para ser manuseada e levada para medições em campo, seria muito útil. Teria tela de alto contraste (possivelmente OLED ou ePaper) com interface gráfica, geolocalização das medições Geiger com GPS/Glonass, fusão de dados de IMU (Inertial Measurement Unit) de 10 graus de liberdade (acelerômetro, magnetômetro, giroscópio e barômetro) para indicar a orientação 3D (para onde o sensor Geiger está apontando) e medir a altitude com mais precisão. Faria registro de dados automático, com publicação dos mesmos via servidor web via ponto de acesso Wifi e/ou via LoRaWan. Seria semelhante ao uRADMonitor model D, porém com custo mais baixo, software e hardware abertos.

6.1.8 Uso de Gateway LoRaWan

Possibilidade do uso de um gateway LoRaWAN certificado na Praia de Meaípe no lugar do NanoGateway, teria um custo maior que o NanoGateway utilizado atualmente (entre U\$300 e U\$1000), porém com a vantagem de ter um alcance maior (se bem posicionado, tendo linha de visada), capacidade de 100 à 1000 nós e ser multicanal. Também poderia ser anunciado como um ponto LoRaWan na praia de Meaípe disponível para qualquer dispositivo nó LoRaWan da comunidade local.

Para o projeto EstGeiger, tal gateway LoRaWan certificado permitiria a instalação de nós EstGeiger mais distantes ao longo da praia de Meaípe, bem como menor taxa de perda de pacotes de dados.

Referências

- ALLIANCE, L. A technical overview of LoRa and LoRaWAN. Technical Marketing Workgroup, 2015. Citado 3 vezes nas páginas 10, 26 e 31.
- ANATEL. *Ato nº 14.448*. 2017. Disponível em: <<http://www.anatel.gov.br/legislacao/atos-de-requisitos-tecnicos-de-certificacao/2017/1139-ato-14448>>. Acesso em: 20 nov. 2018. Citado na página 30.
- BOSCH. *BME280*. 2018. Disponível em: <https://www.bosch-sensortec.com/bst/products/all_products/bme280>. Acesso em: 30 nov. 2018. Citado na página 38.
- CALHEIRO D. S.; PASSAMAI JR., J. L. Estudo da Radiação na areia da praia da Areia Preta. *Anais do VII Encontro Científico de Física Aplicada*, Blucher Physics Proceedings, v. 3, n. 1, p. 149–152, 2016. Citado na página 22.
- CALHEIRO D. S.; PASSAMAI JR., J. L. Estudo da Radiação na areia da praia de Meaípe. *Anais do VII Encontro Científico de Física Aplicada*, Blucher Physics Proceedings, v. 3, n. 1, p. 155–158, 2016. Citado na página 22.
- CAYENNE. *Cayenne*. 2018. Disponível em: <<https://cayenne.mydevices.com/shared/5bbc0df781b1bf1f808ee4af/project/1b2b7edf-dfa2-47b9-8e4a-67f5bb8345df>>. Acesso em: 1 dez. 2018. Citado 4 vezes nas páginas 14, 39, 40 e 67.
- CBN. *Praias de Meaípe e Areia Preta estimulam defesa contra câncer de mama*. 2017. Disponível em: <http://www.gazetaonline.com.br/cbn_vitoria/reportagens/2017/12/praias-de-meaipe-e-areia-preta-estimulam-defesa-contra-cancer-de-mama-1014111516.html>. Acesso em: 22 nov. 2018. Citado na página 22.
- COLISTETE, R. *MicroPython Statistics*. 2018. Disponível em: <https://github.com/rcolistete/MicroPython_Statistics>. Acesso em: 30 nov. 2018. Citado na página 41.
- COLISTETE, R. G. N. *MicroPython BME280 Driver*. 2018. Disponível em: <<https://github.com/neliogodoi/MicroPython-BME280>>. Acesso em: 30 nov. 2018. Citado na página 38.
- COSSINI, F. LoRaWAN: uma rede alternativa para a Internet das Coisas. *Technology Leadership Council Brazil*, n. 274, 2016. Citado na página 27.
- ELECROW. *Mini solar Lipo Charger v1.0*. 2018. Disponível em: <https://www.elecrow.com/wiki/index.php?title=Mini_solar_Lipo_Charger_v1.0>. Acesso em: 30 nov. 2018. Citado na página 69.
- EVANS, D. *A Internet das Coisas - Como a próxima evolução da Internet está mudando tudo*. 2016. Disponível em: <http://www.cisco.com/c/dam/global/pt_br/assets/executives/pdf/internet_of_things_iot_ibsg_0411final.pdf>. Acesso em: 18 nov. 2018. Citado na página 23.
- FUJINAMI N.; T. KOGA, T. M. H. External Exposure Rates from Terrestrial Radiation at Guarapari and Meaípe in Brazil. *Hoken Butsuri*, v. 34, p. 253–267, 1999. Citado 2 vezes nas páginas 16 e 22.

- GAMMA SCOUT. *Gamma Scout Alert*. 2018. Disponível em: <<https://www.gammascout.com/products/geigercounter-gamma-scout-alert>>. Acesso em: 30 nov. 2018. Citado na página 51.
- HELAL, A. A. *Proposta de solução integrada de hardware, software para monitoramento ambiental*. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, 2018. Citado na página 34.
- LND, INC. *End window-alpha-beta-gamma detector*. 2018. Disponível em: <<https://www.lndinc.com/products/geiger-mueller-tubes/712/>>. Acesso em: 30 nov. 2018. Citado na página 70.
- LORA ALLIANCE. *What is the LoRaWAN™ Specification?* 2017. Disponível em: <<https://lora-alliance.org/about-lorawan>>. Acesso em: 19 nov. 2018. Citado 2 vezes nas páginas 27 e 29.
- MARQUES, J. J. B.; BOCHIE, K. LoRa. 2018. Citado na página 26.
- PEREIRA ; CARVALHO, F. V. Internet das Coisas (IoT): Cenário e Perspectivas no Brasil e Aplicações Práticas. *VII SEMINÁRIO DE REDES E SISTEMAS DE TELECOMUNICAÇÕES*, INATEL, 2017. Citado 2 vezes nas páginas 23 e 24.
- PEREIRA, A. M. *A Física das Radiações em Sala de Aula: Do Projeto à Prática*. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, 2014. Citado na página 17.
- PYCOM. *Expansion Board*. 2018. Disponível em: <<https://pycom.io/product/expansion-board-3-0>>. Acesso em: 30 nov. 2018. Citado na página 36.
- PYCOM. *LoPy4*. 2018. Disponível em: <<https://pycom.io/product/lopy4>>. Acesso em: 30 nov. 2018. Citado na página 35.
- PYCOM. *LoRaWAN NanoGateway*. 2018. Disponível em: <<https://docs.pycom.io/tutorials/lora/lorawan-nano-gateway.html>>. Acesso em: 30 nov. 2018. Citado na página 59.
- PYCOM. *Pycom*. 2018. Disponível em: <<https://pycom.io/>>. Acesso em: 30 nov. 2018. Citado na página 36.
- RFID JOURNAL. *That 'Internet of Things' Thing*. 2009. Disponível em: <<https://www.rfidjournal.com/articles/view?4986>>. Acesso em: 10 dez. 2018. Citado na página 16.
- RHELETRONICS. *Geiger Counter Radiation Detector*. 2018. Disponível em: <<https://rhelectronics.net/store/radiation-detector-geiger-counter-diy-kit-second-edition.html>>. Acesso em: 30 nov. 2018. Citado na página 36.
- SAFECAST. *bGeigie Nano*. 2018. Disponível em: <<https://blog.safecast.org/bgeigie-nano/>>. Acesso em: 28 nov. 2018. Citado na página 32.
- SANTIAGO, A. *Radiação - Entenda de uma vez por todas*. 2017. Disponível em: <<https://radioprotecaonapratica.com.br/radiacao-entenda-de-uma-vez-por-todas/#radiacao>>. Acesso em: 22 nov. 2018. Citado 3 vezes nas páginas 20, 21 e 22.

- SPARKFUN. *SparkFun MOSFET Power Controller*. 2018. Disponível em: <<https://www.sparkfun.com/products/11214>>. Acesso em: 30 nov. 2018. Citado na página 69.
- TECINMED. *O contador Geiger-Müller*. 2014. Disponível em: <<http://tecinmed.com/artigos/radiologia/contador-g-m.pdf>>. Acesso em: 12 dez. 2017. Citado na página 37.
- TELECO. *Impactos Econômicos da Internet das Coisas no Brasil*. 2016. Citado na página 23.
- THE THINGS NETWORK. *The Things Network Website*. 2018. Disponível em: <<https://www.thethingsnetwork.org/>>. Acesso em: 25 nov. 2018. Citado na página 39.
- THINGER.IO. *Thinger.io Platform*. 2018. Disponível em: <<https://thinger.io/>>. Acesso em: 30 nov. 2018. Citado na página 70.
- THINGSBOARD.IO. *Thingsboard.io Open Source IoT Platform*. 2018. Disponível em: <<https://thingsboard.io/>>. Acesso em: 30 nov. 2018. Citado na página 70.
- UBIDOTS. *Ubidots*. 2018. Disponível em: <https://app.ubidots.com/ubi/public/getdashboard/page/EiqpY-82xpXpC-IHUYNzdU-BifU#>. Acesso em: 1 dez. 2018. Citado 4 vezes nas páginas 14, 39, 40 e 67.
- URADMONITOR. *Products Overview*. 2018. Disponível em: <<https://www.uradmonitor.com/>>. Acesso em: 28 nov. 2018. Citado na página 33.
- VASCONCELOS, D. MODELLING NATURAL RADIOACTIVITY IN SAND BEACHES OF GUARAPARI. 2013. Citado na página 22.

Apêndices

APÊNDICE A – Software TTN Payload Decoder

Listagem A.1 – Código JavaTM do decoder.

```

1  /* Código Java que realiza a decodificação do payload na TTN
2   Autor: Gabriel Favalezzo Fraga
3   Data: 21/08/2018
4   Versão: 1.1
5 */
6
7 function Decoder(bytes, port) {
8     if(bytes.length > 28){
9         var aux    = bytes[2]<<8;
10        var aux2   = bytes[3];
11        var radiation = aux + aux2;
12
13        var aux3    = bytes[6]<<8;
14        var aux4   = bytes[7];
15        var temperature = (aux3 + aux4)/10;
16
17        var aux5    = bytes[10];
18        var humidity = (aux5)/2;
19
20        var aux6    = bytes[13]<<8;
21        var aux7   = bytes[14];
22        var barometer = (aux6 + aux7)/10;
23
24        var aux8    = bytes[17]<<8;
25        var aux9   = bytes[18];
26        var battery = (aux8 + aux9);
27
28        var aux10   = bytes[21]<<8;
29        var aux11  = bytes[22];
30        var packetCounter = (aux10 + aux11);
31        return {
32            Radiation: radiation ,
33            Temperature: temperature ,
34            Humidity: humidity ,
35            Barometer: barometer ,
36            Battery: battery ,
37            Counter: packetCounter
38        }
39    }else{
40        var auxx    = bytes[2]<<8;
41        var auxx2   = bytes[3];
42        var radiation2 = auxx + auxx2;
43        return {
44            Radiation: radiation2
45        }
46    }
47}
48 }
```

APÊNDICE B – Software do NanoGateway

Listagem B.1 – Código MicroPython do NanoGateway.

```

1 """ LoPy LoRaWAN Nano Gateway. Can be used for both EU868 and US915.
2 Version modified by Gabriel Favalessa Fraga <gfavalessaf@gmail.com> :
3 - (2018/10/16) :
4 * added datalogging of uplink packets received from LoRaWan nodes;
5 - (2018/10/30) :
6 * added datalogging of downlink packets sent to LoRaWan nodes;
7 - (2018/10/31) :
8 * added 'datalogging' argument to NanoGateway class init to enable or not
   datalogging to microSD,
9   default is enabled;
10 * more robust microSD datalogging : card initialization moved to 'start()', test
    if there is no
11   microSD card error;
12 * added 'rgbled' argument to NanoGateway class init to enable or not use of LoPy
    RGBLED to show
13   green if wifi communication is ok, red if is not, blue if LoRaWan packet
    received, default is
14   enabled;
15 - (2018/11/01) :
16 * resend uplink LoRaWan packets feature;
17 - (2018/11/05) :
18 * improved resend uplink LoRaWan packets, sending every 60s;
19 * datalogging with uplink packets status (resend, received, forwarded);
20 - (2018/11/07) :
21 * bug fixes on rgbled and resending packets;
22 * more 'rgbled' colors, so all :
23   . white while nanogateway is starting;
24   . red during starting if datalogging is enabled but there is not microSD;
25   . green if Wifi communication with LoRaWan server is ok;
26   . red if there is no Wifi communication with LoRaWan server;
27   . blue if uplink packet is received and forwarded to LoRaWan server;
28   . yellow if downlink packet is sent to LoRaWan node;
29 * added terminal logging messages when push (every 60s) and pull (every 25s)
   data are sent to
30   LoRaWan server;
31 * RAM memory optimization with 'gc.collect()' when starting (logging on
   terminal), adding packet
32   to resend list (logging on terminal), pushing, pulling and reading data from
   the server;
33 """
34
35 import errno
36 import machine
37 import ubinascii
38 import ujson
39 import os # ?
40 import uos
41 import usocket
42 import utime

```

```

43 import _thread
44 from micropython import const
45 from network import LoRa
46 from network import WLAN
47 from machine import Timer
48 from machine import ADC
49 from machine import SD
50 import pycom
51 import gc
52
53 PROTOCOL_VERSION = const(2)
54
55 PUSH_DATA = const(0)
56 PUSH_ACK = const(1)
57 PULL_DATA = const(2)
58 PULL_ACK = const(4)
59 PULL RESP = const(3)
60
61 TX_ERR_NONE = 'NONE'
62 TX_ERR_TOO_LATE = 'TOO_LATE'
63 TX_ERR_TOO_EARLY = 'TOO_EARLY'
64 TX_ERR_COLLISION_PACKET = 'COLLISION_PACKET'
65 TX_ERR_COLLISION_BEACON = 'COLLISION_BEACON'
66 TX_ERR_TX_FREQ = 'TX_FREQ'
67 TX_ERR_TX_POWER = 'TX_POWER'
68 TX_ERR_GPS_UNLOCKED = 'GPS_UNLOCKED'
69
70 UDP_THREAD_CYCLE_MS = const(20)
71
72 # Setting up battery meter
73 BAT_LED = machine.Pin("G16", machine.Pin.OUT)      # LED to indicates battery is
    running out
74 BAT_LED.value(1)
75 adc = ADC()
76 adc.vref(1126) # put here VRef measured in mV with a voltmeter
77 adcP16 = adc.channel(pin='P16', attn=ADC.ATTN_6DB) # integer [0, 4095], where pin
    P16 has voltage divider (115K + 56K)
78
79 STAT_PK = {
80     'stat': {
81         'time': '',
82         'lati': 0,
83         'long': 0,
84         'alti': 0,
85         'rxnb': 0,
86         'rxok': 0,
87         'rxfw': 0,
88         'ackr': 100.0,
89         'dwnb': 0,
90         'txnb': 0
91     }
92 }
93
94 RX_PK = {
95     'rxpk': [
96         'time': '',
97         'tmst': 0,

```

```

98         'chan': 0,
99         'rfch': 0,
100        'freq': 0,
101        'stat': 1,
102        'modu': 'LORA',
103        'datr': '',
104        'codr': '4/5',
105        'rssr': 0,
106        'lsnr': 0,
107        'size': 0,
108        'data': '',
109    }]
110 }
111
112 TX_ACK_PK = {
113     'txpk_ack': {
114         'error': ''
115     }
116 }
117
118
119 class NanoGateway:
120     """
121     Nano gateway class, set up by default for use with TTN, but can be configured
122     for any other network supporting the Semtech Packet Forwarder.
123     Only required configuration is wifi_ssid and wifi_password which are used for
124     connecting to the Internet.
125     """
126
127     def __init__(self, id, frequency, datarate, ssid, password, server, port,
128                  ntp_server='pool.ntp.org', ntp_period=3600,
129                  datalogging=True, rgbled=True, max_resend_uplink_packets=100):
130         self.id = id
131         self.server = server
132         self.port = port
133
134         self.frequency = frequency
135         self.datarate = datarate
136
137         self.ssid = ssid
138         self.password = password
139
140         self.ntp_server = ntp_server
141         self.ntp_period = ntp_period
142
143         self.datalogging = datalogging
144
145         self.rgbled = rgbled
146
147         self.max_resend_uplink_packets = max_resend_uplink_packets
148         self.resend_uplink_packets = []
149
150         self.server_ip = None
151
152         self.rxnrb = 0
153         self.rxok = 0
154         self.rxfw = 0

```

```

154     self.dwnb = 0
155     self.txnb = 0
156
157     self.sf = self._dr_to_sf(self.datarate)
158     self.bw = self._dr_to_bw(self.datarate)
159
160     self.stat_alarm = None
161     self.pull_alarm = None
162     self.uplink_alarm = None
163
164     self.wlan = None
165     self.sock = None
166     self.udp_stop = False
167     self.udp_lock = _thread.allocate_lock()
168
169     self.lora = None
170     self.lora_sock = None
171
172     self.rtc = machine.RTC()
173
174     self.uSDcard = None
175
176 def start(self):
177     """
178     Starts the LoRaWAN nano gateway.
179     """
180
181     if self.rgbled:
182         pycom.heartbeat(False)
183         pycom.rgbled(0xFFFFF)    # RGBLED white
184
185     self._log('Starting LoRaWAN nano gateway with id: {}', self.id)
186
187     self._log('Free RAM before gc.collect() : {:.3f} KB', gc.mem_free()/1024)
188     gc.collect()
189     self._log('Free RAM after gc.collect() : {:.3f} KB', gc.mem_free()/1024)
190
191     # setup WiFi as a station and connect
192     self.wlan = WLAN(mode=WLAN.STA)
193     self._connect_to_wifi()
194
195     # get a time sync
196     self._log('Syncing time with {} ...', self.ntp_server)
197     self.rtc.ntp_sync(self.ntp_server, update_period=self.ntp_period)
198     while not self.rtc.synced():
199         utime.sleep_ms(50)
200     self._log("RTC NTP sync complete")
201
202     # get the server IP and create an UDP socket
203     self.server_ip = usocket.getaddrinfo(self.server, self.port)[0][-1]
204     self._log('Opening UDP socket to {} ({}), port {}...', self.server, self.
205               server_ip[0], self.server_ip[1])
206     self.sock = usocket.socket(usocket.AF_INET, usocket.SOCK_DGRAM, usocket.
207                               IPPROTO_UDP)
208     self.sock.setsockopt(usocket.SOL_SOCKET, usocket.SO_REUSEADDR, 1)
209     self.sock.setblocking(False)

```

```

209     # push the first time immediately
210     self._push_data(self._make_stat_packet())
211
212     # create the alarms
213     self.stat_alarm = Timer.Alarm(handler=lambda t: self._push_data(self.
214         _make_stat_packet()), s=60, periodic=True)
215     self.pull_alarm = Timer.Alarm(handler=lambda u: self._pull_data(), s=25,
216         periodic=True)
217
218     # start the UDP receive thread
219     self.udp_stop = False
220     _thread.start_new_thread(self._udp_thread, ())
221
222     # initialize the LoRa radio in LORA mode
223     self._log('Setting up the LoRa radio at {} Mhz using {}'.format(
224         self.freq_to_float(self.frequency), self.datarate))
225     self.lora = LoRa(
226         mode=LoRa.LORA,
227         frequency=self.frequency,
228         bandwidth=self.bw,
229         sf=self.sf,
230         preamble=8,
231         coding_rate=LoRa.CODING_4_5,
232         tx_iq=True
233     )
234
235     # create a raw LoRa socket
236     self.lora_sock = usocket.socket(usocket.AF_LORA, usocket.SOCK_RAW)
237     self.lora_sock.setblocking(False)
238     self.lora_tx_done = False
239
240     # Mount the microSD card if datalogging is enabled
241     if self.datalogging:
242         try:
243             sd = SD()
244             os.mount(sd, '/sd')
245             self._log('microSD card mounted with {:.3f} GB free'.format(
246                 os.getfree('/sd')/(1024**2)))
247             self.uSDcard = True
248         except OSError as e:
249             self._log('Error while mounting microSD : {}'.format(e))
250             self.uSDcard = False
251             if self.rgbled:
252                 pycom.rgbled(0xFF0000)      # RGBLED red
253
254             self.lora.callback(trigger=(LoRa.RX_PACKET_EVENT | LoRa.TX_PACKET_EVENT),
255                               handler=self._lora_cb)
256             self._log('LoRaWAN nano gateway online')
257
258     def stop(self):
259         """
260             Stops the LoRaWAN nano gateway.
261         """
262
263         self._log('Stopping ...')
264
265         # send the LoRa radio to sleep

```

```

261         self.lora.callback(trigger=None, handler=None)
262         self.lora.power_mode(LoRa.SLEEP)
263
264     # stop the NTP sync
265     self.rtc.ntp_sync(None)
266
267     # cancel all the alarms
268     self.stat_alarm.cancel()
269     self.pull_alarm.cancel()
270
271     # signal the UDP thread to stop
272     self.udp_stop = True
273     while self.udp_stop:
274         utime.sleep_ms(50)
275
276     # disable WLAN
277     self.wlan.disconnect()
278     self.wlan.deinit()
279     if self.rgbled:
280         pycom.rgbled(0x000000)    # RGBLED turned off
281
282     def _connect_to_wifi(self):
283         self.wlan.connect(self.ssid, auth=(None, self.password))
284         while not self.wlan.isconnected():
285             utime.sleep_ms(50)
286             self._log('WiFi connected to: {}', self.ssid)
287
288     def _dr_to_sf(self, dr):
289         sf = dr[2:4]
290         if sf[1] not in '0123456789':
291             sf = sf[:1]
292         return int(sf)
293
294     def _dr_to_bw(self, dr):
295         bw = dr[-5:]
296         if bw == 'BW125':
297             return LoRa.BW_125KHZ
298         elif bw == 'BW250':
299             return LoRa.BW_250KHZ
300         else:
301             return LoRa.BW_500KHZ
302
303     def _sf_bw_to_dr(self, sf, bw):
304         dr = 'SF' + str(sf)
305         if bw == LoRa.BW_125KHZ:
306             return dr + 'BW125'
307         elif bw == LoRa.BW_250KHZ:
308             return dr + 'BW250'
309         else:
310             return dr + 'BW500'
311
312     def _lora_cb(self, lora):
313         """
314             LoRa radio events callback handler.
315         """
316
317         events = lora.events()

```

```

318     if events & LoRa.RX_PACKET_EVENT:
319         if self.rgbled:
320             pycom.rgbled(0x0000FF)      # RGBLED blue
321             self.rxnb += 1
322             self.rxok += 1
323             rx_data = self.lora_sock.recv(256)
324             stats = lora.stats()
325             packet = self._make_node_packet(rx_data, self rtc.now(), stats.
326                 rx_timestamp, stats.sfrx, self.bw, stats.rssi, stats.snr)
327             self._log('Received packet: {}', packet)
328             rxfwok = self._push_data(packet, resend=True)
329             # Datalogging LoRaWan uplink packet (received from node) to microSD
330             # file
331             # 1st field is the number of packets left to be resent to LoRaWan
332             # server
333             # 2nd field is rxok (number of uplink LoRaWan packets received from
334             # LoRaWan nodes to be forwarded to LoRaWan server)
335             # 3rd field is rxfw (number of uplink LoRaWan packets successfully
336             # forwarded to LoRaWan server)
337             # 4th field is the packet
338             if self.uSDcard:
339                 with open('/sd/gateway_uplink_datalogging.csv', 'a') as f:
340                     f.write("{}\n".format(len(self.
341                         resend_uplink_packets), self.rxok, self.rxfw, packet))
342                     os.sync()
343             if events & LoRa.TX_PACKET_EVENT:
344                 self.txnb += 1
345                 lora.init(
346                     mode=LoRa.LORA,
347                     frequency=self.frequency,
348                     bandwidth=self.bw,
349                     sf=self.sf,
350                     preamble=8,
351                     coding_rate=LoRa.CODING_4_5,
352                     tx_iq=True
353                 )
354
355             def _freq_to_float(self, frequency):
356                 """
357                 MicroPython has some inprecision when doing large float division.
358                 To counter this, this method first does integer division until we
359                 reach the decimal breaking point. This doesn't completely eliminate
360                 the issue in all cases, but it does help for a number of commonly
361                 used frequencies.
362                 """
363
364             divider = 6
365             while divider > 0 and frequency % 10 == 0:
366                 frequency = frequency // 10
367                 divider -= 1
368             if divider > 0:
369                 frequency = frequency / (10 ** divider)
370             return frequency
371
372             def _make_stat_packet(self):
373                 now = self.rtc.now()
374                 STAT_PK["stat"]["time"] = "%d-%02d-%02d %02d:%02d:%02d GMT" % (now[0],

```

```

now[1], now[2], now[3], now[4], now[5])
369 STAT_PK["stat"]["rxnb"] = self.rxnb
370 STAT_PK["stat"]["rxok"] = self.rxok
371 STAT_PK["stat"]["rxfw"] = self.rxfw
372 STAT_PK["stat"]["dwnb"] = self.dwnb
373 STAT_PK["stat"]["txnb"] = self.txnb
374
375     return ujson.dumps(STAT_PK)

376 def _make_node_packet(self, rx_data, rx_time, tmst, sf, bw, rssi, snr):
377     RX_PK["rxpk"][0]["time"] = "%d-%02d-%02dT%02d:%02d.%dZ" % (rx_time
378     [0], rx_time[1], rx_time[2], rx_time[3], rx_time[4], rx_time[5],
379     rx_time[6])
380     RX_PK["rxpk"][0]["tmst"] = tmst
381     RX_PK["rxpk"][0]["freq"] = self._freq_to_float(self.frequency)
382     RX_PK["rxpk"][0]["datr"] = self._sf_bw_to_dr(sf, bw)
383     RX_PK["rxpk"][0]["rssi"] = rssi
384     RX_PK["rxpk"][0]["lsnr"] = snr
385     RX_PK["rxpk"][0]["data"] = ubinascii.b2a_base64(rx_data)[-1]
386     RX_PK["rxpk"][0]["size"] = len(rx_data)
387     return ujson.dumps(RX_PK)

388 #     self._log('Free RAM before collect() when pushing data : {:.3f} KB', gc.
389 mem_free()/1024)
390 #     gc.collect()
391 #     self._log('Free RAM after collect() when pushing data : {:.3f} KB', gc.
392 mem_free()/1024)
393 token = uos.urandom(2)
394 packet = bytes([PROTOCOL_VERSION]) + token + bytes([PUSH_DATA]) +
395     ubinascii.unhexlify(self.id) + data
396 rxfw_before = self.rxfw
397 send_current_packet_from_resend_list = False
398 if resend:
399     self._log('New packet to be sent to LoRaWan server has {} bytes',
400             (packet))
401     if self.max_resend_uplink_packets > 0:
402         if len(self.resend_uplink_packets) < self.
403             max_resend_uplink_packets:
404             self.resend_uplink_packets.append(packet)
405             send_current_packet_from_resend_list = True
406             self._log('Free RAM before gc.collect() when adding packet to
407                     the resend list : {:.3f} KB', gc.mem_free()/1024)
408             gc.collect()
409             self._log('Free RAM after gc.collect() when adding packet to
410                     the resend list : {:.3f} KB', gc.mem_free()/1024)
411     else:
412         self._log('List of resend uplink packets is full, limit is {}',
413             (len(self.max_resend_uplink_packets)))
414 else:
415     self._log('Sending push data with nanogateway status to LoRaWan
416             server : {}', data)
417 with self.udp_lock:
418     try:
419         if len(self.resend_uplink_packets) > 0:
420             self._log('Trying to send {} packet(s) to LoRaWan server',
421                     len(self.resend_uplink_packets))
422             while len(self.resend_uplink_packets) > 0:

```

```

413         self.sock.sendto(self.resend_uplink_packets[0], self.
414                         server_ip)
415         del self.resend_uplink_packets[0]
416         self.rxfw += 1
417         utime.sleep_ms(1500)
418     if not send_current_packet_from_resend_list:
419         self.sock.sendto(packet, self.server_ip)
420         if resend:
421             self.rxfw += 1
422     if (not resend) and self.rgbled:
423         pycom.rgbled(0x00FF00)    # RGBLED green
424         sentok = True
425     except Exception as ex:
426         self._log('Failed to push uplink packet to LoRaWan server: {}', ex)
427     if self.rgbled:
428         pycom.rgbled(0xFF0000)    # RGBLED red
429         sentok = False
430     if self.rxfw - rxfw_before > 0:
431         self._log('{} packet(s) successfully sent to LoRaWan server', self.
432                   rxfw - rxfw_before)
433     if len(self.resend_uplink_packets) > 0:
434         self._log('{} packet(s) left to be resent to LoRaWan server', len(
435                   self.resend_uplink_packets))
436     return sentok
437
438     def _pull_data(self):
439         self._log('Free RAM before gc.collect() when pulling data : {:.3f} KB',
440                  gc.mem_free()/1024)
441         gc.collect()
442         self._log('Free RAM after gc.collect() when pulling data : {:.3f} KB',
443                  gc.mem_free()/1024)
444         token = uos.urandom(2)
445         packet = bytes([PROTOCOL_VERSION]) + token + bytes([PULL_DATA]) +
446                  ubinascii.unhexlify(self.id)
447         with self.udp_lock:
448             try:
449                 self.sock.sendto(packet, self.server_ip)
450                 self._log('Sent pull data to LoRaWan server')
451                 if self.rgbled:
452                     pycom.rgbled(0x00FF00)    # RGBLED green
453             except Exception as ex:
454                 self._log('Failed to pull downlink packets from server: {}', ex)
455                 if self.rgbled:
456                     pycom.rgbled(0xFF0000)    # RGBLED red
457
458     def _ack_pull_rsp(self, token, error):
459         TX_ACK_PK["txpk_ack"]["error"] = error
460         resp = ujson.dumps(TX_ACK_PK)
461         packet = bytes([PROTOCOL_VERSION]) + token + bytes([PULL_ACK]) +
462                  ubinascii.unhexlify(self.id) + resp
463         with self.udp_lock:
464             try:
465                 self.sock.sendto(packet, self.server_ip)
466             except Exception as ex:
467                 self._log('PULL RSP ACK exception: {}', ex)
468                 if self.rgbled:

```

```

462                     pycom.rgbled(0xFF0000)      # RGBLED red
463
464     def _send_down_link(self, data, tmst, datarate, frequency):
465         """
466             Transmits a downlink message over LoRa.
467         """
468
469         self.lora.init(
470             mode=LoRa.LORA,
471             frequency=frequency,
472             bandwidth=self._dr_to_bw(datarate),
473             sf=self._dr_to_sf(datarate),
474             preamble=8,
475             coding_rate=LoRa.CODING_4_5,
476             tx_iq=True
477         )
478         while utime.ticks_cpu() < tmst:
479             pass
480         self.lora_sock.send(data)
481         if self.rgbled:
482             pycom.rgbled(0xFFFF00)      # RGBLED yellow
483         self._log(
484             'Sent downlink packet scheduled on {:.3f}, at {:.3f} MHz using {}:{}\n'
485             ,
486             tmst / 1000000,
487             self._freq_to_float(frequency),
488             datarate,
489             data
490         )
491         # Datalogging LoRaWan downlink packet (sent to node) to microSD file
492         # 1st field is dwnb, number of downlink packets received from LoRaWan
493         # server
494         # 2nd field is the time in seconds the LoRaWan packet is scheduled to be
495         # sent
496         # 3rd field is the frequency in MHz of the LoRaWan packet sending
497         # 4th field is the datarate of the LoRaWan packet sending
498         # 5th field is the data of the LoRaWan packet sending
499         if self.uSDcard:
500             with open('/sd/gateway_downlink_datalogging.csv', 'a') as f:
501                 f.write("{}{}, {:.3f}, {:.3f}, {}, {}\n".format(self.dwnb, tmst
502                     /1000000, self._freq_to_float(frequency), datarate, data))
503                 os.sync()
504
505     def _udp_thread(self):
506         """
507             UDP thread, reads data from the server and handles it.
508         """
509
510         while not self.udp_stop:
511             gc.collect()
512             try:
513                 data, src = self.sock.recvfrom(1024)
514                 _token = data[1:3]
515                 _type = data[3]
516                 if self.rgbled:
517                     pycom.rgbled(0x00FF00)      # RGBLED green
518                 if _type == PUSH_ACK:

```

```

515             self._log("Push ack")
516         elif _type == PULL_ACK:
517             self._log("Pull ack")
518         elif _type == PULL_RESP:
519             self.dwnb += 1
520             ack_error = TX_ERR_NONE
521             tx_pk = ujson.loads(data[4:])
522             tmst = tx_pk["txpk"]["tmst"]
523             t_us = tmst - utime.ticks_cpu() - 15000
524             if t_us < 0:
525                 t_us += 0xFFFFFFFF
526             if t_us < 20000000:
527                 self.uplink_alarm = Timer.Alarm(
528                     handler=lambda x: self._send_down_link(
529                         ubinascii.a2b_base64(tx_pk["txpk"]["data"]),
530                         tx_pk["txpk"]["tmst"] - 50, tx_pk["txpk"]["datr"]
531                         ],
532                         int(tx_pk["txpk"]["freq"] * 1000) * 1000
533                     ),
534                     us=t_us
535                 )
536             else:
537                 ack_error = TX_ERR_TOO_LATE
538                 self._log('Downlink timestamp error!', t_us: {}, t_us)
539                 self._ack_pull_rsp(_token, ack_error)
540                 self._log("Pull rsp")
541     except usocket.timeout:
542         pass
543     except OSError as ex:
544         if ex.errno != errno.EAGAIN:
545             self._log('UDP recv OSError Exception: {}', ex)
546             if self.rgbled:
547                 pycom.rgbled(0xFF0000) # RGBLED red
548     except Exception as ex:
549         self._log('UDP recv Exception: {}', ex)
550         if self.rgbled:
551             pycom.rgbled(0xFF0000) # RGBLED red
552         batt_voltage = (171*adcP16.voltage())//56 # converting to integer
553         voltage, [0, 6702] mV
554         if(batt_voltage < 3200):
555             BAT_LED.value(0)
556         else:
557             BAT_LED.value(1)
558         # wait before trying to receive again
559         utime.sleep_ms(UDP_THREAD_CYCLE_MS)
560
561     # we are to close the socket
562     self.sock.close()
563     self.udp_stop = False
564     self._log('UDP thread stopped')
565
566 def _log(self, message, *args):
567     """
568     Outputs a log message to stdout.
569     """

```

```
570  
571     print( '[{:>10.3f}] {}' .format(  
572         utime.ticks_ms() / 1000,  
573         str(message) .format(*args)  
574     ))
```

APÊNDICE C – Software do EstGeiger Nô

Listagem C.1 – Código MicroPython da EstGeiger Nô.

```

1 # MicroPython Geiger Counter Main Code for Pycom LoPy4 board (TheThingsNetwork
2 # IoT Platform)
3 #
4 # This code interacts with the RadiationD-v1.1 (CAJOE) Geiger counter board
5 # and reports readings in CPM (Counts Per Minute), sending the information
6 # to a LoRaWAN gateway and to the TheThingNetwork IoT Platform, it is using
7 # LoRa modulation (with LoRaWAN protocol).
8 # Updates:
9 # - FTP server
10 # - LoRa nvram save state
11 # - Log Files
12 # - Const on config file
13 # - Debug mode
14 # - using statistic module for mean
15 # - using 'const()' in 'config.py'
16 # - revised the variable payload for temperature, humidity, pressure and
17 #   voltage
18 # - revised wake and reset reason
19 # - use of "region=LoRa.AU915" to initialize LoRaWan
20 # - more terminal debug log
21 # - BME280 statistics on terminal log
22 # - battery voltage statistics on terminal log
23 # - intervals between counts statistics on terminal log
24 # - used ".format()" for formatting strings
25 # - fixed CPM calculation to use interruptCounter and block inside if
26 # - fixed Geiger interruption removing RGBLED usage because it was creating a
27 #   lot of false events
28 # - histogram with intelligent bin size
29 # - uSD files in english
30 # - fixed bug in pressure data sent to Cayenne
31 # - improved payload datalogging
32 # - fixed temperature range sent to Cayenne to include negative values (signed
33 #   integer with 2 bytes, '>h')
34 # ===== Debug Mode =====
35 flagDebug = True
36 if flagDebug:
37     print("===== Start LoPy4 node device =====")
38     from machine import Timer
39     chrono = Timer.Chrono()
40     chrono.start()
41
42
43 # ===== Import Standart Libraries =====

```

```

44 import gc
45 gc.collect()
46 if flagDebug:
47     print('Free RAM before all module imports : {:.3f} KB'.format(gc.mem_free() /1024))
48
49 from machine import Timer, RTC, Pin, I2C, ADC, SD
50 from network import LoRa, WLAN, Bluetooth
51 import binascii
52 import network
53 import machine
54 import socket
55 import struct
56 import pycom
57 import time
58 import os
59
60 # ===== Import Extra Libraries =====
61 import config
62 import bme280
63 import statistics
64 import LoRaAirTimeCalc
65
66
67 # ===== Program initialization =====
68 gc.collect()
69
70 if flagDebug:
71     print('Free RAM after all module imports : {:.3f} KB'.format(gc.mem_free() /1024))
72     print("{{}} firmware v{{}} MicroPython {{}}".format(os.uname()[4], os.uname()[2], os.uname()[3]))
73     print("Import time: {:.3f} ms".format(chrono.read()*1000))
74
75 pycom.heartbeat(False)
76 pycom.rgbled(config.START_COLOUR) # Blink red to show the initialization
77
78 # ===== LoRaWan Configuration =====
79 if flagDebug:
80     print("===== LoRaWan Configuration =====")
81     chrono.reset()
82
83 lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.AU915)
84 lora.nvram_restore()
85
86 if(not lora.has_joined()):
87     print("LoRaWan Join in manual mode... ")
88
89     # create an OTA authentication params
90     dev_addr = struct.unpack(">L", binascii.unhexlify('26011287'))[0]    # these
91     settings can be found from TTN
92     nwk_swkey = binascii.unhexlify('342B1B1E430D1FEC42820744D2463F4B')    # these
93     settings can be found from TTN
94     app_swkey = binascii.unhexlify('BAB37C176C7F0E7F6C228F100D8FD5B1')    # these

```

```

        settings can be found from TTN
93
94     # remove all the channels
95     for channel in range(0, 72):
96         lora.remove_channel(channel)
97
98     # set all channels to the same frequency (must be before sending the OTAA
99     # join request)
100    for channel in range(0, 8):
101        lora.add_channel(channel, frequency=config.LORA_FREQUENCY, dr_min=0,
102                        dr_max=5)
103    lora.add_channel(8, frequency=config.LORA_FREQUENCY, dr_min=6, dr_max=6)
104
105    # join a network using ABP
106    lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey, app_swkey), timeout
107              =0)
108
109    # create a LoRa socket
110    s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
111
112    # set the LoRaWAN data rate
113    s.setsockopt(socket.SOL_LORA, socket.SO_DR, config.LORA_NODE_DR)
114
115    s.setblocking(False)
116
117    gc.collect()
118
119    if flagDebug:
120        print('Free RAM after LoRa configuration : {:.3f} KB'.format(gc.mem_free() /1024))
121        print('LoRaWan frequency {:.1f} MHz, {}, DR (Data Rate) = {}'.format(config.
122                             LORA_FREQUENCY/1E6, config.LORA_GW_DR, config.LORA_NODE_DR))
123        print("LoRaWan Configuration time: {:.3f} ms".format(chrono.read()*1000))
124
125    # ===== Packet Counter =====
126
127    if flagDebug:
128        print("===== Packet Counter =====")
129        chrono.reset()
130
131    if machine.wake_reason()[0] == machine.PWRON_WAKE and pycom.nvs_get('count') != None:
132        pycom.nvs_erase('count')    # 1st time power on
133
134    payloadcount = pycom.nvs_get('count')
135    if payloadcount is not None:
136        payloadcount += 1
137        pycom.nvs_set('count', payloadcount)
138    else:
139        pycom.nvs_set('count', 0)  # Starts from 0
140        payloadcount = pycom.nvs_get('count')
141
142    if flagDebug:
143        print("Packet counter : {}".format(payloadcount))
144        print("Packet Counter time: {:.3f} ms".format(chrono.read()*1000))

```

```

142 # ===== Wake and Reset Reason =====
143 if flagDebug:
144     print("===== Wake and Reset Reason =====")
145     chrono.reset()
146
147 wakeCode = machine.wake_reason()[0]
148 resetCode= machine.reset_cause()
149 wrCode = (wakeCode << 4) | resetCode      # XXXX_ _ _ _ -> wake code - 4 msb
150                                         # _ _ _ X X X X -> reset code - 4 lsb
151
152 if flagDebug:
153     print("Wake and Reset code : 0x{:02x} ".format(wrCode))
154     print("Wake and Reset Reason time: {:.3f} ms".format(chrono.read()*1000))
155
156 # ===== BME280 Sensor =====
157 if flagDebug:
158     print("===== BME280 Sensor =====")
159     chrono.reset()
160
161 i2c = I2C(0, pins=('P9','P10')) # Set pins to I2C -> P9=SDA, P10=SCL (LoPy4)
162 bme = bme280.BME280(i2c=i2c, pressure_mode=bme280.OSAMPLE_8, iir=bme280.FILTER_8)
163                                         # Instantiate sensor, oversampling x2 for temp, x8 for bar, x1 for humid
164                                         # and IIR filter x8
165
166 numreadings = 15
167 samples_temperature = [0.0]*numreadings
168 samples_pressure = [0.0]*numreadings
169 samples_humidity = [0.0]*numreadings
170
171 for i in range(numreadings):
172     samples_temperature[i], samples_pressure[i], samples_humidity[i] = bme.values
173
174 mean_temp = statistics.mean(samples_temperature)    # C
175 mean_pres = statistics.mean(samples_pressure)       # Pa
176 mean_humi = statistics.mean(samples_humidity)      # %
177
178 if flagDebug:
179     print("BME280 mean after {} readings : {:.3f} C, {:.3f}%, {:.2f} Pa".format(
180         numreadings, mean_temp, mean_humi, mean_pres))
181     print("BME280 standart deviation : {:.3f} C, {:.3f}%, {:.2f} Pa".format(
182         statistics.stdev(samples_temperature),
183         statistics.stdev(samples_humidity), statistics.stdev(samples_pressure)))
184
185     print("BME280 Sensor time: {:.3f} ms".format(chrono.read()*1000))
186
187 # ===== Battery Voltage =====
188 if flagDebug:
189     print("===== Battery Voltage =====")
190     chrono.reset()
191
192 adc = ADC()
193 adc.vref(1126) # put here VRef measured in mV with a voltmeter
194 adcP16 = adc.channel(pin='P16', attn=ADC.ATTN_6DB) # integer [0, 4095], where pin
195                                         # P16 has voltage divider (115K + 56K)
196 numreadings = 15
197 samples_voltage = [0.0]*numreadings

```

```

190 meanvoltage = 0
191 for i in range (numreadings):
192     samples_voltage [ i ] = (171*adcP16.voltage ())//56    # converting to integer
193 batt_mV = round( statistics.mean(samples_voltage) )
194
195 if flagDebug:
196     print("Battery voltage minimum : {} mV".format(min(samples_voltage)))
197     print("Battery voltage minimum : {} mV".format(max(samples_voltage)))
198     print("Battery voltage mean after {} readings : {} mV".format(numreadings,
199         batt_mV))
200     print("Battery voltage standart deviation : {} mV".format(statistics.stdev(
201         samples_voltage)))
200     print("Battery Voltage time: {:.3 f} ms".format(chrono.read ()*1000))
201
202 # ===== Initialize Variables
203
203 multiplier = 60 / config.COUNTER_TIME    # CPM is measured each minute, so the
204 constant is 60 seconds
204 cpm = 0
205 interruptCounter = 0
206 totalInterval = 0
207 start = time.ticks_us()
208 intervalList = [0]*config.HIST_SIZE
209
210 # ===== Callback function for Geiger interrupt
210 counter =====
211 def counter_callback(pin):
212     global intervalList
213     global start
214     global interruptCounter
215     global totalInterval
216     interval = time.ticks_diff(time.ticks_us(), start)
217     if interval > config.INTERVAL:
218         totalInterval += interval
219         if (interruptCounter < len(intervalList)):
220             intervalList [interruptCounter] = interval
221             interruptCounter = interruptCounter + 1
222             start = time.ticks_us()
223
224 # ===== Setup interrupt pin
224
225 p0 = Pin('P18', Pin.IN)                  # Geiger Sensor Pin P18 = G30
226 p0.callback(trigger=Pin.IRQ_FALLING, handler=counter_callback)
227
228 # ===== Start Geiger Counting
228
229 if flagDebug:
230     print("===== Counting Geiger events during {}s ... =====".format(
231         config.COUNTER_TIME))
232 pycom.rgbled(config.COUNTER_COLOUR)
233 interruptCounter = 0
234 time.sleep(config.COUNTER_TIME)
235
236 # ===== CPM and Histogram Calculation
236

```

```

237 if flagDebug:
238     chrono.reset()
239
240 intervalListFiltered = list(filter(lambda x: x > 0, intervalList))
241 mean_CPM = round(interruptCounter*multiplier)
242
243 if flagDebug:
244     print("{} Geiger counts during {}s".format(interruptCounter, config.COUNTER_TIME))
245     print("{} Geiger counts saved in list of up to {} values".format(len(intervalListFiltered), config.HIST_SIZE))
246     print("Sum of intervals between Geiger counts : {} s".format(totalInterval/1E6))
247
248 intervalmsfromCPM = 1000/(mean_CPM/60)
249 minIntervalms = min(intervalListFiltered)/1000
250 maxIntervalms = max(intervalListFiltered)/1000
251 # Choose bin size depending on the Geiger data (minimum and maximum values) and
# the number of bins
252 binPossibleIntervalsms = (1, 2, 5, 10, 20, 50, 100, 200, 500)
253 binIndex = 0
254 while (binPossibleIntervalsms[binIndex] < minIntervalms) and (binIndex < len(binPossibleIntervalsms) - 1):
255     binIndex += 1
256 while (binPossibleIntervalsms[binIndex]*config.BIN_LIST_SIZE < maxIntervalms) and
        (binIndex < len(binPossibleIntervalsms) - 1):
257     binIndex += 1
258 binsList = [0]*config.BIN_LIST_SIZE
259 for v in intervalListFiltered:
260     i = int(v/(1000*binPossibleIntervalsms[binIndex]))
261     if i < len(binsList):
262         binsList[i] += 1
263
264 if flagDebug:
265     print("CPM = {}".format(mean_CPM))
266     print("CPS (CPM / 60) = {}".format(mean_CPM/60))
267     print("Mean interval from CPM = {} ms".format(intervalmsfromCPM))
268     print("Minimum of intervals = {} ms".format(minIntervalms))
269     print("Maximum of intervals = {} ms".format(maxIntervalms))
270     print("Mean of intervals = {} ms".format(statistics.mean(intervalListFiltered)/1000))
271     print("Harmonic mean of intervals = {} ms".format(statistics.harmonic_mean(
            intervalListFiltered)/1000))
272     print("Median of intervals = {} ms".format(statistics.median(
            intervalListFiltered)/1000))
273     print("Standart deviation of intervals = {} ms".format(statistics.stdev(
            intervalListFiltered)/1000))
274     print("List of intervals between Geiger counts (us) : {}".format(
            intervalListFiltered))
275     print("List of {} bins with {} ms each : {}".format(config.BIN_LIST_SIZE,
            binPossibleIntervalsms[binIndex], binsList))
276
277 gc.collect()
278
279 if flagDebug:
280     print('Free RAM after Geiger counts and histogram calculation : {:.3f} KB'.
           format(gc.mem_free()/1024))

```

```

281     print("CPM and Histogram Calculation time: {:.3 f} ms".format(chrono.read() *1000))
282
283 # ===== Mounting Package =====
284 if flagDebug:
285     print("----- Mounting Package -----")
286     chrono.reset()
287
288 pycom.rgbled(config.SEND_COLOUR)
289
290 ch = struct.pack("B", 1)      # Channel 01
291 #ch2 = struct.pack("B", 2)      # Channel 02
292 ch3 = struct.pack("B", 3)      # Channel 03
293 ch4 = struct.pack("B", 4)      # Channel 04
294 ch5 = struct.pack("B", 5)      # Channel 05
295 ch6 = struct.pack("B", 6)      # Channel 06
296 ch7 = struct.pack("B", 7)      # Channel 07
297 ch8 = struct.pack("B", 8)      # Channel 08
298 ch9 = struct.pack("B", 9)      # Channel 09
299 lpp = struct.pack("B", 101)    # Using iluminance sensor packet - 2 bytes (for graph)
300 #lpp2 = struct.pack("B", 101) # Using iluminance sensor packet - 2 bytes (for value)
301 lpp3 = struct.pack("B", 103) # Using Temperature sensor packet - 2 bytes (for value)
302 lpp4 = struct.pack("B", 104) # Using Humidity sensor packet - 1 bytes (for value)
303 lpp5 = struct.pack("B", 115) # Using Barometer sensor packet - 2 bytes (for value)
304 lpp6 = struct.pack("B", 101) # Using iluminance sensor packet for battery - 2 bytes
305 lpp7 = struct.pack("B", 101) # Using iluminance sensor packet for Packet Counter - only 2 bytes
306 lpp8 = struct.pack("B", 1)    # Using Digital Output for downlink messages
307 lpp9 = struct.pack("B", 0)    # Using Digital Input for Wake reason code
308
309 cpm = struct.pack(">H", mean_CPM)           # >H = 2 bytes
310 temp = struct.pack(">h", round(mean_temp*10)) # >h = 2 bytes
311 umid = struct.pack("B", round(mean_humi*2))  # >B = 1 byte
312 press = struct.pack(">H", round(mean_pres/10))
313 bat = struct.pack(">H", round(batt_mV))
314 packetNumber = struct.pack(">H", int(payloadcount))
315 downlink = struct.pack("B", int(0))
316 wake = struct.pack("B", int(wrCode))
317
318 # Verify if values are the same and create a variable size payload
319 if round(mean_temp*10) == pycom.nvs_get('temperature'):    # Verify if the
320     temperature value is equal up to decimal
321     if flagDebug:
322         print('Temperature value {:.1f} C repeated, so not included in the
323               payload'.format(round(mean_temp*10)/10))
323     ch3 = ''
324     lpp3 = ''
325     temp = ''
326     pycom.nvs_set('temperature', round(mean_temp*10))
327
328 if round(mean_humi*2) == pycom.nvs_get('humidity'):          # Verify if the

```

```

    humidity value is equal up to 0.5
329 if flagDebug:
330     print('Humidity value {:.1f}% repeated, so not included in the payload'.
            format(round(mean_humi*2)/2))
331 ch4 = ''
332 lpp4 = ''
333 umid = ''
334 else:
335     pycom.nvs_set('humidity', round(mean_humi*2))
336
337 if round(mean_pres/10) == pycom.nvs_get('pressure'):    # Verify if the pressure
            value is equal up to centesimal
338     if flagDebug:
339         print('Pressure value {:.1f} hPa repeated, so not included in the
            payload'.format(round(mean_pres/10)/10))
340     ch5 = ''
341     lpp5 = ''
342     press = ''
343 else:
344     pycom.nvs_set('pressure', round(mean_pres/10))
345
346 if round(batt_mV/10) == pycom.nvs_get('battery'):    # Verify if the voltage value
            is equal up to tens of mV
347     if flagDebug:
348         print('Battery voltage value {} mV repeated, so not included in the
            payload'.format(round(batt_mV/10)*10))
349     ch6 = ''
350     lpp6 = ''
351     bat = ''
352 else:
353     pycom.nvs_set('battery', round(batt_mV/10))
354
355 pkt = ch + lpp + cpm + ch3 + lpp3 + temp + ch4 + lpp4 + umid + ch5 + lpp5 + press
            + ch6 + lpp6 + bat + ch7 + lpp7 + packetNumber + ch8 + lpp8 + downlink + ch9
            + lpp9 + wake
356
357 gc.collect()
358
359 airtime_ms = LoRaAirTimeCalc.airtimetheoretical(13 + len(pkt), config.LORA_SF,
            config.LORA_BW, LoRa.CODING_4_5)[0]*1000
360
361 if flagDebug:
362     print('Free RAM after packet mount : {:.3f} KB'.format(gc.mem_free()/1024))
363     print("CPM: {}, BME280: [ {:.1f} C, {:.1f}%, {:.1f} hPa ], Battery: {} mV,
            Packet length: {} bytes, Air time: {} ms, Payload #: {}, Wake/Reset code:
            0x{:02x}'.format(
364             mean_CPM, round(mean_temp*10)/10, round(mean_humi*2)/2, round(
            mean_pres/10)/10, round(batt_mV), len(pkt), airtime_ms,
            payloadcount, wrCode))
365     print("Mounting Package time : {:.3f} ms".format(chrono.read()*1000))
366
367
368 # ===== Write Log Files =====
369 if flagDebug:
370     print("===== Write Log Files =====")
371     chrono.reset()

```

```

372
373 try:
374     sd = SD()
375     os.mount(sd, '/sd')
376     with open('/sd/payload_datalog.csv', 'a') as f:
377         f.write("{}\n".format(payloadcount,
378                               str(binascii.hexlify(pkt))[2:56], mean_CPM, round(mean_temp*10)
379                               /10, round(mean_humi*2)/2,
380                               round(mean_pres/10)/10, round(batt_mV), len(pkt), airtime_ms))
381     with open('/sd/histogram_datalog.csv', 'a') as f:
382         f.write("{}\n".format(payloadcount, binPossibleIntervalsms[
383                               binIndex], binsList))
384     with open('/sd/intervals_datalog.csv', 'a') as f:
385         f.write("{}\n".format(payloadcount, intervalListFiltered))
386
387     os.sync()
388 except:
389     if flagDebug:
390         print('===== Error while writing log files =====')
391
392 gc.collect()
393
394 if flagDebug:
395     print('Free RAM after logging files : {:.3f} KB'.format(gc.mem_free()/1024))
396     print("Write Log Files time: {:.3f} ms".format(chrono.read()*1000))
397
398 # ===== Send/Receive LoRaWan Package
399
400 if flagDebug:
401     print('===== Send/Receive LoRaWan Package =====')
402     print('Sending: {}'.format(pkt))
403     chrono.reset()
404
405 s.send(pkt)
406 time.sleep(5)
407
408 # Downlink messages to access files by FTP
409 code = s.recvfrom(64)
410
411 if flagDebug:
412     print("Downlink Received Code: {}".format(code))
413     print("Send/Receive LoRaWan Package time: {:.3f} ms".format(chrono.read()
414 *1000))
415
416 if (code[1]==99) and (code[0]==b'\x08\x00\x64\xff'):
417     if flagDebug:
418         print('===== Initializing AP Mode for FTP access ... =====')
419     pycom.rgbled(config.FTP_COLOUR)
420     wlan = WLAN()
421     wlan.init(mode=WLAN.STA_AP, ssid='EstGeiger', auth=(WLAN.WPA2, 'password'))
422     server = network.Server()

```

```

423
424     downlink = struct.pack("B", int(1))
425     ftpPkt = ch8 + lpp8 + downlink
426
427     while True:
428         payloadcount += 1
429         pycom.nvs_set('count', payloadcount)
430
431         time.sleep(20)
432         s.send(ftpPkt)
433         time.sleep(5)
434
435         code = s.recvfrom(64)
436         if flagDebug:
437             print("Downlink Received Code: {}".format(code))
438             if (code[1]==99) and (code[0]==b'\x08\x00\x00\xff'):
439                 break
440
441 # ===== Deep Sleep =====
442 if flagDebug:
443     print('===== Deep Sleep =====')
444     chrono.reset()
445
446 lora.nvram_save()
447
448 if flagDebug:
449     print("LoRaWan NVRAM save time: {}s".format(chrono.read()))
450
451 seconds = config.DEEPSLEEP_TIME + (os.urandom(1)[0] & 0x0F)
452 #seconds = 10
453 if flagDebug:
454     print('===== Deep Sleeping for {}s... ====='.format(seconds))
455     time.sleep(1)
456 machine.deepsleep(seconds*1000)

```