UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Cesar Augusto Benitez Barbosa

CRIAÇÃO E VALIDAÇÃO DE HEATMAP LORAWAN USANDO APLICAÇÃO ANDROID

Santa Maria, RS
2018

Cesar Augusto Benitez Barbosa

CRIAÇÃO E VALIDAÇÃO DE HEATMAP LORAWAN USANDO APLICAÇÃO ANDROID

Engenharia de Computação apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Engenheiro de Computação**.

ORIENTADOR: Prof. Carlos Henrique Barriquello

Santa Maria, RS
2018

End. Eletr.: cesar.augusto@ecomp.ufsm.br

Cesar Augusto Benitez Barbosa

CRIAÇÃO E VALIDAÇÃO DE HEATMAP LORAWAN USANDO APLICAÇÃO ANDROID

Engenharia de Computação apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Engenheiro de Computação**.

Aprovado em 20 de julho de 2018:

Carlos Henrique Barriquello, Dr. (UFSM)
(Orientador)

Cesar Augusto Prior, Dr. (UFSM)

José Eduardo Baggio, Dr. (UFSM)

Santa Maria, RS
2018

AGRADECIMENTOS

Agradeço primeiramente a meus pais que através de seus esforços tornou possível a realização deste sonho, auxiliando no que era necessário para que alcance meus objetivos, também a minha irmã por todo o apoio prestado em mais esta etapa de minha vida.

A minha namorada Angélica, por todo companheirismo e paciência, me ajudando a não perder o foco e a determinação de correr atrás de meus sonhos.

A todos meus amigos e familiares por estarem ao meu lado e não me deixarem desistir em momento algum, principalmente aqueles que estavam presentes em meu dia-a-dia da faculdade, prestando apoio não somente em questões relacionadas a faculdade.

A minha segunda família encontrada entre amigos na universidade, dentre eles Augusto Gouvêa, Cleber Ribas, Mauricio Machado, que me deram todo apoio necessário, seja com trabalhos e matérias da faculdade ou nas horas de descontração, tornando a jornada mais leve e mais fácil de se passar.

Ao meu orientador, Carlos Barriquello, pela disposição a esclarecer as dúvidas relacionadas ao desenvolvimento do trabalho e prestar o apoio para que o mesmo fosse desenvolvido da melhor maneira possível.

Ao CELTAB por permitir que o trabalho fosse desenvolvido mais rapidamente, fazendo uso das dependências do laboratório e permitindo os testes nas imediações da Usina Hidroelétrica de Itaipu.

RESUMO

CRIAÇÃO E VALIDAÇÃO DE HEATMAP LORAWAN USANDO APLICAÇÃO ANDROID

AUTOR: Cesar Augusto Benitez Barbosa
ORIENTADOR: Carlos Henrique Barriquello

Com o avanço da tecnologia há um número cada vez maior de dispositivos que podem comunicar-se através do conceito de Internet das Coisas (*IoT*), assim, considerando que esses dispositivos vêm tornando-se cada vez mais comum e aumentando a quantidade de dispositivos conectados, faz-se necessário o estudo de protocolos de comunicação mais eficientes. Diante disso, essa pesquisa tem o objetivo de apresentar um protocolo de comunicação *LoRaWAN* eficiente para comunicações em grandes áreas e que necessitem de solução em termos energéticos. Este estudo será feito por meio do desenvolvimento e testes de um aplicativo para Android, que desempenhará a função de obtenção dos dados referentes a comunicação e possui a capacidade de conectar-se com o modem RisingHF, possibilitando o envio e recebimento de dados/comandos, diretamente do aplicativo

Palavras-chave: LoRa. LoRaWAN. IoT. Protocolo de Comunicação.

ABSTRACT

CREATION AND VALIDATION OF HEATMAP LORAWAN USING ANDROID APP

AUTHOR: Cesar Augusto Benitez Barbosa
ADVISOR: Carlos Henrique Barriquello

With the advancement of technology there are an increasing number of devices that can communicate through the concept of Internet of Things (IoT), a, thus considering that these devices are becoming increasingly common and increasing the amount of connected devices, it is necessary to study a more efficient communications protocols. In view of this fact, the present project proposes the study of the communication protocol LoRaWAN that aims to be an efficient protocol for communications in a large area and that need an energy efficient solution. This study will be done through the development and testing of an Android application, which will perform the function of obtaining data relating to communication, and which has the ability to connect with the RisingHF modem, enabling the sending and receiving of data/commands directly from the application.

Keywords: LoRa. LoRaWAN. IoT. Communication Protocol.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

$LoRa$	Long Range Technology (Tecnologia de Longo Alcance)
$LoRaWAN$	Long Range Wide Area Network
IoT	Internet of Things (Internet das Coisas)
MAC	Media Access Control (controle de Acesso de Mídia)
$LPWA$	Low Power, Wide Area (Baixo Consumo, Longo Alcance)
ISM	Industrial, Scientific and Medical (Industrial, Científico e Médico)
OSI	Open System Interconnection (Interconexão de Sistemas Aberta)
$MQTT$	Message Queuing Telemetry Transport
$RSSI$	Received Signal Strength Indicator (Indicador de Força de Sinal Recebido)

SUMÁRIO

1 INTRODUÇÃO

A tecnologia está avançando rapidamente na direção de se ter todos dispositivos possíveis comunicando-se entre si, seja para realizar uma melhor captação de dados, automações ou até mesmo gerar estatísticas em tempo real. Em razão disso, surge um um novo desafio: encontrar um protocolo de comunicação capaz de realizar esta função com o maior alcance possível e que consuma a menor quantidade de energia para gerá-lo.

Tendo como base este desafio e fazendo uso da tecnologia *LoRa* (*Long Range*), foi criado o protocolo de comunicação *LoRaWAN* (*Long Range Wide Area Network*), que é um protocolo do tipo *MAC* (*Media Access Control*) desenvolvido para ter um grande alcance e voltado para que dispositivos de baixo consumo energético possam se comunicar com aplicações conectadas à internet por meio de uma conexão sem fio de longo alcance.

O protocolo requer um *gateway*, uma aplicação vinculada ao servidor e um dispositivo que tenha a capacidade de estabelecer a conexão com o *gateway*. Deste modo, para fins desta pesquisa, será utilizado o *gateway* localizado sobre o prédio da reitoria da Universidade Federal de Santa Maria - UFSM, na qual possibilitará a obtenção dos dados por meio dos testes pré-estabelecidos e avaliação da qualidade e estabilidade da comunicação para o uso nas imediações da UFSM e o *gatewway* que está localizado sobre o Edifício das Águas, localizado dentro do Parque Tecnológico de Itaipu, que permitirá a realização dos testes propostos nas imediações da Usina Hidroelétrica de Itaipu.

Para análise do protocolo, será utilizado como base o aplicativo *TTN Mapper*, desenvolvido pela *The Things Network*, que terá novas funções implementadas a ele, visando a comunicação do aplicativo com o modem *RisingHF* (modelo *RHF3M076*), possibilitando desse modo o envio e recebimento de comandos e dados do aplicativo Android para o modem, utilizando-se a conexão via protocolo Serial.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção visa revisar os fundamentos usados como base para a realização do trabalho e justificar a necessidade do mesmo. Desta forma, serão abordados os conceitos das tecnologias utilizadas, fatores relevantes a comunicação e necessidade da realização de testes de qualidade e estabilidade.

2.1 LORAWAN

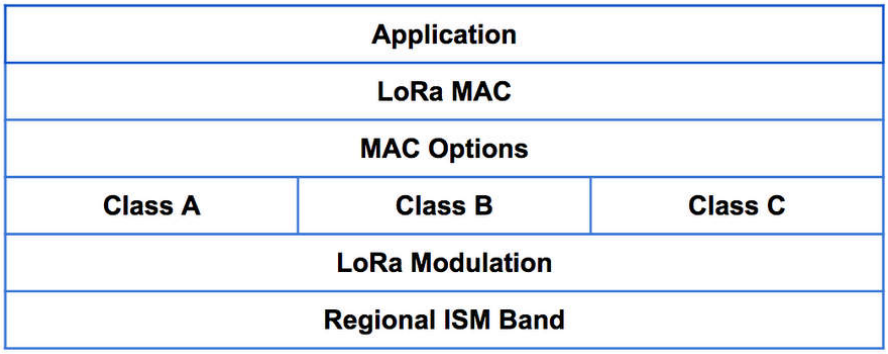
Segundo LORA ALLIANCE (2018) a especificação do *LoRaWAN* é um protocolo de rede *LPWA* (*Low Power - Wide Area*) desenvolvido para conectar 'coisas' operadas por bateria a internet em uma rede regional, nacional ou global, e mira as necessidades chaves do *IoT* (*Internet of Things*) como comunicação bi-direcional, segurança ponta-a-ponta (*end-to-end*), mobilidade e serviços de localização.

Segundo THE THINGS NETWORK (2018) *LoRaWAN* é um protocolo *MAC* para redes de grande alcance. É desenvolvida para possibilitar dispositivos de baixo consumo a se conectar com aplicações conectadas a internet através de uma conexão sem fio de longo alcance. *LoRaWAN* pode ser mapeada pela segunda e terceira camada do modelo *OSI* (*Open System Interconnection*). É implementada em cima do *LoRa* ou modulação *FSK* em rádio frequências *ISM*. Os protocolos *LoRaWAN* são definidos pela *LoRa Alliance* e formalizados pelas *Especificaçoes LoRaWAN*, (*LoRaWAN Specification*), que podem ser obtidas através do site da *LoRa Alliance*.

2.1.1 Topologia

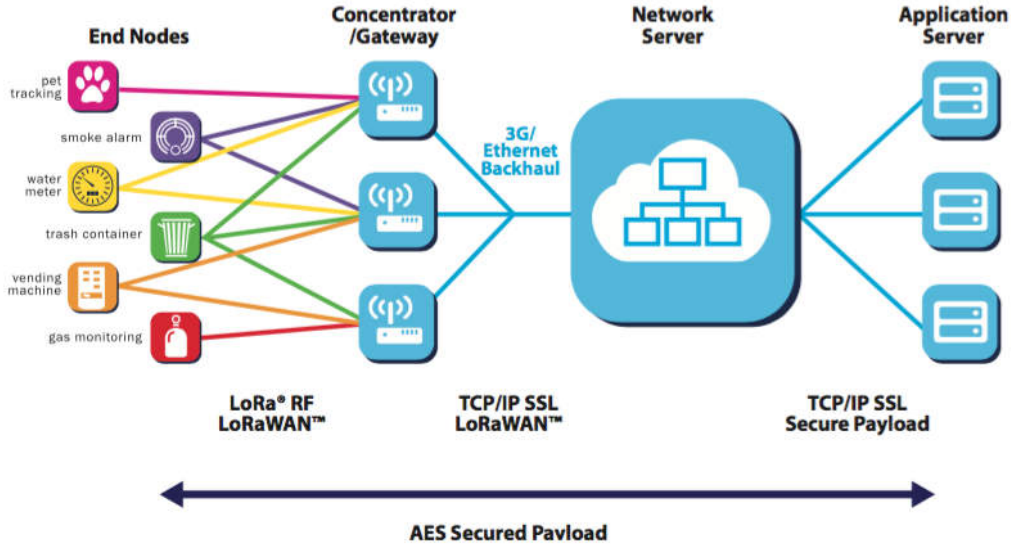
A arquitetura de rede *LoRaWAN* foi feita seguindo a topologia de rede em estrela, (*star-of-stars*), na qual os *gateways* retransmitem as mensagens entre os nós, (*nodes*), e um servidor de rede central. Os *gateways* são conectados ao servidor via conexão padrão IP e agem como um *bridge*, simplesmente convertendo os pacotes *RF* em pacotes IP e vice-versa. A comunicação sem fio tira vantagem das características de longo alcance da camada física do *LoRa*, possibilitando um acesso de um único *hop* entre um nó e um ou vários *gateways*. Todos modos tem capacidade de comunicação bi-direcional, e tem suporte para grupos de endereçamento *multicast* para fazer um uso eficiente do espectro durante tarefas como atualizações de *Firmware Over-The-Air (FOTA)* ou outra distribuição em massa de mensagens (LORA ALLIANCE, 2018).

Figura 2.1 – Arquitetura Protocolo LoRaWAN



Fonte: Imagem retirada do site: 3GLTEINFO (2017)

Figura 2.2 – Arquitetura de Rede Protocolo LoRaWAN



Fonte: Imagem retirada do site: 3GLTEINFO (2017)

2.1.2 Segurança

Qualquer tecnologia de comunicação que lida com muitos dispositivos conectados necessita de uma segurança robusta *end-to-end*. A tecnologia *LoRa* alcança isso implementando segurança em duas camadas diferentes (3GLTEINFO, 2017).

- Camada de Rede
- Camada de Aplicação

A segurança de rede garante a autenticidade do *Nó* na rede e a segurança de aplicação garante que o operador não tenha acesso aos dados de aplicação do usuário final (3GLTEINFO, 2017).

Segundo 3GLTEINFO (2017), a tecnologia *LoRa* usa chaves de segurança *AES* (*Advanced Encryption Standard*). Para alcançar o nivel de segurança requerido para redes *LoRa* várias camadas de encriptação foram empregadas:

- Chave de Rede Única (EUI64) garante segurança a camada de Rede
- Chave de Aplicação Única (EUI64) garante segurança de ponta-a-ponta, (*end-to-end*), a camada de aplicação
- Chave de Dispositivo Única (EUI128)

2.1.3 Nós

As especificaçõs do *LoRaWAN* definem três tipos de dispositivos. Todos os dispositivos *LoRaWAN* devem implementar *Classe A*, onde a *Classe B* e *Classe C* são extensões da especificação de dispositivos de *Classe A* (THE THINGS NETWORK, 2018).

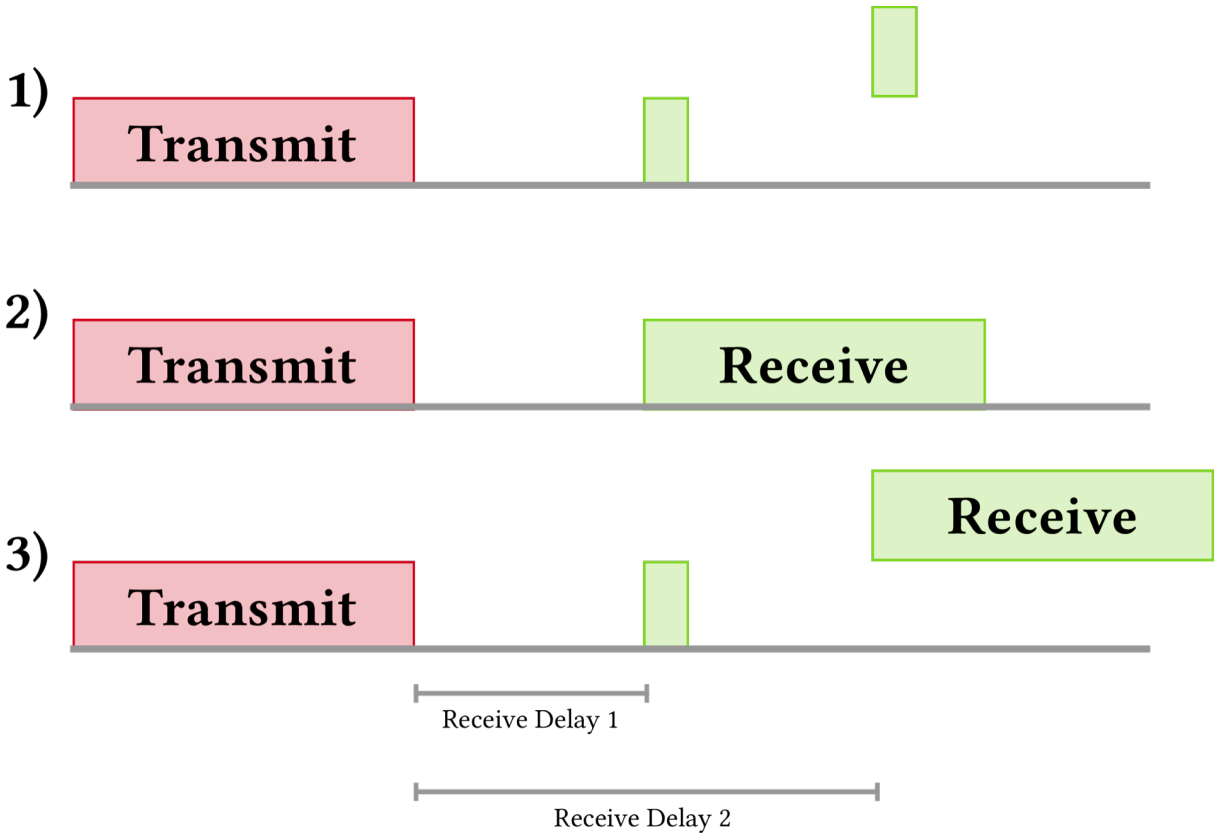
2.1.3.1 Classe A

Dispositivos suportam comunicação bi-direcional entre um dispositivo e um *gateway*. Mensagens de *uplink* (do dispositivo para o servidor) podem ser enviadas a qualquer tempo. O dispositivo então abre duas janelas de recepção em tempos especificados (1s e 2s) após a transmissão de *uplink*. Se o servidor não responder em nenhuma destas janelas (situação 1 da figura 2.3), a próxima oportunidade será após a próxima mensagem de *uplink* do dispositivo. O servidor pode responder ambas na primeira janela de recepção, ou na segunda janela de recepção, porém nunca deve usar ambas as janelas (THE THINGS NETWORK, 2018).

2.1.3.2 Classe B

Dispositivos extendem a *Classe A* adicionando janelas de recepção programadas para mensagens de *downlink* do servidor. Usando sinais de sincronização de tempo transmitidos pelo *gateway*, os dispositivos abrem a janela de recepção periodicamente (THE THINGS NETWORK, 2018).

Figura 2.3 – Janela de Transmissão Classe A.

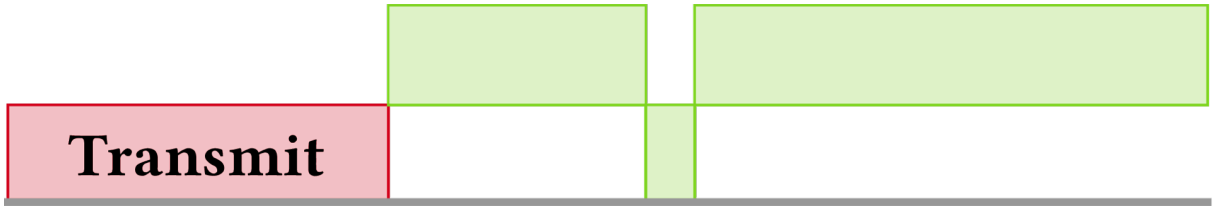


Fonte: Adaptado de THE THINGS NETWORK (2018)

2.1.3.3 Classe C

Dispositivos extendem a *Classe A* mantendo as janelas de recepção abertas a não ser enquanto estão transmitindo, (conforme a *Figura 2.4*). Isso permite uma comunicação de baixa latência mas consome muito mais energia do que dispositivos de *Classe A* (THE THINGS NETWORK, 2018).

Figura 2.4 – Janela de Transmissão Classe C.



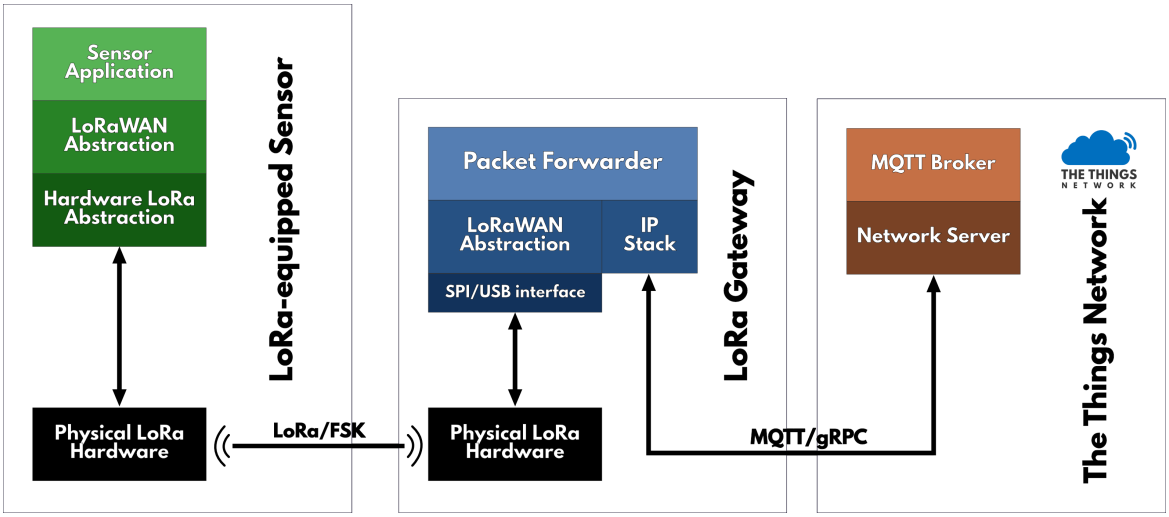
Fonte: Adaptado de THE THINGS NETWORK (2018)

2.2 PACKET FORWARDER

Packet Forwarder é o termo usado para descrever um programa rodando em um *Gateway* que tenha a interação:

- Com o chip *LoRa*, para receber e transmitir pacotes *LoRa*
- Com a rede (internet), para transmitir esses pacotes para as aplicações

Figura 2.5 – Esquemático do Packet Forwarder



Fonte: Imagem retirada do site: THE THINGS NETWORK (2018)

2.3 TTN MAPPER

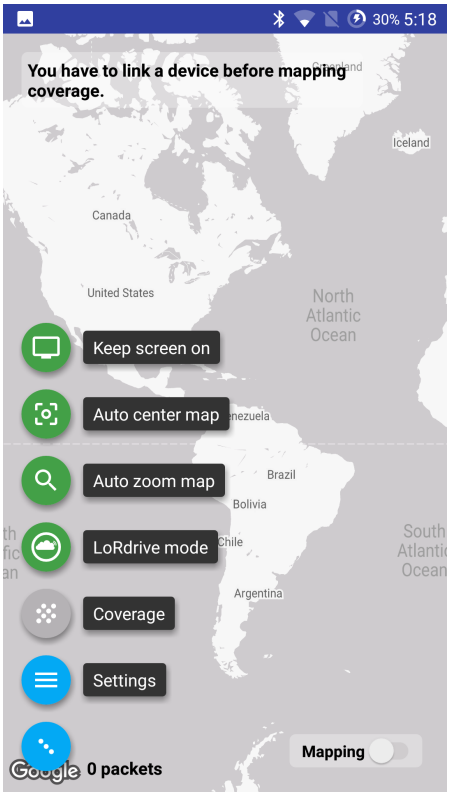
O aplicativo *TTN Mapper* foi escolhido para ser usado como base para o desenvolvimento da aplicação proposta por este trabalho, o mesmo foi desenvolvido pela *The Things Network*, sob a licença *PDDL v1.0*.

O objetivo do *TTN Mapper* é de prover um mapa da cobertura atual dos *gateways* vinculados a *The Things Network*. Contribuidores ao *TTN Mapper* fazem a medição da performance dos *gateways* em sua vizinhança e fazem o upload destas informações para o site do *TTN Mapper* (TIMOTHY SEALY, 2018).

Figura 2.6 – Interface TTN Mapper.

Fonte: Captura de tela do aplicativo.

Figura 2.7 – Menus Interface TTN Mapper.



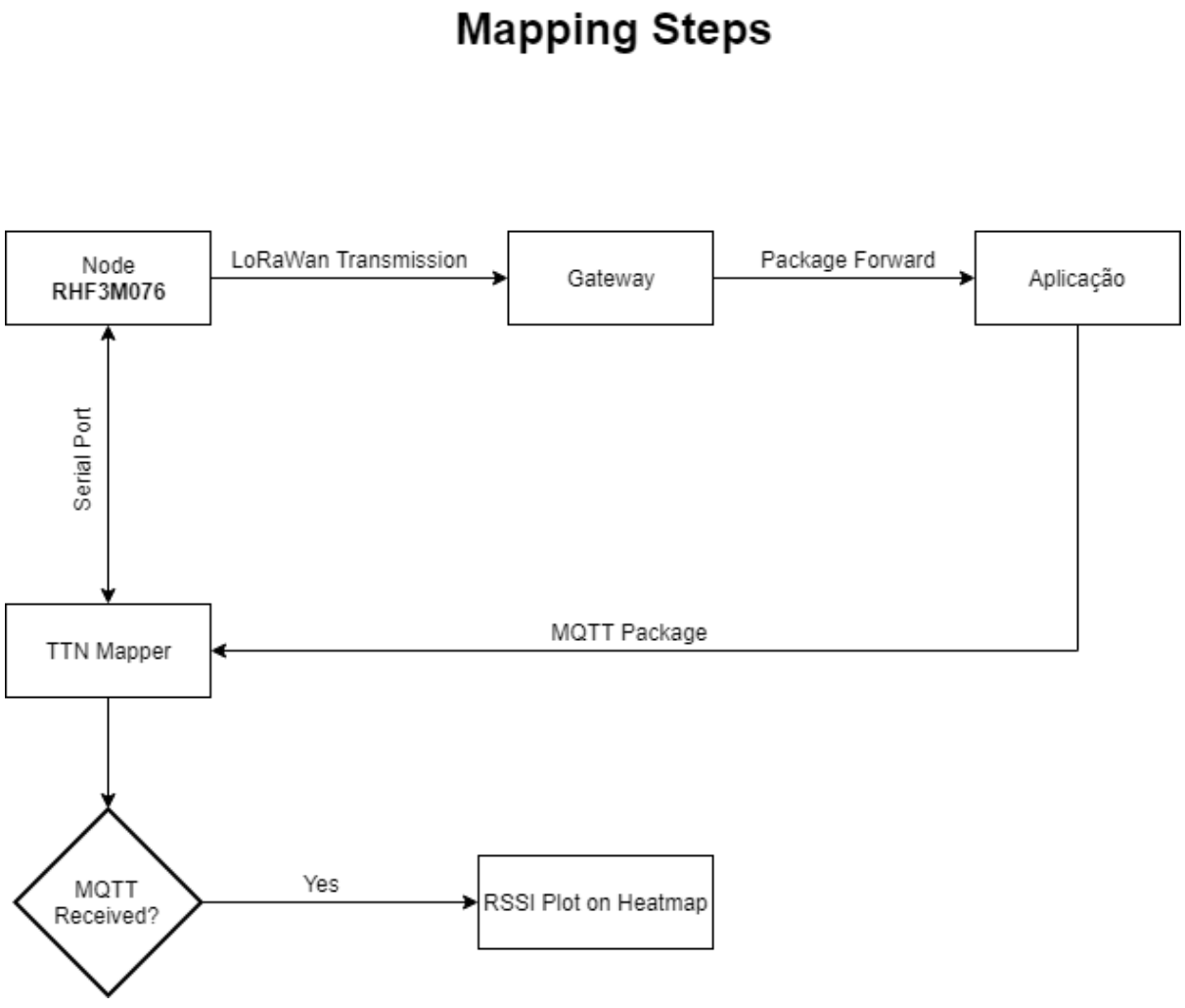
Fonte: Captura de tela do aplicativo.

3 METODOLOGIA

Esta seção tem como objetivo demonstrar a metodologia escolhida para resolver os problemas propostos para este trabalho, como o desenvolvimento e escolhas feitas para que o mesmo possa atender aos requisitos.

O mapeamento será realizado tanto no Campus de Santa Maria da *UFSM* quanto nas intermediações da Usina Hidroelétrica de Itaipu, em Foz do Iguaçu - PR, e deverá seguir o fluxograma da *Figura 3.1* para ser realizado.

Figura 3.1 – Fluxograma Para Mapeamento

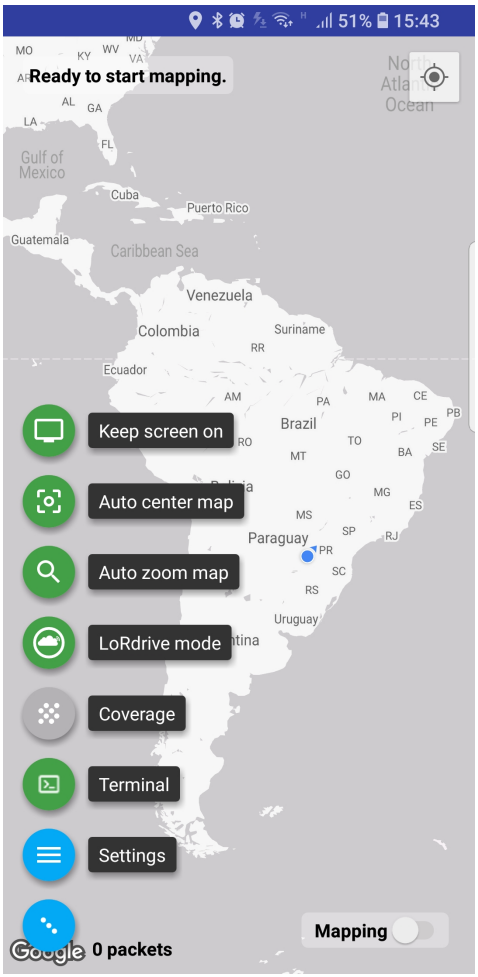


Fonte: UML gerado.

3.1 TERMINAL

Tendo como um dos objetivos o aperfeiçoamento de um aplicativo já existente, adicionando funcionalidades ao mesmo com a finalidade de torná-lo um aplicativo capaz de realizar as atividades requeridas para o andamento do projeto, foi desenvolvida uma *activity de Terminal* e adicionado ao aplicativo, conforme indica a *Figura 3.2*.

Figura 3.2 – Menus Interface TTN Mapper Com Terminal



Fonte: Captura de tela do aplicativo.

3.1.1 Activity

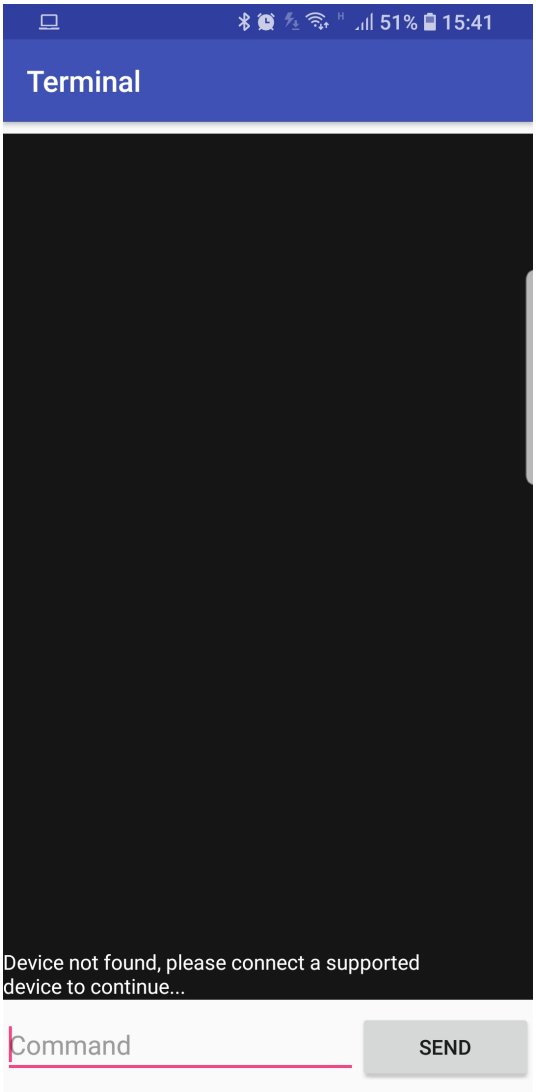
É um componente de aplicativo que fornece uma tela com a qual os usuários podem interagir para fazer algo, como discar um número no telefone, tirar uma foto, enviar um e-mail ou ver um mapa. Cada atividade recebe uma janela que exibe a interface do usuário. Geralmente, a janela preenche a tela, mas pode ser menor que a tela e flutuar sobre outras janelas (GOOGLE DEVELOPERS, 2018).

3.1.2 Activity Terminal

A *activity do Terminal* foi feita com a finalidade de emular um terminal que tenha capacidade de se comunicar com um *nó* através da porta *USB*, fazendo uso de um conector *USB OTG* (*On The Go*), permitindo também o envio de comandos *AT* e outros comandos programados com a inteção de facilitar os testes e o mapeamento.

O terminal foi concebido para que consiga realizar a comunicação com o *nó* escolhido para a realização deste projeto, portanto está programado para aceitar e enviar comandos apenas para *nós* iguais a este, não realizando a conexão de qualquer outro dispositivo não adicionado ao código fonte, porém com a possiblidade de se adicionar mais dispositivos futuramente.

Figura 3.3 – Activity Terminal TTN-Mapper



Fonte: Captura de tela do aplicativo.

3.1.3 Comunicação

Para que a comunicação entre o aplicativo *TTN-Mapper* e o node escolhido fosse possivel fez-se necessário o uso da comunicação *Serial*, sendo usado um cabo *OTG* conectado ao smartphone e ao node.

3.1.3.1 Comunicação Serial

A comunicação *Serial* foi realizada com o auxilio da biblioteca *UsbSerial* desenvolvida por FELHR85 (2018).

Esta biblioteca contém todas as funções necessárias para o correto funcionamento do aplicatvio, facilitando no processo de desenvolvimento do mesmo.

3.1.4 FUNÇÕES IMPLEMENTADAS

Para tornar os testes mais fáceis foram implementadas algumas funções ao código fonte do Terminal, listadas no *Quadro 3.1*.

Quadro 3.1 – Funções Implementadas

Comando:	Função:
cls/clear	Limpa tela do terminal.
cloop	Cria um loop com número de instruções e intervalo definidos pelo usuário, onde é enviado o valor de um contador como mensagem para o gateway.
close/disconnect	Encerra a conexão (desconecta) o nó que estiver conectado.
loop	Cria um loop com função, número de instruções e intervalo definidos pelo usuário.
ping	Faz o teste de conexão do aplicativo com o gateway, enviando 4 pacotes, em seguida gerando uma estatística com o tempo de resposta, quantidade de pacotes recebidos e perdidos durante o teste.
UFSM	Script para configurar o nó junto ao aplicativo.
gpsmap	Captura a posição gps do celular e o codifica, enviando esta posição codificada para o gateway a cada 5 segundos.

Fonte: Funções Implementadas no Código Fonte.

3.1.4.1 Função CLOOP

Após enviar o comando *cloop*, será solicitado que o usuário entre com a quantidade de mensagens que serão enviadas, e por último será solicitado o intervalo, em *ms*, entre um envio e o outro.

Devido ao tempo necessário para que o *nó* consiga enviar com segurança a informação o intervalo mínimo é de *5s*, ou *5000ms*.

Ao término do envio das mensagens será apresentado ao usuário quantos pacotes chegaram ao servidor e foram lidos pelo aplicativo, via *MQTT* (*Message Queuing Telemetry Transport*), gerando assim uma estatística de quantos pacotes enviados foram recebidos pelo aplicativo e posteriormente mapeados com sucesso.

3.1.4.2 Função LOOP

Após entrar com o comando *loop*, será solicitado qual comando o usuário deseja que seja efetuado, seguido da quantidade de vezes que este deve ser executado e por último o intervalo, em *ms*, entre um comando e o outro.

3.1.4.3 Função UFSM

Ao entrar com a função *UFSM*, o usuário estará definindo as configurações do aplicativo para funcionar com a aplicação da UFSM.

3.1.4.4 Função GPSMAP

Ao enviar o comando *gpsmap* o aplicativo passa a coletar os dados de localizações, (Latitude, Longitude, Altitude e Precisão), codificar eles e enviar periodicamente, com intervalo de 5 segundos, via *LoRaWAN*.

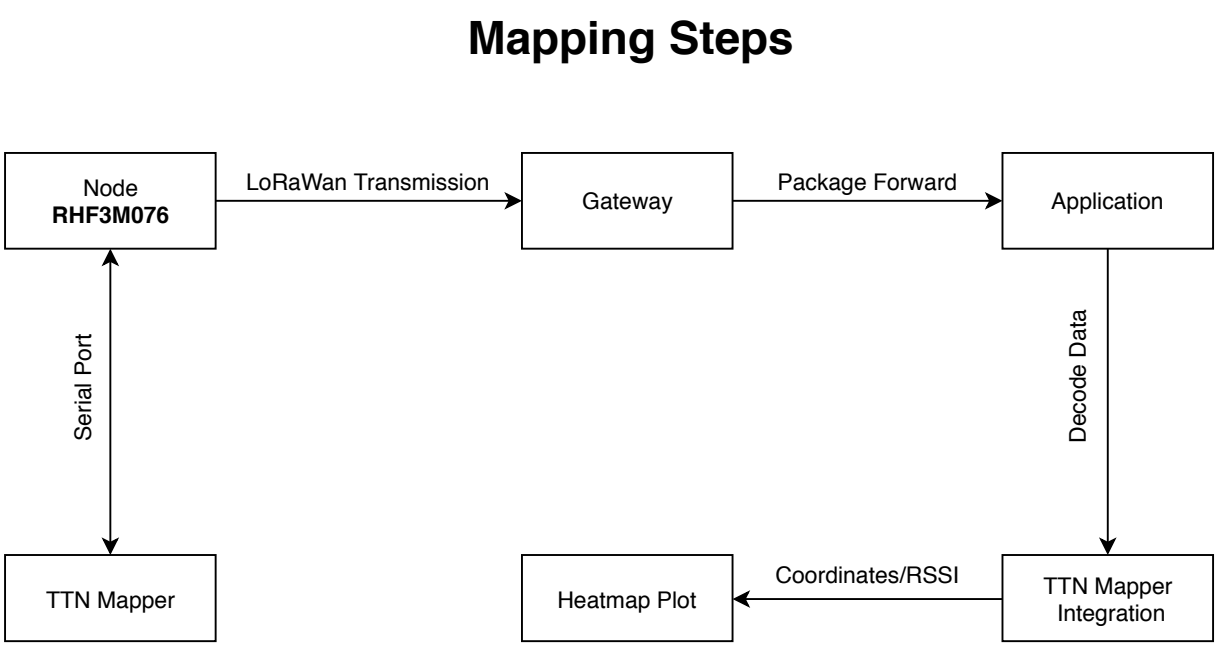
Para que o envio possa ser realizado mantendo a precisão pretendida, de 7 casas decimais, os dados precisam ser codificados, esta codificação estará disponível nos *Anexos* deste documento.

Esta função foi implementada para que possa ser feito o mapeamento mesmo em locais onde não haja a disponibilidade de uma rede de internet móvel de boa qualidade, sendo independente de seu uso diferentemente das demais funções de mapeamento presentes na aplicação. Isto se dá ao fato de o mapeamento não ser feito com a necessidade do recebimento de um pacote *MQTT*.

De acordo com o fluxograma mostrado abaixo, pode-se observar que ao receber o pacote a aplicação o decodifica e a envia para o *TTN Mapper* através da integração do aplicativo ao servidor de aplicação, este por sua vez, ao receber o dado, faz o plot de acordo com o *RSSI* (*Received Signal Strength Indicator*), indicador de força do sinal recebido.

A integração é feita diretamente da página do *The Things Network*, seguindo o fluxograma da *Figura 3.4*, esta se faz após a escolha/criação da aplicação desejada, na seção de integração, em seguida selecionando a integração referente ao *TTN Mapper*.

Figura 3.4 – Fluxograma de Mapeamento - GPSMap



Fonte: UML Gerado.

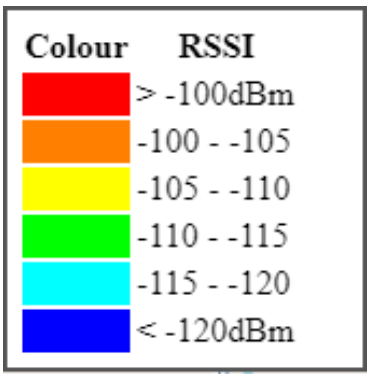
Para uma melhor referência/documentação a respeito da integração da aplicação a mesma pode ser encontrada a partir da documentação feita por TIMOTHY SEALY (2018), presente na seção de documentação do site do *The Things Network*.

3.2 HEATMAP

Segundo OPTIMIZELY (2018) *Heatmap* é uma representação gráfica de dados que usa um sistema de codificação de cores para representar diferentes valores. Os *Heatmaps* são usadas em várias formas de análises.

A leitura do *Heatmap*, ou mapa de calor, se dá analisando a cor a qual a localização está sendo representada e seguindo um código de cores, correspondente a *Figura 3.5*, para que a leitura dos resultados seja feita de forma mais intuitiva e prática.

Figura 3.5 – Legenda Heatmap UFSM



Fonte: Imagem retirada da Legenda do site TTN Mapper.

3.2.1 Experimentos

Para que seja feito o *Heatmap* e a obtenção dos dados foram realizados os seguintes experimentos:

- Mapeamento em Área Aberta
- Mapeamento em Área Fechada

Cada experimento foi realizado sob diferentes condições climáticas, com a finalidade de obter dados que possam ser usados para validar a aplicação e os resultados obtidos.

3.2.2 Mapeamento em Área Aberta

Este teste serve para mostrar a Área de cobertura e a qualidade de conexão para *nós LoRa* instalados em ambientes externos e sob diferentes condições climáticas. Lembrando ainda que apenas o fato de o *nó* estar instalado em ambiente interno não o isenta da ocorrência de enfraquecimentos do sinal, dependendo também se há alguma barreira no caminho do sinal do dispositivo até o *gateway*.

3.2.2.1 Ensolarado

Sob estas condições, dia ensolarado, considerando céu limpo, e em ambiente externo, se espera a obtenção dos melhores resultados em relação a qualidade do sinal, pois neste caso o meio impõe menos interferência ao sinal e a instalação do *nó* em local aberto, fora de prédios e construções, evita que haja um enfraquecimento do sinal devido a refração e reflexão da onda.

3.2.2.2 Chuvoso

Sob estas condições, dia chuvoso e nublado, e levando em consideração a instalação do *nó* em ambiente externo, espera-se a obtenção de resultados que apontem uma pequena piora na qualidade do sinal, podendo até mesmo ser irrelevante, se comparado a uma condição climática melhor, esta perda pode ser atribuida a uma atenuação na onda causada pelas condições climáticas.

3.2.3 Mapeamento em Área Fechada

Este teste serve para mostrar a Área de cobertura e a qualidade de conexão para *nós LoRa* instalados em ambientes internos e sob diferentes condições climáticas, a fim de se verificar se há diferenças, do mesmo modo que no experimento anterior.

4 RESULTADOS

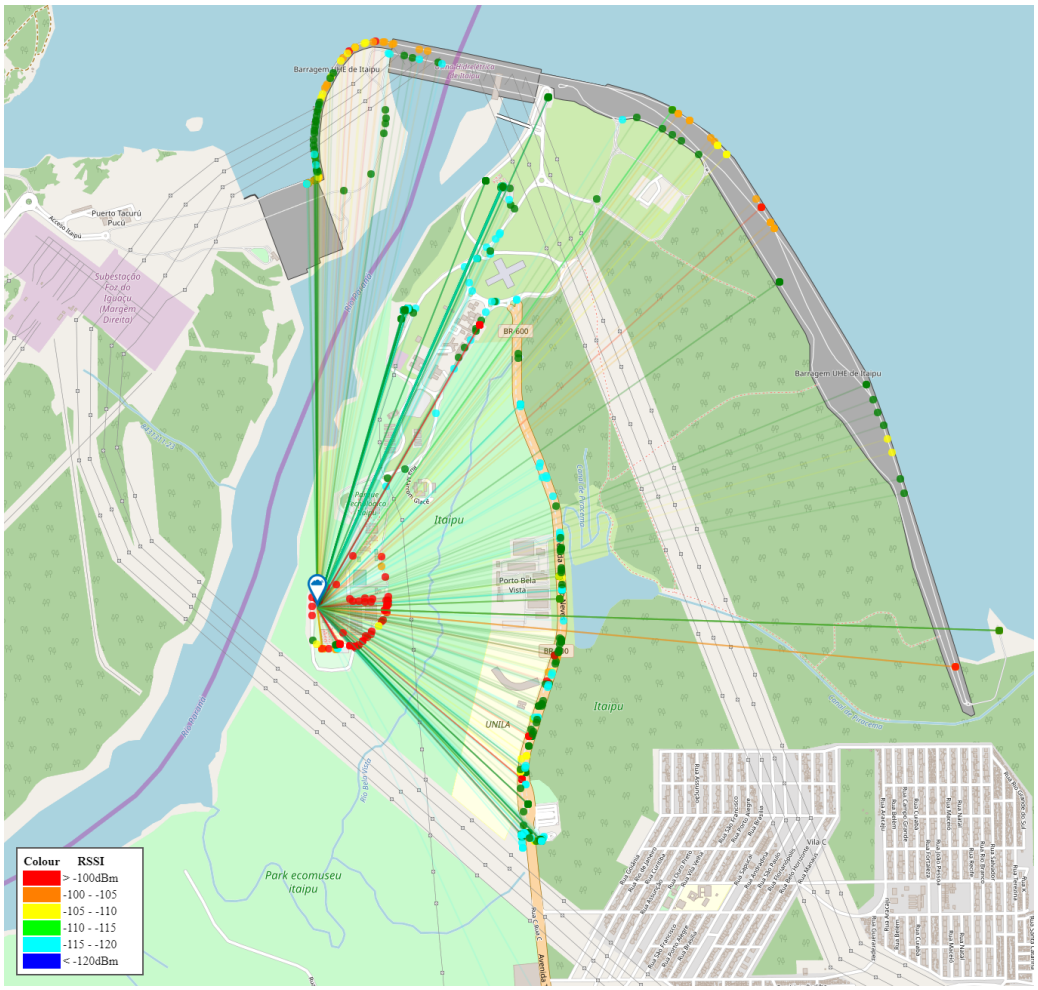
4.1 MAPEAMENTO EM ÁREA ABERTA

Nesta seção serão apresentados os resultados obtidos através do mapeamento em Área Aberta de acordo com a metodologia sugerida neste trabalho.

4.1.1 Caso: Itaipu

4.1.1.1 Mapeamento em Dia Ensolarado

Figura 4.1 – Heatmap Itaipu - Dia Ensolarado



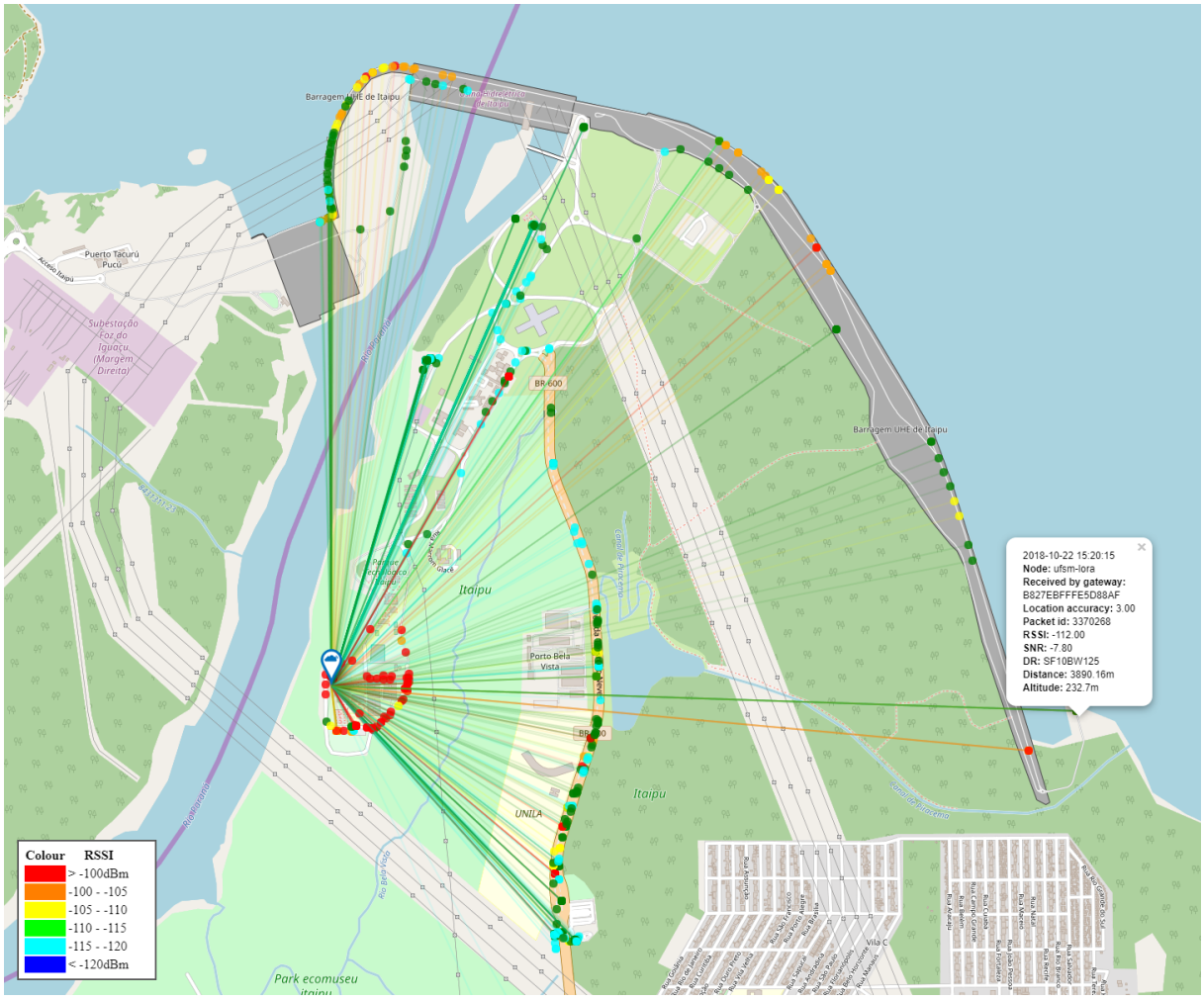
Fonte: Imagem retirada do site TTN Mapper.

A realização do mapeamento nas imediações da Usina Hidroelétrica de Itaipu, foi realizada no dia 22 de Outubro de 2018, gerando o *Heatmap* visto acima.

A partir da *Figura 4.1* podemos observar que alguns dados foram compatíveis com os que eram esperados, demonstrando por exemplo que quanto mais perto do *Gateway* melhor a qualidade e intensidade deveriam ser.

Na *Figura 4.2* podemos observar os dados obtidos do ponto que foi mapeado a maior distância do gateway, considerando o mapeamento feito nesta ocasião.

Figura 4.2 – Heatmap Itaipu - Dia Ensolarado - Maior Distância

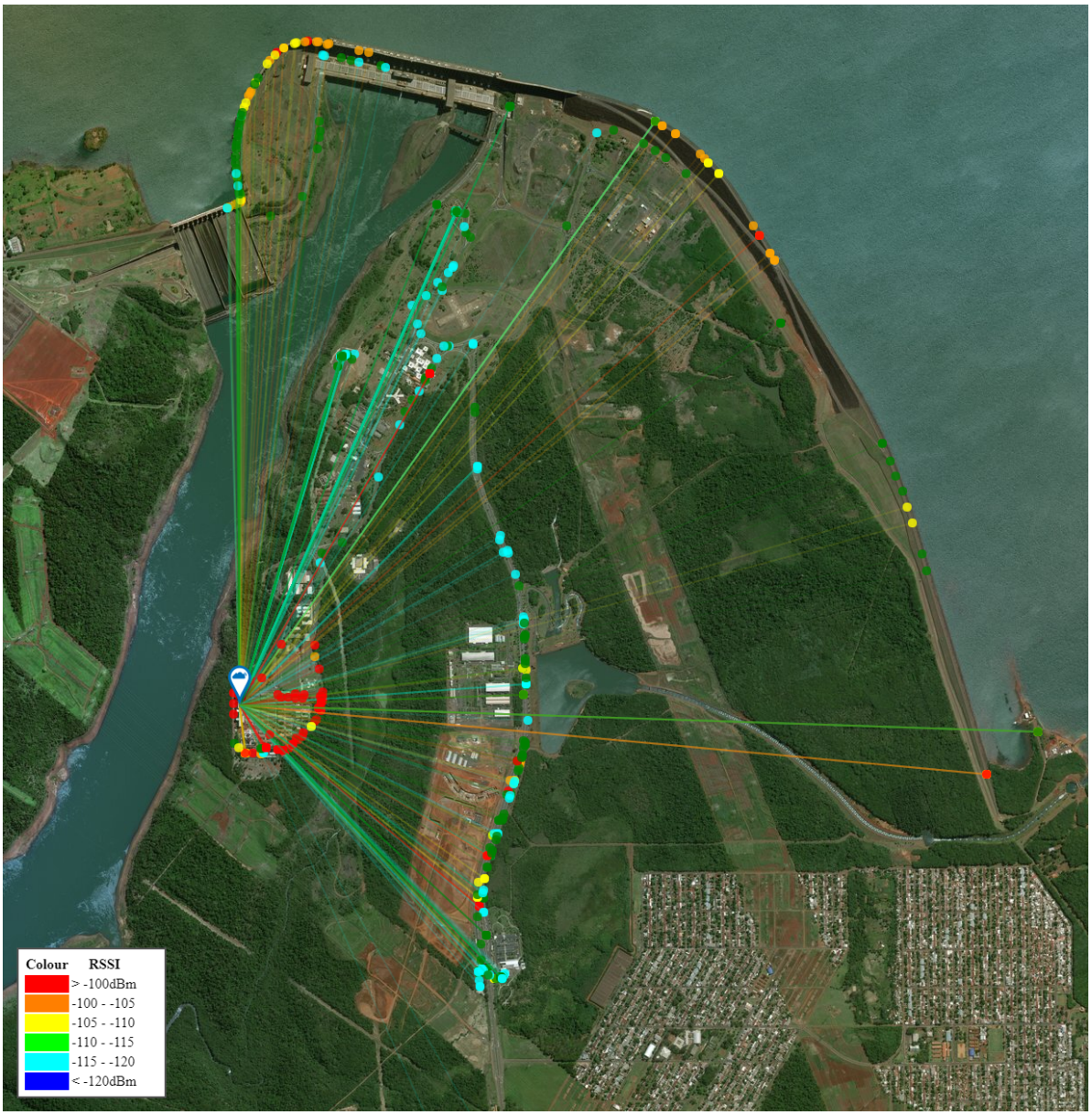


Fonte: Imagem retirada do site TTN Mapper.

Ao analisar a *Figura 4.2* percebemos que há coerência nos dados representados pelo *Heatmap* criado, onde o ponto apresentado demonstra um *RSSI* menor do que os pontos mapeados mais próximos ao *Gateway*.

A fim de demonstrar que há consistência nos dados apresentados, demonstrando que os pontos que apresentaram uma variação considerável com relação a pontos próximos, ou mais próximos ao *Gateway*, tiveram esta alteração em decorrência de interferência na transmissão, por não ter uma linha de visada limpa, ou seja sem obstáculos, esta situação pode ser observada na *Figura 4.3*, onde o *Heatmap* é demonstrado sobre uma visão de satélite da Usina Hidroelétrica de Itaipu.

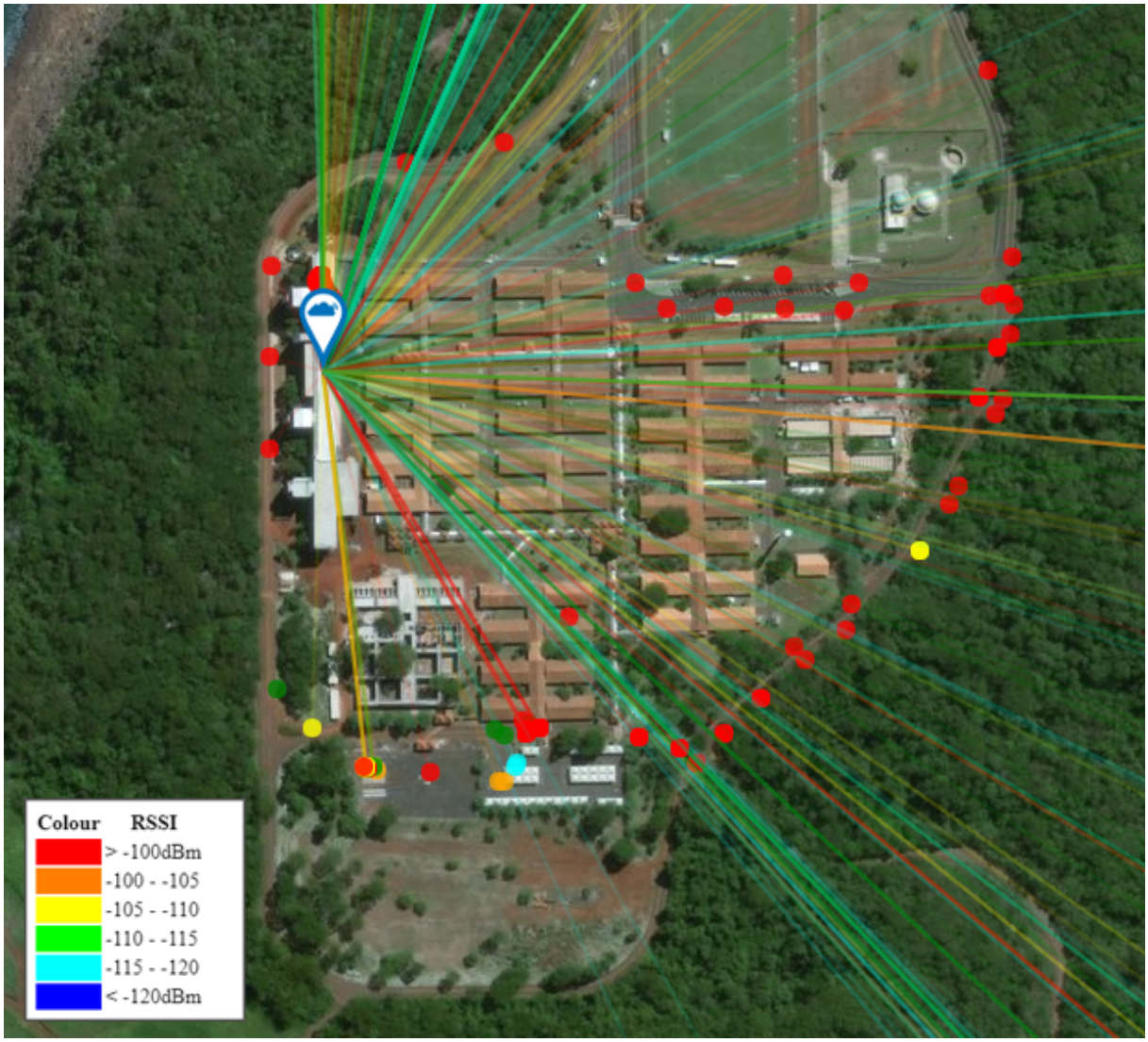
Figura 4.3 – Heatmap Itaipu - Dia Ensolarado - Satélite



Fonte: Imagem retirada do site TTN Mapper.

Considerando apenas a área do *Parque Tecnológico de Itaipu - PTI*, *Figura 4.4*, a fim de facilitar a identificação dos pontos, podemos então validar os dados plotados no mapa, onde a obstrução da linha de visada, seja pelos prédios ou árvores, ocasionou um menor valor de *RSSI*, ou seja, o sinal teve uma perda de potência.

Figura 4.4 – Heatmap PTI - Dia Ensolarado - Satélite



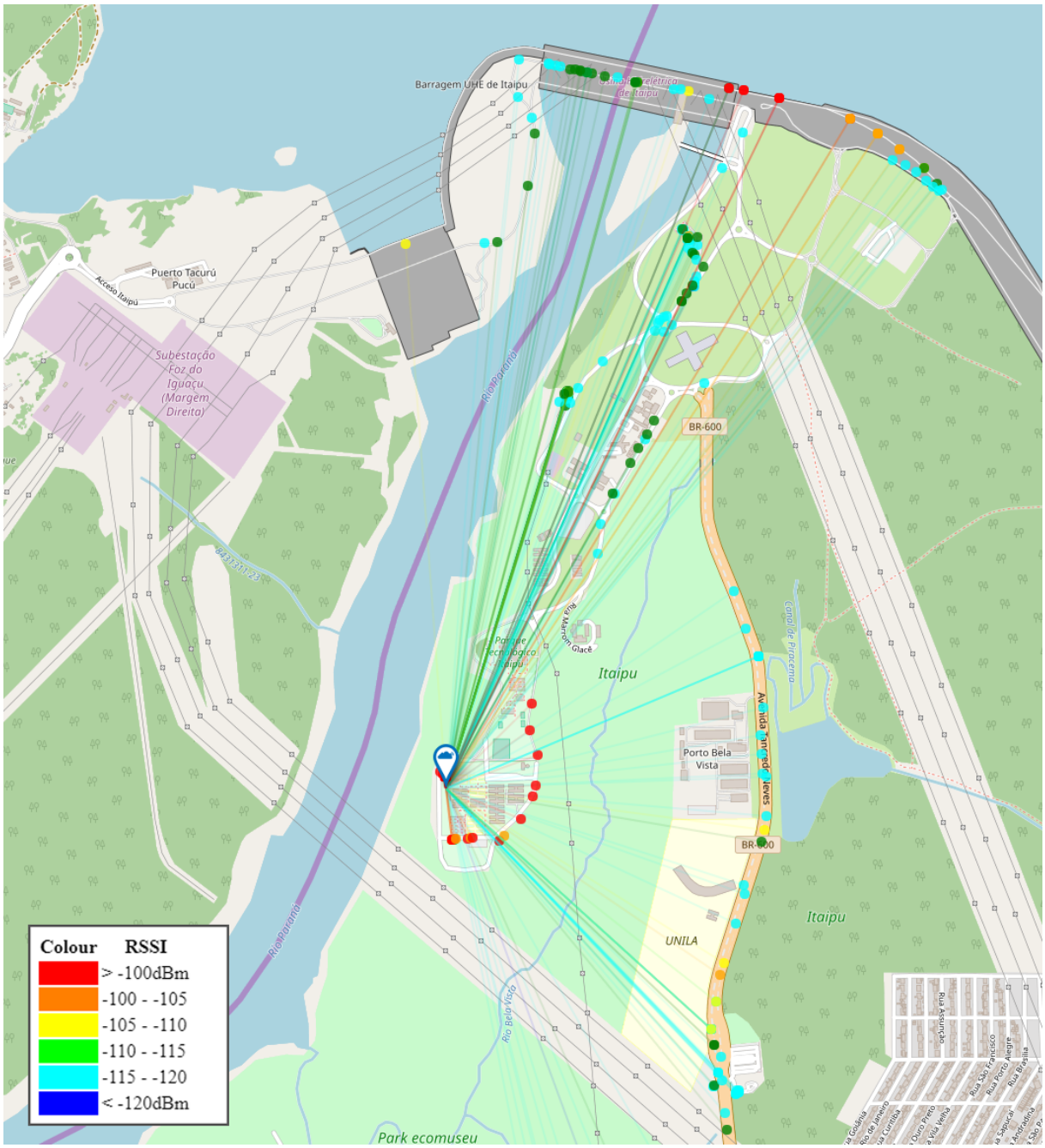
Fonte: Imagem retirada do site TTN Mapper.

Pode-se osbervar também que as árvores acabam ocasionando uma maior perda na força de sinal, pois onde há apenas os prédios e construções obstruindo a linha de visada o *RSSI* se mostrou de uma grandeza maior.

4.1.1.2 Mapeamento em Dia Chuvoso

A realização do mapeamento nas imediações da Usina Hidroelétrica de Itaipu, foi realizada no dia 26 de Outubro de 2018, gerando o *Heatmap* da *Figura 4.5*.

Figura 4.5 – Heatmap Itaipu - Dia Chuvoso

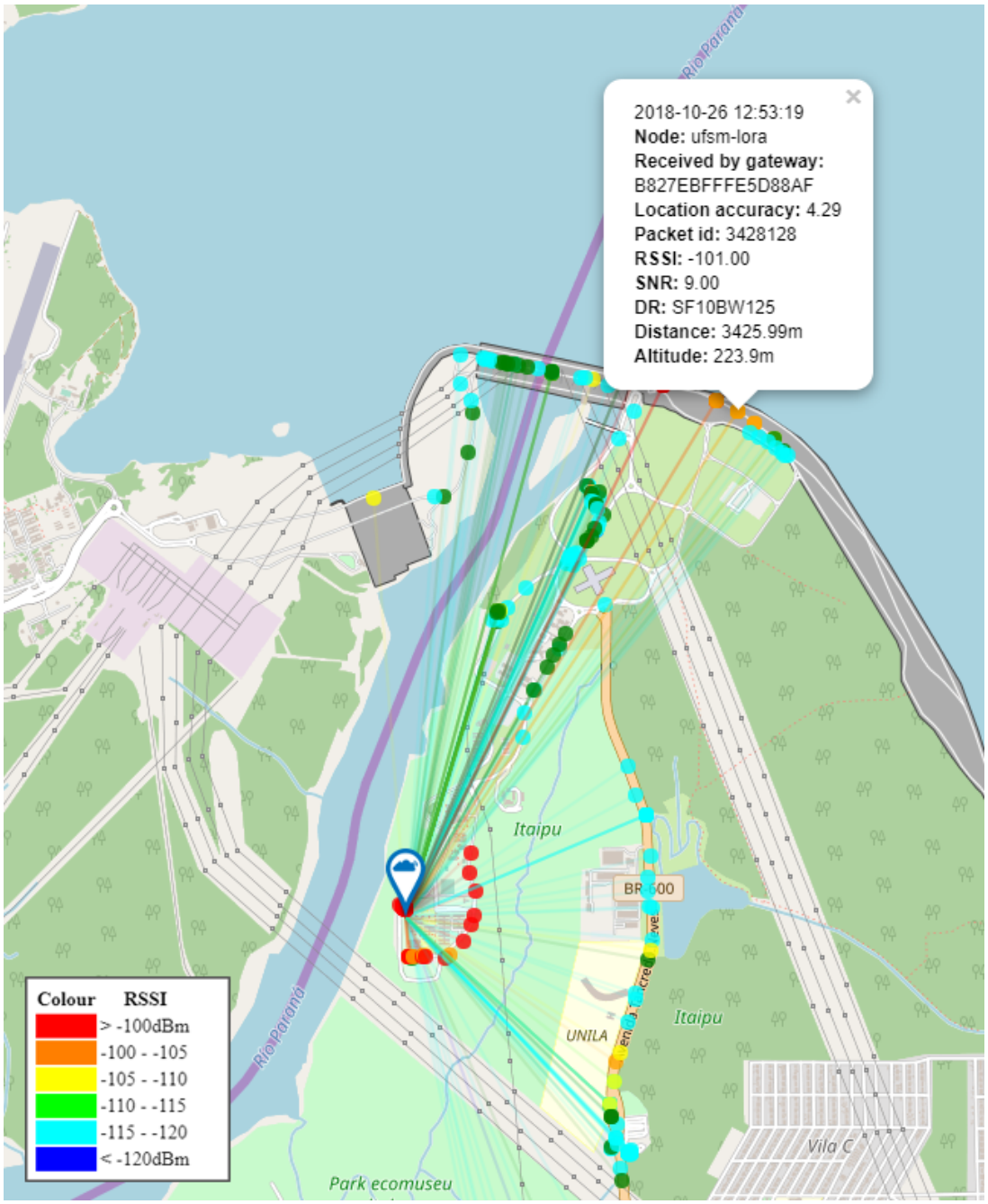


Fonte: Imagem retirada do site TTN Mapper.

A partir da *Figura 4.5* podemos observar que alguns dados foram compatíveis com os que eram esperados, demonstrando por exemplo que quanto mais perto do *Gateway* melhor a qualidade e intensidade deveriam ser.

Na *Figura 4.6* podemos observar os dados obtidos do ponto que foi mapeado a maior distância do gateway, considerando o mapeamento feito nesta ocasião.

Figura 4.6 – Heatmap Itaipu - Dia Chuvoso - Maior Distância

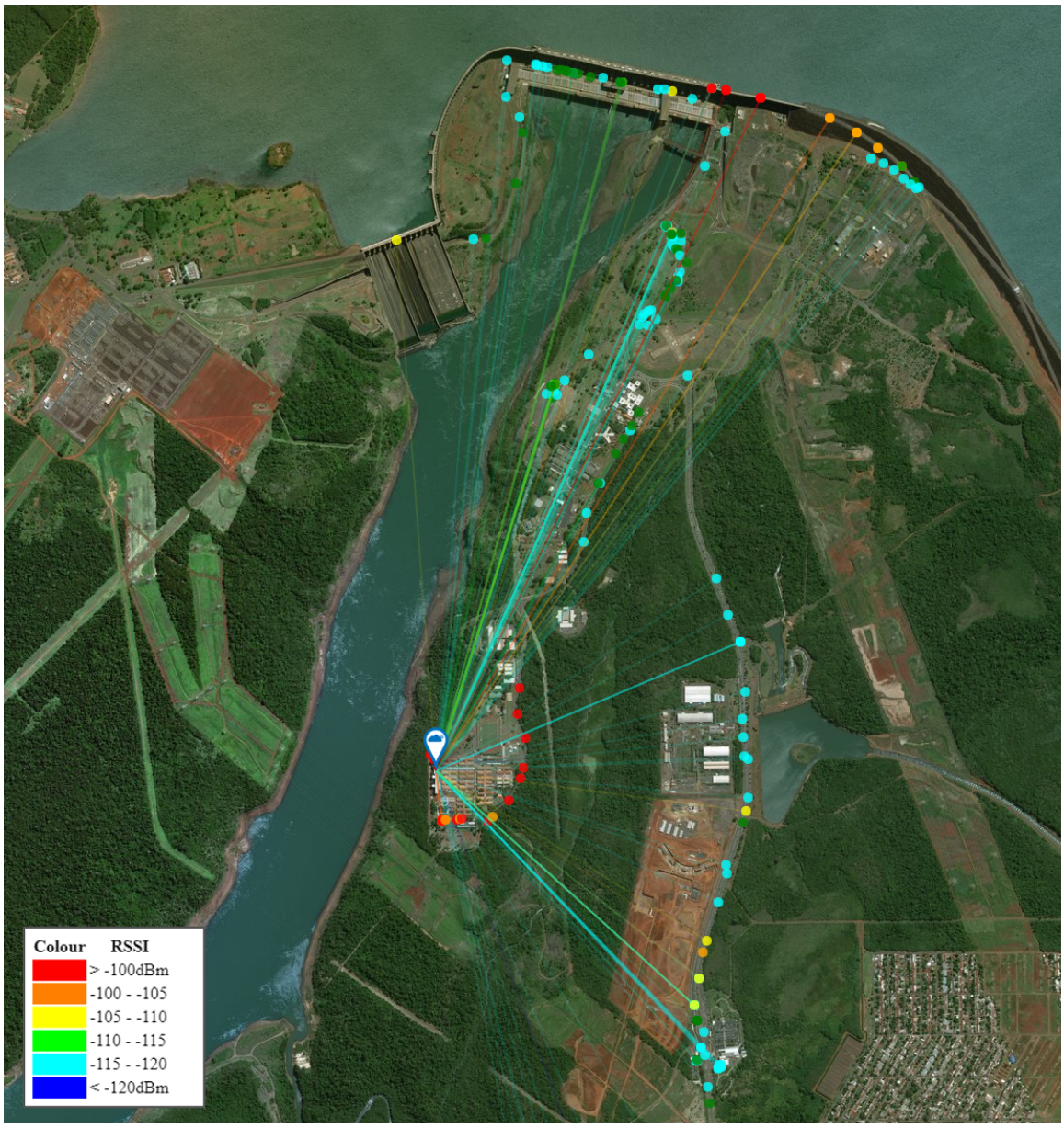


Fonte: Imagem retirada do site TTN Mapper.

Ao analisar a *Figura 4.6* percebemos que há coerência nos dados representados pelo *Heatmap* criado, onde o ponto apresentado demonstra um *RSSI* menor do que os pontos mapeados mais próximos ao *Gateway*.

A fim de demonstrar que há consistência nos dados apresentados, demonstrando que os pontos que apresentaram uma variação considerável com relação a pontos próximos, ou mais próximos ao *Gateway*, tiveram esta alteração em decorrência de interferência na transmissão, por não ter uma linha de visada limpa, ou seja sem obstáculos, esta situação pode ser observada na *Figura 4.7*, onde o *Heatmap* é demonstrado sobre uma visão de satélite da Usina Hidroelétrica de Itaipu.

Figura 4.7 – Heatmap Itaipu - Dia Chuvoso - Satélite



Fonte: Imagem retirada do site TTN Mapper.

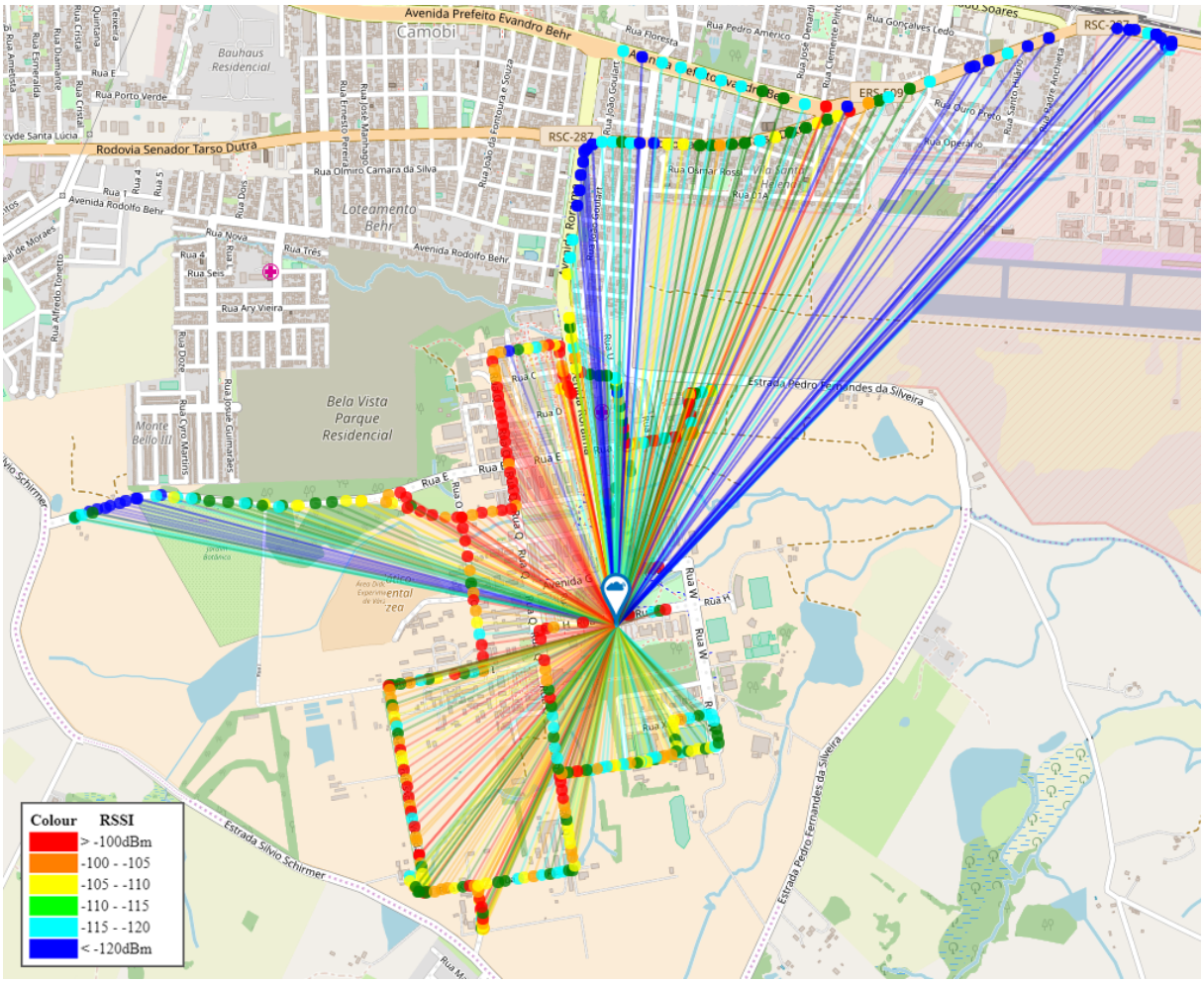
4.1.2 Caso: UFSM

Os resultados do caso do Campus de Santa Maria da *UFSM* serão menos abrangentes em relação aos testes propostos, em questão de quantidade de testes feitos e resultados obtidos, porém será apresentado a fim de validação aos dados obtidos no caso da Usina Hidroelétrica de Itaipu.

4.1.2.1 Teste em Dia Ensolarado

A realização do mapeamento nas imediações do Campus de Santa Maria da *UFSM*, foi realizada no dia 2 de Agosto de 2018, gerando o *Heatmap* da *Figura 4.8*.

Figura 4.8 – Heatmap UFSM - Dia Ensolarado

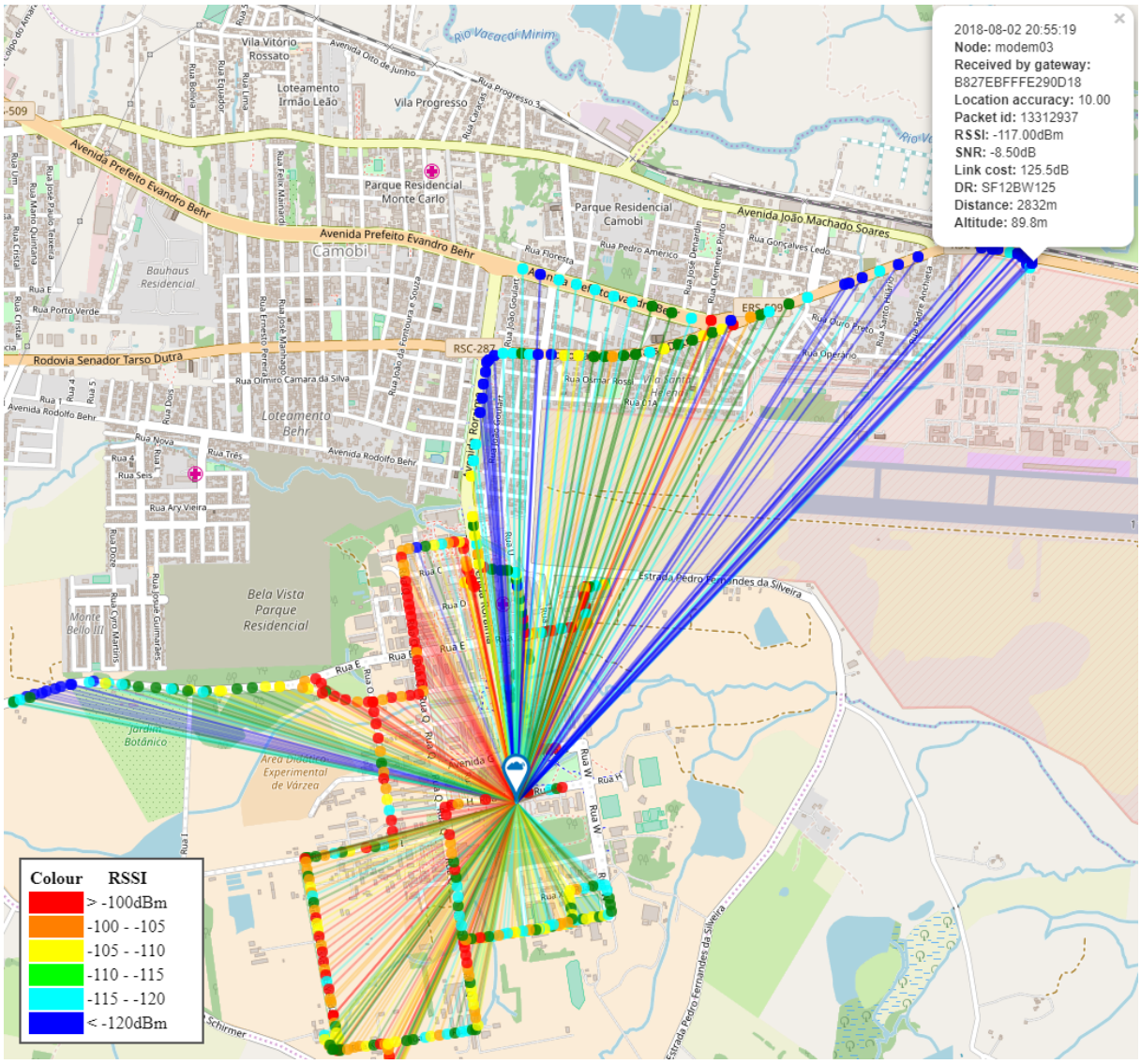


Fonte: Imagem retirada do site TTN Mapper.

A partir da *Figura 4.8* podemos observar que alguns dados foram compatíveis com os que eram esperados, demonstrando por exemplo que quanto mais perto do *Gateway* melhor a qualidade e intensidade deveriam ser.

Na *Figura 4.9* podemos observar os dados obtidos do ponto que foi mapeado a maior distância do gateway, considerando o mapeamento feito nesta ocasião.

Figura 4.9 – Heatmap UFSM - Dia Ensolarado - Maior Distância

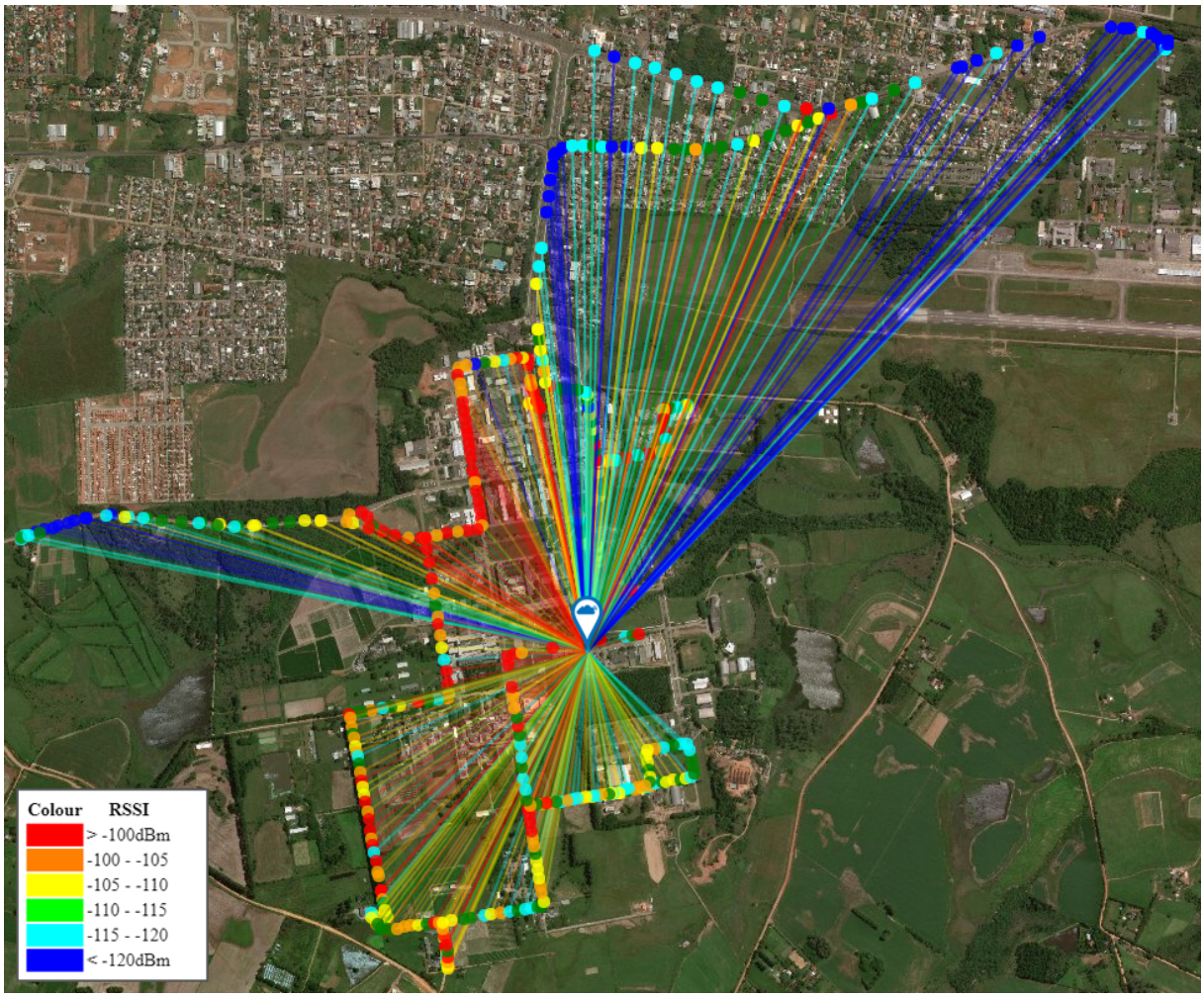


Fonte: Imagem retirada do site TTN Mapper.

Ao analisar a *Figura 4.9* percebemos que há coerência nos dados representados pelo *Heatmap* criado, onde o ponto apresentado demonstra um *RSSI* menor do que os pontos mapeados mais próximos ao *Gateway*.

A fim de demonstrar que há consistência nos dados apresentados, demonstrando que os pontos que apresentaram uma variação considerável com relação a pontos próximos tiveram esta alteração em decorrência de interferência na transmissão, por não ter uma linha de visada limpa, ou seja sem obstáculos, esta situação pode ser observada na *Figura 4.10*, onde o *Heatmap* é demonstrado sobre uma visão de satélite do Campus da UFSM.

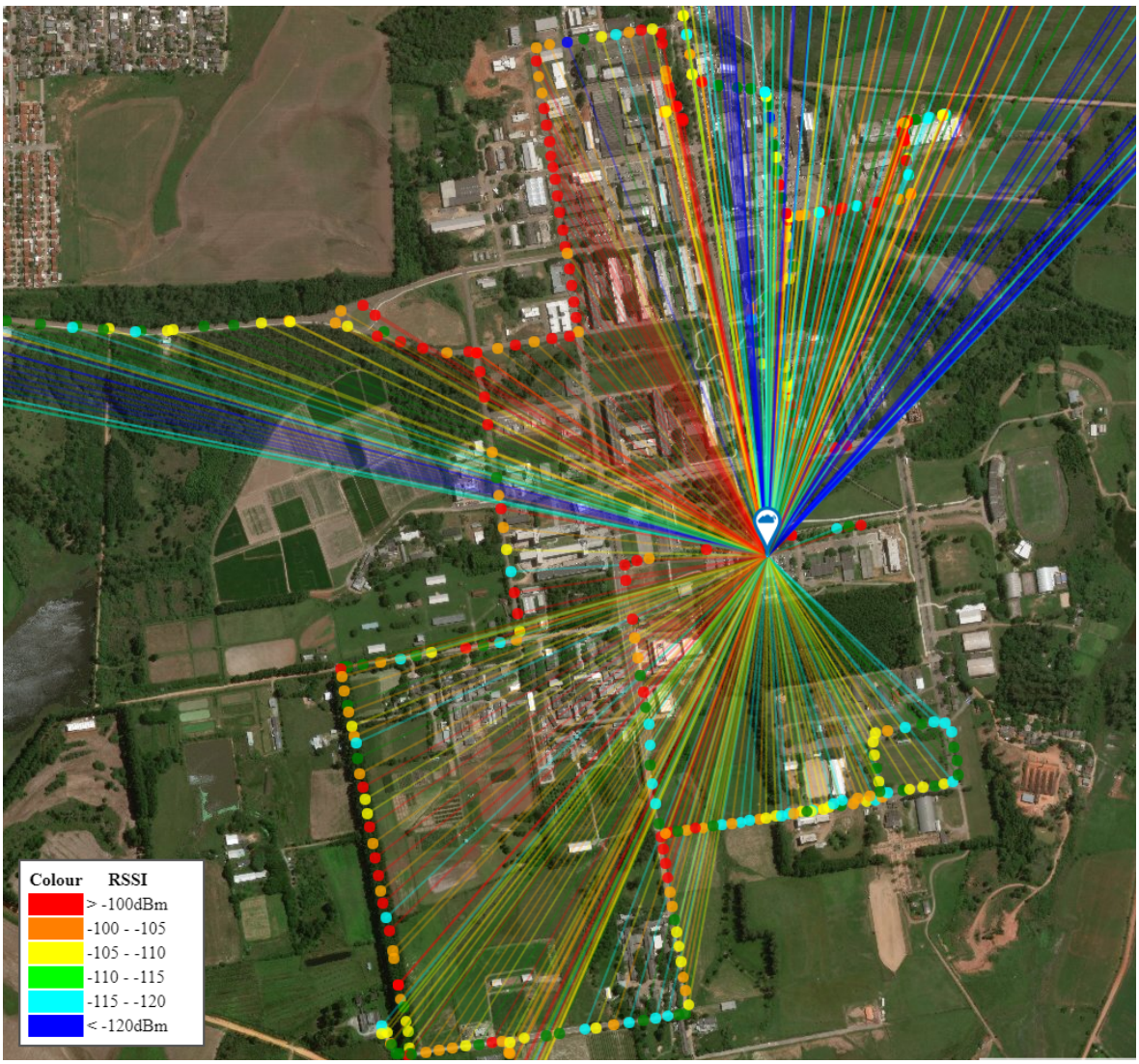
Figura 4.10 – Heatmap UFSM - Dia Ensolarado - Satélite



Fonte: Imagem retirada do site TTN Mapper.

Considerando uma área reduzida da *UFSM*, *Figura 4.11*, a fim de facilitar a identificação dos pontos, podemos então validar os dados plotados no mapa, onde a obstrução da linha de visada, seja pelos prédios ou árvores, ocasionou um menor valor de *RSSI*, ou seja, o sinal teve uma perda de potência.

Figura 4.11 – Heatmap UFSM - Dia Ensolarado - Satélite - Visão Aproximada



Fonte: Imagem retirada do site TTN Mapper.

Pode-se osbervar também que as árvores acabam ocasionando uma maior perda na potência do sinal, pois onde há apenas os prédios e construções obstruindo a linha de visada o *RSSI* se mostrou de uma grandeza maior.

4.2 MAPEAMENTO EM ÁREA FECHADA

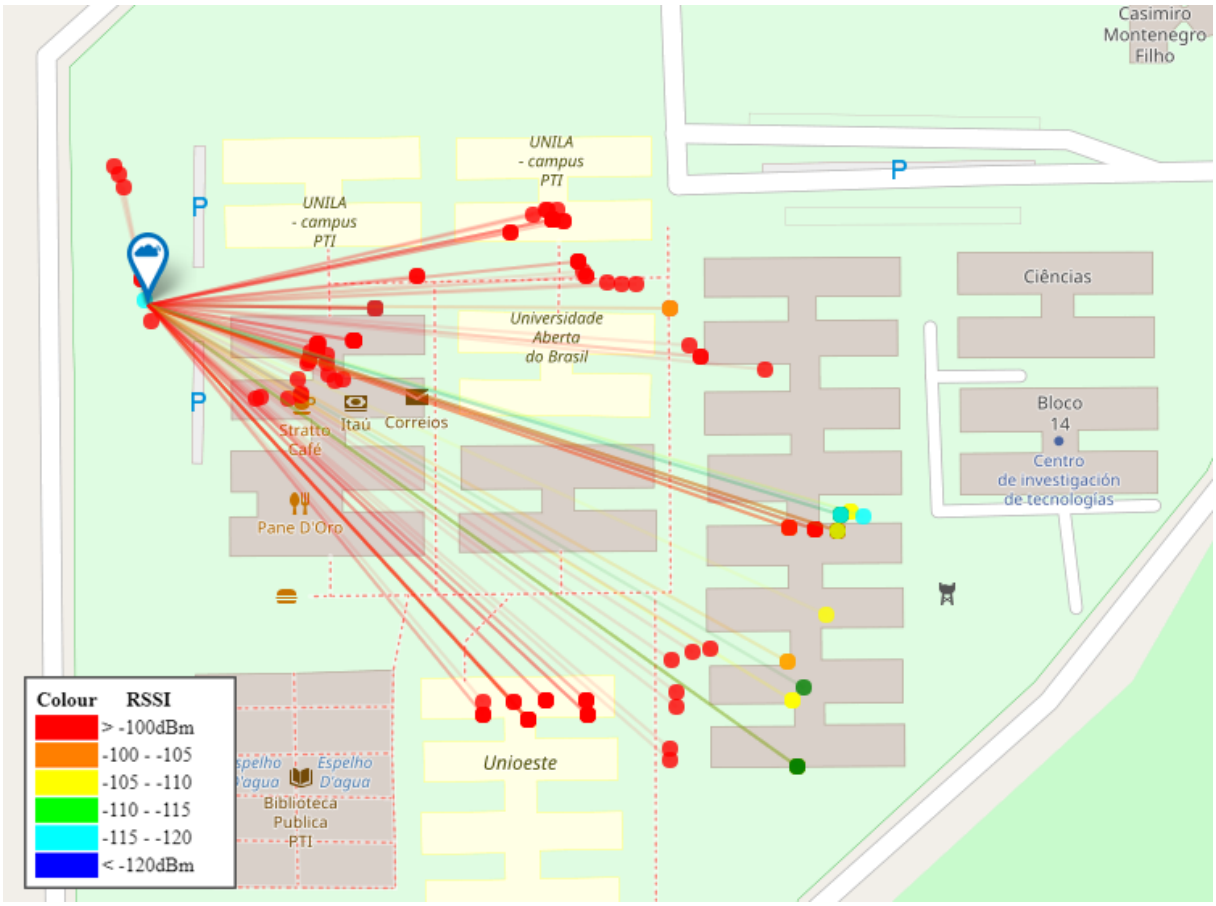
Nesta seção serão apresentados os resultados obtidos através do mapeamento em Área Fechada de acordo com a metodologia sugerida neste trabalho.

4.2.1 Caso: Itaipu

4.2.1.1 Mapeamento em Dia Ensolarado

A realização do mapeamento nas imediações do Parque Tecnológico de Itaipu, localizado dentro da Usina Hidroelétrica de Itaipu, foi realizada no dia 22 de Outubro de 2018, gerando o *Heatmap* da *Figura 4.12*.

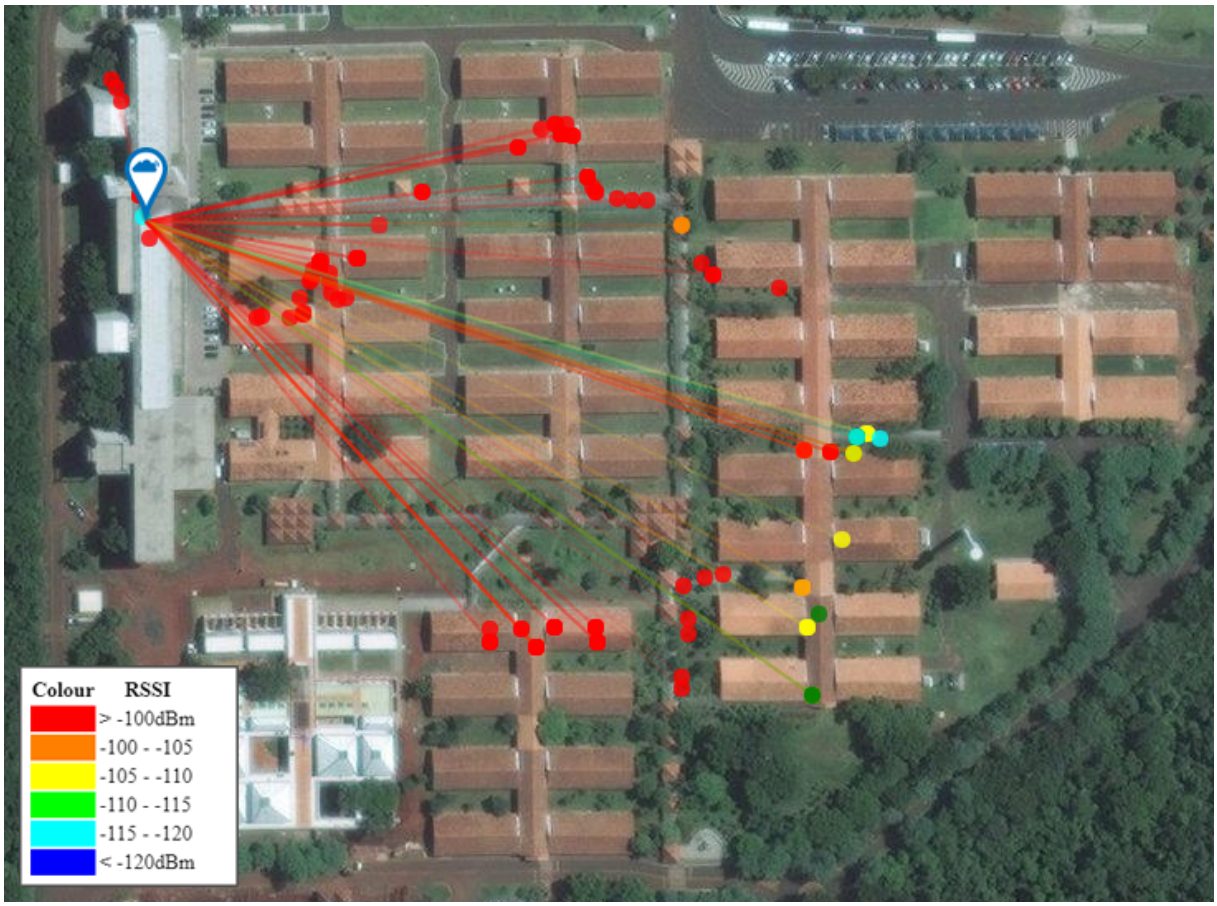
Figura 4.12 – Heatmap PTI - Dia Ensolarado



Fonte: Imagem retirada do site TTN Mapper.

Conforme demonstrado na imagem de satélite da *Figura 4.13*, vemos que há alguns prédios que não aparecem na representação anterior, e que os pontos medidos fora dos mesmos é realizada sob local coberto, com a finalidade de se obter dados mais completos sob qualidade de recepção de sinal e a sua relação com a ateuação quando dentro de construções.

Figura 4.13 – Heatmap PTI - Dia Ensolarado - Satélite

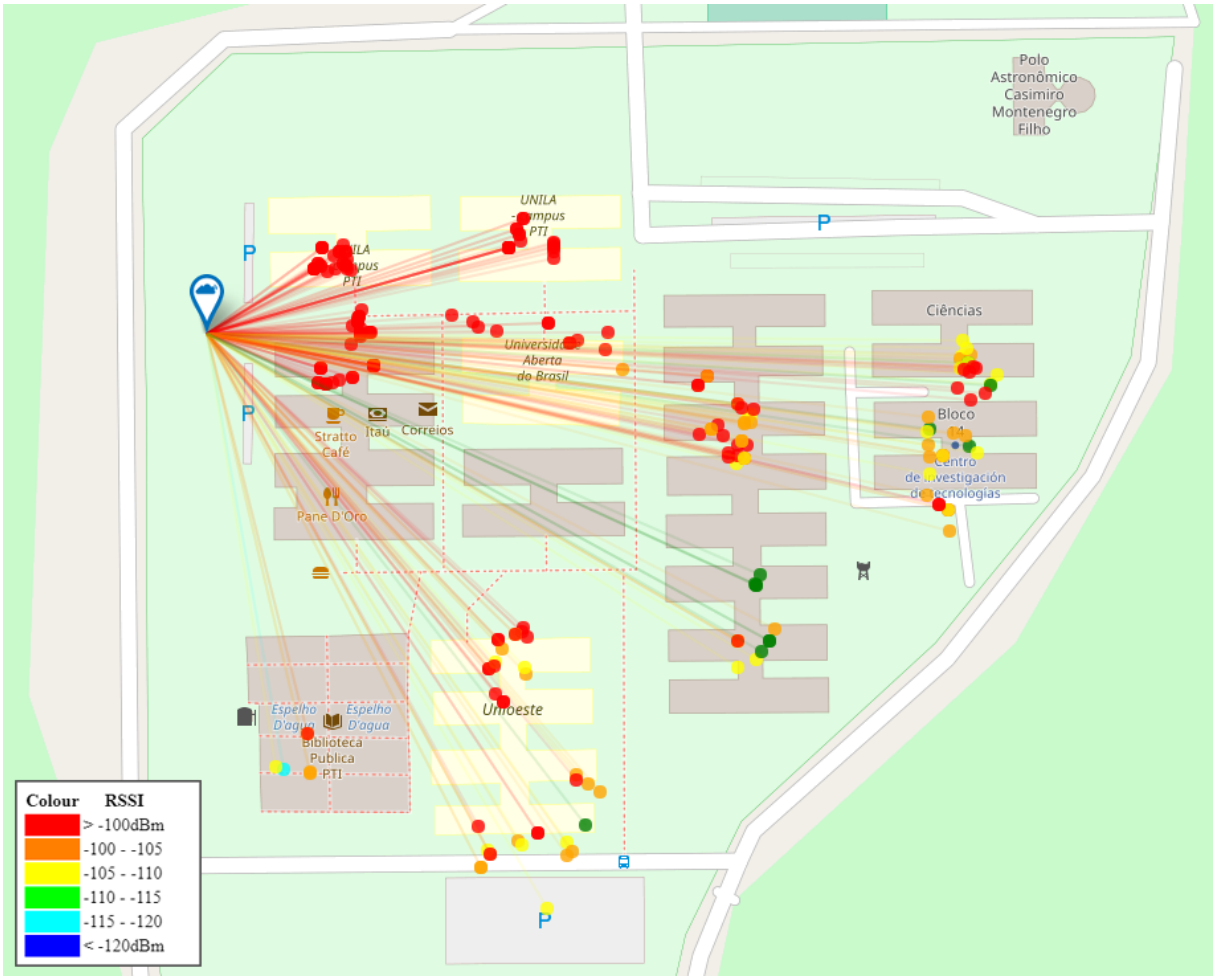


Fonte: Imagem retirada do site TTN Mapper.

4.2.2 Mapeamento em Dia Chuvoso

A realização do mapeamento nas imediações do Parque Tecnológico de Itaipu, localizado dentro da Usina Hidroelétrica de Itaipu, foi realizada no dia 1 de Novembro de 2018, gerando o *Heatmap* da *Figura 4.14*.

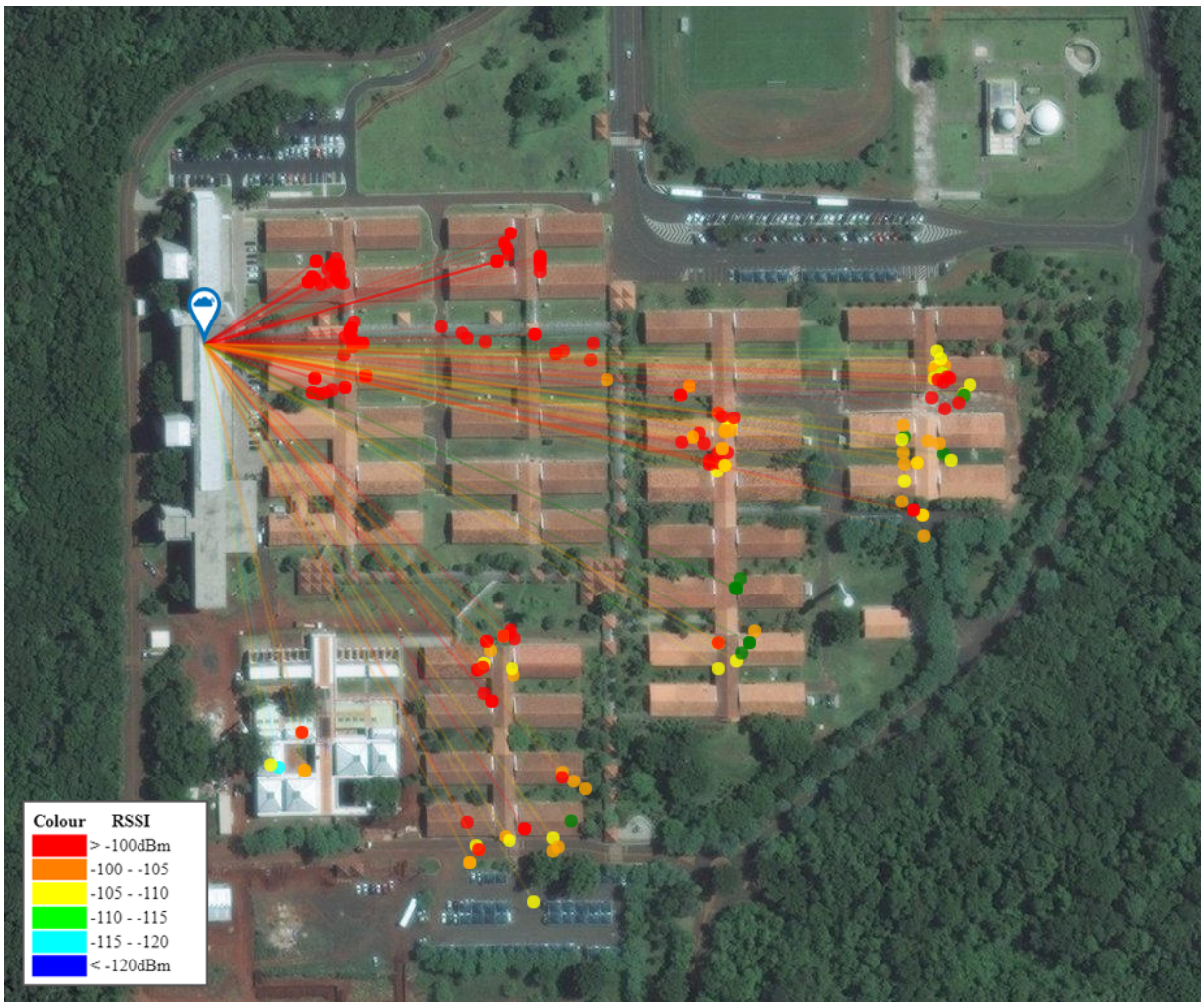
Figura 4.14 – Heatmap PTI - Dia Chuvoso



Fonte: Imagem retirada do site TTN Mapper.

Conforme demonstrado na imagem de satélite da *Figura 4.15*, vemos que há alguns prédios que não aparecem na representação anterior, e que os pontos medidos fora dos mesmos é realizada sob local coberto, com a finalidade de se obter dados mais completos sob qualidade de recepção de sinal e a sua relação com a perca quando dentro de construções.

Figura 4.15 – Heatmap PTI - Dia Chuvoso - Satélite



Fonte: Imagem retirada do site TTN Mapper.

A partir da *Figura 4.15* podemos observar que não foi possível obter mais pontos dentro dos prédios do PTI, são vários os motivos que podem levar isto a acontecer, mais a frente iremos ver quais são estes.

4.3 ANALISE DOS RESULTADOS

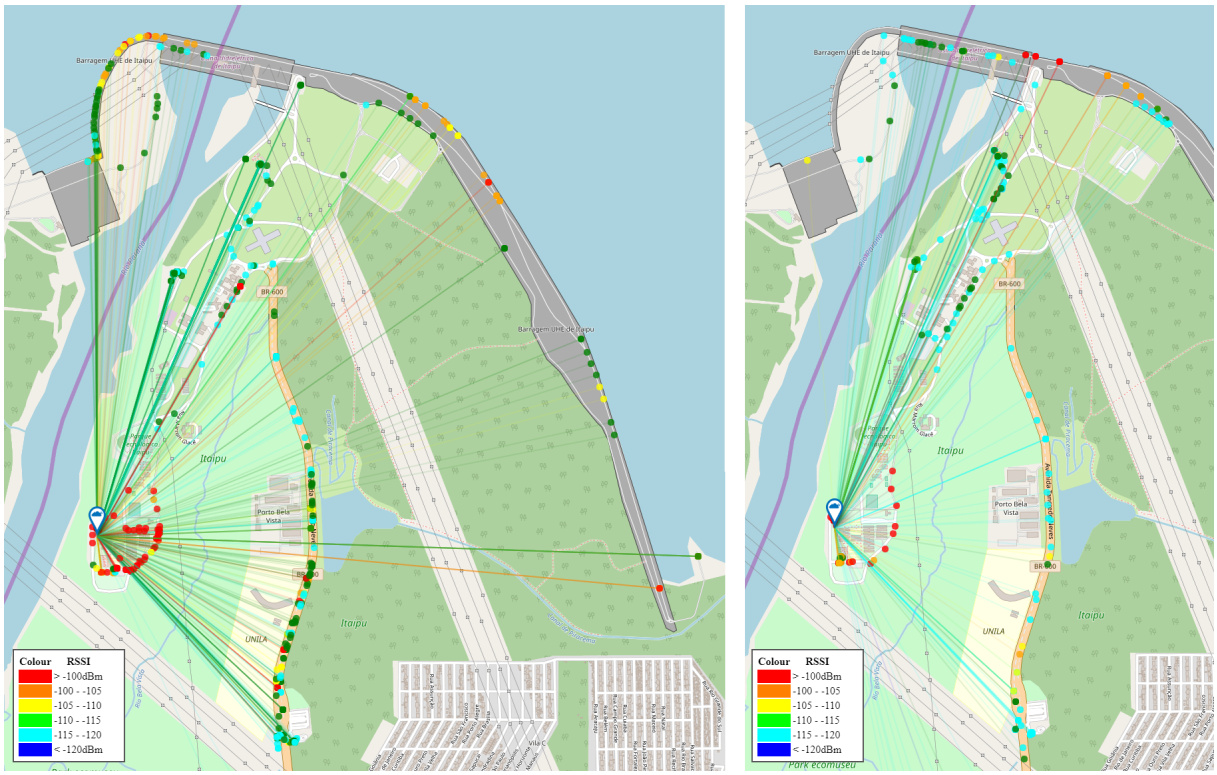
Agora passaremos a analisar os resultados obtidos e comparar estes entre si, a fim de validar os testes que foram feitos.

Os resultados a serem comparados a seguir dizem respeito aos testes realizados nas imediações da Usina Hidroelétrica de Itaipu e no Parque Tecnológico de Itaipu - PTI, o teste realizado na Universidade Federal de Santa Maria será usado como um caso separado a fim de validar que o resultado obtido após os testes é ou não satisfatório.

4.3.1 Heatmap: Dia Ensolarado x Dia Chuvoso

Podemos observar na *Figura 4.16*, onde a esquerda se encontra o *Heatmap* gerado em Dia Ensolarado e a direita o que foi gerado em Dia Chuvoso, ambos feitos em Área Aberta.

Figura 4.16 – Comparação Heatmap Itaipu (Área Aberta): Dia Ensolarado x Dia Chuvoso



Fonte: Imagem retirada do site TTN Mapper.

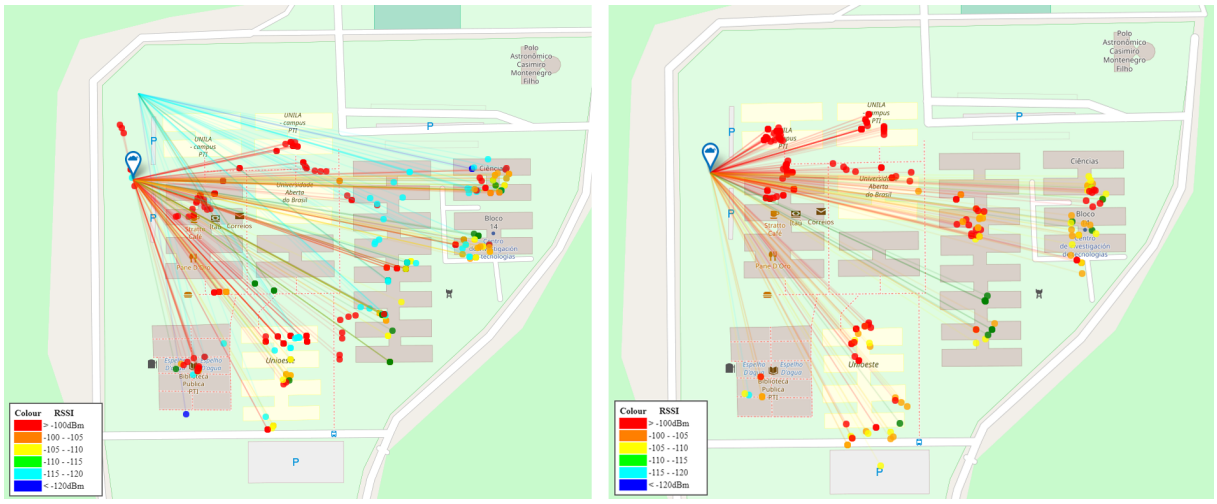
A partir da *Figura 4.16* é possível afirmar que a chuva impos uma atenuação ao sinal, isso acontece pois a água da chuva além de gerar obstáculos a onda de rádio enviada pode acabar absorvendo parte de sua energia, fazendo assim com que o sinal enfraqueça.

Entretanto, a chuva não causou uma atenuação tão grande no sinal em grande

parte dos pontos medidos, sendo uma atenuação mais discreta em relação a onda.

Na *Figura 4.17* iremos observar o mesmo caso apresentado anteriormente, com a imagem ao lado esquerdo sendo do *Heatmap* em Dia Ensolarado e a do lado direito referente ao gerado em Dia Chuvoso, porém com os mapeamentos sendo feitos desta vez em Área Fechada.

Figura 4.17 – Comparação Heatmap Itaipu (Área Fechada): Dia Ensolarado x Dia Chuvoso



Fonte: Imagem retirada do site TTN Mapper.

Assim como no caso anterior, podemos perceber que a chuva neste caso também impôs atenuação ao sinal, gerando enfraquecimento do mesm a ponto de não permitir a recepção do sinal.

Entretanto, pode-se dizer que a chuva não gerou uma atenuação tão grande ao sinal.

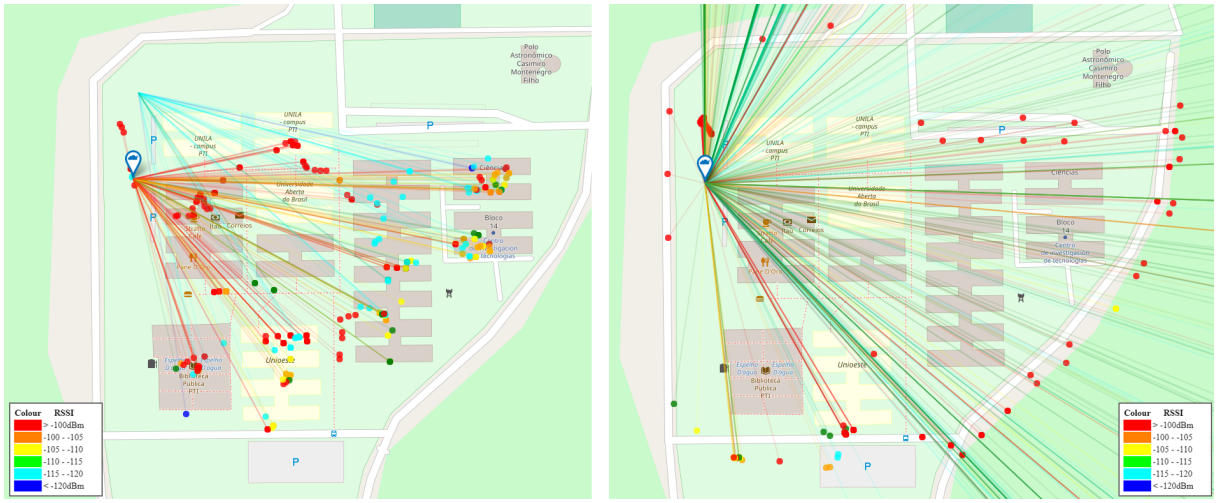
Porém, vale lembrar que a atenuação pode ser maior em casos de uma condição climática mais extrema, como por exemplo um temporal, consequentemente fazendo com que o mesmo perca ainda mais potência ou seja atenuando o sinal em locais onde o mesmo teria um *RSSI* aceitável, entretanto necessita-se a realização de mais experimentos para averiguar o comportamento nestas condições.

4.3.2 Heatmap: Área Fechada x Área Aberta

Analizaremos agora se o fato do *nó* estar em Área Fechada ou Área Aberta gera uma atenuação considerável ao sinal.

Na *Figura 4.18* podemos ver, a esquerda o *Heatmap* gerado em Área Fechada e a direita o que foi gerado em Área Aberta, ambas feitas em Dia Ensolarado.

Figura 4.18 – Comparação Heatmap Itaipu (Dia Ensolarado): Área Fechada x Área Aberta



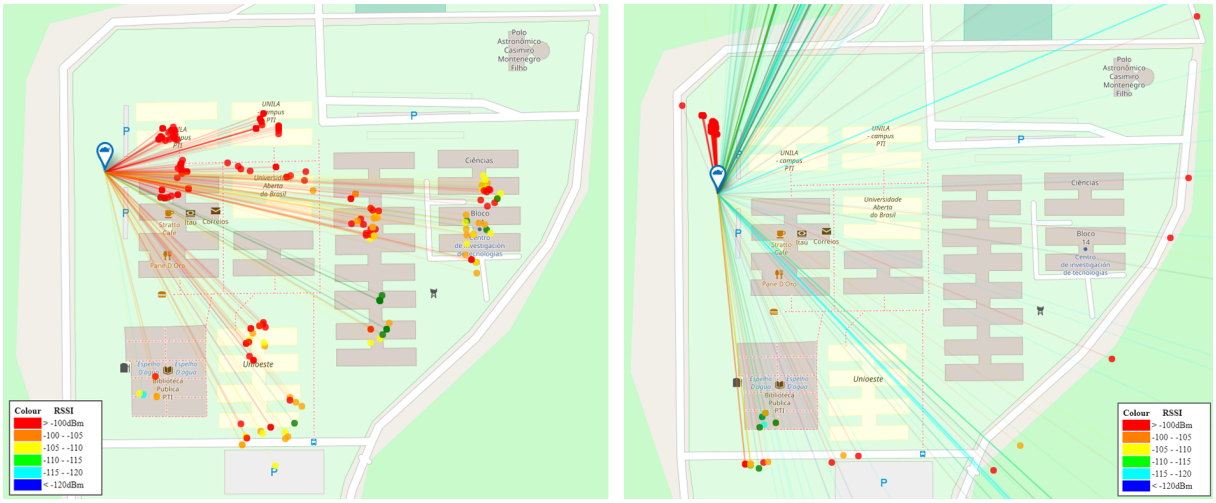
Fonte: Imagem retirada do site TTN Mapper.

Após analise da *Figura 4.18* pode-se afirmar que a intensidade do sinal, *RSSI*, em pontos mapeados em Área Aberta são maiores do que os que foram mapeados em Área Fechada, isso se deve aos obstáculos que estas construções apresentam a onda.

Esta perda na potência do sinal pode variar, dependendo da construção a qual se está sendo feita a medição, por exemplo, a atenuação em construções onde se usa muito ferro para as estruturas pode acabar gerando uma atenuação maior ao sinal, pois esta estrutura pode acabar criando uma *Gaiola de Faraday*, absorvendo assim boa parte da energia da onda.

Na *Figura 4.19* vemos o mesmo caso anterior, porém agora sendo mapeado em Dia Chuvoso, a esquerda tem-se o *Heatmap* dos pontos mapeados em Área Fechada e a direita temos os pontos que foram mapeados em Área Aberta.

Figura 4.19 – Comparação Heatmap Itaipu (Dia Chuvoso): Área Fechada x Área Aberta



Fonte: Imagem retirada do site TTN Mapper.

Percebe-se que neste caso os pontos mapeados em Área Fechada, assim como no caso anterior, obtiveram uma atenuação por estar dentro de uma construção e neste a atenuação foi ainda maior, devido a condição climática a qual foi medido, conforme mostrado anteriormente.

5 CONCLUSÃO

Após a obtenção e analise dos resultados gerados, observa-se que os dados obtidos através do experimento são consistentes e podem ser uteis no auxilio a instalação e aplicação de sistemas embarcados que utilizem o protocolo de comunicação *LoRaWAN*, como na escolha do local de instalação do *nó*, podendo ver se o mesmo está em uma área com boa qualidade de recepção/emissão do sinal, usando o *RSSI* como parâmetro para definir a qualidade da conexão, ou no desenvolvimento de um sistema de localização que faça uso do *RSSI* para medir a distância, a exemplo do demonstrado nos artigos N. Salman, Y. Jay Guo, A. H. Kemp, M. Ghogho (2012), Ka-Ho Lam, Chi-Chung Cheung, Wah-Ching Lee (2017), Bernat Carbonés Fargas, Martin Nordal Petersen (2017), Zhe He, You Li, Ling Pei, Kyle O'Keefe (2018), Ka-Ho Lam, Wah-Ching Lee, Chi-Chung Cheung (2018).

Os *Heatmaps* que foram gerados servem para se ter uma visualização gráfica dos dados que foram coletados durante os testes, servindo também como ferramenta para identificação de possíveis causas que podem levar a conexão a estar/ser instável, se esta baixa qualidade tem relação com fatores físicos como o clima ou local de instalação, podendo vir a gerar perdas de dados, visto que o pacote enviado pode não chegar ao *gateway* e consequêntemente não alcançar a aplicação para processamento do mesmo.

Mesmo que os dados obtidos se mostrem suficientes para a validação do aplicativo, onde o mesmo possa ser usado para a realização deste tipo de experimento, se faz necessária a realização de mais testes, com a finalidade de obter dados mais precisos com relação a diferença de qualidade de conexão em diferentes locais e situações climáticas, sendo indicados a realização destes testes em localidades diferentes, repetindo os testes já realizados, como em ambientes rurais, urbanos e industriais, onde pode haver interferência de outros sistemas, como redes de telefonia celular, redes wi-fi e demais equipamentos que possam causar interferência.

A partir da analise dos dados obtidos, pode-se ver que houve uma perda considerável na qualidade do sinal em alguns locais medidos, onde não havia uma linha de visada limpa para esta transmissão, pode-se observar ainda que as árvores mostraram ser um obstáculo maior do que construções, no que diz respeito ao obstáculo ao sinal, absorvendo mais energia da onda do que os demais obstáculos.

REFERÊNCIAS BIBLIOGRÁFICAS

3GLTEINFO. *HomeLoRa Tutorial*: What is lora wireless for iot? 3GLTEinfo, 2017. Acesso em 12 set. 2018. Disponível em: <http://www.3glteinfo.com/lora/>.

Bernat Carbonés Fargas, Martin Nordal Petersen. **GPS-free geolocation using LoRa in low-power WANs**. IEEE, 2017. Acesso em 16 nov. 2018. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8016251>.

FELHR85. **UsbSerial**. GitHub, 2018. Acesso em 16 jul. 2018. Disponível em: <https://github.com/felHR85/UsbSerial>.

GOOGLE DEVELOPERS. **Activities**. Android Developers, 2018. Acesso em 13 nov. 2018. Disponível em: <https://developer.android.com/guide/components/activities?hl=pt-br>.

Ka-Ho Lam, Chi-Chung Cheung, Wah-Ching Lee. **LoRa-based localization systems for noisy outdoor environment**. IEEE, 2017. Acesso em 16 nov. 2018. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8115843>.

Ka-Ho Lam, Wah-Ching Lee, Chi-Chung Cheung. **New RSSI-Based LoRa Localization Algorithms for Very Noisy Outdoor Environment**. IEEE, 2018. Acesso em 16 nov. 2018. Disponível em: <https://ieeexplore.ieee.org/document/8377968>.

LORA ALLIANCE. **About LoRaWAN**: about-lorawan. LoRa Alliance, 2018. Acesso em 26 jul. 2018. Disponível em: <https://lora-alliance.org/about-lorawan>.

N. Salman, Y. Jay Guo, A. H. Kemp, M. Ghogho. **Analysis of linear least square solution for RSS based localization**. IEEE, 2012. Acesso em 16 nov. 2018. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6380846>.

OPTIMIZELY. **Heatmap**: What is a heatmap? Optimizely, 2018. Acesso em 25 set. 2018. Disponível em: <https://www.optimizely.com/optimization-glossary/heatmap/>.

THE THINGS NETWORK. **Documentation**: lorawan. The Things Network, 2018. Acesso em 26 jul. 2018. Disponível em: <https://www.thethingsnetwork.org/docs/>.

TIMOTHY SEALY. **Applications**: Ttn mapper. The Things Network, 2018. Acesso em 26 jul. 2018. Disponível em: <https://www.thethingsnetwork.org/docs/applications/ttnmapper/>.

Zhe He, You Li, Ling Pei, Kyle O'Keefe. **Enhanced Gaussian Process based Localization using a Low Power Wide Area Network**. IEEE, 2018. Acesso em 16 nov. 2018. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8515235>.

ANEXO A – TERMINALACTIVITY - CÓDIGO FONTE

```
1  package org.ttnmapper.ttnmapperv2;
2  /**
3   *  (en) Android Library Import:
4   */
5  import android.app.PendingIntent;
6  import android.content.BroadcastReceiver;
7  import android.content.Context;
8  import android.content.Intent;
9  import android.content.IntentFilter;
10 import android.hardware.usb.UsbDevice;
11 import android.hardware.usb.UsbDeviceConnection;
12 import android.hardware.usb.UsbManager;
13 import android.os.Build;
14 import android.os.VibrationEffect;
15 import android.os.Vibrator;
16 import android.support.v4.content.LocalBroadcastManager;
17 import android.support.v7.app.AppCompatActivity;
18 import android.os.Bundle;
19 import android.text.method.ScrollingMovementMethod;
20 import android.util.Log;
21 import android.view.View;
22 import android.widget.Button;
23 import android.widget.EditText;
24 import android.widget.TextView;
25 import android.widget.Toast;
26
27 import com.felhr.usbserial.UsbSerialDevice;
28 import com.felhr.usbserial.UsbSerialInterface;
29
30 import java.io.UnsupportedEncodingException;
31 import java.util.Map;
32
33 public class TerminalActivity extends AppCompatActivity {
34
35     /**
36      *  (en) Application Variables Declaration
37      *
38      *  (pt) Declaração de Variáveis do Aplicativo
39      */
40     private static final String ACTION_USB_PERMISSION = "com.cesar.ttnmapper.
    USB_PERMISSION";
41     private static final String TAG = "TerminalActivity";
42     private final String startString = "Please connect a device to continue..."
```

```
    ;
43  private final String pingMsg = "AT+CMSG=ping";
44  private static boolean ping = false;
45  private static boolean GPSMap = false;
46  private static long mediumPingTime = 0;
47  private static int pingCounter = 0;
48  private static int successPing = 0;
49  private static int failedPing = 0;
50  private static int defaultPingCnt = 4;
51  private final int pingDelay = 500;
52  private static String Mode = "N/A";
53  private static boolean ModeTest = true;
54  private static boolean join = false;
55  private static int joinCnt = 20;
56  private static boolean pingReceived = false;
57  private static long elapsedtime;
58  private static long startTime;
59  private int cloopcnt = 0;
60  private boolean loop = false;
61  private boolean cloop = false;
62  private int[] loopSettings = {0, 0};
63  private int state = 0;
64  private String loopCommand;
65
66  /**
67  *   (en) Input Config Strings:
68  *   Replace ALL the strings ahead to the strings desired
69  *
70  *   (pt) Entre com as Strings de Configuração:
71  *   Substitua TODAS as strings a seguir com as strings desejadas
72  */
73  private final String UFSM_AppID     = AppIDString;
74  private final String UFSM_DevID     = DevIDString;
75  private final String UFSM_AccessKey = AccessKeyString;
76  private final String UFSM_Broker    = BrokerString;
77
78  PendingIntent mPermissionIntent;
79
80  UsbDevice deviceFound = null;
81  UsbDeviceConnection connection = null;
82  UsbSerialDevice serial;
83
84  TextView terminalOut;
85  EditText lineCommand;
86  Button btnSend;
87
88  @Override
```

```
89      protected void onCreate(Bundle savedInstanceState) {
90          super.onCreate(savedInstanceState);
91          setContentView(R.layout.activity_terminal);
92
93          terminalOut = (TextView) findViewById(R.id.terminalOutput);
94          lineCommand = (EditText) findViewById(R.id.commandLine);
95          btnSend = (Button) findViewById(R.id.sendBtn);
96
97          terminalOut.setMovementMethod(new ScrollingMovementMethod());
98
99          terminalOut.setText(startString);
100
101         // register the broadcast receiver
102         mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(
103                 ACTION_USB_PERMISSION), 0);
104         IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
105
106         registerReceiver(mUsbReceiver, filter);
107
108         registerReceiver(mUsbDeviceReceiver, new IntentFilter(
109                 UsbManager.ACTION_USB_DEVICE_ATTACHED));
110         registerReceiver(mUsbDeviceReceiver, new IntentFilter(
111                 UsbManager.ACTION_USB_DEVICE_DETACHED));
112
113         registerGPS();
114         registerMqtt();
115         connectUsb();
116
117         LocalBroadcastManager.getInstance(this).registerReceiver(GPSService,
    new IntentFilter("ttn-mapper-gpsservice-answer"));
118
119         btnSend.setOnClickListener(new View.OnClickListener() {
120             @Override
121             public void onClick(View v) {
122                 commandHandler();
123             }
124         });
125     }
126
127     /**
128      *  (en) On mMessageReceiver Broadcast Received
129      */
130     private BroadcastReceiver mMessageReceiver = new BroadcastReceiver() {
131         @Override
132         public void onReceive(Context context, Intent intent) {
133             String msg = intent.getStringExtra("message");
134
```

```
135             runOnUiThread(new Runnable() {
136                 @Override
137                 public void run() {
138                     terminalOut.append("\n\n————————————————————————————————————————" +
    "————————————————————————————————————————\n" +
139                                        "MQTT Received:\n" + msg + "" +
140                                        "\n————————————————————————————————————————" +
    "————————————————————————————————————————\n");
141                 }
142             });
143 
144             if (cloop) {
145                 cloopcnt++;
146             }
147         }
148     };
149 
150     /**
151      *  (en) On GPSService Broadcast Received
152      */
153     private BroadcastReceiver GPSService = new BroadcastReceiver() {
154         @Override
155         public void onReceive(Context context, Intent intent) {
156             String status = intent.getStringExtra("stats");
157 
158             if (status.equals("true")) {
159                 GPSReq();
160             }else {
161                 runOnUiThread(new Runnable() {
162                     @Override
163                     public void run() {
164                         terminalOut.append("\n
    ————————————————————————————————————————————————————————————————————————\n" +
165                                            "Logging service not running, try
    to restart it." +
166                                            "\n
    ————————————————————————————————————————————————————————————————————————\n");
167                         vibrateNotify();
168                     }
169                 });
170 
171                 stop();
172             }
173         }
174     };
175 
176     /**
```

```
177      *  (en) Emits a vibration notification
178      *
179      *  (pt) Emite uma notificação de vibração
180      */
181     private void vibrateNotify() {
182         if (Build.VERSION.SDK_INT >= 26) {
183             ((Vibrator) getSystemService(VIBRATOR_SERVICE)).vibrate(
    VibrationEffect.createOneShot(300,VibrationEffect.DEFAULT_AMPLITUDE));
184         } else {
185             ((Vibrator) getSystemService(VIBRATOR_SERVICE)).vibrate(300);
186         }
187     }
188
189     /**
190      *  (en) On mGPSReceiver Broadcast Received
191      */
192     private BroadcastReceiver mGPSReceiver = new BroadcastReceiver() {
193         @Override
194         public void onReceive(Context context, Intent intent) {
195             String msg = intent.getStringExtra("message");
196
197             if (msg.equals("selfstop")) {
198                 String reason = intent.getStringExtra("payload");
199
200                 stop();
201
202                 runOnUiThread(new Runnable() {
203                     @Override
204                     public void run() {
205                         terminalOut.append("\n
    ————————————————————————————————————————————\n" +
206                                 "                    GPSMap Error:\n
    " + reason +
207                                 "\n
    ————————————————————————————————————————————\n\n");
208                     }
209                 });
210             } else {
211                 String lat = intent.getStringExtra("lat");
212                 String lon = intent.getStringExtra("lon");
213                 String alt = intent.getStringExtra("alt");
214                 String acc = intent.getStringExtra("acc");
215
216                 String dlat = intent.getStringExtra("dlat");
217                 String dlon = intent.getStringExtra("dlon");
218                 String dalt = intent.getStringExtra("dalt");
219                 String dacc = intent.getStringExtra("dacc");
```

```
220
221             String toSend = "AT+MSGHEX=" + lat + lon + alt + acc;
222
223
224             Log.d(TAG, "GPSMap: Pos Received");
225             Log.d(TAG, "GPSMap: Pos encoded: " + toSend);
226
227             if (lat != "not accurate enought" && lon != "not accurate
enought") {
228
229                 runOnUiThread(new Runnable() {
230                     @Override
231                     public void run() {
232                         terminalOut.append("\n
————————————————————————————————————————\n" +
233                                 "GPS Pos:\n" +
234                                 "\nLat: " + dlat + "\nLat(HEX): " + lat +
235                                 "\nLon: " + dlon + "\nLon(HEX): " + lon +
236                                 "\nAlt: " + dalt + "\nAlt(HEX): " + alt +
237                                 "\nAcc: " + dacc + "\nAcc(HEX): " + acc +
238                                 "\n\n" + toSend +
239                                 "\n
————————————————————————————————————————\n\n");
240                     }
241                 });
242
243                 try {
244                     serial.write(toSend.getBytes());
245                 } catch (Exception e) {
246                     e.printStackTrace();
247                     runOnUiThread(new Runnable() {
248                         @Override
249                         public void run() {
250                             terminalOut.append("\n
————————————————————————————————————————————————————————————————————————\
n" +
251                                     "
      Error:" +
252                                     "\nError while trying to
send serial command.\n" +
253                                     "Please make sure that the
device is connected." +
254                                     "\n
————————————————————————————————————————————————————————————————————————\
n\n");
255                         }
256                     });
```

```
257                     }
258                 } else {
259                     float tAcc = Float.parseFloat(dacc);
260                     String temp = String.format("%.02f", tAcc);
261                     runOnUiThread(new Runnable() {
262                         @Override
263                         public void run() {
264                             terminalOut.append("\n
    ------------------------------------------------------------\n" +
265                                     "GPS Pos Received but was not accurate
    enought (>20m).\n" +
266                                     "Current accuracy: " + temp +
267                                     "\n
    ------------------------------------------------------------\n\n");
268                         }
269                     });
270                 }
271                 Log.d(TAG, "GPSMap: Pos Printed");
272             }
273         }
274     };
275
276     /*  (en) commandHandler function is meant to interpret
277      *       the user input and decide wich function to call
278      *
279      *  (pt) A função commandHandler destina-se a interpretar
280      *       a entrada do usuário e decidir qual função chamar
281      */
282     private void commandHandler() {
283         String command = lineCommand.getText().toString();
284
285         if (!loop) {
286             if (command.toUpperCase().equals("DISCONNECT") || command.
    toUpperCase().equals("CLOSE")) {
287                 releaseUsb();
288             } else if (command.toUpperCase().equals("CLS") || command.
    toUpperCase().equals("CLEAR")) {
289                 terminalOut.setText("");
290             } else if (command.toUpperCase().equals("LOOP")) {
291                 if (GPSMap) {
292                     terminalOut.append("\n
    ------------------------------------------------------------\n" +
293                             "        Can't do this opperation while GPSMap is
    active     " +
294                             "\n
    ------------------------------------------------------------\n");
295                 }else {
```

```
296                     loop = true;
297                     terminalOut.append("\nEnter the command:");
298                     state = 1;
299                 }
300             } else if (command.toUpperCase().equals("CLOOP")) {
301                 if (GPSMap) {
302                     terminalOut.append("\n
    ----------------------------------------------------------------\n" +
303                             "        Can't do this opperation while GPSMap is
    active     " +
304                             "\n
    ----------------------------------------------------------------\n");
305                 }else {
306                     loop = true;
307                     cloop = true;
308                     terminalOut.append("\nType the number of events desired: ")
    ;
309                     state = 10;
310                 }
311             }else if (command.toUpperCase().equals("PING")) {
312                 if (GPSMap) {
313                     terminalOut.append("\n
    ----------------------------------------------------------------\n" +
314                                     "        Can't do this opperation while
    GPSMap is active     " +
315                                     "\n
    ----------------------------------------------------------------\n");
316                 }else {
317                     doPing();
318                 }
319             }else if (command.toUpperCase().equals("GPSMAP")) {
320                 startGPSMap();
321             }else if (command.toUpperCase().equals("STOP")) {
322                 stop();
323
324             }else if (command.toUpperCase().equals("UFSM")) {
325                 MyApplication mApplication = (MyApplication)
    getApplicationContext();
326
327                 terminalOut.append("\n\n
    ----------------------------------------------------------------\n" +
328                                     "        Running UFSM Config:\n");
329                 mApplication.setTtnApplicationId(UFSM_AppID);
330                 terminalOut.append("\nApplicationID = " + UFSM_AppID);
331                 mApplication.setTtnDeviceId(UFSM_DevID);
332                 terminalOut.append("\nDevID = " + UFSM_DevID);
333                 mApplication.setTtnAccessKey(UFSM_AccessKey);
```

```
334                 terminalOut.append("\nAccessKey = " + UFSM_AccessKey);
335                 mApplication.setTtnBroker(UFSM_Broker);
336                 terminalOut.append("\nMQTT Broker: " + UFSM_Broker);
337 
338                 terminalOut.append("\n
    ————————————————————————————————————————————————————————————\n\n");
339             }else {
340                 serial.write(command.getBytes());
341             }
342         }else {
343             /**
344              *  (en) Switch state variable is meant to get all the
345              *       data required to the loop and cloop functions
346              *
347              *  (pt) O switch na variável state destina-se a obter todos
348              *       os dados necessários para as funções loop e cloop
349              */
350             switch (state) {
351                 case 1:
352                     loopCommand = lineCommand.getText().toString();
353                     terminalOut.append("\nCommand: " + loopCommand + "\n\nType
    cancel to abort...\n\nType the number of events desired:");
354                     state = 10;
355                     break;
356                 case 10:
357                     if(lineCommand.toString().toUpperCase().equals("CANCEL")) {
358                         terminalOut.append("\n\nOperation canceled.\n");
359                         state = 1010;
360                         break;
361                     }else {
362                         loopSettings[0] = Integer.parseInt(lineCommand.getText
    ().toString());
363                         terminalOut.append("\n\nNumber of events: " +
    loopSettings[0] + "\n\nType the interval in ms:");
364                         state = 100;
365                         break;
366                     }
367                 case 100:
368                     loopSettings[1] = Integer.parseInt(lineCommand.getText().
    toString());
369                     terminalOut.append(" " + loopSettings[1] + " ms\n\n");
370                     loopSend();
371                     state = 1010;
372                     break;
373             }
374 
375         }
```

```
376         if(state == 1010) {
377             state = 0;
378             loop = false;
379         }
380 
381         lineCommand.setText("");
382     }
383 
384     /**
385      *  (en) GPSMap Function
386      */
387     private void startGPSMap() {
388         GPSMap = true;
389         terminalOut.append("\n-----------------------------------------\n" +
390                 "GPS Mapping Started\n-----------------------------------------\n\n")
    ;
391 
392         new Thread(new Runnable() {
393             @Override
394             public void run() {
395                 while (GPSMap) {
396                     GPSServiceVer();
397 
398                     try {
399                         //  Log.d(TAG, "GPSMap: Sleep");
400                         Thread.sleep(5000);
401                         //  Log.d(TAG, "GPSMap: Wake");
402                     } catch (InterruptedException e) {
403                         e.printStackTrace();
404                     }
405                 }
406             }
407         }).start();
408     }
409 
410     /**
411      *  (en) Request the GPS Service Status
412      *
413      *  (pt) Requisita o Status do Serviço de GPS
414      */
415     private void GPSServiceVer() {
416         Intent intent = new Intent("ttn-mapper-gpsservice-event");
417         intent.putExtra("message", "statsrequest");
418 
419         LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
420     }
421 
```

```
422     /**
423      *  (en) stop Function
424      */
425     private void stop() {
426         GPSMap = false;
427 
428         try {
429             Thread.sleep(500);
430         } catch (InterruptedException e) {
431             e.printStackTrace();
432         }
433     }
434 
435     /**
436      *  (en) GPSReq Function is meant to send the request
437      *       of GPS coordinates to another Activity
438      *
439      *  (pt) A função GPSReq é destinada a enviar uma requisição
440      *       das coordenadas GPS para outra Activity
441      */
442     private void GPSReq() {
443         Intent intent = new Intent("ttn-mapper-gpsreq-service");
444         intent.putExtra("message", "posrequest");
445 
446         LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
447         Log.d(TAG, "GPSMap: Pos Req");
448     }
449 
450     /**
451      *  (en) joinReq is meant to trying to join an OTAA Network
452      *
453      *  (pt) A função joinReq é destinada a tentar se conectar a uma rede OTAA
454      */
455     private void joinReq() {
456         join = false;
457         int sleepTime = 6000;
458         int joinStep = 0;
459 
460         while(!join && joinStep < joinCnt) {
461             Log.d(TAG, "OTAA: Trying to join the network....");
462             final String joinStats = "\nJoin attempt: " + (joinStep+1) + " of "
    + joinCnt;
463             runOnUiThread(new Runnable() {
464                 @Override
465                 public void run() {
466                     terminalOut.append(joinStats);
467                 }
```

```
468             });
469             serial.write("AT+JOIN".getBytes());
470             try {
471                 Thread.sleep(sleepTime);
472             } catch (InterruptedException e) {
473                 e.printStackTrace();
474             }
475             joinStep++;
476         }
477 
478         final String joinStats;
479         try {
480             Thread.sleep(4000);
481         } catch (InterruptedException e) {
482             e.printStackTrace();
483         }
484 
485         if (join) {
486             joinStats = "\n
    ————————————————————————————————————————————————————————\n" +
487                         "                    Network Joined." +
488                         "\n
    ————————————————————————————————————————————————————————\n\n";
489         }else {
490             joinStats = "\n
    ————————————————————————————————————————————————————————\n" +
491                         "                    Network Join Failed." +
492                         "\n
    ————————————————————————————————————————————————————————\n\n";
493         }
494         runOnUiThread(new Runnable() {
495             @Override
496             public void run() {
497                 terminalOut.append(joinStats);
498             }
499         });
500     }
501 
502     /**
503      *  (en) doPing function is meant to test the quality of connection
504      *       by trying to send a message with ACK to the gateway 4 times and
505      *       analysing if the ACK was received, returning the receive statistcs
506      *
507      *  (pt) A função doPing é destinada a fazer o teste de qualidade da
508      *       conexão, a partir do envio de mensagens com ACK para o gateway
509      *       4 vezes e analizando se o ACK foi recebido, retornando uma
510      *       estatística de recebimento
```

```
511      */
512     private void doPing() {
513         if (pingCounter == 0) {
514             ping = true;
515             lineCommand.setEnabled(false);
516             btnSend.setEnabled(false);
517             startTime = System.currentTimeMillis();
518             String test = pingMsg + pingCounter;
519             terminalOut.append("\n\n
    ------------------------------------------------------------------\n" +
520                                     "                          Starting Ping:\n" +
521                                 "\nSending Message: " + test + "\n");
522             serial.write(test.getBytes());
523             pingCounter++;
524         }else if (pingCounter < defaultPingCnt) {
525             try {
526                 Thread.sleep(pingDelay);
527             } catch (InterruptedException e) {
528                 e.printStackTrace();
529             }
530             startTime = System.currentTimeMillis();
531             String test = pingMsg + pingCounter;
532             terminalOut.append("\n------------------------------------------------\
    nSending Message: " + test + "\n");
533             serial.write(test.getBytes());
534             pingCounter++;
535         }else {
536             ping = false;
537
538             float successRate = (successPing*100) / pingCounter;
539             float failRate = 100 - successRate;
540             float medium = (float) mediumPingTime/pingCounter;
541
542             String pingSuccess = String.format("%.02f", successRate);
543             String pingFail = String.format("%.02f", failRate);
544             String mediumTime = String.format("%.02f", medium);
545
546             Log.d(TAG, "PING Stats: \n" + "\n
    ------------------------------------------------" +
547                     "\nSuccess ping: " + successPing +
548                     "\nFailed  ping: " + failedPing +
549                     "\n\nSuccess rate: " + pingSuccess + "%\nFail rate: " +
    pingFail + "%" +
550                     "\nMedium ping Time: " +  mediumTime +
551                     "\n------------------------------------------------\n\n");
552
553             runOnUiThread(new Runnable() {
```

```
554                 @Override
555                 public void run() {
556                     terminalOut.append("\n
    ------------------------------------------------------" +
557                             "\nSuccess ping: " + successPing +
558                             "\nFailed  ping: " + failedPing +
559                             "\n\nSuccess rate: " + pingSuccess + "%\nFail rate:
     " + pingFail + "%" +
560                             "\nMedium ping Time: " +  mediumTime +
561                             "ms\n
    ------------------------------------------------------\n"+"" +
562                             "
    ------------------------------------------------------------------------\n\n");
563                     lineCommand.setEnabled(true);
564                     btnSend.setEnabled(true);
565
566                     pingCounter = 0;
567                     successPing = 0;
568                     failedPing = 0;
569                     mediumPingTime = 0;
570                 }
571             });
572         }
573     }
574
575     /**
576      *  (en) loopSend function is responsible to do the loop with the
577      *       parameters desired, command, number of times and interval
578      *
579      *  (pt) A função loopSend é responsável por fazer o loop com os
580      *       parâmetros desejados, comando, número de vezes e intervalo
581      */
582     private void loopSend() {
583         lineCommand.setEnabled(false);
584         btnSend.setEnabled(false);
585
586         if(!cloop && loopCommand.toUpperCase().contains("MSG") && loopSettings
    [1] < 5000) {
587             runOnUiThread(new Runnable() {
588                 @Override
589                 public void run() {
590                     terminalOut.append("\n
    ------------------------------------------------------\n" +
591                                     "        Interval too short, changing
    to 5000ms." +
592                                     "\n
    ------------------------------------------------------\n");
```

```
593                 }
594             });
595             loopSettings[1] = 5000;
596         }else if (cloop) {
597             runOnUiThread(new Runnable() {
598                 @Override
599                 public void run() {
600                     terminalOut.append("\n
    ————————————————————————————————————————————————————————\n" +
601                             "        Interval too short, changing to 5000ms." +
602                             "\n
    ————————————————————————————————————————————————————————\n");
603                 }
604             });
605             loopSettings[1] = 5000;
606         }
607         new Thread(new Runnable() {
608             @Override
609             public void run() {
610                 int done = 0;
611                 for (int step = loopSettings[0]; step > 0; step--){
612                     String stepsDone = String.valueOf(done+1);
613                     runOnUiThread(new Runnable() {
614                         @Override
615                         public void run() {
616                             terminalOut.append("
    ————————————————————————————————————————————————————————\n\n" +
617                                                "
    ————————————————————————————————————————————————————" +
618                                                "\nRunning " + stepsDone + " of
    " + String.valueOf(loopSettings[0]) + ":\n");
619                         }
620                     });
621
622                     if(!cloop) {
623                         runOnUiThread(new Runnable() {
624                             @Override
625                             public void run() {
626                                 terminalOut.append("Command: " + loopCommand +
    "\n");
627                             }
628                         });
629
630                         serial.write(loopCommand.getBytes());
631                     }else {
632                         String toSend = "AT+MSG=" + stepsDone;
633                         runOnUiThread(new Runnable() {
```

```
634                             @Override
635                             public void run() {
636                                 terminalOut.append("Command: " + toSend + "\n")
    ;
637                             }
638                         });
639 
640                         serial.write(toSend.getBytes());
641                     }
642                     try {
643                         done++;
644                         Thread.sleep(loopSettings[1]);
645                     } catch (InterruptedException e) {
646                         e.printStackTrace();
647                     }
648                 }
649 
650                 try {
651                     Thread.sleep(500);
652                 } catch (InterruptedException e) {
653                     e.printStackTrace();
654                 }
655                 loopCompleted();
656                 runOnUiThread(new Runnable() {
657                     @Override
658                     public void run() {
659                         lineCommand.setEnabled(true);
660                         btnSend.setEnabled(true);
661                     }
662                 });
663             }
664         }).start();
665     }
666 
667     /**
668      *  (en) loopCompleted functions is called when the loopSend function
669      *       completes their loop, returning the number of MQTT packages
670      *       received during the loop
671      *
672      *  (pt) A função loopCompleted é chamada quando a função loopSend
673      *       completa seu loop, retornando o número de pacotes MQTT
674      *       recebidos durante o loop
675      */
676     private void loopCompleted() {
677         runOnUiThread(new Runnable() {
678             @Override
679             public void run() {
```

```
680                 terminalOut.append("\n\nLoop completed.\n");
681                 try {
682                     Thread.sleep(500);
683                 } catch (InterruptedException e) {
684                     e.printStackTrace();
685                 }
686                 float successrate = cloopcnt*100 / loopSettings[0];
687                 String mqttSuccess = String.format("%.02f", successrate);
688 
689                 terminalOut.append("\n
    ----------------------------------------------------------\n" +
690                                    "   MQTT Packages recived during CLOOP: "
    + String.valueOf(cloopcnt) + "\n" +
691                                    "\n  Success Rate: " + mqttSuccess +
692                                    "%\n
    ----------------------------------------------------------\n\n");
693                 cloopcnt = 0;
694             }
695         });
696     }
697 
698     /**
699      *  (en) connectUsb is the function responsible to test if the
700      *       application have the permissions and ask for then if needed
701      *
702      *  (pt) A função connectUsb é a função responsável por testar se
703      *       o aplicativo tem as permissões e pedir por elas se necessário
704      */
705     private void connectUsb() {
706         UsbManager usbManager = getSystemService(UsbManager.class);
707         Map<String, UsbDevice> connectedDevices = usbManager.getDeviceList();
708         for(UsbDevice device : connectedDevices.values()) {
709             if(device.getVendorId() == 1155) {
710                 if (!usbManager.hasPermission(device)) {
711                     terminalOut.append("Please give the device permission to
    connect.\n");
712                     mPermissionIntent = PendingIntent.getBroadcast(this, 0, new
     Intent(ACTION_USB_PERMISSION), 0);
713                     usbManager.requestPermission(device, mPermissionIntent);
714                     try {
715                         Thread.sleep(2000);
716                     } catch (InterruptedException e) {
717                         e.printStackTrace();
718                     }
719                     if (!usbManager.hasPermission(device)){
720                         terminalOut.append("Error: Device has no permissions.")
    ;
```

```
721                         return;
722                     }else {
723                         SerialConnectionSetup(usbManager, device);
724                         break;
725                     }
726                 }else {
727                     SerialConnectionSetup(usbManager, device);
728                     break;
729                 }
730             }
731         }
732 
733         if (terminalOut.getText().toString().equals(startString)) {
734             terminalOut.setText("Device not found, please connect a supported
    device to continue...");
735             return;
736         }
737     }
738 
739     /**
740      *  (en) SerialConnectionSetup function is responsible to
741      *       configure the USB connection and try to make it
742      *
743      *  (pt) A função SerialConnectionSetup é responsável por
744      *       configurar a conexão USB e tentar estabelecê-lá
745      */
746     private void SerialConnectionSetup(UsbManager usbManager, UsbDevice device)
     {
747         try {
748             connection = usbManager.openDevice(device);
749             serial = UsbSerialDevice.createUsbSerialDevice(device, connection);
750 
751             if (serial != null && serial.open()) {
752                 serial.setBaudRate(115200);
753                 serial.setDataBits(UsbSerialInterface.DATA_BITS_8);
754                 serial.setStopBits(UsbSerialInterface.STOP_BITS_1);
755                 serial.setParity(UsbSerialInterface.PARITY_NONE);
756                 serial.setFlowControl(UsbSerialInterface.FLOW_CONTROL_OFF);
757                 serial.read(mCallback);
758 
759                 terminalOut.setText("Device connected: " + device.
    getProductName() + "\nProductID: " + device.getProductId() + "\nVendorID:" +
     device.getVendorId() + "\n\n");
760                 if(!ModeTest) {
761                     ModeTest = true;
762                 }
763                 serial.write("AT+MODE".getBytes());
```

```
764                 terminalOut.append("\n\n");
765             }
766 
767         } catch (Exception e) {
768             e.printStackTrace();
769             terminalOut.append("\n\n
    ----------------------------------------------------------------------------" +
770                                "\nError:\n" + e.toString() +
771                                "
    ----------------------------------------------------------------------------\n\n
    ");
772             Toast.makeText(getApplicationContext(), "Error: " + e.toString(),
    Toast.LENGTH_SHORT).show();
773         }
774     }
775 
776     private void releaseUsb(){
777         if (serial != null) {
778             serial.close();
779             if (loop) {
780                 loop = false;
781                 state = 0;
782             }
783             if (cloop) {
784                 cloop = false;
785             }
786             serial = null;
787             deviceFound = null;
788 
789             terminalOut.append("\n\n..... Device disconnected ......");
790         }
791     }
792 
793     @Override
794     protected void onDestroy() {
795         stop();
796         releaseUsb();
797         unRegisterMqtt();
798         unregisterReceiver(mUsbReceiver);
799         unregisterReceiver(mUsbDeviceReceiver);
800         super.onDestroy();
801     }
802 
803     @Override
804     protected void onPause() {
805         super.onPause();
806     }
```

```
807
808     @Override
809     protected void onResume() {
810         super.onResume();
811     }
812
813     private void registerMqtt () {
814         LocalBroadcastManager.getInstance(this).registerReceiver(
    mMessageReceiver, new IntentFilter("ttn-terminal-service-event"));
815     }
816
817     private void unRegisterMqtt() {
818         LocalBroadcastManager.getInstance(this).unregisterReceiver(
    mMessageReceiver);
819     }
820
821     private void registerGPS () {
822         LocalBroadcastManager.getInstance(this).registerReceiver(mGPSReceiver,
    new IntentFilter("ttn-mapper-gpspos-event"));
823     }
824
825     private void unRegisterGPS() {
826         LocalBroadcastManager.getInstance(this).unregisterReceiver(mGPSReceiver
    );
827     }
828
829     /**
830      *  (en) UsbReadCallback mCallback function is called when
831      *       the application receives a message via USB and
832      *       handle it
833      *
834      *  (pt) A função UsbReadCallback mCallback é chamada quando
835      *       a aplicação recebe uma mensagem via USB e faz seu
836      *       tratamento
837      */
838     UsbSerialInterface.UsbReadCallback mCallback = (data) -> {
839         String dataStr = null;
840         boolean exit = false;
841         try {
842             dataStr = new String(data, "UTF-8");
843             String finalDataStr;
844             if(ModeTest) {
845                 if (dataStr.toUpperCase().contains("OTAA")) {
846                     finalDataStr = "OTAA Detected....\n\n";
847                     Mode = "OTAA";
848                     ModeTest=false;
849
```

```
850                     new Thread(new Runnable() {
851                         @Override
852                         public void run() {
853                             try {
854                                 Thread.sleep(600);
855                             } catch (InterruptedException e) {
856                                 e.printStackTrace();
857                             }
858 
859                             Log.d(TAG, "OTAA: Starting JoinReq");
860                             joinReq();
861                         }
862                     }).start();
863                 }else if (dataStr.toUpperCase().contains("ABP")) {
864                     finalDataStr = "ABP Detected....\n\n";
865                     Mode = "ABP";
866                     ModeTest=false;
867                 }else {
868                     finalDataStr = "";
869                 }
870             }else if(ping) {
871                 if (dataStr.contains("Received")) {
872                     elapsedtime = System.currentTimeMillis() - startTime;
873                     finalDataStr = dataStr;
874                     pingReceived = true;
875                 }else if (pingReceived){
876                     finalDataStr = dataStr + "\nPing received in: " +
    elapsedtime +  "ms\n------------------------------------------";
877                     successPing++;
878                     mediumPingTime += elapsedtime;
879                     exit = true;
880                     pingReceived = false;
881                 }else if(dataStr.contains("ERROR")||dataStr.contains("Done") &&
     !pingReceived){
882                     finalDataStr = dataStr + "\nPing Failed\n
    ------------------------------------------";
883                     failedPing++;
884                     exit = true;
885                 }else {
886                     finalDataStr = dataStr;
887                 }
888             }else if (dataStr.toUpperCase().contains("NETWORK JOINED") ||
    dataStr.toUpperCase().contains("JOINED ALREADY")){
889                 finalDataStr = dataStr;
890                 join = true;
891             }else {
892                 finalDataStr = dataStr;
```

```
893             }
894
895             runOnUiThread(new Runnable() {
896                 @Override
897                 public void run() {
898                     terminalOut.append(finalDataStr);
899                 }
900             });
901
902             if(ping && exit) {
903                 doPing();
904             }
905
906         } catch (UnsupportedEncodingException e) {
907             Log.d(TAG, "Error: " + e.toString());
908             Toast.makeText(getApplicationContext(), "Error:\n" + e.toString(),
    Toast.LENGTH_SHORT).show();
909         }
910     };
911
912     /**
913      *  (en) mUsbReceiver is called when the device detects that
914      *       the modem was connected or disconneceted via OTG
915      *
916      *  (pt) mUsbReceiver é chamada quando o dispositivo detecta
917      *       que o modem foi conectado ou desconectadio via OTG
918      */
919     private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
920         @Override
921         public void onReceive(Context context, Intent intent) {
922             String action = intent.getAction();
923
924             if(ACTION_USB_PERMISSION.equals(action)) {
925
926                 synchronized (this) {
927                     UsbDevice device = (UsbDevice) intent.getParcelableExtra(
    UsbManager.EXTRA_DEVICE);
928
929                     if (intent.getBooleanExtra(UsbManager.
    EXTRA_PERMISSION_GRANTED, false)) {
930                         if (device != null) {
931                             connectUsb();
932                         }
933                     }
934                 }
935             }
936         }
```

```
937     };
938
939     private final BroadcastReceiver mUsbDeviceReceiver = new BroadcastReceiver
        () {
940         @Override
941         public void onReceive(Context context, Intent intent) {
942             String action = intent.getAction();
943
944             if (UsbManager.ACTION_USB_DEVICE_ATTACHED.equals(action)) {
945
946                 deviceFound = (UsbDevice) intent.getParcelableExtra(UsbManager.
        EXTRA_DEVICE);
947
948                 connectUsb();
949             }else if (UsbManager.ACTION_USB_DEVICE_DETACHED.equals(action)) {
950                 UsbDevice device = (UsbDevice) intent.getParcelableExtra(
        UsbManager.EXTRA_DEVICE);
951
952                 if (device != null) {
953                     if (device == deviceFound) {
954                         releaseUsb();
955                     }
956                 }
957             }
958         }
959     };
960 }
```

Listing A.1: Terminal Activity Source Code

ANEXO B – TTNMAPPERSERVICE - ADICIONADO AO CÓDIGO FONTE

```
@Override
public void onCreate() {
    LocalBroadcastManager.getInstance(this).registerReceiver(mGPSReq, new
    IntentFilter("ttn-mapper-gpsreq-service"));
}

private BroadcastReceiver mGPSReq = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        GPSPosReq();
    }
};

private void GPSPosReq() {
    //String msg = intent.getStringExtra("message");

    try {
        Intent gpsIntent = new Intent("ttn-mapper-gpspos-event");
        MyApplication myApplication = (MyApplication) getApplicationContext();
        Log.d(TAG, "GPSMap: Req Received");
        //add data
        gpsIntent.putExtra("message", "notification");

        if(myApplication.getLatestAcc() < 20 && myApplication.getLatestLat()
    !=0 && myApplication.getLatestLon() !=0) {
            Log.v(TAG,"-----------------------------------------\n" +
                    "Location:\nLat: " + myApplication.getLatestLat() + "\nLon:
    " + myApplication.getLatestLon() +
                    "\n-----------------------------------------\n");

            String lat = coordinateToHex(myApplication.getLatestLat());
            String lon = coordinateToHex(myApplication.getLatestLon());
            String alt = coordinateToHexUn(myApplication.getLatestAlt()/10);
            String acc = coordinateToHexUn(myApplication.getLatestAcc());

            gpsIntent.putExtra("lat", lat);
            gpsIntent.putExtra("dlat", Double.toString(myApplication.
    getLatestLat()));
            gpsIntent.putExtra("lon", lon);
            gpsIntent.putExtra("dlon", Double.toString(myApplication.
    getLatestLon()));
            gpsIntent.putExtra("alt", alt);
            gpsIntent.putExtra("dalt", Double.toString(myApplication.
```

```
    getLatestAlt()));
39              gpsIntent.putExtra("acc", acc);
40              gpsIntent.putExtra("dacc", Double.toString(myApplication.
    getLatestAcc()));
41              Log.d(TAG, "GPSMap: GPS Send");
42          }else {
43              gpsIntent.putExtra("lat", "not accurate enought");
    gpsIntent.putExtra("lon", "not accurate enought");
44              gpsIntent.putExtra("alt", "not accurate enought");
45              gpsIntent.putExtra("acc", "not accurate enought");
46              gpsIntent.putExtra("dacc", Double.toString(myApplication.
    getLatestAcc()));
47          }
48
49          LocalBroadcastManager.getInstance(this).sendBroadcast(gpsIntent);
50          Log.d(TAG, "\n-----------------------------------------------\n" +
51                  "              Send GPS:\n\nLat: " +  myApplication.
    getLatestLat() +
52                  "\nLon: "+ myApplication.getLatestLon() +
53                  "\nAlt: "+ myApplication.getLatestAlt() +
54                  "\nAcc: "+ myApplication.getLatestAcc() +    "\n
    -----------------------------------------------");
55      } catch (Exception e) {
56          e.printStackTrace();
57          Log.e(TAG, "\n
    -----------------------------------------------\nError: " + e.
    toString());
58      }
59  }
60
61  private String coordinateToHex(double in) {
62      String out = "";
63      in = in * Math.pow(10, 7);
64      int abs = (int) in;
65      if (abs < 0) {
66          abs = Math.abs(abs);
67          out = Integer.toString(abs, 16);
68          if(out.length() > 8) {
69              Log.d(TAG, "Coordinates Error: Hex String > 8 bytes.");
70          }
71          while (out.length() < 8) {
72              out = "0" + out;
73          }
74
75          out = "FF" + out;
76
77      }else {
```

```
78          out = Integer.toString(abs, 16);
79          if(out.length() > 8) {
80              Log.d(TAG, "Coordinates Error: Hex String > 8 bytes.");
81          }
82          while (out.length() < 8) {
83              out = "0" + out;
84          }
85
86          out = "00" + out;
87      }
88      return out;
89  }
90
91  private String coordinateToHexUn(double in) {
92      String out = "";
93      in = in * Math.pow(10, 7);
94      int abs = (int) in;
95      out = Integer.toString(abs, 16);
96      if(out.length() > 8) {
97          Log.d(TAG, "Coordinates Error: Hex String > 8 bytes.");
98      }
99      while (out.length() < 8) {
100         out = "0" + out;
101     }
102
103     return out;
104 }
105
106 private void sendToTerminal (String toSend) {
107     Intent intent = new Intent("ttn-terminal-service-event");
108
109     //add data
110     intent.putExtra("message", toSend);
111
112     LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
113     Log.d(TAG, "-------------------------------------------------" +
114                 "Send to terminal:\n" + toSend +  "
    -------------------------------------------------");
115 }
```

Listing B.1: Added to TTNMapperService Source Code

ANEXO C – MAPSACTIVITY - ADICIONADO AO CÓDIGO FONTE

```
@Override
public void onCreate() {
    LocalBroadcastManager.getInstance(this).registerReceiver(GPSStatsReq, new
    IntentFilter("ttn-mapper-gpsservice-event"));
}

private BroadcastReceiver GPSStatsReq = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        GPSStatsVer();
    }
};

private void GPSStatsVer() {
    Intent serviceStats = new Intent("ttn-mapper-gpsservice-answer");
    if(isMyServiceRunning(TTNMapperService.class)){
        serviceStats.putExtra("stats", "true");
    }else {
        serviceStats.putExtra("stats", "false");
    }
    LocalBroadcastManager.getInstance(this).sendBroadcast(serviceStats);
}
```

Listing C.1: Added to MapsActivity Source Code