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

Matheus Henrique Baumgarten Rabuske

ESTUDO DO PROTOCOLO LORAWAN E IMPLEMENTAÇÃO DE UMA REDE PRIVADA COM NÓS DISTRIBUÍDOS

Santa Maria, RS
2017

Matheus Henrique Baumgarten Rabuske

ESTUDO DO PROTOCOLO LORAWAN E IMPLEMENTAÇÃO DE UMA REDE PRIVADA COM NÓS DISTRIBUÍDOS

Trabalho de Conclusão de Curso 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 **Bacharel em Engenharia de Computação**.

ORIENTADOR: Prof. Carlos Henrique Barriquello

Santa Maria, RS
2017

RESUMO

ESTUDO DO PROTOCOLO LORAWAN E IMPLEMENTAÇÃO DE UMA REDE PRIVADA COM NÓS DISTRIBUÍDOS

AUTOR: Matheus Henrique Baumgarten Rabuske
ORIENTADOR: Carlos Henrique Barriquello

O presente trabalho mostra um estudo geral realizado sobre o LoRaWAN, um protocolo de longo alcance e baixo custo energético desenvolvido para a Internet das Coisas. Serão abordadas suas características principais, diferenças para outros protocolos comumente utilizados na área, arquitetura da rede, funcionamento, segurança, uso legal no Brasil e partes de *software* e *hardware* que o implementam. No final do trabalho, será apresentada a implementação ponta-a-ponta de uma rede LoRaWAN e o desenvolvimento de uma aplicação cliente-servidor simples que utiliza a rede implementada para comunicação bidirecional entre dois ou mais dispositivos.

Palavras-chave: Protocolo. Internet. Coisas. Comunicação. Alcance. Economia. Energia. Bidirecional.

ABSTRACT

STUDY OF THE LORAWAN PROTOCOL AND IMPLEMENTATION OF A PRIVATE NETWORK WITH DISTRIBUTED NODES

AUTHOR: Matheus Henrique Baumgarten Rabuske
ADVISOR: Carlos Henrique Barriquello

The present work shows a general study on LoRaWAN, a long-range, low-energy protocol developed for the Internet of Things. Its main characteristics, differences to other protocols commonly used in the area, network architecture, operation, security, legal use in Brazil and parts of software and hardware that implement it will be addressed. At the end of the work, will be presented the end-to-end implementation of a LoRaWAN network and the development of a simple client-server application that uses the implemented network for bidirectional communication between two or more devices.

Keywords: Protocol. Internet. Things. Communication. Range. Saving. Energy. Bidirectional.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet of Things
$P2P$	Peer-To-Peer
$LPWAN$	Low Power Wide Area Network
MAC	Media Access Control
OSI	Open System Interconnection
ISM	Instrumentation, Scientific and Medical
ADR	Adaptive Data Rate
ABP	Activation By Personalization
$OTAA$	Over The Air Activation
MIC	Message Integrity Checksum
DR	Data Rate
$DDRO$	Downlink Data Rate Offset
SPI	Serial Peripheral Interface
FSK	Frequency-Shift Keying
$JSON$	JavaScript Object Notation
TTN	The Things Network
$MQTT$	Message Queueing Telemetry Transport
$RSSI$	Received Signal Strenght Indication
SNR	Signal-to-Noise Ration
SSD	Solid-State Drive

SUMÁRIO

1 INTRODUÇÃO

A Internet das Coisas (*Internet of Things*, ou IoT), de acordo com Xia et al. (2012), é a área da tecnologia que trata da comunicação via rede entre os diversos dispositivos eletrônicos utilizados no dia a dia e que possuem algum grau de inteligência, como: sensores, *smartphones*, dispositivos de segurança, objetos de uso pessoal, etc. Devido ao fato de estarem interconectados e trocando informações entre si, esses dispositivos auxiliam em tarefas nas quais incluem-se:

- Automação e/ou gerenciamento de processos;
- Coleta de dados de um ambiente ou aplicação;
- Otimização de recursos empregados para a realização de tarefas;
- Predição de eventos;

De acordo com Cossini (2016), previsões indicam que o número de dispositivos conectados via rede ultrapassará a marca dos 20 milhões em 2020. Além disso, economicamente falando, no Brasil "[...] a estimativa é de 50 a 200 bilhões de dólares de impacto econômico anual em 2025 [causado pela uso da IoT]" (BNDES, 2017). Conforme essas previsões, a tendência é que a economia do país passe a ser fortalecida através dos avanços tecnológicos nessa área, como resultado da melhora na prestação de serviços em geral e otimização dos processos de produção (como, por exemplo, na área da agricultura, reduzindo perdas e melhorando a qualidade dos produtos agrícolas) causados pelo uso de dispositivos conectados entre si.

Uma das necessidades da IoT é que existam protocolos de comunicação para serem utilizados entre os dispositivos dentro de cada rede. Esses protocolos devem ser adequados dentro do contexto em que serão utilizados (como, por exemplo, em redes cujos nós são alimentados por energia proveniente de baterias, o protocolo escolhido deve utilizar um baixo custo de energia). Devem também proporcionar comunicação segura (criptografada), com o propósito de evitar que terceiros interceptem a comunicação, o que resultaria no risco de ocorrerem vazamentos indesejados de informações ou comportamento errôneo dos nós da rede causado por mensagens modificadas.

Como alternativa aos protocolos de comunicação disponíveis para IoT, foi criado recentemente o LoRaWAN, um protocolo LPWAN (*Low Power Wide Area Network*) que, conforme a classificação sugere, foi desenvolvido visando cobrir uma área relativamente grande (com raio de cobertura entre 3 km e 4 km em áreas urbanas, e de até 12 km em áreas rurais, de acordo com Junior (2016)) a um custo energético baixo.

Esse protocolo será utilizado como objeto de estudo deste trabalho, cuja motivação, objetivos e metodologia serão descritos a seguir.

1.1 MOTIVAÇÃO

Por ser um protocolo de longo alcance, o LoRaWAN é muito promissor dentro de áreas como, por exemplo, agricultura e pecuária, pois proporciona uma grande área de cobertura com o uso de um único *gateway* (receptor) para recebimento dos dados enviados pelos nós (transmissores) da rede, conforme será comentado nos capítulos seguintes. Entretanto, por se tratar de uma tecnologia nova, informações sobre esse protocolo e sobre implementações completas de redes que o utilizam ainda são escassas e estão fragmentadas pela *internet*. Deve-se considerar também que essas informações e implementações estão em constante mudança, e, em alguns casos, defasadas entre si, visto que os *softwares* e *hardwares* que compõem uma rede LoRaWAN, apesar de funcionais, ainda estão em desenvolvimento continuo por empresas entusiastas dessa tecnologia e membros de comunidades pró-LoRaWAN. Logo, uma informação que é válida hoje pode não ser mais válida amanhã - por exemplo, a especificação LoRaWAN pode ser mudada, fazendo com que as implementações de *software* e *hardware* também tenham que ser modificadas para atenderem as mudanças presentes nessa nova especificação.

Portanto, são necessários estudos sobre o protocolo e sobre a implementação completa de uma rede considerando o seu estado da arte atual, ou seja, usando o que existe atualmente nas implementações de redes que o utilizam. Esses estudos são necessários para que os pesquisadores, estudantes e entusiastas brasileiros de tecnologia também possam contribuir para a divulgação do protocolo e para o desenvolvimento de ferramentas que implementem e auxiliem seu uso, enriquecendo e ajudando na produtividade das áreas carentes de tecnologias LPWAN, futuramente.

1.2 OBJETIVOS

Com base nas motivações citadas na seção anterior, este trabalho tem como objetivos:

- Mostrar alguns protocolos comumente utilizados na IoT, como *background* para o item a seguir;
- Introduzir ao leitor o protocolo LoRaWAN e suas características, de forma que,

baseado no item anterior, o leitor possa entender sobre em quais aplicações seu uso se sobressai em relação aos demais protocolos utilizados na IoT;

- Mostrar o funcionamento do LoRaWAN, bem como as partes de *software* e *hardware*, a arquitetura geral de uma rede que o implementa e como o mesmo deve ser utilizado de forma legal no Brasil. Também será explicado, de modo geral, tópicos como o funcionamento da segurança no protocolo e modo de operação dos nós da rede, que são preocupações comuns em protocolos para IoT;
- Explicar uma implementação ponta-a-ponta de uma rede LoRaWAN privada e segura (criptografada), apresentando possibilidades de *software* a serem utilizados e mostrando quais o *softwares* e *hardwares* foram escolhidos para realizar a implementação, bem como suas configurações;
- Apresentar uma aplicação simples que utiliza a rede LoRaWAN implementada, comprovando ao leitor o funcionamento da mesma, bem como testes que demonstram o desempenho da rede em diferentes ambientes implementados;

1.3 METODOLOGIA

A metodologia utilizada na pesquisa para a realização deste trabalho segue a ordem em que os capítulos foram divididos no decorrer do mesmo.

No capítulo 2 serão mostrados alguns dos protocolos de comunicação disponíveis para serem utilizados na área da IoT, bem como os prós e contras de sua utilização e aplicações em que cada protocolo se destaca em relação aos demais, devido às suas particularidades.

No capítulo 3 será realizado o estudo do protocolo LoRaWAN em si, bem como suas diferenças em relação aos protocolos citados no capítulo 2, suas especificações técnicas, seu funcionamento e arquitetura de uma rede que o utiliza. Também será definida a sua utilização nos padrões de comunicação do Brasil, de forma que o mesmo seja utilizado legalmente dentro das especificações da Anatel.

No capítulo 4 serão feitos estudos e a implementação ponta-a-ponta de uma rede LoRaWAN totalmente privada (com dados acessíveis somente por pessoas autorizadas, sem o uso de locais de rede publicamente acessíveis), fazendo comparativos entre diferentes implementações de *softwares* que podem ser utilizadas e selecionando as melhores dentro do contexto deste trabalho, juntamente com os *hardwares* e configurações utilizadas na implementação;

Finalmente, no capítulo 5, será realizada a implementação e execução de uma aplicação utilizando a rede implementada no capítulo 4. Além disso, serão feitos tes-

tes de desempenho desta rede (privada) e de outra implementação de rede acessível através de uma plataforma disponível publicamente na *internet*.

2 ALGUNS PROTOCOLOS DE COMUNICAÇÃO PARA IOT

Antes de estudar o protocolo LoRaWAN, é interessante comentar sobre alguns outros protocolos possíveis de serem utilizados na área da IoT, a fim de, posteriormente, realizar um comparativo entre esses protocolos e o citado anteriormente. Para tal, foram escolhidos três protocolos *wireless* (ou seja, que usam o ar como meio de transmissão de dados), da camada de enlace de dados, com prós, contras e aplicações distintas entre si, que serão tratados a seguir neste capítulo.

2.1 WI-FI (802.11)

O Wi-Fi não é um protocolo de comunicação por si só, mas sim, um conjunto de várias versões ou extensões do protocolo 802.11. Sua primeira versão foi lançada em 1997 e foi chamada simplesmente de 802.11, ou 802.11 *Legacy*, de acordo com Alecrim (2013). Novas versões do protocolo foram lançadas nos anos seguintes, buscando principalmente um aumento na taxa de transmissão de dados, visto que, na versão *Legacy*, a taxa de transmissão era de apenas 1 Mbps ou 2 Mbps, conforme afirma Alecrim (2013). Também de acordo com o autor citado, em versões mais atuais do protocolo, como a 802.11n, lançada em 2009, a taxa de transmissão pode chegar a 300 Mbps ou 600 Mbps, dependendo da faixa de frequência utilizada.

De maneira geral, os protocolos Wi-Fi operam nas faixas de 2.4 GHz e 5 GHz, sendo que as faixas exatas de frequências possíveis e utilizadas variam entre as versões em uso. Sua topologia é na forma de estrela, onde múltiplos nós se conectam a um único *gateway* (ponto central da rede), o qual se conecta à *internet* usando qualquer outro protocolo (Ethernet via cabo, geralmente).

Os principais prós do protocolo são sua alta taxa de transmissão de dados e custo baixo de implementação, devido ao fato de ser uma tecnologia já consolidada no mercado. No entanto, o alto custo energético empregado na comunicação e baixo raio de cobertura (teoricamente em torno de 30 metros em ambientes fechados e 150 metros em ambientes abertos, apesar de na prática esses valores serem superestimados, conforme afirma Morimoto (2009)) são os principais contras de sua utilização. Sendo assim, na área de IoT, o protocolo Wi-Fi surge como uma opção interessante em aplicações que não exigem um raio de cobertura extenso e em que os nós da rede (sensores e afins) estão conectados à rede de energia elétrica.

Um exemplo de caso de uso que pode se encaixar dentro dos prós e contras do protocolo é o de uma automação residencial (onde os nós estão conectados na

tomada), principalmente pelo fato de, geralmente, já existir um *gateway* Wi-Fi instalado na residência, reduzindo o custo de implementação da rede e tornando necessária apenas a conexão dos sensores e atuadores na mesma. Também pode-se considerar casos em que o tráfego dos dados deve ocorrer de forma rápida, como em aplicações onde a detecção de um evento e atuação sobre o mesmo precisa acontecer em um intervalo de tempo muito pequeno.

2.2 BLE (BLUETOOTH LOW ENERGY)

Também conhecido como Bluetooth Smart, é a evolução do protocolo Bluetooth original e com foco na economia energética, conforme o nome sugere. Além do custo energético baixo, também possui 1 Mbps de taxa máxima de transmissão, com um raio máximo de transmissão de 50 metros e transmite na frequência de 2.4 GHz (logo, não utiliza faixa licenciada de frequência), de acordo com Pessoa (2016).

O BLE opera em três topologias de rede, como apontado por Pessoa (2016): P2P (*peer-to-peer*), onde é realizado um canal único de conexão entre dois dispositivos; estrela, onde vários dispositivos se conectam com um dispositivo central; e *mesh*, que é uma rede de várias estrelas interligadas entre si. Seu funcionamento consiste em passar maior parte do tempo em modo ocioso e só saindo desse modo no momento de uma transmissão. Sendo assim, para que a característica de baixo custo energético seja mantida, esse protocolo não deve ser utilizado em aplicações com fluxo intenso de dados; nesses casos, recomenda-se utilizar o Bluetooth clássico ou outro protocolo sem fio.

Os maiores prós do protocolo em relação aos demais são seu baixo custo energético e, de acordo com Pessoa (2016), financeiro, pois o custo do uso do mesmo é muito baixo. Por outro lado, seus contras ficam em relação ao baixo raio de cobertura e inviabilidade do seu uso em aplicações que exigem grande fluxo de dados (grande volume de dados trafegados em um intervalo de tempo pequeno).

Entre as aplicações onde esse protocolo se destaca, podem-se citar aquelas onde a comunicação entre dispositivos próximos ocorre de maneira esporádica e com baixo fluxo de dados, como no controle remoto de dispositivos residenciais ou automação dos mesmos, em que um ou mais dispositivos não possuem acesso à rede elétrica.

2.3 SIGFOX

Diferentemente dos protocolos anteriores, este protocolo é um protocolo fechado, ou seja, deve ser obtida uma licença, a partir de um provedor, para que sua utilização seja possível, semelhante às redes de telefonia celular. Ainda assim, utiliza a faixa de frequência não licenciada de 902-928 MHz nas Américas (variando em outras regiões), com canais com largura de banda de 600 Hz e taxa de transmissão de 600 bps, de acordo com WND Brasil (2017). Devido a essas características, o protocolo possui grande economia energética e uma área de cobertura muito maior em relação ao Wi-Fi e ao BLE, sendo entre 3 km e 10 km em áreas urbanas e entre 30 km e 50 km em áreas rurais, conforme citado por Santos et al. (200-?). Por outro lado, a taxa de transmissão dos dados é menor em relação às taxas dos protocolos citados anteriormente, sendo projetado para envio de pequenas mensagens. Cada transmissor pode não estar conectado à apenas uma estação de recepção (antena); os dados enviados pelo transmissor são recebidos por todas as estações que o tiverem em suas áreas de cobertura.

Sendo assim, sua área de cobertura e baixo consumo energético são seus maiores prós, enquanto a baixa velocidade de transmissão, necessidade de pagar por sua utilização e a não possibilidade de implementação de uma rede totalmente privada são seus maiores contras.

Como aplicações para esse protocolo encaixam-se aquelas onde transmissores estão espalhados a quilômetros de distância dos receptores, como em sensores meteorológicos espalhados pela cidade ou em sensores agrícolas espalhados em uma lavoura.

2.4 COMPARAÇÃO ENTRE OS PROTOCOLOS

Com base nas informações citadas, o quadro 2.1 foi criado, a fim de facilitar a comparação entre os protocolos pelo leitor. Esse quadro considera a frequência no qual cada protocolo é utilizado no Brasil, a taxa máxima de transmissão de dados, o máximo raio de cobertura considerando o melhor caso (onde há poucos ou nenhum obstáculo) e o gasto energético desses protocolos.

Quadro 2.1 – Comparação entre os dados dos protocolos citados no decorrer do capítulo

Protocolo	**Frequência**	**Taxa Transm.**	**Raio**	**Gasto Energético**
Wi-Fi (802.11n)	2.4 GHz	300 Mbps	150 m	Alto
BLE	2.4 GHz	1 Mbps	50 m	Baixo
Sigfox	902-928 MHz	600 bps	50 km	Baixo

Fonte: Autor.

3 O PROTOCOLO LORAWAN

O LoRaWAN é um protocolo LPWAN sem fio da camada MAC (*Media Access Control*, ou Controle de Acesso ao Meio) do modelo OSI (*Open System Interconnection*), com foco principal em aplicações da IoT que exigem grande área de cobertura e baixo custo energético, e em que alta taxa de transmissão de dados não é um requisito. Seu raio de cobertura pode chegar a até 4 km em áreas urbanas e a mais de 12 km em áreas rurais, conforme afirma Junior (2016), enquanto sua taxa de transmissão varia entre 300 bps e 50 kbps, dependendo da distância entre o transmissor e o receptor (*gateway*), de acordo com Committee (2017b). O protocolo foi criado pela LoRa Alliance, uma organização aberta e sem fins lucrativos composta por diversas empresas da área da IoT, e que é responsável também pelo lançamento de novas versões, *features* e especificações do mesmo.

Esse protocolo foi desenvolvido com base na utilização da tecnologia de modulação LoRa, criada e patenteada pela empresa americana Semtech, e utiliza faixas de frequências ISM (*(Instrumentation, Scientific and Medical*) não-licenciadas, ou seja, que podem ser utilizadas gratuitamente sem licenciamento prévio.

Para a transmissão dos dados, além da frequência, o protocolo LoRaWAN também utiliza variados *data rates*, resultantes de combinações entre diferentes larguras de banda (*bandwidth*) e diferentes fatores de espalhamento espectral (*spread factor*), o que reduz o risco de colisão dos dados, uma vez que, para que ocorra colisão, os dados devem chegar no receptor no mesmo intervalo de tempo, na mesma frequência, com mesma largura de banda e mesmo fator de espalhamento espectral. No entanto, ao variar-se o *data rate*, o alcance da comunicação mudará inversamente; ou seja, se o *data rate* for aumentado, o alcance da comunicação diminuirá, e vice-versa. As combinações entre frequências e *data rates* resultam nos canais que serão utilizados para a comunicação, onde um canal tem apenas uma faixa de frequência, mas pode utilizar um ou mais *data rates* diferentes. O canal a ser utilizado na transmissão é escolhido de forma pseudo-aleatória pelo dispositivo, conforme Committee (2017b).

Em comparação aos protocolos citados no capítulo 2, pode-se dizer que o LoRaWAN é uma alternativa de protocolo aberto em relação ao Sigfox, ainda que ligeiramente inferior em termos de área de cobertura. Também se sobressai em relação ao Wi-Fi na questão de economia energética, e em relação ao BLE, na questão de área de cobertura, apesar de ser inferior a ambos Wi-Fi e BLE no que diz respeito à taxa de transmissão de dados, de acordo com os valores apresentados.

Entre as aplicações nas quais esse protocolo é útil, estão aquelas em que não é necessária uma grande taxa de transmissão de dados, como, por exemplo, em casos onde pacotes de dados, como *status* do dispositivo, são enviados em períodos de,

pelo menos, alguns segundos. Outros casos em que o uso do protocolo é encorajado são aqueles onde os nós da rede situam-se em locais de difícil acesso, como áreas com vegetação densa ou de solo irregular, ou em ambientes rurais, onde é necessário cobrir uma área grande com o uso de poucos *gateways* devido a inviabilidade, do ponto de vista de infraestrutura, de instalar *gateways* de outros protocolos de menor alcance nesses locais.

Nas seções seguintes serão mostradas informações essenciais para a compreensão do funcionamento de uma rede LoRaWAN, e que foram obtidas utilizando como base, principalmente, o documento de especificações fornecido por Committee (2017b) e a *wiki* da Network (2017d).

3.1 CARACTERÍSTICAS

A seguir serão apontados as principais características do protocolo, dividindo-as entre prós e contras do mesmo.

3.1.1 Prós

- Cobre uma grande área enquanto mantém um baixo custo energético, em comparação aos demais protocolos comentados anteriormente;
- Comunicação bidirecional segura (criptografada);
- Uso gratuito (não são necessários os serviços de um provedor, ao contrário do Sigfox) e em faixa de frequência não-licenciada;
- *Softwares* desenvolvidos pela comunidade *open-source*, fazendo com que *bugs* sejam detectados e corrigidos e novas *features* implementadas em questão de semanas, em projetos ativos;
- Possui envio de mensagens com ou sem confirmação de entrega (*Acknowledgement*);
- Dados podem ser enviados em diferentes frequências, larguras de banda e fatores de espalhamento, reduzindo a probabilidade de ocorrência de colisões de dados e aumentando o desempenho do sistema em redes com muitos nós realizando transmissões simultaneamente;

3.1.2 Contras

- Tecnologia ainda em ascensão, portanto, o custo dos *hardwares* ainda é muito alto em comparação ao custo dos *hardwares* de outros protocolos. Por exemplo, os *gateways* considerados pela comunidade LoRaWAN como sendo de "baixo custo" são implementados utilizando uma Raspberry Pi 3, e apenas o custo deste dispositivo já é superior ao preço de um *modem* roteador Wi-Fi residencial comum;
- Nem todos os países possuem definição de uso específica, realizada pelo orgão regulamentador local, de faixas de frequência a serem utilizadas, como é o caso do Brasil, que será comentado posteriormente;
- No momento, todos os *gateways* possuem ou são baseados no *chip* SX1301 e suas evoluções, desenvolvidos somente pela Semtech; portanto, se a empresa encerrar a produção desses *chips*, o protocolo corre riscos de "morrer" por falta de *hardware* compatível;
- A baixa taxa de transmissão e tamanho máximo reduzido de dados a serem enviados em uma transmissão inviabilizam seu uso em aplicações onde pacotes grandes de dados precisam ser enviados, sob o risco de congestionar a rede devido às varias transmissões necessárias para realizar o envio de um pacote completo;

3.2 COMPONENTES E ARQUITETURA GERAL DA REDE LORAWAN

A fim de demonstrar com mais clareza o funcionamento da rede LoRaWAN, nesta seção serão explicados cada componente da rede e serão mostrados diagramas que demonstram o funcionamento dos messes componentes.

3.2.1 *Motes* (Nós da Rede)

"*Mote*" é o termo utilizado pela LoRa Alliance para se referir aos nós da rede LoRaWAN. Em geral, os *motes* são dispositivos que realizam alguma tarefa de aquisição e/ou envio de dados, e que são capazes de enviar dados (*uplinks*) e de receber dados (*downlinks*) através da rede LoRaWAN graças a um *chip* modulador conectado ao seu hardware. Esse *chip* é responsável pela modulação dos dados utilizando a técnica

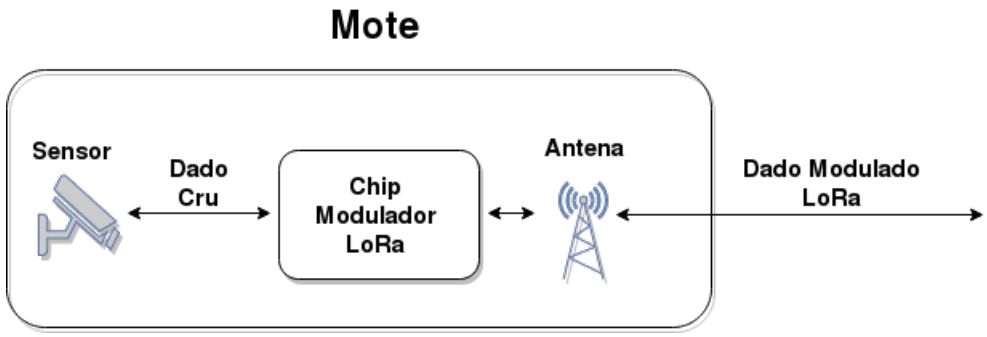
LoRa, antes da transmissão dos mesmos a partir de uma antena, também conectada ao *hardware* do *mote*.

Esses dispositivos possuem dois tipos de mensagens que podem ser enviadas: em um destes tipos, a mensagem é enviada e o *mote* aguarda o recebimento de um *downlink* chamado de ACK (*Acknowledgement*), que sinaliza ao *mote* que o *uplink* chegou corretamente ao seu destino, no caso, o *network server*; no outro tipo, a mensagem é enviada sem a necessidade do retorno de um ACK e, consequentemente, sem aguardá-lo.

Um recurso interessante é que, a fim de fazer com que os *motes* na rede possuam a máxima taxa de transmissão de dados possível, os mesmos dispõem do ADR (*adaptive data rate*), recurso onde o *network server* gerencia e varia os *data rates* e as potências das transmissões realizadas pelos *motes* dentro da rede, melhorando o desempenho da mesma como um todo.

O *mote* deve ser configurado previamente para operar dentro das especificações da rede LoRaWAN em funcionamento, ou seja, deverão ser especificados quais canais serão utilizados, juntamente com quais frequências e *data rates* esses canais operarão, assim como demais parâmetros relacionados à conexão com o *network server*, conforme será visto posteriormente neste trabalho.

Figura 3.1 – Diagrama simplificado do funcionamento de um *mote* na rede LoRaWAN



Fonte: Autor.

3.2.2 *Gateways*

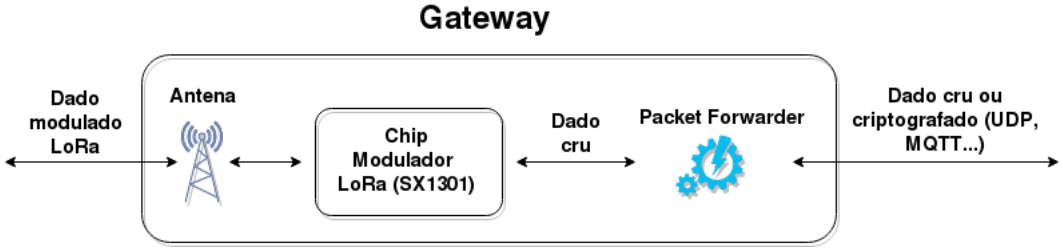
Os *gateways* (ou *concentrators*, como são chamados pela LoRa Alliance, apesar de tal denominação não ser comum entre os fóruns e artigos disponíveis) são os dispositivos responsáveis pela recepção dos pacotes modulados em LoRa, demodulação e redirecionamento dos mesmos pela *internet* para o *network server*, no caso de *uplink*, e pelo caminho reverso, no caso de *downlink*.

Esses dispositivos atuam como uma camada transparente na rede LoRaWAN, ou seja, os *gateways* redirecionam todo e qualquer pacote LoRa autêntico (sem erros ou malformações) que recebem dentro de sua área de cobertura. Sendo assim, os *motes* enviam *uplinks* sem saberem em qual ou quais *gateways* os mesmos foram recebidos, e recebem *downlinks* sem saberem a partir de qual ou quais *gateways* os mesmos foram enviados.

Um dos principais componentes desses dispositivos é o *chip* SX1301, citado anteriormente, que realiza a demodulação dos pacotes LoRa recebidos na antena conectada ao *gateway*. Outro componente essencial dos *gateways* é um *software* chamado de *packet forwarder*, que é encarregado de gerenciar, verificar e redirecionar pacotes LoRa demodulados para o *network server* (utilizando algum protocolo de rede implementado) definido em suas configurações, e de gerenciar, verificar e redirecionar os downlinks recebidos pelo *network server* para os *motes* corretos. Atualmente, existem diversas implementações de *packet forwarders*, das quais algumas serão comentadas no próximo capítulo.

Assim como no caso dos *motes*, o *gateway* deve ser configurado (através do *packet forwarder*) com os canais, frequências, *data rates* e fatores de espalhamento espectral que serão utilizados. Também deve ser configurado o endereço do *network server* para o qual serão redirecionados os *uplinks* e a partir do qual serão recebidos os *downlinks*.

Figura 3.2 – Diagrama simplificado do funcionamento de um *gateway* na LoRaWAN



Fonte: Autor.

3.2.3 *Network Server*

Esse componente é responsável pelo gerenciamento dos *uplinks*, *downlinks* e pelo funcionamento dos *gateways* e *motes* da rede como um todo. Trata-se de um computador ou servidor de uso geral (ou seja, não é e nem usa nenhum *hardware* específico para LoRaWAN) com acesso à *internet* ou, pelo menos, à mesma rede utilizada pelos *gateways*, embarcando uma implementação em *software* que realiza

as tarefas de gerenciamento da rede, como:

- Rotear *uplinks* para a *application* (contextos isolados nos quais os *motes* são inseridos) correta, contida dentro do *application server*, e *downlinks* para os *motes* corretos, selecionando o melhor *gateway* em relação ao *mote* para envio do *downlink*;
- Descriptografar a parte de controle dos *uplinks* (como a contagem de quadros, ou *frame counter*), validando-os e desprezando-os em caso de *uplinks* inconsistentes ou duplicados;
- Gerenciar os *data rates* e potências de transmissão dos *motes* que possuem ADR ativo dentro da rede;
- Gerenciar o envio dos *downlinks* com base nas aberturas das janelas de recepção dos *motes* destinos;
- No caso do *packet forwarder* da The Things Network, conforme será abordado posteriormente, o *network server* também é responsável pelo envio das configurações corretas dos canais para o *gateway*, de acordo com a região especificada;

Dependendo da implementação em *software*, o *network server* pode ser a parte de um sistema completo do qual também faz parte o *application server*, o qual será abordado a seguir.

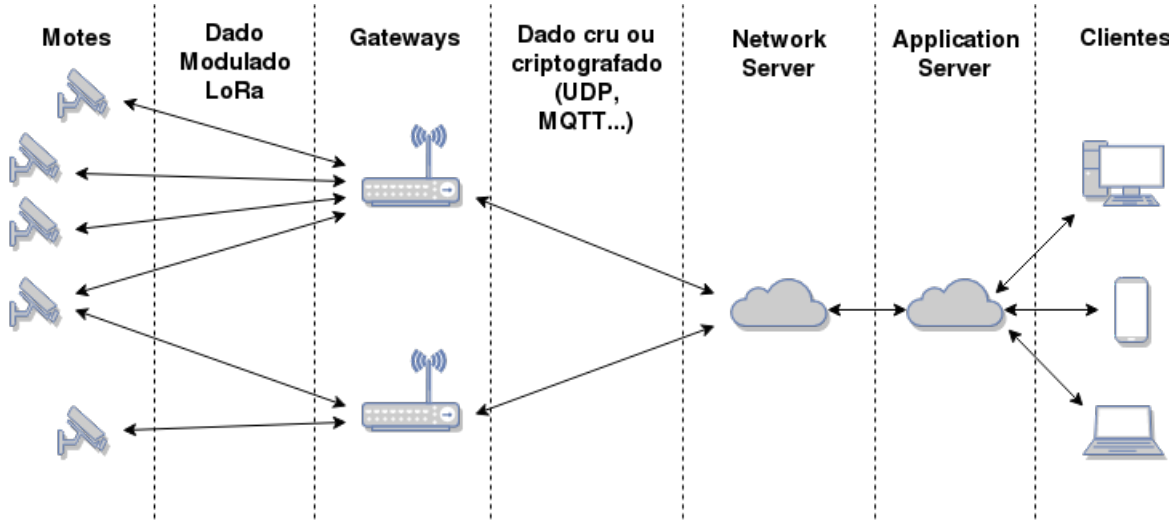
3.2.4 *Application Server*

Penúltima camada da rede LoRaWAN, o *application server* é o componente que irá tratar, descriptografar os *payloads* (dado enviado pelos *motes*, como uma medida ou informação) e disponibilizar os *uplinks* transmitidos pelos *motes*, preparar os *downlinks* a serem enviados para os mesmos e disponibilizar um ambiente de configuração e registro dos *gateways*, aplicações e *motes*, integrando-os à rede LoRaWAN. Este componente também não exige nenhum *hardware* específico para LoRaWAN, apesar de ser necessário utilizar implementações de software compatíveis com a implementação utilizada no *network server*.

O *application server* mantém os *payloads* dos *uplinks* e *downlinks*, e deve proporcionar alguma forma de acesso a esses dados a partir dos dispositivos clientes, através de chamadas de API, bibliotecas, acesso ao banco de dados, etc. Também deve gerar os ACKs necessários em caso de recepção de *uplinks* com requisição de confirmação de entrega, e chaves de criptografia para os dados no momento do registro dos *motes*.

A figura 3.3 mostra, de maneira geral, a arquitetura final da rede LoRaWAN, representando também a interação entre o *network server*, o *application server* e os dispositivos clientes.

Figura 3.3 – Diagrama geral da arquitetura de uma rede LoRaWAN



Fonte: Autor.

3.3 CLASSES DE *MOTES*

A LoRa Alliance define classes de *motes* que, de forma simplificada, são perfis de funcionamento dos mesmos que modificam a forma como o dispositivo gerencia suas janelas de recepção dos *downlinks*. Um *mote* pode ter entre uma e três classes diferentes implementadas, que podem ser modificadas durante o período de funcionamento do *mote* para atender algum cenário específico. As classes definidas, no momento, são as seguintes:

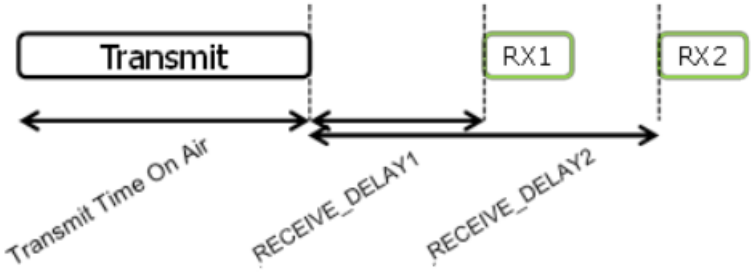
3.3.1 Classe A

Essa é a única classe que deve, obrigatoriamente, ser implementada nos *motes*. Também, é a classe mais econômica energeticamente, pois realiza a abertura de apenas duas janelas para recepção de *downlinks*, ambas após atrasos (configuráveis) desde o término do processo de modulação do *uplink* realizado pelo *mote*, conforme mostrado na figura 3.4.

Na primeira janela de recepção (RX1), a frequência e o *data rate* do canal de recepção são selecionados em função da frequência e do *data rate*, respectivamente,

do canal através do qual foi realizado o *uplink*. Por outro lado, na segunda janela (RX2), a frequência e *data rate* do canal de recepção são fixos. Os valores específicos para essas janelas variam, dependendo das especificações da LoRa Alliance para a região na qual o protocolo está sendo implementado.

Figura 3.4 – Ilustração do funcionamento das janelas de *uplink* e *downlink* em *motes* classe A



Fonte: Retirado de Committee (2017b).

Nessa classe, alguns detalhes importantes são:

- Dois quadros idênticos devem ser enviados pelo *gateway*, sendo um para cada janela de recepção do *mote*;
- Se um *downlink* for recebido e demodulado corretamente na primeira janela, a segunda janela não deve ser aberta, logo, o segundo quadro é ignorado;
- Após o envio de um *uplink*, o *mote* deve aguardar até o recebimento de algum *downlink* ou a expiração de suas duas janelas de recepção, para então poder enviar outro *uplink*;

3.3.2 Classe B

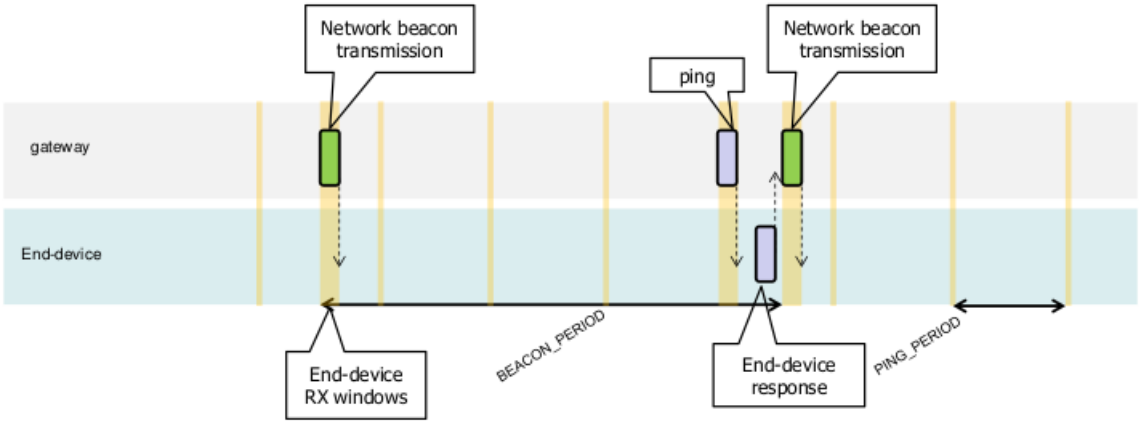
Conforme pôde-se concluir anteriormente, um dos maiores problemas da classe A é a imprevisibilidade, do ponto de vista do *network server*, dos intervalos de tempo em que os *motes* abrirão suas janelas de recepção. Isso faz com que os *downlinks* gerados a partir do *application server* fiquem enfileirados, enquanto *network server* aguarda a chegada de um *uplink*, sinalizando que o mesmo pode enviar os *downlinks* na fila de envio.

Com o propósito de resolver esse problema, a LoRa Alliance definiu, na última versão do protocolo divulgada até o momento (1.1), especificações para a classe B, onde o *mote*, além de abrir as duas janelas de recepção pertencentes à classe A,

também é capaz de abrir mais janelas de recepção em intervalos de tempo pré definidos, eliminando a necessidade de realizar *uplinks* para que possam ser recebidos *downlinks*, caso existentes. Por outro lado, o uso dessa classe resulta em um gasto energético maior por parte do *mote*, devido ao maior número de janelas de recepção abertas.

Para que *motes* operem nessa classe, os mesmos devem se conectar à rede operando como classe A. Os *gateways* então enviam, via *broadcast*, uma referência de tempo para os *motes* na rede chamada de *beacon*. Depois de receberem o *beacon*, *os motes* passam a operar na classe B, abrindo pequenas janelas de recepção, chamadas de *ping slots*, com base nessa mesma referência temporal enviada pelo *gateway*. Dessa forma, o momento da abertura desses *ping slots* se torna previsível para o *network server*, possibilitando que *downlinks* sejam enviados também nesses *ping slots* e eliminando a necessidade do *mote* realizar *uplinks* para receber os *downlinks*, conforme ilustrado a figura 3.5. A periodicidade, frequência e *data rate* dos *ping slots* são configuráveis a partir do *network server*.

Figura 3.5 – Ilustração do funcionamento das janelas de *uplink* e *downlink* e dos *ping slots* em *motes* classe B



Fonte: Retirado de Committee (2017b).

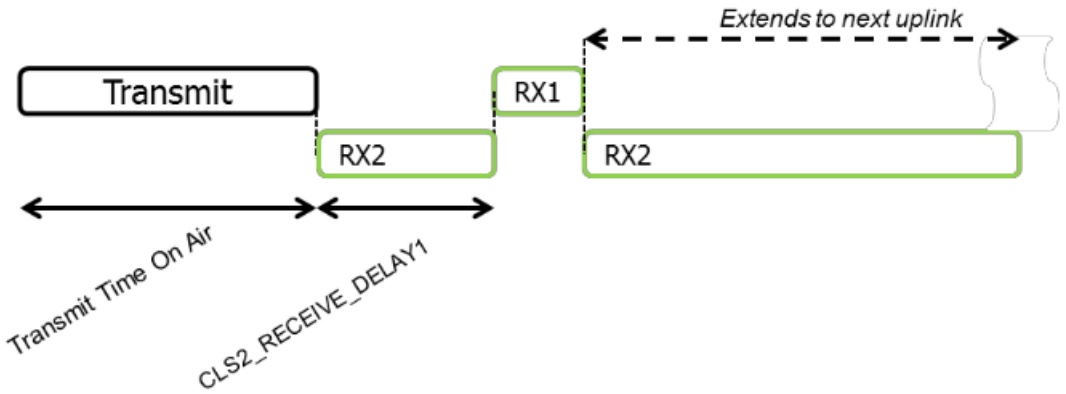
3.3.3 Classe C

Essa classe é a mais simples, caracterizando um modo de operação onde o *mote* mantém uma janela de recepção aberta durante todo o tempo, exceto durante o processo de transmissão de *uplinks*. Por conta disso, o consumo energético por parte dos *motes* que utilizam essa classe é o maior entre as três classes, sendo aconselhável utilizá-lo apenas em *motes* conectados a baterias com grande quantidade de energia armazenada ou à rede elétrica, ou em casos onde um gasto energético

maior já é esperado e não é um obstáculo para a aplicação. Além disso, *motes* que implementam a classe B não devem implementar a classe C.

O funcionamento dessa classe é baseado no uso das janelas RX1 e RX2, também utilizadas na classe A. A diferença fica por conta da duração da RX2: essa janela fica aberta por todo o tempo, exceto durante os *uplinks* (único momento em que nenhuma janela de recepção está aberta) e o intervalo em que RX1 está aberta. Ou seja, RX2 é aberta no intervalo entre o término da modulação do *uplink* e da abertura de RX1, e entre o término da RX1 e o ínicio de outro *uplink*. A figura 3.6 demonstra esse procedimento de maneira mais clara.

Figura 3.6 – Ilustração do funcionamento das janelas de *uplink* e *downlink* em *motes* classe C



Fonte: Retirado de Committee (2017b).

3.4 SEGURANÇA

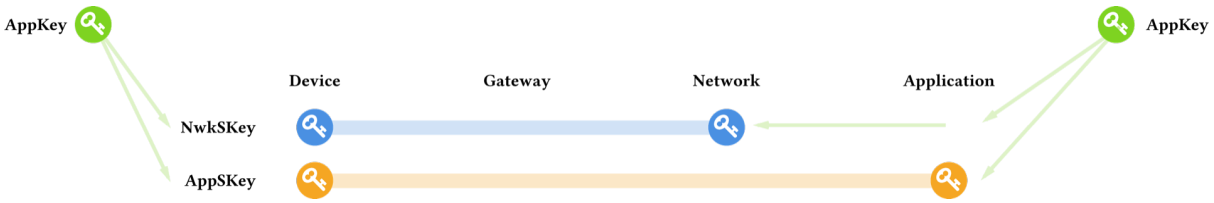
Conforme citado anteriormente neste trabalho, a comunicação na rede LoRaWAN é criptografada, ou seja, após o dado ser enviado pelo *mote*, ele somente será descoberto pelo *application server*. Isso acontece graças à utilização de três chaves de 128 *bits* distintas:

- AppKey, que é conhecida somente pelo *mote* e pelo *application server*. Utiliza o algoritmo AES-128, e é utilizada apenas em dispositivos ativados utilizando OTAA para, durante a ativação do *mote*, criar as duas chaves que serão abordadas a seguir;
- NwkSKey, conhecida pelo *mote* e pelo *network server*, que serve para criptografar e descriptografar os dados do pacote relacionados ao MIC (*Message Integrity*

Checksum), que validam a integridade do pacote. A criptografia realizada com essa chave utiliza o algoritmo AES-128;

- AppSKey, conhecida pelo *mote* e pelo *application server*, que é utilizada para criptografar e descriptografar o *payload* do pacote. A criptografia realizada com essa chave utiliza o algoritmo AES-CMAC;

Figura 3.7 – Funcionamento geral das chaves de criptografia da rede LoRaWAN



Fonte: Retirado de Network (2017d).

No caso da definição 1.1 da LoRaWAN, além da AppKey mencionada acima, também é utilizada a NwkKey, que é usada para gerar outras chaves com o propósito de tornar a rede ainda mais segura. No entanto, devido ao fato de a definição 1.1 ser muito nova, fazendo com que poucos dispositivos a implementem, a NwkKey não será abordada nesse trabalho.

3.5 ATIVAÇÃO DE *MOTES*

Para que trafeguem dados dentro da rede LoRaWAN, os *motes* devem ser registrados em aplicações no *application server* para, então, serem ativados. Cada *application* possui um identificador único de 64 *bits* chamado de AppEUI, ao qual o *mote* é vinculado durante seu registro. Após o registro, é gerado um identificador único para o *mote*, também de 64 *bits*, chamado de DevEUI. O AppEUI e o DevEUI serão utilizados pelo *network server* e pelo *application server*, para que o *mote* seja identificado durante a ativação e seja gerado para o mesmo um endereço de 32 *bits* chamado de DevAddr, usado na identificação e roteamento dos *uplinks* e *downlinks*.

Um *mote* pode operar ativado somente para uma *application*, porém, podem haver diferentes registros para um *mote* no *application server*, bastando que o mesmo seja reconfigurado e reativado para operar em uma *application* diferente. Considerando a definição 1.0 da LoRaWAN, existem dois processos de ativação, os quais que serão abordados a seguir.

3.5.1 ABP

ABP (*Activation By Personalization*) é a forma mais simples de ativação dos *motes*. Ela consiste em realizar a ativação manualmente no *application server*, gerando as chaves de criptografia (NwkSKey e AppSKey) e inserindo-as manualmente nos *motes*, juntamente com seus DevAddr e DevEUI gerados e AppEUI ao qual estão vinculados. Por outro lado, esse tipo de ativação torna o *setup* inicial da rede mais trabalhoso em caso de redes maiores, uma vez que cada *mote* deverá ser configurado manualmente. Além disso, esse método não é tão seguro, pois as chaves de criptografia permanecem iguais até que sejam recriadas e reinseridas, também manualmente.

3.5.2 OTAA

A OTAA (*Over The Air Activation*) é uma forma de ativação mais complexa do que a ABP, porém, também é mais segura e prática, pois, ao utilizá-la, os *motes* são capazes de realizar suas próprias ativações. Para isso, antes do processo de ativação, deve ser gerado um DevEUI, assim como na ABP, e inseri-lo no *mote*, juntamente com o JoinEUI, que é o identificador único do *Join Server* e, em alguns casos, como na plataforma aberta da *The Things Network*, é igual ao AppEUI. Também se faz necessária a inserção da AppKey no *mote*, utilizando-se para isso alguma conexão criptografada com outro protocolo (por exemplo, Bluetooth), para que não ocorra o risco da mesma ser interceptada e a AppKey descoberta por terceiros, ou inserindo-a manualmente no *mote*, sendo essa a opção mais segura, ainda que seja menos prática em casos de *deploy* de *motes* em massa.

No caso da OTAA, o DevAddr é gerado durante o processo de ativação remota do *mote*, assim como as chaves de criptografia, que são derivadas da AppKey e trocadas entre o *application server* e o *mote*. Um mesmo *mote* pode ser ativado periodicamente, dependendo de sua configuração. Dessa forma, a cada ativação, novas chaves de criptografia são geradas (ainda que a AppKey permaneça igual), o que explica a afirmação anterior sobre a OTAA ser mais segura do que a ABP.

3.6 USO DA LORAWAN NO BRASIL

Conforme comentado anteriormente, a rede LoRaWAN utiliza faixas do espectro de frequência que não necessitam de licenciamento prévio. Entretanto, é importante observar que LoRaWAN não opera somente em uma faixa específica; cada país

ou região tem sua própria faixa de frequência não-licenciada onde a rede LoRaWAN pode operar. Sendo assim, esta seção visa estudar e definir a faixa em que se pode utilizar a rede LoRaWAN no Brasil, visto que ainda não existe nenhuma definição oficial por parte da Anatel ou da LoRa Alliance sobre o assunto.

3.6.1 Definição da Faixa de Frequência

De acordo com Network (2017a), a faixa de frequência ser utilizada no Brasil é a mesma que é utilizada nos Estados Unidos, ou seja, a faixa compreendida entre 902 MHz e 928 MHz. Esse dado serve como um bom ponto de partida para a pesquisa, porém, a faixa indicada por Network (2017a) tem como justificativa a resolução 506 da Anatel (2017a), que foi revogada e, em seu lugar, passou a operar a resolução 680 (Anatel (2017b)).

A faixa de frequência definida pela Network (2017a) não está dentro das faixas de radiofrequência com restrições de uso para dispositivos de radiação restrita, conforme o artigo 7º da resolução 680 da Anatel (2017b), o que serve como indício de que a faixa pode ser utilizada. Entretanto, essa faixa não pode ser completamente utilizada. Existe uma subfaixa, situada entre as frequências de 907.5 MHz e 915 MHz, destinada para uso licenciado. Por este motivo, para que a faixa definida para os Estados Unidos possa ser utilizada no Brasil, os *gateways* e *motes* que a utilizarem devem ser configurados levando em conta essa restrição. Também é importante comentar que, conforme o Ato nº 11542 citado na resolução 680, o tempo médio de uso da frequência em dispositivos que utilizam tecnologia de espalhamento espectral (que é o caso dos dispositivos que utilizam modulação LoRa), nas faixas de 902-907.5 MHz e 915-928 MHz, não deve ser maior do que 400 ms durante períodos de 14 e 7 segundos, em casos de canais com largura de banda menor do que 250 kHz e largura de banda situada entre 250 kHz e 500 kHz, respectivamente.

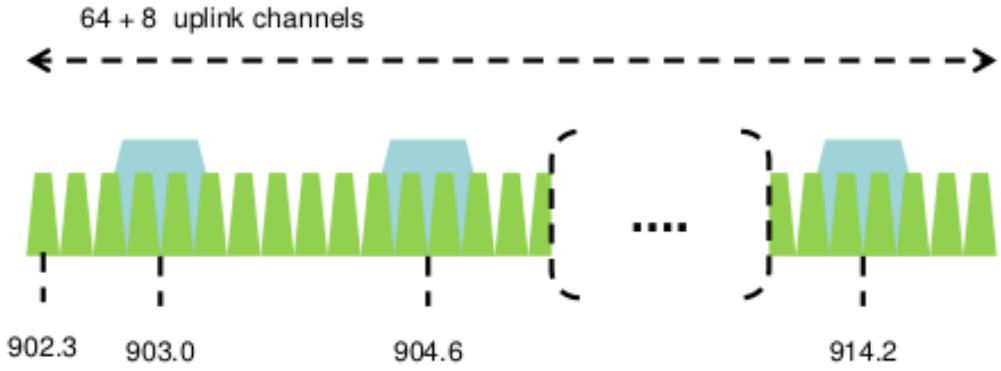
3.6.2 Especificações de Uso

A fim de regulamentar as especificações de uso da LoRaWAN dentro das restrições regionais, foi desenvolvido pela Committee (2017a) um documento com as especificações para diversas regiões, incluindo a dos Estados Unidos, que também pode ser utilizada no Brasil, conforme explicado anteriormente.

Sendo assim, de acordo com as especificações dos Estados Unidos, as seguintes definições com relação a canais de *uplink* e *downlink* são feitas:

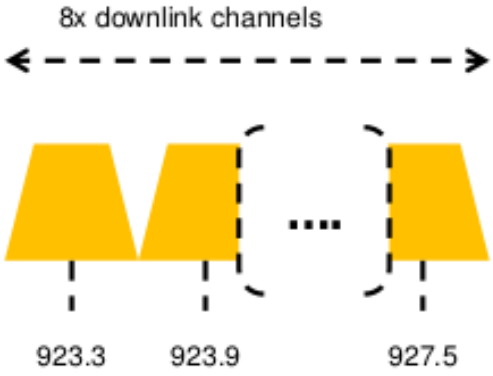
- Para *uplinks*, podem ser utilizados 64 canais, começando em 902.3 MHz e aumentando 200 kHz até 914.9 MHz, com largura de banda de 125 kHz e utilizando *data rates* de 0 a 3, sendo a numeração dos *data rates* atribuída com base nas combinações entre fatores de espalhamento e frequências indicadas no quadro 3.1. Adicionalmente, também podem ser utilizados outros 8 canais, começando em 903 MHz e aumentando 1.6 MHz até 914.2 MHz, com largura de banda de 500 kHz e utilizando *data rate* 4. É valido lembrar que, no Brasil, a faixa entre 907.5 MHz e 915 MHz é licenciada, logo, não deve ser utilizada;
- Para *downlinks*, podem ser utilizados 8 canais, começando em 923.3 MHz e aumentando 600 kHz até 927.5 MHz, com largura de banda de 500 kHz e utilizando *data rates* de 8 a 13. Essa especificação pode ser utilizada normalmente no Brasil;

Figura 3.8 – Diagrama dos canais de *uplink* no padrão dos Estados Unidos



Fonte: Retirado de Committee (2017a).

Figura 3.9 – Diagrama dos canais de *downlink* no padrão dos Estados Unidos



Fonte: Retirado de Committee (2017a).

O quadro 3.1 mostra as configurações de fator de espalhamento e frequência que compõem a numeração de cada *data rate*, bem como suas taxas de transmissão relacionadas, enquanto o quadro 3.2 mostra o tamanho máximo do *payload* enviado em cada *data rate*, considerando que o *mote* não irá operar com repetidor.

De acordo com a especificação, devem ser utilizados pelo menos 4 canais e a potência máxima de transmissão do *mote* não deve ser superior a 21 dBm. A respeito das especificações das janelas de *downlink*, a única janela com configuração fixa é a RX2, que opera na frequência de 923.3 MHz e DR8 (*data rate 8*). A RX1 tem canal e *data rate* dependentes do canal e do *data rate* utilizados no *uplink* realizado antes da abertura das janelas; o número do canal é definido com base no resto da divisão entre o número do canal do *uplink* e 8, enquanto o *data rate* é definido com base no quadro 3.3, onde DDRO significa *Downlink Data Rate Offset* e indica o *data rate* resultante dependendo de um *offset* configurado no *network server*. Por exemplo, se o uplink foi enviado com canal 8, em *data rate* 2 e o *offset* configurado é 0, a RX1 vai abrir com canal 0 e em *data rate* 12.

Para a transmissão dos *beacons* da classe B, o *data rate* utilizado é o 8, enquanto a escolha do canal depende das configurações de temporização dos *beacons* e não será abordada nesse trabalho.

Quadro 3.1 – Relação das configurações de cada *data rate* utilizável

Data Rate	**Fator de Espalhamento e Frequência**	**Taxa de Transmissão**
0	10 e 125 kHz	980 *bits*/s
1	9 e 125 kHz	1760 *bits*/s
2	8 e 125 kHz	3125 *bits*/s
3	7 e 125 kHz	5470 *bits*/s
4	8 e 500 kHz	12500 *bits*/s
5-7	Reservado	
8	12 e 500 kHz	980 *bits*/s
9	11 e 500 kHz	1760 *bits*/s
10	10 e 500 kHz	3900 *bits*/s
11	9 e 500 kHz	7000 *bits*/s
12	8 e 500 kHz	12500 *bits*/s
13	7 e 500 kHz	21900 *bits*/s
14	Reservado	
15	Definido com Comando MAC	

Fonte: Adaptado de Committee (2017a).

Quadro 3.2 – Relação dos tamanhos máximos de *payload* em cada *data rate* utilizável

Data Rate	**Tamanho Máximo do *Payload***
0	19 *bytes*
1	61 *bytes*
2	133 *bytes*
3	250 *bytes*
4	250 *bytes*
5-7	Não Definido
8	61 *bytes*
9	137 *bytes*
10	250 *bytes*
11	250 *bytes*
12	250 *bytes*
13	250 *bytes*
14-15	Não Definido

Fonte: Adaptado de Committee (2017a).

Quadro 3.3 – Relação dos *data rates* utilizados na RX1, com base no *data rate* do *uplink* e em um *offset* configurável

***Uplink* DR**	**DDRO 0**	**DDRO 1**	**DDRO 2**	**DDRO 3**
0	10	9	8	8
1	11	10	9	8
2	12	11	10	9
3	13	12	11	10
4	13	13	12	11

Fonte: Adaptado de Committee (2017a).

Além das informações citadas no decorrer desta subseção, a LoRa Alliance também sugere algumas outras parametrizações a serem realizadas, como os tempos de abertura da RX1 e RX2 sendo 1 e 2 segundos após o término do *uplink*, respectivamente. No entanto, caso o *mote* tenha parâmetros diferentes dos especificados, os mesmos devem ser configurados igualmente no *network server*. Caso o *mote* e o *network server* possuam configurações diferentes, a comunicação pode não funcionar conforme esperado, ou, talvez, nem mesmo ocorrer comunicação.

4 IMPLEMENTAÇÃO DA REDE LORAWAN

Após a fundamentação teórica realizada nos capítulos anteriores, neste capítulo serão realizados a implementação da rede LoRaWAN, bem como testes utilizando-a, a fim de extrair dados de desempenho da rede.

4.1 *MOTE*

O *mote* a ser utilizado será o modelo RHF3M076, da RisingHF, devido à facilidade de configurá-lo, pois o mesmo tem uma porta mini USB que pode ser utilizada para conectá-lo a um computador. Dessa forma, a configuração do *mote* pode ser feita pelo computador, através de comandos AT. Esse modelo suporta as classes A e C de operação, cobrindo várias faixas de frequência diferentes (incluindo a 902-928 MHz que será utilizada) e permitindo uma potência de transmissão de até 20 dBm em faixas de frequências baixas e 14 dBm em faixas de frequências altas.

Figura 4.1 – Imagem de um *mote* do modelo RHF3M076

Fonte: Retirado de RisingHF (2017b).

A configuração do *mote* foi realizada utilizando um *script* Python, definindo os valores descritos a seguir. As demais configurações que não foram citadas, foram mantidas com seus valores padrão.

- ***Uplinks* e *Downlinks*:** A configuração dos canais de *uplink* foi realizada com base nos valores sugeridos para a faixa US902-928 MHz pela Network (2017b), sendo gerados 8 canais com *data rate* entre 0 e 3, e um canal com *data rate* 4. Os demais canais não eram úteis e foram desativados.

  Um fato interessante é que a faixa utilizada compreende valores de frequência

entre 903.9 e 905.3 MHz, que se encaixa normalmente dentro da faixa não licenciada brasileira.

Em relação ao *downlink*, não foi possível realizar a configuração correta da RX1, de forma que a mesma operasse sem problemas. Portanto, essa janela foi desativada, ficando ativa somente a RX2, que foi configurada na frequência de 923.3 MHz e *data rate* 8, e que operou da forma esperada. Sua configuração define que a mesma é aberta 2 segundos após o final do *uplink*.

Os *data rates* utilizados (0 a 4, e 8) foram configurados conforme o quadro 3.1, atendendo as especificações da LoRa Alliance. Como *data rate default*, foi escolhido arbitrariamente o *data rate* 0.

- **ADR Habilitado:** Sim;
- **Potência de Transmissão:** 14 dBm, por ser o maior valor possível para ser utilizado em faixas de frequência altas e por estar dentro do limite especificado pela LoRa Alliance para a faixa utilizada;
- **Classe:** A. Essa classe é a que mais se aproxima de um caso de uso real da LoRaWAN, onde o gasto energético por parte do *mote* deve ser pequeno. O uso da classe C, apesar de mais prático, resultaria em um consumo energético muito alto;
- **Modo de Ativação:** ABP. OTAA é mais seguro e mais prático em casos de redes extensas, mas no caso deste trabalho, o uso do ABP agiliza o *deploy* das configurações do *mote* utilizando chaves de criptografia já criadas;

4.2 GATEWAY

Devido ao alto custo dos *gateways* comerciais desenvolvidos especificamente para LoRaWAN, nesta implementação será utilizado um *gateway* composto pelo módulo *concentrator* RHF0M301, da RisingHF, conectado em uma Raspberry Pi 3, ambos fornecidos pelo professor orientador deste trabalho. Esse módulo é responsável pela recepção e demodulação LoRa (ou modulação LoRa e transmissão) dos pacotes LoRaWAN recebidos do (ou enviados para o) *gateway* via protocolo SPI (*Serial Peripheral Interface*). O *gateway*, por sua vez, transmite (ou recebe) os pacotes através de algum protocolo da *internet*.

Assim como o *mote* RHF3M076, esse módulo *concentrator* opera em diversas faixas de frequências, atendendo diferentes especificações regionais, incluindo a

faixa 902-928 MHz que será utilizada. Também possui 8 canais de *uplink* configuráveis, além de mais um canal LoRa padrão e um canal FSK (*Frequency-Shift Keying*, utilizado em algumas regiões).

Figura 4.2 – Imagem de um módulo *concentrator* do modelo RHF0M301

Fonte: Retirado de RisingHF (2017a).

Após a montagem do *hardware* do *gateway*, é necessário realizar a instalação de algum *packet forwarder*, que redirecionará os pacotes LoRaWAN dos *motes* para o *network server*, e vice-versa. Existem algumas alternativas de *packet forwarder* possíveis de serem utilizadas atualmente, como:

- ***Semtech Packet Forwarder*:** É o pioneiro dos *packet forwarders*, tendo sido desenvolvido (e sendo mantido até hoje) pela própria Semtech, na linguagem C. Seu código-fonte está disponível no Github[1], permitindo *deploy* do mesmo para diversas arquiteturas diferentes, incluindo a arquitetura utilizada na Raspberry Pi 3.

  Esse *packet forwarder* possui um arquivo de configurações, na qual é especificado o ID único do *gateway* (registrado no *application server*), bem como configurações de canais e potência de transmissão do *gateway* e endereço do *network server* para o qual serão redirecionados os pacotes LoRaWAN recebidos. Os dados contidos nos *uplinks* e *downlinks* são enviados em uma estrutura JSON (*JavaScript Object Notation*).

  É uma opção interessante, porém, tem como seus maiores defeitos o uso do protocolo UDP e a utilização de uma conexão descriptografada para envio (e

[1] Disponível em: https://github.com/Lora-net/packet_forwarder

recepção) dos dados para o *network server*, de acordo com Network (200-?). Logo, não há nenhuma confirmação de integridade dos dados que chegaram no *network server* a partir do *gateway*, devido ao uso do UDP, e qualquer dispositivo que interceptar a conexão poderá descobrir quais dados estão sendo trafegados; mesmo que os dados estejam criptografados pelo protocolo LoRaWAN, alguém que intercepte a comunicação pode tentar encontrar brechas na criptografia das mensagens utilizada pela LoRaWAN (como, por exemplo, encontrar padrões ou detectar momentos em que uma mensagem diferente das demais é enviada, sinalizando a ocorrência de um evento ou exceção) ou, até mesmo, enviar mensagens modificadas com o intuito de prejudicar o funcionamento da rede. Sendo assim, esse *packet forwarder* é uma opção viável somente em casos de pesquisa ou prototipagem, ou em casos onde o mesmo é utilizado como ponte entre o *concentrator* e algum outro software com o propósito de enviar (ou receber) os dados que saem do (ou entram no) *packet forwarder* em outro protocolo mais robusto e seguro. Outro ponto negativo é a falta de documentação dos parâmetros utilizados no arquivo de configurações, pois não foi encontrada nenhuma documentação oficial para esse arquivo, obrigando o desenvolvedor a deduzir informações a partir dos arquivos-modelo existentes ou em fóruns na *internet*.

Existem outras implementações de *packet forwarder* que são derivadas do *Semtech Packet Forwarder*, buscando melhorias como a utilização de outros protocolos de comunicação com o *network server*, mas todos acabam também afetados pela falta de documentação. Sendo assim, essas versões derivadas não serão abordadas neste trabalho.

- ***TTN Packet Forwarder*:** Também conhecido como *The Things Network Packet Forwarder*, foi desenvolvido pela The Things Network Core Team na linguagem Go. Foi criado para corrigir os defeitos do *Semtech Packet Forwarder*, como fraca confiabilidade da entrega de dados e segurança baixa, e, para isso, utiliza um protocolo que também foi criado pela TTN Core Team, chamado de *Gateway Connector Protocol*. Esse protocolo, de acordo com a Network (200-?), fornece conexão criptografada entre o *gateway* e o *network server* utilizando gRPC ou MQTT, além de fornecer autenticação de *gateways*, impedindo que *gateways* não autenticados enviem dados para o *network server*.

  Apesar de utilizável, esse *packet forwarder* não possui compatibilidade com os demais *network servers* senão o da TTN, sobre o qual será comentado posteriormente. Outro ponto negativo é que, de acordo com Team (2017a), o desenvolvimento do *packet forwarder* foi paralisado em setembro de 2017. Apesar disso, seu código fonte está disponível no Github[2], junto com tutoriais para a realização

[2]Disponível em: https://github.com/TheThingsNetwork/packet_forwarder

de seu *deploy* nos principais *gateways* existentes, incluindo naqueles baseados em Raspberry Pi 3.

Com base nas características de cada *packet forwarder* comentadas, o escolhido será o *Semtech Packet Forwarder*, pela facilidade de realizar seu *deploy* e por já existir um arquivo de configurações para a faixa 902-928 MHz, disponível no Github[3] pelo The Things Network Core Team. Essas configurações definem como são os canais usados pelo *gateway* e são iguais às especificadas na configuração do *mote*, portanto, não serão comentadas novamente.

4.3 NETWORK SERVER E APPLICATION SERVER

Nessa seção, serão mostradas duas implementações diferentes: na primeira delas, existem implementações individuais para o *application server* e para o *network server*, sendo algo próximo da arquitetura sugerida pela LoRa Alliance. Na segunda, o *network server* e o *application server* podem ser consideradas como *features* de um único grande *software*. São os seguintes:

- ***LoRa Server* e *LoRa App Server*:** Ambos *softwares* são implementações de código aberto, partes de uma mesma solução. Seus códigos fonte estão no Github[4], e foram criados por Orne Brocaar, com contribuição da comunidade *open-source*.

  O *LoRa Server* é a implementação correspondente ao *network server*, e os dados são trafegados entre o mesmo e o *gateway* usando o protocolo MQTT (*Message Queueing Telemetry Transport*). Seu funcionamento considera a utilização do *Semtech Packet Forwarder* por parte do *gateway* e, para que os dados em JSON que saem do *packet forwarder* e que deveriam ser enviados via UDP cheguem no *LoRa Server* em MQTT, existe um outro *software* (que também é parte do conjunto *LoRa Server* + *App Server*) desenvolvido, e é chamado de *LoRa Gateway Bridge*. O *LoRa Gateway Bridge* é melhor utilizado quando implementado dentro do próprio *gateway*, fazendo com que nenhum dado seja enviado via UDP, mas sim, sempre via MQTT, garantindo segurança na conexão (MQTT possui suporte à criptografia). Apesar disso, esse *software* também pode ser implementado no *network server* ainda que essa prática não seja muito comum devido ao fato de não proporcionar a segurança e a robustez que a transmissão utilizando MQTT proporciona, e tendo como caso de uso apenas redes em que

[3]Disponível em: https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-global_conf.json

[4]Disponível em: https://github.com/brocaar/loraserver

o *LoRa Server* deve ser compatível com dados transmitidos por *gateways* já em operação.

A outra implementação refere-se ao *LoRa App Server* que, além de implementar o *application server* com as funcionalidades especificadas pela LoRa Alliance, também fornece uma *interface web* para gerência e configuração de usuários da rede, *organizations*, *applications*, *gateways* e *motes*, comunicando-se com o *LoRa Server* via gRPC. Esse *software* também se comunica com um *broker* MQTT, no qual ficam armazenados os *uplinks* e *downlinks* descriptografados; cada *application* é um tópico, e cada *DevEUI* em conjunto com cada tipo de dado (*uplink*, *downlink*, ACK...) formam uma estrutura de subtópicos dentro do tópico da *application*. A estrutura de cada *application* também é acessada pelas aplicações clientes, as quais leem os *payloads* enviados pelos *motes* e publicam dados para serem enviados aos os mesmos. A figura 4.3 mostra uma visão geral da arquitetura da comunicação entre os *gateways*, *LoRa Server*, *LoRa App Server* e aplicações clientes.

Mais informações sobre o *LoRa Server*, além das citadas aqui, podem ser encontradas na documentação presente no *site* oficial da aplicação[5].

- ***The Things Network Backend*:** Esse *software* também possui seu código fonte disponível no Github[6], e é desenvolvido pelo *The Things Network Core Team*. Apesar de possibilitar implementações privadas, também é disponibilizado como uma plataforma *online*, facilitando prototipagem e possibilitando seu uso até mesmo em ambientes de produção, desde que a segurança dos dados trafegados não seja algo tão crítico.

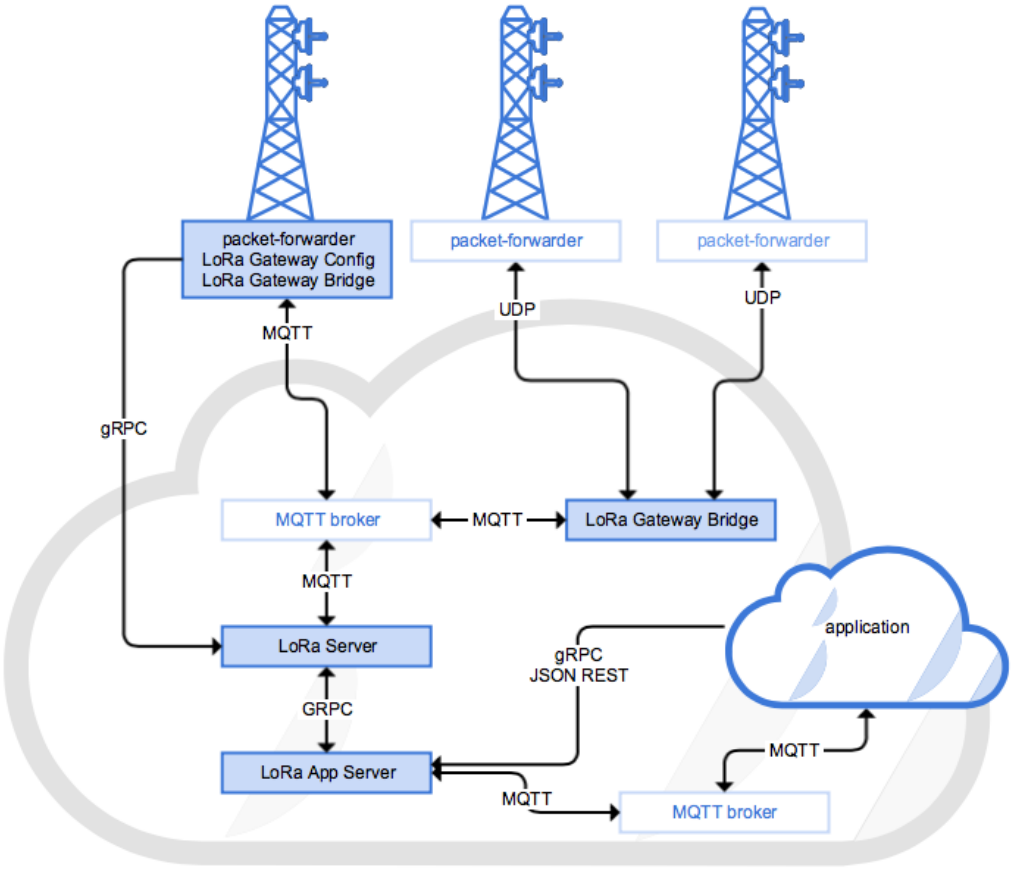
  Possui compatibilidade com diversos *packet forwarders* diferentes, incluindo o *Semtech Packet Forwarder* e seus derivados, além de, é claro, o *TTN Packet Forwarder*.

  Essa implementação não divide o sistema em *network server* e *application server*, conforme às especificações da LoRa Alliance; ao invés disso, ambos são somados como um sistema completo chamado de *backend*, que, por sua vez, é subdividido em módulos menores distribuídos e com funções específicas, que são:

  - ***Router*:** É responsável por todas as tarefas relacionadas aos *gateways*, como gerenciamento de seus *status*; cálculo de *scores* dos *gateways* com base nos dados físicos das transmissões dos *uplinks*, como RSSI (*Received Signal Strenght Indication*), SNR (*Signal-to-Noise Ratio*), etc.; extração do

[5] *Site* oficial do *LoRa Server*: https://www.loraserver.io/
[6] Disponível em: https://github.com/TheThingsNetwork/ttn

Figura 4.3 – Diagrama geral da arquitetura da rede LoRaWAN implementada com *LoRa Server*



Fonte: Retirado de Server (200-?).

endereço dos *motes* e redirecionamento dos pacotes para o *broker* correto, de acordo com o endereço extraído; e agendamento das transmissões dos *downlinks*, de forma que os mesmos sejam enviados no momento correto de acordo com a classe e o tempo de abertura das janelas de recepção de cada *mote*;

– ***Broker*:** Esse módulo é encarregado das tarefas relacionadas aos *uplinks* e *downlinks* em geral, como deduplicação dos *uplinks*, caso algum pacote duplicado seja detectado; seleção do melhor *gateway* para a realização dos *downlinks* com base nos *scores* calculados no *router*; roteamentos dos pacotes, direcionando os *uplinks* para as *applications* corretas (de acordo com a *application* a qual cada *mote* pertence) e os *downlinks* para os *routers* corretos (de acordo com o *router* a qual o melhor *gateway*, que possui o *mote* destino em sua área de cobertura, está conectado); e verificação de integridade dos *uplinks*, ignorando-os caso sejam irregulares;

– ***Network Server*:** Não devendo-se confundi-lo com o *network server* es-

pecificado pela LoRa Alliance, esse módulo realiza o gerenciamento dos *motes* em geral. Possui o mapeamento entre *motes* e DevAddrs, característica que o encarrega de auxiliar o *broker* informando-o quais *motes* possuem determinado DevAddr, possibilitando ao *broker* selecionar o *mote* de origem correto para direcionar os *uplinks* à *application* correta. É responsável também por gerenciar os *frame counters* de cada *mote* e gerar os MICs de cada *downlink*. Por fim, é encarregado de, no caso dos *downlinks*, gerar a parte de mensagem que não possui *payload* (como dados de controle da rede), gerenciando os *frame counters* e gerando os MICs corretos, para que, por fim, o *handler* complete o pacote adicionando o *payload* ao mesmo;

- ***Handler*:** É responsável por descriptografar, converter para um formato apropriado para a aplicação e publicar as mensagens em um *broker* MQTT, o qual será acessado pelas aplicações. Também é encarregado de gerar os ACKs e obter os *downlnks* contidos dentro do *broker* MQTT, para depois redirecioná-los ao *broker* da rede, que encaminhará seus envios;

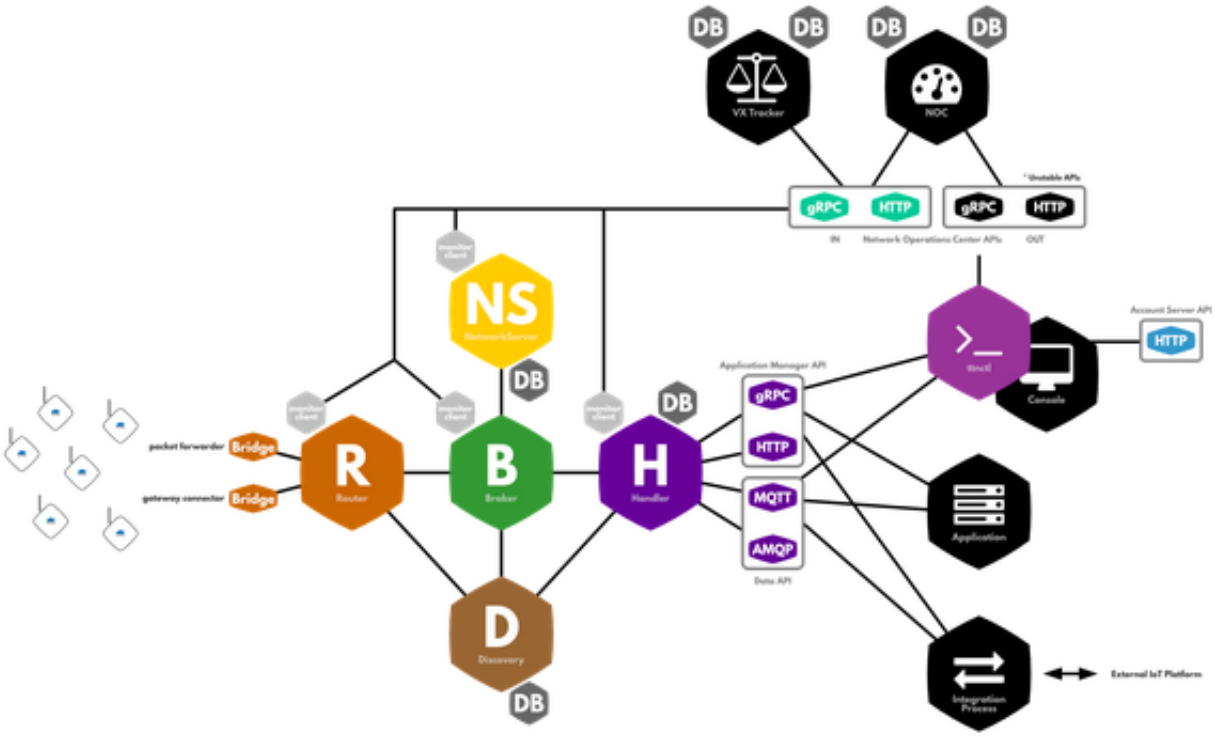
Esses módulos foram projetados dessa forma visando uma implementação distribuída e, no futuro, possibilitar implementações mistas, de forma que alguns módulos possam ser utilizados dentro do ambiente privado, enquanto outros possam ser utilizados dentro do ambiente público (disponível na *internet*) da The Things Network. A figura 4.4 mostra uma visão geral da arquitetura da implementação, onde as iniciais G, R, B, NS, H e A representam *gateway*, *router*, *broker*, *network server*, *handler* e *application*, respectivamente.

As informações sobre essa implementação neste item, assim como outras mais detalhadas, podem ser encontradas em Team (2017b);

Com base nos dados apresentados sobre as duas implementações, a escolhida foi a *LoRa Server*, pela facilidade de configurá-la e simplicidade de sua arquitetura, uma vez que uma implementação totalmente distribuída como a da *TTN Backend* é interessante apenas em situações específicas, como quando existe tráfego de dados muito grande ou quando o serviço não pode, em nenhuma hipótese, parar (justificando modularização do sistema para proporcionar escalabilidade e alta-disponibilidade dos módulos). No entanto, a plataforma *online* pública da TTN será utilizada futuramente neste trabalho, onde testes de comparação com a implementação privada serão feitos.

A implementação privada do *LoRa Server* foi realizada com ambos *LoRa Server* e *LoRa App Server* em um instância na nuvem, hospedada na Digital Ocean. A instância *host* possui 1 GB de RAM, 1 núcleo de CPU, 30 GB de armazenamento SSD e um endereço IP público.

Figura 4.4 – Diagrama geral da arquitetura de uma rede LoRaWAN implementada com *TTN Backend*



Fonte: Retirado de Team (2017b).

A respeito das dependências do *LoRa Server*, foram instalados: Mosquitto, um broker MQTT no qual serão inseridos os *uplinks* e *downlinks*; Redis, um serviço de armazenamento de dados transientes em memória; e PostgreSQL, um serviço de banco de dados relacional.

Considerando-se que é desejável que os dados trafeguem de maneira segura e com uma conexão robusta, o *LoRa Gateway Bridge* foi instalado no *gateway*, de forma que os dados sejam enviados criptografados usando MQTT.

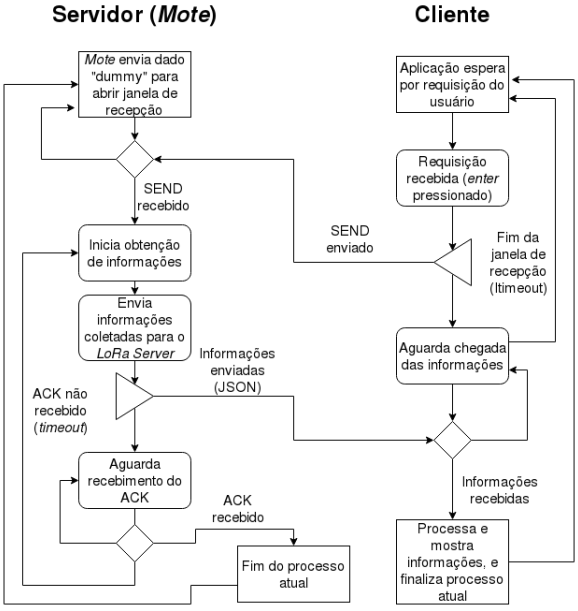
5 DESENVOLVIMENTO DE UMA APLICAÇÃO E RESULTADOS

5.1 DESENVOLVIMENTO DA APLICAÇÃO

Com o propósito de contextualizar o trabalho e mostrar que a rede LoRaWAN implementada funciona, foi desenvolvida uma aplicação cliente-servidor em Python, onde os clientes são sistemas com acesso à *internet* (computadores e sistemas embarcados em geral) e os servidores são sistemas sem acesso à internet e com sistema operacional Linux, conectados a um módulo RHF3M076 e formando, dessa forma, um *mote* dentro da rede LoRaWAN. Enquanto o código do servidor foi de autoria do autor deste trabalho, o código dos clientes foi uma adaptação do código apresentado por Santos (2017).

Em geral, a aplicação consiste no pedido de informações por parte dos clientes aos *motes*. Após receberem esse pedido, os *motes* coletam informações como: porcentagem de uso de RAM e de disco, e temperatura da CPU. Esse dado é retornado aos clientes que fizeram a requisição, onde os dados são organizados e mostrados na tela, além de dados gerais da comunicação da rede LoRa. A figura 5.1 mostra um fluxograma do funcionamento da comunicação entre um cliente e um servidor.

Figura 5.1 – Fluxograma do funcionamento da comunicação entre cliente e servidor (*mote*)



Fonte: Autor.

5.2 APLICAÇÃO EM FUNCIONAMENTO

Após o término da implementação da rede LoRaWAN e da aplicação que a utilizará, serão mostradas figuras do conjunto em funcionamento, considerando o uso de um único *mote* conectado à rede.

A fim de mostrar as informações dos dispositivos, as figuras 5.2, 5.3 e 5.4 mostram os dados dos registros da *application*, do *gateway* e do *mote* no *LoRa Server*, respectivamente. Enquanto isso, a figura 5.5 mostra a *AppSKey* e *NwkSKey* geradas para o *mote* registrado.

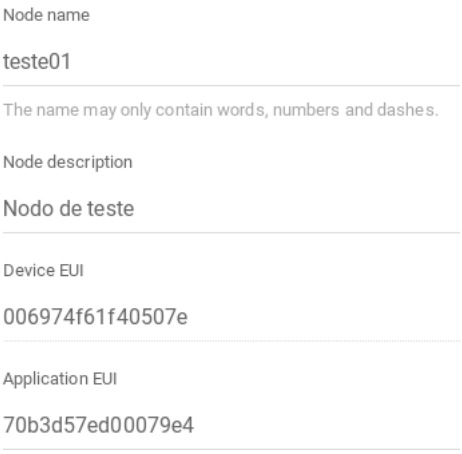
Figura 5.2 – Dados do registro da *application* no *LoRa Server*

Fonte: Autor.

Figura 5.3 – Dados do registro do *gateway* no *LoRa Server*

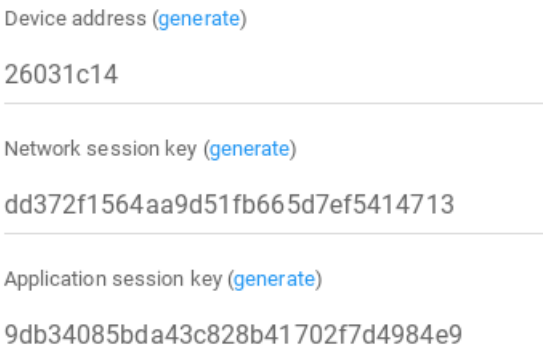
Fonte: Autor.

Figura 5.4 – Dados do registro do *mote* no *LoRa Server*



Fonte: Autor.

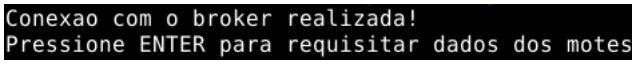
Figura 5.5 – Chaves de criptografia do *gateway* no *LoRa Server*



Fonte: Autor.

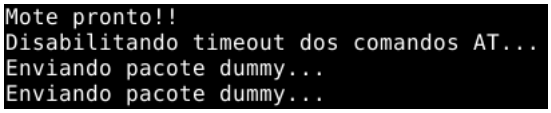
Após a realização do registro, as informações mostradas foram utilizadas para configurar os motes, o cliente e o servidor da aplicação. A seguir, o cliente e o servidor (*mote*) foram inicializados, gerando os diálogos conforme as figuras 5.6 e 5.7.

Figura 5.6 – Cliente da aplicação pronto para enviar requisições de dados ao servidor (*mote*)



Fonte: Autor.

Figura 5.7 – Servidor (*mote*) da aplicação pronto para receber requisições de dados do cliente



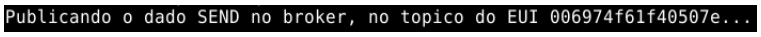
Fonte: Autor.

Estando o cliente apto a enviar requisição de dados, e o *mote* apto a receber as requisições, foi pressionado o botão "ENTER" do teclado no cliente, conforme a figura 5.8.

A requisição, depois de recebida no *mote*, fez com que o mesmo buscasse as informações de porcentagem de uso de memória RAM, de disco e temperatura da CPU. Depois de buscadas as informações, elas foram enviadas de volta ao cliente, e o servidor aguarda a chegada do ACK para realizar novos envios de pacote *dummy*, conforme a figura 5.9.

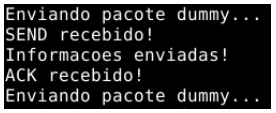
O cliente, após receber as informações, organiza-as e mostra-as na tela para o usuário. Após o término da janela de recepção de dados do *mote*, o mesmo retorna ao estado inicial, conforme a figura 5.10.

Figura 5.8 – Mensagem resultante após o envio do comando de requisição de dados pelo cliente ao *mote*



Fonte: Autor.

Figura 5.9 – Diálogo resultante no *mote* após a chegada da requisição de dados do cliente, aquisição e envio dos mesmos, e espera por ACK



Fonte: Autor.

A figura 5.10 mostra o término do processo completo da aplicação executado com sucesso. Conforme se pode ver, a contagem de pacotes (*FrameCount*) está em 473, indicando que esse é o número do pacote com os dados requisitados. Isso significa que o pacote *dummy* que resultou na janela de recepção dos dados requisitados tinha contagem 472, e o *dummy* a ser enviado após o recebimento dos dados terá contagem 474. A partir dessas informações, é possível observar nos *logs* de funcionamento do *gateway* o trecho em que essa troca de informações ocorre, conforme a figura 5.11.

Também, com base na figura 5.10, é possível ver que o *MAC Address* do *gateway* e o DevAddr do *mote* correspondem aos dados registrados no *LoRa Server*, enquanto o uso de frequência, largura de banda, fator de espalhamento e técnica de modulação utilizada correspondem aos valores configurados no *gateway* e no *mote*, ou seja, está operando conforme esperado.

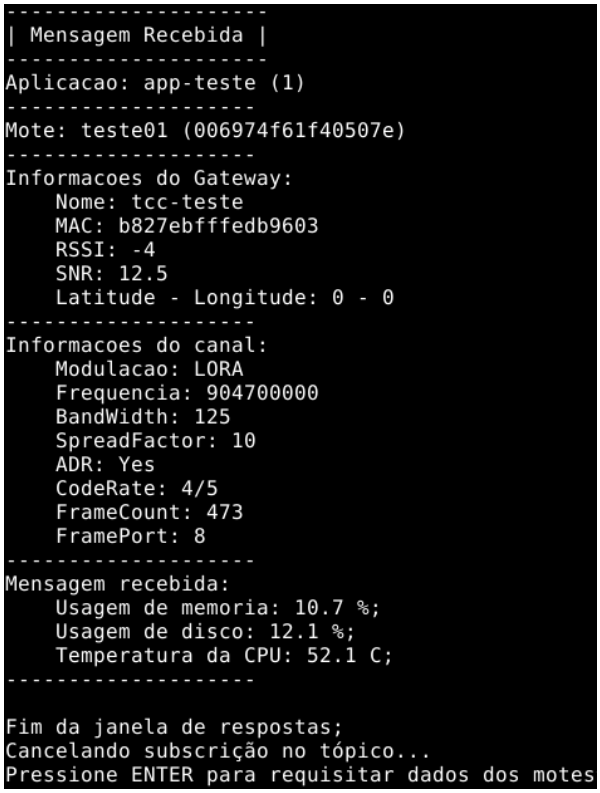
Casos excepcionais, como erros e falhas de conexão, apesar de tratados nos códigos e apresentarem avisos ao usuário, não serão abordados nessa seção.

5.3 TESTES DE PERDAS DE ACKNOWLEDGEMENT

Seguindo o término da demonstração do funcionamento da aplicação, a mesma não será mais utilizada. Ao invés disso, será utilizado um *script*, programado em Python, que faz um *mote* enviar 200 mensagens com um caractere e com espera por ACK. Quatro experimentos são realizados utilizando o esse *script*, onde o mesmo é executado dez vezes consecutivas em cada um dos seguintes cenários:

1. **Experimento 1:** Utilização de *mote* classe A e *LoRa Server* + *LoRa App Server* implementados no capítulo anterior;

Figura 5.10 – Dados recebidos no cliente pelo *mote*, descriptografados e organizados



Fonte: Autor.

2. **Experimento 2:** Utilização de *mote* classe C e *LoRa Server* + *LoRa App Server* implementados no capítulo anterior;

3. **Experimento 3:** Utilização de *mote* classe A e plataforma pública da The Things Network;

4. **Experimento 4:** Utilização de *mote* classe C e plataforma pública da The Things Network;

Em cada cenário, o *mote* foi reconfigurado e os *frame counters* das plataformas foram reiniciados. No total, foram realizadas 40 execuções do *script*. Cada execução resultou no tempo total e médio decorrido no envio das mensagens e recebimento dos ACKs, e número de ACKs perdidos (ou seja, casos em que a mensagem foi enviada mas nenhum ACK foi recebido). Esses resultados foram tabelados nos quadros 5.1, 5.2, 5.3, 5.4 e 5.5.

Figura 5.11 – Trecho dos *logs* do *gateway* que mostra as informações trocadas entre o cliente e o servidor (*mote*) para requisição e recebimento dos dados

```
INFO: Received pkt from mote: 26031C14 (fcnt=472)

JSON up: {"rxpk":[{"tmst":226578636,"chan":0,"rfch":0,"freq":903.900000,"stat":1,"modu":
"LORA","datr":"SF10BW125","codr":"4/5","lsnr":9.5,"rssi":0,"size":14,"data":"QBQcAyaA2AE
IWzEpi7I="}]}
INFO: [up] PUSH_ACK received in 3 ms
INFO: [down] PULL_RESP received  - token[0:0] :)

JSON down: {"txpk":{"imme":false,"tmst":228578636,"freq":923.3,"rfch":0,"powe":20,"modu"
:"LORA","datr":"SF12BW500","codr":"4/5","ipol":true,"size":17,"data":"oBQcAyaASwAI/PL0pc
RmGZE="}}
INFO: tx_start_delay=1497 (1497.000000) - (1497, bw_delay=0.000000, notch_delay=0.000000
)

INFO: Received pkt from mote: 26031C14 (fcnt=473)

JSON up: {"rxpk":[{"tmst":230746940,"chan":4,"rfch":1,"freq":904.700000,"stat":1,"modu":
"LORA","datr":"SF10BW125","codr":"4/5","lsnr":12.5,"rssi":-4,"size":27,"data":"gBQcAyag2
QEIyrVWrqjQ2Ii7fT0LQRpjh4wS"}]}
INFO: [up] PUSH_ACK received in 3 ms
INFO: [down] PULL_RESP received  - token[0:0] :)

JSON down: {"txpk":{"imme":false,"tmst":232746940,"freq":923.3,"rfch":0,"powe":20,"modu"
:"LORA","datr":"SF12BW500","codr":"4/5","ipol":true,"size":12,"data":"YBQcAyagTACHeEP0"}
}
INFO: tx_start_delay=1497 (1497.000000) - (1497, bw_delay=0.000000, notch_delay=0.000000
)
INFO: [down] PULL_ACK received in 1 ms

INFO: Received pkt from mote: 26031C14 (fcnt=474)

JSON up: {"rxpk":[{"tmst":234310100,"chan":2,"rfch":0,"freq":904.300000,"stat":1,"modu":
"LORA","datr":"SF10BW125","codr":"4/5","lsnr":6.0,"rssi":-3,"size":14,"data":"QBQcAyaA2g
EIgxfAJo4="}]}
INFO: [up] PUSH_ACK received in 0 ms
```

Fonte: Autor.

Quadro 5.1 – Resultado das 10 execuções do *script* no Experimento 1

Execução	Tempo Total (s)	Tempo Médio (s)	ACKs Perdidos
1	734.596299171	3.67298149586	0
2	908.557182074	4.54278591037	16
3	750.541357756	3.75270678878	2
4	728.641026497	3.64320513248	0
5	742.634352446	3.71317176223	0
6	744.632679939	3.72316339970	0
7	729.620444298	3.64810222149	0
8	748.642654419	3.74321327209	0
9	731.684502125	3.65842251062	0
10	750.111641645	3.75055820823	1

Fonte: Autor.

Quadro 5.2 – Resultado das 10 execuções do *script* no Experimento 2

Execução	**Tempo Total (s)**	**Tempo Médio (s)**	**ACKs Perdidos**
1	766.427670717	3.83213835359	2
2	728.632747889	3.64316373944	0
3	719.615826130	3.59807913065	0
4	738.663762331	3.69331881166	0
5	750.547234774	3.75273617387	2
6	728.631921530	3.64315960765	0
7	740.645980120	3.70322990060	0
8	739.641636848	3.69820818424	0
9	725.678102016	3.62839051008	0
10	738.666998625	3.69333499312	0

Fonte: Autor.

Quadro 5.3 – Resultado das 10 execuções do *script* no Experimento 3

Execução	**Tempo Total (s)**	**Tempo Médio (s)**	**ACKs Perdidos**
1	676.780563354	3.38390281677	0
2	682.436291218	3.41218145609	0
3	677.463097811	3.38731548905	0
4	695.451015234	3.47725507617	0
5	671.419818401	3.35709909201	0
6	695.544038773	3.47772019386	0
7	706.815012217	3.53407506108	0
8	677.686697006	3.38843348503	0
9	703.649684906	3.51824842453	0
10	704.675527334	3.52337763667	0

Fonte: Autor.

Quadro 5.4 – Resultado das 10 execuções do *script* no Experimento 4

Execução	Tempo Total (s)	Tempo Médio (s)	ACKs Perdidos
1	671.792918682	3.35896459341	0
2	681.423914909	3.40711957455	0
3	698.627050638	3.49313525319	0
4	693.547338486	3.46773669243	0
5	668.429683685	3.34214841843	0
6	663.422055244	3.31711027622	0
7	658.420864344	3.29210432172	0
8	668.440393686	3.34220196843	0
9	662.421781540	3.31210890770	0
10	678.425157309	3.39212578654	0

Fonte: Autor.

Quadro 5.5 – Médias dos resultados obtidos em cada experimento

Experimento	Tempo Total (s)	Tempo Médio (s)	ACKs Perdidos
1	756.966214037	3.78483107018	1.9
2	737.715188098	3.68857594049	0.4
3	689.192174625	3.44596087313	0
4	674.495115852	3.37247557926	0

Fonte: Autor.

Os resultados obtidos mostram que os tempos entre a transmissão e recepção do ACK nos experimentos utilizando o *LoRa Server* são maiores, além de terem ocorrido algumas perdas de ACK, algo que não aconteceu em ambos experimentos utilizando o *TTN Backend*. Nota-se também que, no caso do *LoRa Server*, o uso da classe C resultou em um menor número médio de perda de ACKs, algo esperado devido ao fato de *motes* classe C manterem a janela de recepção quase sempre aberta.

Estudos mais aprofundados, considerando toda a rede e realizando medições precisas em cada ponto da mesma, são necessários para diagnosticar o porquê de terem ocorrido perdas de ACK apenas no *LoRa Server*, e, inclusive, na utilização da classe C. Essa afirmação é reforçada pelo fato de os resultados mostrarem que o *packet forwarder* que utiliza um protocolo mais confiável (MQTT) perdeu ACKs, enquanto o que utiliza um protocolo de menor confiabilidade (UDP) não perdeu nenhum ACK. Ainda que esses estudos fujam do escopo deste trabalho, ainda pode-se supor que as perdas de ACK tenham ocorrido por atrasos na chegada do *downlink* no *gateway*, uma vez que qualquer atraso gerado no caminho do *downlink* pode fazer com que o

mesmo não chegue no *gateway* a tempo de ser transmitido ao *mote* enquanto sua janela de recepção está aberta, fazendo com que o ACK seja perdido. Esses atrasos podem ter ocorrido devido a:

- Instabilidade da rede, levando a perdas, retransmissões ou baixa taxa de transmissão de pacotes;
- Diferenças nas implementações: enquanto nos experimentos com *TTN Backend* o *gateway* redireciona diretamente os dados recebidos via UDP, os experimentos com *LoRa Server* adicionam uma camada de conversão, de MQTT para UDP, fazendo com que os *downlinks* recebidos via MQTT do *LoRa Server* precisem, primeiramente, passar pelo *LoRa Gateway Bridge*, para somente então passarem pelo *packet forwarder*, para, finalmente, poderem ser transmitidos ao *mote*, aumentando o tempo em que o dado fica "parado" no *gateway*;
- Diferenças entre as infraestruturas do *LoRa Server* + *LoRa App Server* e do *TTN Backend*, gerando atrasos no processamento e armazenamento dos *uplinks*, e envio dos ACKs;

Apesar dos resultados obtidos servirem como um bom ponto de partida para futuras pesquisas de desempenho das implementações, os testes foram realizados com apenas um *mote*; portanto, cabem aqui estudos futuros sobre o comportamento e desempenho das implementações quando submetidas ao acesso de vários *motes* simultaneamente.

6 CONCLUSÃO

Neste trabalho foi apresentado o protocolo LoRaWAN, tendo sido citadas suas características, funcionamento, diferenças em relação a outros protocolos comuns na área da IoT, partes de *software* e *hardware* que compõem uma rede que o implementa, configurações necessárias para o seu funcionamento correto e uma aplicação que o utiliza. Conclui-se que, apesar de ser um protocolo relativamente novo, já pode ser utilizado em alguns projetos menores, e deve começar a ter seu uso difundido na comunidade acadêmica, pois pode vir a ser de grande auxílio no desenvolvimento da IoT no Brasil, futuramente.

A implementação da rede e os testes realizados mostram que uma rede com LoRaWAN funciona e que pode ser implementada no Brasil, mesmo sem nenhuma regulamentação específica existente. Entretanto, conforme dito anteriormente, os testes realizados foram muito simplórios, considerando um caso mínimo e poucas variáveis, devido a limitações como a falta de um número considerável de *motes* disponíveis e tempo escasso para a realização de experimentos mais completos. Sendo assim, não é possível concluir-se qual implementação da rede, em termos de *software*, terá melhor desempenho em ambientes mais realistas, onde muitos *motes* enviam dados paralelamente, congestionando a rede.

Portanto, pesquisas futuras devem ser realizadas, investigando mais a fundo as implementações de *packet forwarder* e *application + network server* existentes e realizando testes de desempenho das mesmas quando implementadas em redes congestionadas, enquanto verifica-se o impacto do uso das janelas de recepção em diferentes cenários (somente RX1 ou RX2 aberta, ou ambas abertas), com diferentes classes de *motes* e com diferentes períodos de transmissões de dados. Além disso, são necessários estudos práticos sobre o impacto no desempenho, na área de cobertura e no custo energético causado pelo uso da LoRaWAN em relação a outros protocolos utilizados na IoT, como o Sigfox.

REFERÊNCIAS BIBLIOGRÁFICAS

ALECRIM, E. **O que é Wi-Fi (IEEE 802.11)?** 2013. Acesso em: 20/11/2017. Disponível em: <https://www.infowester.com/wifi.php>.

ANATEL. **Resolução Nº 506**. 2017. Acesso em: 19/11/2017. Disponível em: <http://www.anatel.gov.br/legislacao/resolucoes/23-2008/104-resolucao-506>.

____. **Resolução Nº 680**. 2017. Acesso em: 18/11/2017. Disponível em: <http://www.anatel.gov.br/legislacao/resolucoes/2017/936-resolucao-680>.

BNDES. Relatório do plano de ação - iniciativas e projetos mobilizadores. **Internet das Coisas: Um Plano de Ação para o Brasil**, v. 8, p. 5, 2017. Acesso em 15/11/2017. Disponível em: <https://www.bndes.gov.br/wps/wcm/connect/site/269bc780-8cdb-4b9b-a297-53955103d4c5/relatorio-final-planodeacao-produto-8.pdf?MOD=AJPERES&CVID=lXysvoX&CVID=lXysvoX&CVID=lXysvoX&CVID=lXysvoX&CVID=lXysvoX&CVID=lXysvoX&CVID=lXysvoX&CVID=lXyrzvY>.

COMMITTEE, L. A. T. **LoRaWAN 1.1 Regional Parameters**. Beaverton, 2017. Final release, 13–18 p.

____. **LoRaWAN 1.1 Specification**. Beaverton, 2017. Final release, 8–88 p.

COSSINI, F. Lorawan: Uma rede alternativa para a internet das coisas. **Technology Leadership Council Brazil**, v. 274, p. 1, 2016. Acesso em 15/11/2017. Disponível em: <https://www.ibm.com/developerworks/community/blogs/tlcbr/resource/TLC-BR_Mini_Paper_Ano_12_N_274_IoT.pdf?lang=en>.

JUNIOR, V. P. da S. **Conheça a Tecnologia LoRa e o Protocolo LoRaWAN**. 2016. Acesso em: 20/11/2017. Disponível em: <https://www.embarcados.com.br/conheca-tecnologia-lora-e-o-protocolo-lorawan/>.

MORIMOTO, C. E. **Entendendo a Questão do Alcance em Redes Wireless**. 2009. Acesso em: 25/11/2017. Disponível em: <http://www.hardware.com.br/dicas/alcance-redes-wireless.html>.

NETWORK, T. T. **Gateway Connection to TTN**. 200–? Acesso em: 19/11/2017. Disponível em: <https://www.thethingsnetwork.org/docs/gateways/start/connection.html\#gateway-connector-protocol>.

____. **Frequency Plan and Regulations by Country**. 2017. Acesso em: 18/11/2017. Disponível em: <https://www.thethingsnetwork.org/wiki/LoRaWAN/Frequencies/By-Country\#frequency-plan-and-regulations-by-country_b>.

____. **Frequency Plans**. 2017. Acesso em: 19/11/2017. Disponível em: <https://www.thethingsnetwork.org/wiki/LoRaWAN/Frequencies/Frequency-Plans>.

____. **gateway-conf**. 2017. Acesso em: 25/11/2017. Disponível em: <https://github.com/TheThingsNetwork/gateway-conf/blob/master/US-global_conf.json>.

____. **LoRaWAN**. 2017. Acesso em: 16/11/2017. Disponível em: <https://www.thethingsnetwork.org/wiki/LoRaWAN/Home>.

PESSOA, L. **Introdução ao Bluetooth Smart (BLE)**. 2016. Acesso em: 23/11/2017. Disponível em: <https://www.embarcados.com.br/bluetooth-smart-ble/>.

RISINGHF. **RHF0M301 – LoRa Gateway and Concentrator Module**. 2017. Acesso em: 19/11/2017. Disponível em: <http://www.risinghf.com/product/rhf0m301/?lang=en>.

____. **RHF3M076 USB LoRaWAN AT Modem**. 2017. Acesso em: 19/11/2017. Disponível em: <http://www.risinghf.com/product/rhf3m076/?lang=en>.

SANTOS, B. P. et al. **Internet das Coisas: da Teoria à Prática**. 200–? 10 p. Acesso em: 25/11/2017. Disponível em: <http://homepages.dcc.ufmg.br/~mmvieira/cc/papers/internet-das-coisas.pdf>.

SANTOS, N. **Python: Subscribing to MQTT Topic**. 2017. Acesso em: 23/11/2017. Disponível em: <https://techtutorialsx.com/2017/04/23/python-subscribing-to-mqtt-topic/>.

SERVER, L. **System Architecture**. 200–? Acesso em: 21/11/2017. Disponível em: <https://docs.loraserver.io/overview/architecture/>.

TEAM, T. T. N. C. **New TTN Packet Forwarder Available**. 2017. Acesso em: 19/11/2017. Disponível em: <https://www.thethingsnetwork.org/forum/t/new-ttn-packet-forwarder-available/7644/46>.

____. **The Things Network Backend**. 2017. Acesso em: 23/11/2017. Disponível em: <https://www.thethingsnetwork.org/wiki/Backend/Home>.

WND Brasil. **Uma Visão Técnica da Rede Sigfox**. 2017. Acesso em: 25/11/2017. Disponível em: <https://www.embarcados.com.br/uma-visao-tecnica-da-rede-sigfox/>.

XIA, F. et al. Internet of things. **Internation Journal of Communication Systems**, v. 25, p. 1101–1102, 2012. Acesso em 15/11/2017. Disponível em: <https://pdfs.semanticscholar.org/930c/4981e87584afa7e6f1f4977323e365aae097.pdf>.

APÊNDICE A – *SCRIPT* DE CONFIGURAÇÃO DOS *MOTES*

```
import serial
from time import sleep

# Chaves de criptografia
NwkSKey = "123456789abcdefghijklmnopqrstuvw"
AppSKey = "abcdefghijklmnopqrstuvw123456789"
# EUI da aplicacao
AppEui = "12345abcdefghijk"
# Endereco do mote
DevAddr = "1a2b3c4d"


# Executa o comando AT e pausa a execucao do codigo ate que o mesmo tenha a
    resposta esperada
def error_check(command2write, answer_expected):
  global serial_connection
  serial_connection.write(command2write)
  while True:
    serial_answer = serial_connection.readlines()
    for current_answer in serial_answer:
      if current_answer == answer_expected:
        return
      sleep(1)


# Abre a conexao com a porta serial
serial_connection = serial.Serial('/dev/ttyACM0', 9600, timeout=1)

try:
  # Verifica a possibilidade de enviar comandos
  error_check('AT\r\n', '+AT: OK\r\n')

  # Desabilita o timeout da porta serial
  error_check('AT+UART=TIMEOUT, 0\r\n', '+UART: TIMEOUT is disabled\r\n')

  # Realiza um reset de fabrica no mote e aguarda 5 segundos ate sua
      finalizacao
  serial_connection.write(b"AT+FDEFAULT=RISINGHF\r\n")
  serial_connection.readlines()
  sleep(5)

  # Definicao do esquema de taxa de dados para 902-928 (Padrao Americano/
      Sul Americano)
```

```
serial_connection.write(b"AT+DR=US915\r\n")
serial_connection.readlines()

# Configura os canais utilizados pelo mote
serial_connection.write(b"AT+CH=0,903.9,DR0,DR3\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=1,904.1,DR0,DR3\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=2,904.3,DR0,DR3\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=3,904.5,DR0,DR3\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=4,904.7,DR0,DR3\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=5,904.9,DR0,DR3\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=6,905.1,DR0,DR3\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=7,905.3,DR0,DR3\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=8,904.6,DR4\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=9,0\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=10,0\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=11,0\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=12,0\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=13,0\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=14,0\r\n")
serial_connection.readlines()
serial_connection.write(b"AT+CH=15,0\r\n")
serial_connection.readlines()

# Desabilita RX1
serial_connection.write(b"AT+RXWIN1=OFF\r\n")
serial_connection.readlines()

# Configura RX2 para funcionar em 923.3 MHz e em DR8
serial_connection.write(b"AT+RXWIN2=923.3,DR8\r\n")
serial_connection.readlines()

# Configura a abertura de RX2 em 2 segundos apos o uplink
serial_connection.write(b"AT+DELAY=RX2,2000\r\n")
```

```
    serial_connection.readlines()

    # Configura o DR padrao
    serial_connection.write(b"AT+DR=DR0\r\n")
    serial_connection.readlines()

    # Potencia de transmissao
    serial_connection.write(b"AT+POWER=14\r\n")
    serial_connection.readlines()

    # Ativa o Adaptative Data Rate
    serial_connection.write(b"AT+ADR=ON\r\n")
    serial_connection.readlines()

    # Configura a operacao do mote como classe A
    serial_connection.write(b"AT+CLASS=A\r\n")
    serial_connection.readlines()

    # Configura a ativacao do mote como Activation By Personalization
    serial_connection.write(b"AT+MODE=LWABP\r\n")
    serial_connection.readlines()

    # Configura as chaves de criptografia, EUI da aplicacao e o endereco do
        mote
    serial_connection.write(b"AT+KEY=NwkSKey,\"%s\"\r\n" % (NwkSKey))
    serial_connection.readlines()
    serial_connection.write(b"AT+KEY=AppSKey,\"%s\"\r\n" % (AppSKey))
    serial_connection.readlines()
    serial_connection.write(b"AT+ID=AppEui,\"%s\"\r\n" % (AppEui))
    serial_connection.readlines()
    serial_connection.write(b"AT+ID=DevAddr,\"%s\"\r\n" % (DevAddr))
    serial_connection.readlines()

except:
    print("Erro ocorrido!")
    serial_connection.close()
```

APÊNDICE B – CLIENTE DA APLICAÇÃO

```
import paho.mqtt.client as mqttClient
import json, base64, time

# Especifica o endereco do broker MQTT
broker_address = "192.168.2.1"
# Porta do broker MQTT
broker_port = 1883
# Username do broker MQTT
broker_user = "loraroot"
# Password do broker MQTT
broker_password = "admin"
# AppEUI da aplicacao
application_id_number = 1
# Tempo, em segundos, em que este programa ira aguardar por mensagens
    chegarem no broker
wait_packet_time = 10
# Lista dos DevEUIs dos motes aos quais deseja-se receber informacoes
eui_list = ["a1b2c3d4e5f67890"]


# Metodo executado quando a conexao com o broker eh realizada
def connection_established(client, userdata, flags, rc):
  if rc == 0:
    print("Conexao com o broker realizada!")
    global Connected
    Connected = True
  else:
    print("Conexao com o broker falhou...")


# Metodo executado quando um dado chega no broker
def message_received(client, userdata, message):
  # Obtem a mensagem completa em JSON, convertida para ASCII
  full_message_json = json.loads(message.payload.decode('ascii'))
  # Obtem o AppEUI
  application_id = full_message_json["applicationID"]
  # Obtem o nome da aplicacao
  application_name = full_message_json["applicationName"]
  # Obtem o DevEUI do mote
  dev_eui = full_message_json["devEUI"]
  # Obtem o nome do mote
  node_name = full_message_json["nodeName"]
  # Lista de informacoes relacionadas ao gateway e a transmissao do pacote
```

```
# (MAC e nome do gateway, RSSI, SNR e coordenadas)
gateways_info = full_message_json["rxInfo"]
# Frequencia em que o pacote foi enviado
frequency = full_message_json["txInfo"]["frequency"]
# Modulacao do pacote, pode ser LoRa ou FSK
modulation = full_message_json["txInfo"]["dataRate"]["modulation"]
# Bandwith em que o pacote foi enviado
band_width = full_message_json["txInfo"]["dataRate"]["bandwidth"]
# SpreadFactor em que o pacote foi enviado
spread_factor = full_message_json["txInfo"]["dataRate"]["spreadFactor"]
# Verifica se o mote utiliza ADR
if (full_message_json["txInfo"]["adr"]):
  adr = "Yes"
else:
  adr = "No"
# Obtem o code rate, frame count e porta em que foi enviado
code_rate = full_message_json["txInfo"]["codeRate"]
fcnt = full_message_json["fCnt"]
fport = full_message_json["fPort"]
# Extrai o dado enviado pelo mote
data_received = base64.b64decode(full_message_json["data"]).decode("ascii
    ")
# Verifica se o dado enviado pelo mote eh um dado "dummy"
# Se for, sai do metodo
if data_received == "?":
  return

print("")
print("————————————————————")
print("| Mensagem Recebida |")
print("————————————————————")
print("Aplicacao: {} ({})".format(application_name, application_id))
print("————————————————————")
print("Mote: {} ({})".format(node_name, dev_eui))
print("————————————————————")
for gateway_info in gateways_info:
  print("Informacoes do Gateway:")
  print("    Nome: {}".format(gateway_info["name"]))
  print("    MAC: {}".format(gateway_info["mac"]))
  print("    RSSI: {}".format(gateway_info["rssi"]))
  print("    SNR: {}".format(gateway_info["loRaSNR"]))
  print("    Latitude - Longitude: {} - {}".format(gateway_info["latitude
      "], gateway_info["longitude"]))
print("————————————————————")
print("Informacoes do canal:")
print("    Modulacao: {}".format(modulation))
print("    Frequencia: {}".format(frequency))
```

```
    print("     BandWidth: {}".format(band_width))
    print("     SpreadFactor: {}".format(spread_factor))
    print("     ADR: {}".format(adr))
    print("     CodeRate: {}".format(code_rate))
    print("     FrameCount: {}".format(fcnt))
    print("     FramePort: {}".format(fport))
    print("-----------------------")
    print("Mensagem recebida:")
    print("     Usagem de memoria: {} %;".format(data_received.split("-")[0]))
    print("     Usagem de disco: {} %;".format(data_received.split("-")[1]))
    print("     Temperatura da CPU: {} C;".format(data_received.split("-")[2])
        )
    print("-----------------------")
    print("")


# Flag que indica se a conexao com o broker foi realizada
Connected = False

# Cria o cliente para conexao ao broker e configura o usuario e senha para
    conexao
mqtt_client = mqttClient.Client("LoRa Server MQTT Client")
mqtt_client.username_pw_set(broker_user, password=broker_password)
# Inicializa os listeners que aguardam o estabelecimento da conexao e
    chegada de novas mensagens
mqtt_client.on_connect = connection_established
mqtt_client.on_message = message_received

# Inicializa a conexao com o broker
mqtt_client.connect(broker_address, port=broker_port)

# Cria a thread que gerencia reconexoes com o broker e entrada/saida de
    dados
mqtt_client.loop_start()

# Aguarda ate que a conexao seja estabelecida
while Connected != True:
    time.sleep(0.1)

try:
    while True:
        # Aguarda que o usuario aperte "Enter" para requistar dados aos motes
        input("Pressione ENTER para requisitar dados dos motes")

        # "Inscreve-se" no broker
        mqtt_client.subscribe("application/{}/node/+/rx".format(
            application_id_number))
```

```
        # Envia o dado "SEND" para todos os DevEUIs especificados
        for current_eui in eui_list:
          print("Publicando o dado SEND no broker, no topico do EUI {}...".
              format(current_eui))
          mqtt_client.publish("application/{}/node/{}/tx".format(
              application_id_number, current_eui),
                              '{"reference": "ref",'
                              '"confirmed": true,'
                              '"fPort": 8,'
                              '"data": "U0VORA=="}')

        # Espera pelos dados que os motes deverao transmitir
        time.sleep(wait_packet_time)
        # Fecha a janela de recebimento de dados dos motes, e volta a aguardar
            entrada do usuario
        print("Fim da janela de respostas;")
        print("Cancelando subscricao no topico...")
        mqtt_client.unsubscribe("application/{}/node/+/rx".format(
            application_id_number))

    except KeyboardInterrupt:
      print("Saindo...")
      mqtt_client.disconnect()
      mqtt_client.loop_stop()
```

APÊNDICE C – SERVIDOR (*MOTE*) DA APLICAÇÃO

```
import serial, time, psutil, os


# Metodo que obtem os dados do sistema, no formato:
# {porcentagem de uso de memoria}-{porcentagem de uso de disco}-{
    temperatura da CPU (Celsius)}
def get_system_infos():
  # Get the current memory usage (percentage)
  memory_usage = psutil.virtual_memory()[2]
  # Get the current disk usage (percentage)
  disk_usage = psutil.disk_usage('/')[3]
  # Get the current CPU temperature (Celsius)
  cpu_temperature = os.popen('vcgencmd measure_temp').readline().replace("
      temp=", "").replace("\'C\n", "")
  return "{}-{}-{}".format(memory_usage, disk_usage, cpu_temperature)


# Cria a comunicacao serial entre o host e o mote
serial_connection = serial.Serial('/dev/ttyACM0', 9600, timeout=1)

# Aguarda o mote estar disponivel para receber comandos
serial_connection.write(b"AT\r\n")
while True:
  if b'+AT: OK\r\n' in serial_connection.readlines():
    print('Mote pronto!!')
    break
  else:
    print('Esperando pela disponibilidade do mote...')
    time.sleep(0.5)

# Destativa o timeout dos comandos AT
print('Disabilitando timeout dos comandos AT...')
serial_connection.write(b"AT+UART=TIMEOUT, 0\r\n")
serial_connection.readlines()

while True:
  # Envia pacote dummy (com '?'), abrindo a janela de recepcao
  print('Enviando pacote dummy...')
  serial_connection.write(b'AT+MSG="?"\r\n')
  while True:
    done_received = False
    send_received = False
    # Verifica se chegou o comando SEND (53 45 4E 44) na janela de recepcao
```

```
serial_output = serial_connection.readlines()
for current_output in serial_output:
  if b'RX: "53 45 4E 44 "' in current_output:
    print("SEND recebido!")
    send_received = True
    break
  if b'+MSG: Done' in current_output:
    done_received = True
    break
# Se o SEND foi recebido, coleta as informacoes e envia-as
if send_received:
  system_infos = get_system_infos()
  serial_connection.write(b"AT+CMSG=\"" + system_infos.encode("ascii")
     + b"\"\r\n")
  print("Informacoes enviadas!")
  # Aguarda por ACK. Se ocorrer timeout, re-obtem as informacoes e
     tenta envia-las
  while True:
    ack_received = False
    for current_output in serial_connection.readlines():
      if b'+CMSG: ACK Received\r\n' in current_output:
        print("ACK recebido!")
        ack_received = True
        break
      elif b'+MSG: Done' in current_output:
        print("ACK nao recebido, re-obtendo informacoes...")
        system_infos = get_system_infos()
        serial_connection.write(b"AT+CMSG=\"" + system_infos.encode("
           ascii") + b"\"\r\n")
        print("ACK recebido!")
        time.sleep(0.5)
    if ack_received:
      break
  break
elif done_received:
  break
time.sleep(0.5)
```

APÊNDICE D – *SCRIPT* DE UTILIZADO NO TESTE DE PERFORMANCE DOS *NETWORK SERVERS* + *APPLICATION SERVERS* A PARTIR DOS *MOTES*

```
import serial, time

# Numero de pacotes que devem ser enviados para medicao
sends_number = 200


# Cria a comunicacao serial entre o host e o mote
serial_connection = serial.Serial('/dev/ttyACM0', 9600, timeout=1)

# Aguarda o mote estar disponivel para receber comandos
serial_connection.write(b"AT\r\n")
while True:
  if b'+AT: OK\r\n' in serial_connection.readlines():
    print('Mote pronto!!')
    break
  else:
    print('Esperando pela disponibilidade do mote...')
    time.sleep(0.5)

# Destativa o timeout dos comandos AT
print('Disabilitando timeout dos comandos AT...')
serial_connection.write(b"AT+UART=TIMEOUT, 0\r\n")
serial_connection.readlines()

# Numero total de ACKs perdidos
lost_acks = 0
# Somatorio de todos os tempos de transmissao e recepcao de ACK
total_time = 0


for i in range(0,sends_number):
  print("Enviando pacote {}...".format(i))
  # Inicia o contador de tempo
  t0 = time.time()
  # Envia um pacote qualquer, com '?'
  serial_connection.write(b'AT+CMSG=\"?"\r\n')
  # Verifica a resposta, se ocorreu ACK, ou, senao, Done (ACK timeout)
  while True:
    lines = serial_connection.readlines()
    # Se ACK chegou, para o contador de tempo, soma o tempo de transmissao
    # e recebimento e envia outro pacote
    if '+CMSG: ACK Received\r\n' in lines:
```

```
            current_time = time.time() - t0
            total_time += current_time
            break
        # Se ACK nao chegou, para o contador de tempo, soma o tempo de
            transmissao
        # e recebimento, incrementa o contador de ACKs perdidos, e envia outro
            pacote
        elif '+CMSG: Done\r\n' in lines:
            current_time = time.time() - t0
            total_time += current_time
            lost_acks += 1
            break

# Informa dados da performance
print("Tempo total: {}".format(total_time))
print("Tempo medio: {}".format(total_time/float(sends_number)))
print("ACKs perdidos: {}".format(lost_acks))
```

ANEXO A – ONFIGURAÇÕES DO *SEMTECH PACKET FORWARDER* PARA FAIXA 902-928 MHZ


```
{
        "SX1301_conf": {
                "lorawan_public": true,
                "clksrc": 1,
                "clksrc_desc": "radio_1 provides clock to concentrator",
                "antenna_gain": 0,
                "antenna_gain_desc": "antenna gain, in dBi",
                "radio_0": {
                        "enable": true,
                        "type": "SX1257",
                        "freq": 904300000,
                        "rssi_offset": -166.0,
                        "tx_enable": true,
                        "tx_freq_min": 923000000,
                        "tx_freq_max": 928000000
                },
                "radio_1": {
                        "enable": true,
                        "type": "SX1257",
                        "freq": 905000000,
                        "rssi_offset": -166.0,
                        "tx_enable": false
                },
                "chan_multiSF_0": {
                        "desc": "Lora MAC, 125kHz, all SF, 903.9 MHz",
                        "enable": true,
                        "radio": 0,
                        "if": -400000
                },
                "chan_multiSF_1": {
                        "desc": "Lora MAC, 125kHz, all SF, 904.1 MHz",
                        "enable": true,
                        "radio": 0,
                        "if": -200000
                },
                "chan_multiSF_2": {
                        "desc": "Lora MAC, 125kHz, all SF, 904.3 MHz",
                        "enable": true,
                        "radio": 0,
                        "if": 0
                },
                "chan_multiSF_3": {
```

```
        "desc": "Lora MAC, 125kHz, all SF, 904.5 MHz",
        "enable": true,
        "radio": 0,
        "if": 200000
},
"chan_multiSF_4": {
        "desc": "Lora MAC, 125kHz, all SF, 904.7 MHz",
        "enable": true,
        "radio": 1,
        "if": -300000
},
"chan_multiSF_5": {
        "desc": "Lora MAC, 125kHz, all SF, 904.9 MHz",
        "enable": true,
        "radio": 1,
        "if": -100000
},
"chan_multiSF_6": {
        "desc": "Lora MAC, 125kHz, all SF, 905.1 MHz",
        "enable": true,
        "radio": 1,
        "if": 100000
},
"chan_multiSF_7": {
        "desc": "Lora MAC, 125kHz, all SF, 905.3 MHz",
        "enable": true,
        "radio": 1,
        "if": 300000
},
"chan_Lora_std": {
        "desc": "Lora MAC, 500kHz, SF8, 904.6 MHz",
        "enable": true,
        "radio": 0,
        "if": 300000,
        "bandwidth": 500000,
        "spread_factor": 8
},
"chan_FSK": {
        "desc": "FSK disabled",
        "enable": false
},
"tx_lut_0": {
        "desc": "TX gain table, index 0",
        "pa_gain": 0,
        "mix_gain": 8,
        "rf_power": -6,
        "dig_gain": 0
```

```
},
"tx_lut_1": {
        "desc": "TX gain table, index 1",
        "pa_gain": 0,
        "mix_gain": 10,
        "rf_power": -3,
        "dig_gain": 0
},
"tx_lut_2": {
        "desc": "TX gain table, index 2",
        "pa_gain": 0,
        "mix_gain": 12,
        "rf_power": 0,
        "dig_gain": 0
},
"tx_lut_3": {
        "desc": "TX gain table, index 3",
        "pa_gain": 1,
        "mix_gain": 8,
        "rf_power": 3,
        "dig_gain": 0
},
"tx_lut_4": {
        "desc": "TX gain table, index 4",
        "pa_gain": 1,
        "mix_gain": 10,
        "rf_power": 6,
        "dig_gain": 0
},
"tx_lut_5": {
        "desc": "TX gain table, index 5",
        "pa_gain": 1,
        "mix_gain": 12,
        "rf_power": 10,
        "dig_gain": 0
},
"tx_lut_6": {
        "desc": "TX gain table, index 6",
        "pa_gain": 1,
        "mix_gain": 13,
        "rf_power": 11,
        "dig_gain": 0
},
"tx_lut_7": {
        "desc": "TX gain table, index 7",
        "pa_gain": 2,
        "mix_gain": 9,
```

```
        "rf_power": 12,
        "dig_gain": 0
},
"tx_lut_8": {
        "desc": "TX gain table, index 8",
        "pa_gain": 1,
        "mix_gain": 15,
        "rf_power": 13,
        "dig_gain": 0
},
"tx_lut_9": {
        "desc": "TX gain table, index 9",
        "pa_gain": 2,
        "mix_gain": 10,
        "rf_power": 14,
        "dig_gain": 0
},
"tx_lut_10": {
        "desc": "TX gain table, index 10",
        "pa_gain": 2,
        "mix_gain": 11,
        "rf_power": 16,
        "dig_gain": 0
},
"tx_lut_11": {
        "desc": "TX gain table, index 11",
        "pa_gain": 3,
        "mix_gain": 9,
        "rf_power": 20,
        "dig_gain": 0
},
"tx_lut_12": {
        "desc": "TX gain table, index 12",
        "pa_gain": 3,
        "mix_gain": 10,
        "rf_power": 23,
        "dig_gain": 0
},
"tx_lut_13": {
        "desc": "TX gain table, index 13",
        "pa_gain": 3,
        "mix_gain": 11,
        "rf_power": 25,
        "dig_gain": 0
},
"tx_lut_14": {
        "desc": "TX gain table, index 14",
```

```
                        "pa_gain": 3,
                        "mix_gain": 12,
                        "rf_power": 26,
                        "dig_gain": 0
                },
                "tx_lut_15": {
                        "desc": "TX gain table, index 15",
                        "pa_gain": 3,
                        "mix_gain": 14,
                        "rf_power": 27,
                        "dig_gain": 0
                }
        },
        "gateway_conf": {
                "server_address": "router.us.thethings.network",
                "serv_port_up": 1700,
                "serv_port_down": 1700,
                "servers": [ {
                        "server_address": "router.us.thethings.network",
                        "serv_port_up": 1700,
                        "serv_port_down": 1700,
                        "serv_enabled": true
                } ]
        }

}
```


Fonte: Retirado de Network (2017c).