UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

CAIO CÉSAR OLIVEIRA RABELO

DESENVOLVIMENTO DE PROTOCOLO BASEADO EM AODV USANDO CAMADA FÍSICA DO LORA

PATOS DE MINAS
2018

CAIO CÉSAR OLIVEIRA RABELO

DESENVOLVIMENTO DE PROTOCOLO BASEADO EM AODV USANDO CAMADA FÍSICA DO LORA

Trabalho final como um dos requisitos parciais para aprovação na disciplina trabalho de conclusão de curso 2, do curso em Engenharia eletrônica e telecomunicações.

Orientador: Prof. Me. Alexander Bento Melo

PATOS DE MINAS
2018

CAIO CÉSAR OLIVEIRA RABELO

DESENVOLVIMENTO DE PROTOCOLO BASEADO EM AODV USANDO CAMADA FÍSICA DO LORA

Trabalho final como um dos requisitos parciais para aprovação na disciplina trabalho de conclusão de curso 2, do curso em Engenharia eletrônica e telecomunicações.

Patos de minas – MG, 10 de agosto de 2018

BANCA EXAMINADORA:

_____

Prof. Me. Alexander Bento Melo
Universidade Federal de Uberlândia
Orientador

_____

Prof. Me. Gustavo Nozella Rocha
Universidade Federal de Uberlândia
Examinador

_____

Prof. Dr. Laurence Rodrigues do Amaral
Universidade Federal de Uberlândia
Examinador

RESUMO

Nos últimos anos, o conceito de I*nternet of Things* (IoT) está cada vez mais presente e o número de dispositivos IoT cresce em uma escala logarítmica. Grande parte desses dispositivos IoT se localizam em áreas urbanas, onde a concentração de sensores sem fio, entre outros dispositivos, é grande. Também não podemos descartar as aplicações remotas de IoT como, por exemplo, em áreas agrícolas, em subestações de energia, hidrelétricas etc, qualquer que seja o cenário a demanda por comunicação será considerável. Existem várias propostas de redes e protocolos para atender as demandas de IoT, entre elas temos as LPWAN's, redes de baixa potência com grande área de cobertura. Em uma dessas tecnologias podemos encontrar o LoRa ™, que possui uma série de especificações definidas pela LoRaWAN *Specification* 1R0, que é basicamente uma topologia em estrela concentrando os dispositivos em um *gateway* por onde as mensagens são encaminhadas para a rede externa (internet), e a nível físico uma modulação proprietária baseada em *Chirp Spread Spectrum* (CSS). O trabalho proposto tem como objetivo adaptar um protocolo de roteamento na modulação LoRa de forma que dispositivos dentro da área de cobertura possam trocar pacotes de informação, ampliando ainda mais a área de cobertura da rede e possibilitando a troca de informações entre dispositivos sem a necessidade de um concentrador (*gateway*). Para isso será feito um estudo nas áreas de: redes Ad hoc, *Carrier Sense* para possibilitar a implementação de um protocolo de roteamento baseado no AODV.

Palavras-chave: IoT, LoRa, LPWAN, AODV, Redes de sensores.

ABSTRACT

In recent years, the concept of Internet of Things (IoT) is increasingly present, and the number of IoT devices grows on a logarithmic scale. Most of these IoT devices are located in urban areas, where the concentration of wireless sensors, among other devices, is large. But we can not rule out remote IoT applications such as agricultural areas, power substations, hydroelectric plants, whatever the scenario is, the demand for communication will be considerable. There are several proposals of networks and protocols to meet the demands of IoT, among them we have the LPWAN's, low power networks with large coverage area. In one of these technologies we can find LoRa ™, which has several specifications defined by LoRaWAN Specification 1R0, which is basically a star network topology concentrating all devices in a gateway where they carry the messages to the external network (internet). At the physical level a proprietary modulation based on Chirp Spread Spectrum (CSS). The proposed work aims to fit a routing protocol in LoRa modulation so that devices within the coverage area can exchange information packets, further extending the coverage area of the network and enabling the exchange of information between devices without the need for a hub. For this, a study will be made in these areas: Ad hoc networks, Carrier Sense to enable the implementation of an AODV-based protocol.

Keywords: IoT, LoRa, LPWAN, AODV, Sensor Networks.

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API - Application program interface

BW - Largura de banda (*Bandwidth*)

CSMA/CA - *Carrier-sense multiple access with collision avoidance*

CD – Taxa de código (*Coding rate*)

CSS – *Chirp Spread Spectrum*

ISM – Industrial científico e médico (Industrial Scientific and Medical)

IoT – Internet das coisas (Internet of things)

LPWAN – Rede de longa distância e baixa potência (*Low Power Wide Area Network*)

M2M – Máquina a Máquina (*Machine to Machine*)

NAV - Vetor de alocação de rede (*Network Allocation Vector*)

RTS – *Request to send*

CTS – *Clear to send*

RREQ – *Route Request*

RREP – *Route Reply*

RERR – *Route Error*

OSI – *Open System Interconnection*

SUMÁRIO

1. INTRODUÇÃO

O crescente número de dispositivos conectados a internet, na qual possibilitou enxergar um novo conceito: *Internet of things* (IoT), ou internet das coisas, onde temos uma nova forma de interagirmos com o mundo por meio de dispositivos embarcados. Extraindo mais dados do mundo a nossa volta e os interpretando podemos ter uma ampla gama de tecnologias e soluções. A próxima revolução será a interconexão entre vários sensores e atuadores para formar um ambiente inteligente.[1]

Nesse contexto, vemos a necessidade de integrar esses dispositivos à internet entrando em cena as tecnologias de comunicação de baixa potência, as *Low Power Wide Area Network* (LPWAN).

Atualmente existem várias tecnologias de comunicação para IoT, tais como: Ingenu, LoRa, Weightless e SigFox. Cada umas dessas empresas utilizaram um modelo de negócios para alcançar seus objetivos. Duas delas se destacam no mundo do IoT: LoRa e SigFox.

A tecnologia da SigFox se consiste no uso de bandas não licenciadas, *industrial, scientific and medical* (ISM) que varia de acordo com a regulamentação de cada região, os rádios possuem as seguintes zonas configuráveis[2]:

- Europa, Irã, Omã e Africa do Sul usam 868MHz (14 dBm);
- EUA, México e Brasil em 902MHz (22 dBm);
- Japão em 923MHz (14 dBm);
- Argentina, Colômbia, Austrália, Nova Zelândia, Hong Kong, Singapura e Taiwan em 920MHz (22 dBm).

No caso da SigFox, a empresa possui todos os direitos da tecnologia, desde a rede até os end-points. Para manter o preço baixo dos end-points, a SigFox fornece a tecnologia dos end-points para fabricantes de chips como a STMicroelectronics, Atmel e Texas Instruments. Porém, a venda dos end-points não é a principal fonte de renda da empresa. A SigFox lucra por meio dos royalties das operadoras de rede. Apesar dos end-points serem relativamente baratos, o uso da rede irá depender dos modems licenciados e do restante da infraestrutura, então o cliente terá que depender da infra estrutura proprietária da SigFox[3]. A arquitetura de

rede da SigFox possui uma topologia em estrela, na qual vários objetos (*end-points*) se conectam em *gateways* da SigFox, os gateways por sua vez se conectam na nuvem por meio de pacotes IP e, posteriormente, são organizados nos servidores também da SigFox, onde a coleta de dados pode ser feita pela *application program interface* (API) da própria empresa. Essa arquitetura pode ser observada conforme Figura 1.1.

Figura 1.1 - Arquitetura de rede da SigFox.

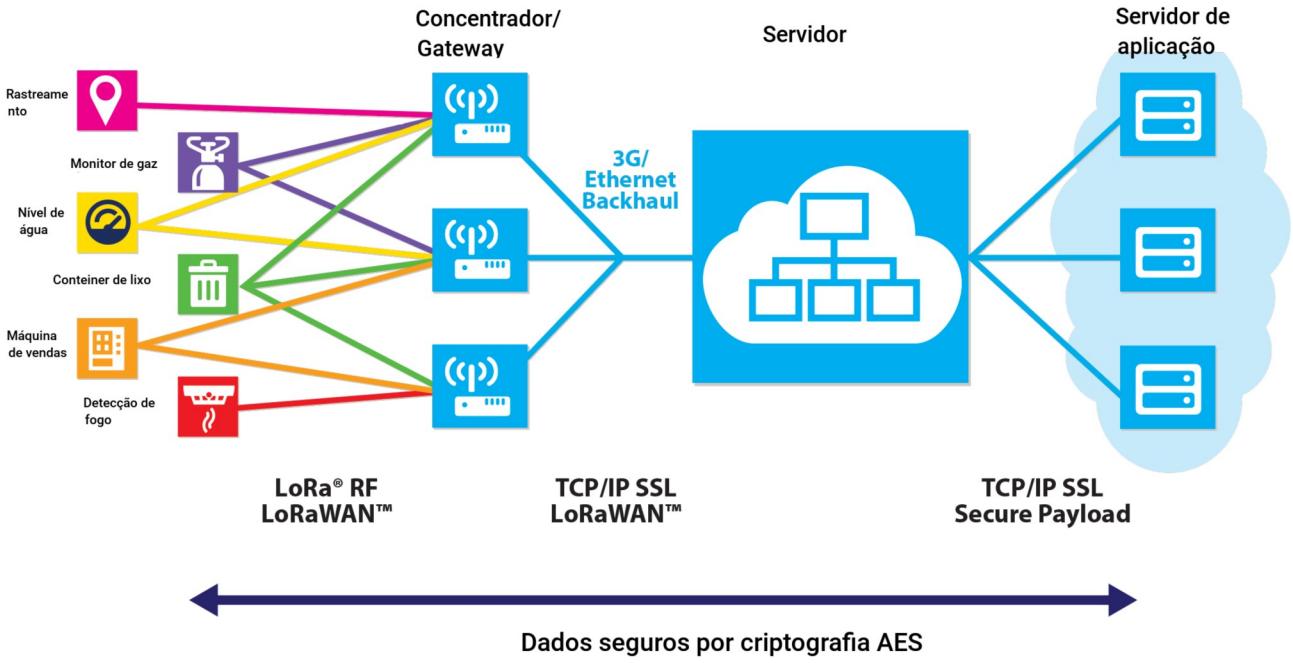
Fonte: Site oficial da SigFox. [4]

Com isso essa tecnologia só é realmente acessível para as corporações, dificultando o desenvolvimento dos projetos na plataforma da SigFox por micro empresas e *freelancers*.

A tecnologia LoRa™ foi primeiramente desenvolvida pela Cycleo, uma empresa francesa adquirida pela SemTech [5] e atualmente mantida pela *LoRa™ Alliance*, uma organização sem fins lucrativos formada por várias empresas interessadas em LPWAN para IoT, M2M, *smart city* e aplicações industriais [6]. É uma tecnologia relativamente mais aberta que a da SigFox pois qualquer pessoa pode adquirir no site da LoRa *alliance* as especificações do LoRaWAN™ e fabricar os próprios componentes de rede (*gateways* e módulos). Porém na prática, observamos que a maior parte (se não todos) dos chips são fabricados pela SemTech. Esse modelo de negócios tem como resultado a maior facilidade para comprar e desenvolver tecnologias usando LoRa, principal característica que viabiliza esse projeto.

Assim com a maioria das LPWAN's, o LoRa usa a banda não licenciada ISM e é composta por uma topologia em estrela. Podemos ainda separar essa estrutura em três componentes principais: *End-devices,* gateways e servidores. Os end-devices são os sensores e/ou atuadores conectados na LoRaWAN, fazendo interface por meio da camada física LoRa/OOK com os gateways, que por sua vez são concentradores, conectando a vários end-devices. Os gateways se conectam nos servidores por meio de pacotes IP. Os servidores por sua vez rodam softwares em back-end para realizar a interface com a aplicação. Tal estrutura de rede pode ser visualizada na Figura 1.2.

Figura 1.2 - Topologia das redes LoRaWAN



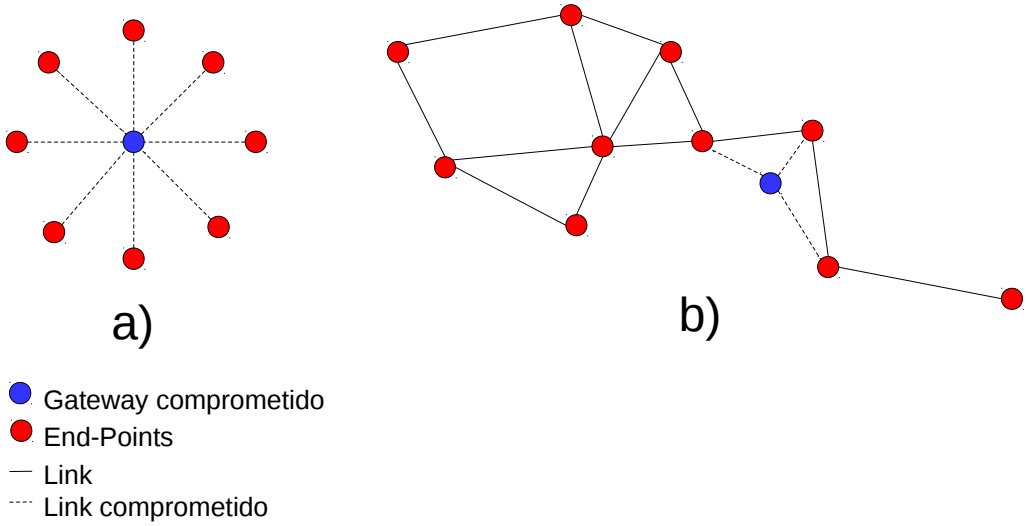
Fonte: Site oficial da SemTech.[7]

1.1 PROBLEMA

Como essas redes possuem o tipo de topologia escrito nas seçoes anteriores, temos que a área de cobertura (apesar de ser vasta se comparada com outras tecnologias sem fio) depende da quantidade de gateways espalhados em uma região. Os *gateways* das redes LPWAN tendem a ser bem caros, já que possuem uma eletrônica mais elaborada, podendo inviabilizar projetos de baixo investimento que demandem uma longa área de cobertura. E por se tratar de uma topologia centralizada, o mal funcionamento de um *gateway* compromete

toda a rede, isso não ocorre em uma topologia descentralizada como a *mesh*, a situação pode ser visualizada na Figura 1.3.

Figura 1.3 - Comparação entre uma topologia centralizada (a) e descentralizada (b) quanto a falha no *gateway*



1.2 TEMA

Adaptar um protocolo de roteamento já existente para ser usado de forma integrada com a camada física do LoRa afim de que ocorra a transmissão dos pacotes de dados entre os nós da rede em um contexto de rede de sensores onde a capacidade de processamento e energia são recursos limitados.

1.3 OBJETIVO

1.3.1 Objetivo Geral

Nesse projeto foi implementado o protocolo AODV no contexto das redes de sensores, onde a capacidade computacional é limitada. Para que seja possível, será feita algumas alterações no protocolo AODV e a implementação do algoritmo CSMA/CA para evitar as colisões nas transmissões já que a modulação LoRa não possui esse tipo de recurso.

Todo o sistema será testado paralelamente a construção do protótipo. Experimentos ao longo do desenvolvimento do protótipo irá fornecer os dados necessário para certificar o coreto funcionamento nos diversos cenários.

1.3.2 Objetivo Específico

O protótipo será desenvolvido em duas plataformas diferentes, microcontroladores Atmega328 (plataforma Arduíno) e uma placa Raspberry Pi (Ver. 3) usando Python juntamente com o sistema operacional, o Raspibian. O algoritmo será comum a todas as plataformas, mínimas alterações deverão ser necessárias devido a diferença entre as plataformas e linguagens de programação utilizadas.

O AODV a ser implementado irá passar por mudanças em sua estrutura com o objetivo de adaptá-lo para o microcontrolador. Dentro dessas mudanças, a redução da tabela de roteamento e a eliminação de alguns mecanismos de manutenção de rota. Tudo para que o protocolo seja leve o suficiente para caber no *payload* de 255bits da camada física do transceptor LoRa.

Os cenários para avaliar o correto funcionamento do protocolo é, em uma primeira etapa, testes estáticos onde todos os elementos da rede em um ponto fixo e testar a formação de rotas. Disposição dos dispositivos será modificada e todos os dados de tabelas e listas serão zeradas a cada novo teste. Dois cenários essenciais serão criados: todos os dispositivos em um mesmo raio e cada dispositivo enxergando apenas seu vizinho (formação em série). Com esses dois cenários poderá analisar o funcionamento do mecanismo de descoberta de rota do protótipo.

1.4 REFERENCIAL TEÓRICO

O referencial teórico será baseado em artigos e publicações do IEEE e documentos técnicos como: RFC 3561, especificações da *LoRa Alliance e data-sheet* dos módulos SX1278 que serão usados no protótipo.

2. PROTOCOLOS NECESSÁRIOS

2.1 A CAMADA FÍSICA DO LORA

A tecnologia de rádio do LoRa usa um tipo de modulação em espalhamento espectral, modulação baseada no Chirp Spread Spectrum (CSS). O motivo do uso dessa modulação é explicado pela teorema da capacidade do canal.

De acordo com o teorema de capacidade do canal de Shannon, a capacidade de transmissão de um canal depende da relação sinal/ruído e da largura de banda do canal de acordo com (1).

$$c = B \cdot \log_2\left(1 + \frac{S}{N}\right) \quad (1)$$

Na qual “c” é a capacidade do canal em [bit/s] e “S/N” é a relação sinal ruido.

Com isso podemos tirar como conclusão que: Com baixas taxas de transmissão e uma alta largura de banda, conseguimos um canal funcionando mesmo com uma péssima relação sinal/ruido. Isso viabiliza os links de longo alcance obtido pela modulação CSS [8].

2.1.1 Fator de espalhamento

Na modulação LoRa, podemos manipular a taxa de bit afim de conseguir a melhor relação de distância e taxa respeitando o teorema da capacidade do canal citado, para alcançarmos uma distância maior temos que abrir mão da taxa e bit e vice-versa. Essa manipulação é feita alterando um parâmetro da modulação chamado fator de espalhamento, ou *spreading factor.*

O fator de espalhamento é expresso em bits por símbolo, ou seja quantos bits em cada alteração na frequência. A taxa de bit dessa modulação pode ser calculada em (2), onde SF é o fator de espalhamento e BW a largura de banda[8].

$$R_b = SF \cdot \frac{1}{\frac{2^{SF}}{BW}} \quad (2)$$

Com (2) pode-se obter o período de cada símbolo dado em (3). [8]

$$T_S = \frac{s^{SF}}{BW} \quad (3)$$

Olhando no *datasheet*, temos os seguintes fatores de espalhamento mostrado na Tabela 2.1:

Tabela 2.1 - Relação entre o fator de espalhamento e valor do registrador

Fator de espalhamento (chips/símbolo)	Valor do registrador
64	0b0110
128	0b0111
256	0b1000
512	0b1001
1024	0b1010
2048	0b1011
4096	0b1100

2.1.2 Largura *de banda*

Outro parâmetro importante é a largura de banda (BW) que essa modulação irá usar. De acordo com o *datasheet* do SX1278, podemos variar a largura de banda de acordo com o valor do registrador do chip, Tabela 2.2 [9].

Tabela 2.2 - Relação entre largura de banda e valor do registrador

Largura de banda (kHz)	Valor no registrador Bw
7.8 kHz	0b0000
10.4 kHz	0b0001
15.6 kHz	0b0010
20.8 kHz	0b0011
31.25 kHz	0b0100
41.7 kHz	0b0101
62.5 kHz	0b0110
125 kHz	0b0111
250 kHz	0b1000
500 kHz	0b1001

2.1.3 Coding rate

E para termos uma taxa de erro arbitrariamente baixa, temos o parâmetro de taxa de codificação, *coding rate* (CR), no caso dos módulos da SemTech usam codificação de Huffman nas seguistes taxas mostrada na Tabela 2.3:

Tabela 2.3 - Relação entre a taxa de codificação e o valor do registrador

Taxa de codificação	Valor do registrador
4/5	0b001
4/6	0b010
4/7	0b011
4/8	0b100

2.1.4 Estrutura de pacotes

Os pacote de transmissão do LoRa possui basicamente:

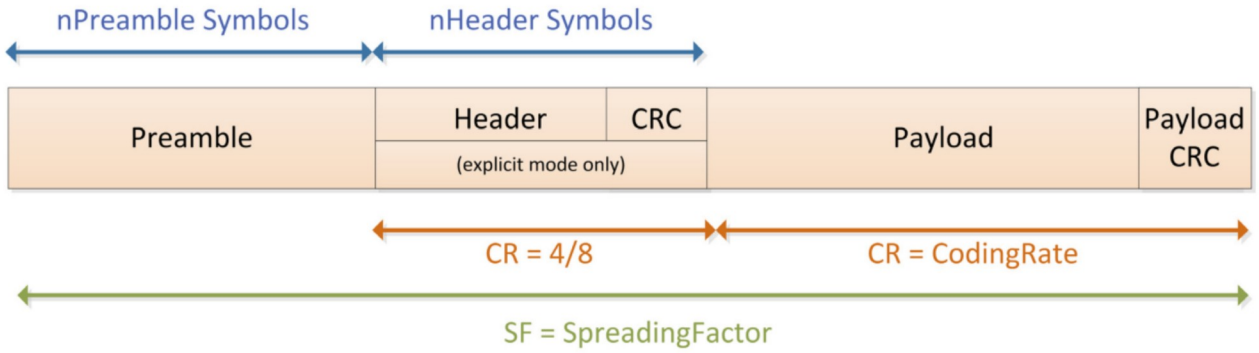
- Preâmbulo;
- Cabeçalho (*header*);
- Carga útil (*payload*).

O preâmbulo é a parte inicial do pacote, ele é responsável pela sincronia, engatilhando algumas interrupções no receptor para que assim possa receber o restante do pacote. O preâmbulo é um parâmetro configurável, podemos alterar seu comprimento pelo registrador "*PreambleLength*", por padrão é 12. [9]

O cabeçalho é dividido em dois modos: modo implícito e explícito. No modo implícito, o cabeçalho carrega a informação de todos os parâmetros da modulação e é usado nos casos em que o receptor não conhece os parâmetros da modulação. Já no explícito, não é informado nenhum parâmetro da modulação, partindo do ponto em que o receptor conheça todos os parâmetros. [9]

E por fim a carga útil, onde a informação a ser transmitida será contida. Tanto o cabeçalho quanto a carga útil possuem seu próprio código corretor de erro. A estrutura do pacote pode ser visualizado graficamente pela Figura 2.1.

Figura 2.1 - Estrutura do pacote



Fonte: Datasheet SX1278

2.1.5 *Time on Air*

Um parâmetro importantíssimo que devemos saber para que todo o sistema funcione como esperado é o *time on air*, o tempo de duração de um pacote. Sabendo esse tempo pode-se alertar aos dispositivos dentro do raio de alcance, o período em que o canal será usado. Esse tempo será usado pelo algoritmo CSMA/CA, abordado nas seções seguintes.

De acordo com a documentação técnica do SX1278, o tempo que um pacote ocupa pode ser calculado somando o tempo de cada parte do pacote. Para o preâmbulo, o cálculo é demonstrado em (4).

$$T_{preamble} = (n_{preamble} + 4.25).T_{sym} \quad (4)$$

Onde o “n” é o comprimento do preâmbulo, definido pelos registradores “RegPreambleMsb” e “RegPreambleLsb” que por padrão é 12.

Para o cálculo do período da carga útil, temos que saber a quantidade de símbolos que pode ser calculada em (5), e multiplicando pelo período de cada símbolo obtemos o período completo do *payload*, de acordo com (6).

$$n_{payload} = 8 + max\left(ceil\left[\frac{8.PL - 4SF + 28 + 16.CRC - 20.IH}{4.(SF - 2.DE)}\right].(CR + 4), 0\right) \quad (5)$$

$$T_{payload} = n_{payload} . Ts \quad (6)$$

Sabendo o período das partes do pacote, basta somá-las para obter o tempo total de transmissão ou duração do pacote de acordo com (7).

$$T_{packet} = T_{preamble} + T_{payload} \quad (7)$$

2.2 O CSMA/CA

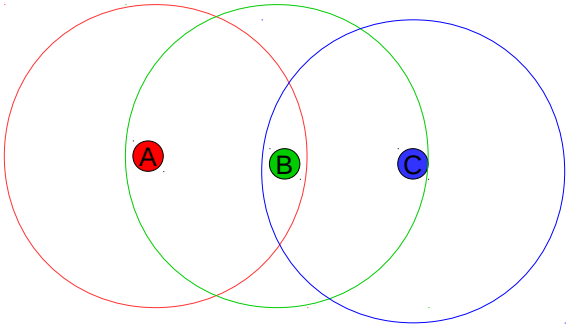
No caso dos sistemas de transmissão sem fio, os pacotes chegam a todas as estações dentro de uma área, e na maior parte dos casos esse meio é uma modulação com a frequência compartilhada entre todas as estações, ou seja dentro de um mesmo domínio de interferência eletromagnética. *O Carrier-sense multiple access with collision avoidance* (CSMA/CA) é um protocolo de acesso múltiplo que opera na camada de data link no modelo OSI. Como a sigla sugere, o CSMA/CA é um protocolo responsável por evitar essas colisões de pacotes dentro do meio. [10]

A primeira etapa do *carrier sense multiple access* é a nível físico, simplesmente escutar o canal e retornar se está sendo usado ou não. Com o canal livre, é feita uma verificação a nível lógico, por meio dos mecanismo de RTS/CTS (também conhecido como *virtual carrier sense*) afim de evitar o problema do terminal oculto (*Hidden Node Problem*). [10]

2.2.1 O problema do terminal oculto

O problema do terminal oculto é um problema comum nas redes sem fio, pois um terminal não consegue fazer o carrie sense (escutar por outras transmissões) em outro terminal que esteja fora de seu raio de alcance. Assim, em caso de uma tentativa de estabelecer a comunicação entre dois terminais na presença de um terceiro(terminal oculto), há uma chance dos terminais em questão realizar transmissões simultâneas e gerar uma interferência na recepção. Situação exemplificada na Figura 2.2, onde o terminal “escondido” é apenas um, se multiplicarmos o número de terminais desse exemplo, a probabilidade de transmissões simultâneas aumenta ainda mais.

Figura 2.2 - Problema do terminal oculto com apenas um terminal oculto.



Uma abordagem que o padrão 802.11 usa para resolver o problema do terminal oculto é o uso de pacotes de aperto de mão (*handshake*) conhecido como RTS/CTS *exchange*, explicado com mais detalhe na próxima seção.

2.2.2 O mecanismo RTS/CTS

A estação que deseja enviar um pacote, e sabendo que o meio físico está disponível, primeiro fica em espera por um tempo aleatório (*backoff*) até enviar uma requisição de transmissão chamada de *Request to Send (RTS)*, contendo em seu *frame o* tempo para a transmissão do pacote (que pode ser determinado pelo cálculo demostrado na secção 2.2). Nesse estágio, todos as estações dentro do raio escutarão essa requisição, e a nível lógico, se tornarão indisponíveis para transmitir durante o período de tempo contido na requisição recebida. Essa indisponibilidade é definida em cada estação com o nome de *Network Allocation Vector* (NAV). Após esse período se qualquer estação deseja transmitir um pacote, entrarão primeiramente em um tempo de espera aleatório antes de transmitir, a importância desse tempo de espera aleatório é que ele evita a colisão de pacotes (requisições) após a transmissão de uma estação dentro do raio. No caso do CSMA esse período de espera é definido pelo algorítimo chamado de *binary exponential backoff*. Após N tentativas de retransmissão, o tempo de espera é aumentado em uma ração exponencial 2^N que dobra o tempo de espera a cada tentativa até que a quantidade de tentativa máximas seja alcançada. O tempo de espera exponencial binário é essencial para funcionamento do algoritmo, pois a cada tentativa o espaço temporal entre as requisições aumenta, reduzindo a chance de colisões.

Todo esse mecanismo funciona a custo de um aumento no *overhead* e consequentemente uma redução do *throughput*. Para as *sensor networks*, o baixo *throughput* não chega a ser um problema justamente por essas redes funcionarem a baixa taxa de dados.

2.3 MANET's

Dentro das *Mobile wireless sensor network* (MWSN), rede de sensores de alta capacidade de adaptação, se encontra as MANET's, ou *Mobile ad hoc network. As MANET's* são redes para dispositivos móveis que usam o conceito de rede ad-hoc, rede que dispensa um gateway visto que cada dispositivo (node) opera como um host e gateway ao mesmo tempo com a possibilidade de mudança na posição geográfica de cada dispositivo graças a topologia dinâmica. [11]

Existem três formas de classificar os protocolos que compõe as MANET's:

- Protocolos Proativos;
- Protocolos Reativos;
- Protocolos Híbridos.

Os protocolos proativos ou "*table driven*", como o nome sugere, são protocolos que usam uma tabela de roteamento em todos os nodes e essa tabela é atualizada periodicamente. Com isso temos a desvantagem de sobrecarregar a rede caso ela não comporte a quantidade de tráfego. Para que os protocolos proativos sejam implementados em uma topologia sujeita a mudanças, a maior parte desses protocolos precisam guarda algumas tabelas de roteamento para cada nó na rede, tornando uma solução não atrativa para grandes redes MANET's (Ade & Tijare, 2009) ou mesmo redes de sensores já que os recursos de processamento e armazenamento de dados são limitados.

Já os protocolos reativos são mais relevantes em links de baixa taxa, justamente por formarem rotas apenas quando elas são necessárias (*on demand*), reduzindo a quantidade de pacotes de controle pelas funções de descoberta e manutenção de rotas. Por esse motivo, protocolos reativos das MANET's melhor se adéquam no contexto das redes de sensores.

E os híbridos juntam os mecanismos tanto dos protocolos proativos quanto dos reativos, permitindo que dispositivos mais próximos possam trabalhar juntos formando um

grupo com o objetivo de reduzir o *overhead* causado pelos mecanismos de descoberta de rota.[12]

2.3.1 Protocolos reativos

Nos protocolos reativos há em essência funções para criação de rotas e funções para manutenção de rotas. O processo de descoberta de rota começa quando uma fonte deseja se comunicar com um destino sem possuir uma rota válida em sua tabela, enviando um broadcast para todos os dispositivos vizinhos um pacote de requisição de rota. Há algumas estrategias para a disseminação desses pacotes de *broadcast*, contudo o trabalho em questão irá descrever apenas uma das estrategias usada pelo AODV. Neste cenário a disseminação dos pacotes ocorre com a retransmissão do pacote de requisição pelos dispositivos intermediários de toda a rede, com isso todos os dispositivos recebem a requisição e a rede é inundada com requisições. Para evitar que cada dispositivo retransmita uma requisição, esses pacotes utilizam o conceito de números sequenciais (s*equence numbers*). A cada nova requisição feita por uma fonte, um número local é acrescido em 1 e essa informação é encapsulada no pacote de requisição. Desta forma cada requisição pode ser unicamente identificada, evitando duplicação de pacotes. [11]

A manutenção de rotas é feita toda vez que um dispositivo detecta um link quebrado. Sabendo todos os pulos de uma determinada rota, o dispositivo que detecta a link falho ao tentar repassar algum pacote de *unicast*, começa um reparo em todas as rotas que tenham como destino, a rota falha em questão e procura uma rota alternativa para o próximo pulo. [11]

2.4 O AODV

No protocolo AODV, citado na seção 2.3.1, as rotas são definidas de acordo com a demanda, caracterizando-o como um protocolo de roteamento reativo. Cada nó é responsável por armazenar uma tabela de rotas. A criação/manutenção de rotas é somente feita caso dois nós precisem se comunicar, evitando ao máximo pacotes de *broadcast*.

Caso um ponto (*source*) deseje comunicar-se com outro ponto (*destiny*), é feita uma consulta em sua tabela por uma rota ativa. Caso não exista uma rota ativa, é iniciado o processo de descoberta de rota ou *path discovery*.*[13]*

2.4.1 *Route Request* (RREQ)

A descoberta de rota é iniciada com a fonte mandando uma requisição de rota, *route request* (RREQ). Esse pacote é disseminado para os nós vizinhos por meio do *broadcast* e para evitar que uma requisição seja passada para frente mais de uma vez, causando loops na rede, cada requisição RREQ pode ser unicamente identificada por meio dos valores de *broadcast_id e source_addr em seu cabeçalho* como pode ser visto na Tabela 2.4. Então, antes de iniciar o processo de descoberta de rota (*path discovery*) enviando o pacote RREQ, o nó deve armazenar o seu endereço de rede e o broadcast_id e fazer o broadcast do pacote. Fazendo isso, evita-se que nó fonte processe seu próprio pacote RREQ.[14][15]

Tabela 2.4 - Cabeçalho do pacote RREQ.

source_addr
source_sequence_number
*broadcast*_id
destination_id
destination_sequence_number
hop_count

Então o processamento dos pacotes de requisição de rotas (RREQ) é feito somente pelos nós intermediários e no nó destino, da seguinte forma. Quando um nó recebe um pacote RREQ, é feita uma consulta em uma tabela local dos pares source_addr *e broadcast_id*, se for encontrado esses mesmos pares de variáveis localmente, quer dizer que esse mesmo *broadcast* já foi recebido e processado, então descarta-se esse pacote.[13] Caso contrário adiciona-se o par *source_addr e broadcast_id localmente e continua com o processamento da requisição.* A informação contida no RREQ então é atualizada, incrementando seu valor de hop_count em 1 e usado para atualizar ou criar uma rota até o nó fonte (source_addr) chamado de *reverse path*, ou caminho reverso. Então o pacote é retransmitido, mantendo as mesmas informações do pacote original, com exceção do contador de pulos (hop_count) que é incrementado por cada nó. [16]

Eventualmente o pacote RREQ chegara a seu nó destino, então o processamento desse pacote pelo destino é feito de forma semelhante ao no intermediário. Mas desta vez o nó destino não repassa o *broadcast* e começa uma outra parte da descoberta de rota enviando um pacote de reply, *route reply* (RREP).[17]

2.4.2 *Route Reply (RREP)*

Quando a requisição RREQ chega ao destino, o mesmo é responsável por mandar uma resposta de rota, *route reply* (RREP). O cabeçalho desse pacote contem o endereço fonte, endereço do destinatário (o mesmo que enviou o pacote RREP), o sequence number do destinatário, contagem de pulos e a vida util. A Tabela 2.5 contem os campos do pacote de resposta (*reply)*.

Tabela 2.5 - cabeçalho do pacote RREP.

source_addr
destination_addr
destination_squence_number
hop_count
lifetime

Para o controle de transmissão desses pacotes, não há a necessidade de identificar unicamente esse pacote, como é feito no *broadcast* do pacote RREQ. Pois a retransmissão do RREP é feita em *unicast*, já que dessa vez os nós conhecem o caminho reverso, *reverse path*, possibilitado pelo RREQ.[18][13]

2.4.3 Parâmetros do protocolo

Existem alguns parâmetros no protocolo que regulam o funcionamento de alguns mecanismos. Para uma descoberta de rota, *path discobery*, são usados o net traversal time, ou tempo, o tempo necessário para um pacote atravessar todo o diâmetro da rede. Com esse tempo o protocolo pode estimar o tempo em que um possível *reply* irá chegar, dimensionando com mais precisão o timeout da descoberta de rota. Ultrapassando esse limite de tempo, uma outra requisição de rota é enviada.[13] O número de requisições para estabelecer uma rota também é um parâmetro importante do protocolo e possui um número fixo até abortar o processo.

3. METODOLOGIA

Dentro do *LoRa alliance specification* temos vários parâmetros que devemos seguir para criar redes LoRaWAN, esse projeto irá utilizar apenas os parâmetros que priorise a qualidade do link dado uma taxa de dados que comporte toda a pilha de protocolos, independente das normas da *LoRa alliance.* Na primeira etapa um breve estudo foi feito, variando alguns parâmetros na modulação com o objetivo de obter uma relação entre distância, latência e taxa de bit que se adéque no contexto do projeto. Algoritmos para a realização dos experimentos em ambiente real foram feitos, evitando ao máximo simulações.

O trabalho será desenvolvido (na fase de planejamento) a parte lógica, desenvolver o protocolo reativo, cabeçalho e biblioteca do chip que irá em cada ponto e ainda um algoritmo CSMA/CA para evitar a colisão de pacotes nos pontos mais próximos. Posteriormente iremos aplicar essa lógica (na fase de execução) na prática em chips LoRa SX1278 fabricado pela SemTech com placas de desenvolvimento Arduino nano, e para o concentrador o mesmo chip SX1278 mais uma placa embarcada Raspberry Pi 3 que irá comunicar com a rede TCP/IP.

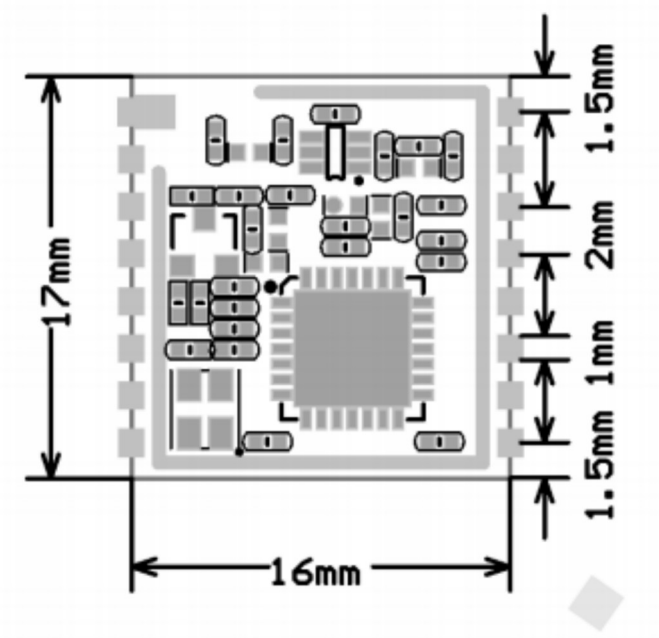
Para o teste inicial da rede, 4 módulos da SemTech foram usados. Um concentrador e três nós da rede configurados em diferentes distâncias para analisarmos a performance da rede. Note que o experimento será executado com poucos dispositivos pela falta de investimento no projeto e pelo grau de complexidade que a rede teria com mais dispositivos, dificultando ainda mais a execução do experimento.

4. CONSTRUÇÃO DO PROTÓTIPO

O protótipo serve como prova de conceito do trabalho, por ele podemos observar o desempenho da rede em ambiente real e documentar os problemas durante a implementação.

Para o protótipo do sistema, temos a divisão em dois dispositivos: um sistema embarcado Linux atuando como um concentrador de dados e microcontroladores para os demais pontos da rede. Ambos dispositivos usam o mesmo módulo transceptor AI-thinker RA-01, um módulo SMD de 17 por 16 mm usando em seu núcleo o chip Semtech SX1278. Todo o módulo é encapsulado com uma blindagem metálica e uma antena simples do tipo *spring* de 2.5 dBi.

Figura 4.1 - Dimensões do módulo Ra-01



O sistema embarcado é uma placa de desenvolvimento Raspberry Pi ver.3 com o módulo AI-Thinker RA-01 em seu barramento de entradas e saídas. Usando como linguagem de programação, o Python pela facilidade de desenvolvimento que a linguagem oferece. E com a conexão IEEE 802.11 AdHoc a um notebook, para debugar a rede e/ou levar as informações dos demais pontos da rede para uma rede IP.

Já para os demais pontos da rede, não há necessidade de um sistema embarcado custoso como o RaspberryPi, um microcontrolador que tenha os requisitos mínimos já é o

suficiente. Os requisitos para o acesso e bom funcionamento do protótipo, o microcontrolador escolhido tem necessariamente que ter suporte ao protocolo SPI de forma nativa e memória dinâmica suficiente para armazenar toda a tabela de roteamento. Para tal, a placa de desenvolvimento Arduíno nano com um microcontrolador Atmega328 é uma boa escolha inicial para a construção do protótipo. Devido a diferença entre as plataformas de desenvolvimento e as linguagens utilizadas, algumas mudanças sutis no algoritmo foram necessárias, por exemplo, no programa dos microcontroladores foi feito rotinas para tratar as interrupçoes usando apenas um temporizador. No decorrer das seções seguintes será abordado com mais detalhe a implementação nas duas plataformas.

4.1 Biblioteca para o chip SX1278

Para a criação da biblioteca do módulo Ra-01, é necessário conhecer seu funcionamento interno. Começando pela interface eletrônica, o módulo Ra-01 possui 16 pinos descritos na Tabela 4.1:

Tabela 4.1 - Relação dos pinos do módulo Ra-01.

Grupo	Pino	Descrição
Antena	ANT	Entrada para a antena
Alimentação	GND	Terra
	3.3V	Alimentação positiva tipicamente +3.3Volts (podendo variar de 2.5 a 3.7 Volts)
Reset	RESET	Pino de reset
SPI	NSS	Entrada do chip select
	MOSI	Entrada de dados da interface SPI
	MISO	Saida de dados da interface SPI
	SCK	Entrada do clock da interface SPI
Uso geral	DIO0	Digital I/O, pino configurado via software
	DIO1	Digital I/O, pino configurado via software
	DIO2	Digital I/O, pino configurado via software
	DIO3	Digital I/O, pino configurado via software
	DIO4	Digital I/O, pino configurado via software
	DIO5	Digital I/O, pino configurado via software

Fonte:

Apesar de programável, os pinos DIO (Digital I/O) possuem funções pré-determinadas acessadas pelos registradores RegDioMapping1 e RegDioMapping2 que mapeiam os pinos de DIO0 a DIO3 e DIO4 a DIO5 respectivamente. A relação dos registradores e sua função se encontra na Tabela 4.2.

Tabela 4.2 - Relação dos registradores e função dos pinos DIOX.

Valor	DIO5	DIO4	DIO3	DIO2	DIO1	DIO0
00	ModeReady	CadDetected	CadDone	FhssChange Channel	RxTimeout	RxDone
01	ClkOut	PllLock	ValidHeader	FhssChange Channel	FhssChange Channel	TxDone
10	ClkOut	PllLock	PayloadCrcE rror	FhssChange Channel	CadDetected	CadDone
11	-	-	-	-	-	-

As entradas e saídas do módulo utilizam tensão de 3,3 Volts, e como o chip é sensível à variação de tensão, é recomendável utilizar conversores lógicos no caso entradas e saídas diferentes de 3,3 Volts. É o caso do microcontrolador Atmega 328, que possui nível lógico alto de 5 Volts.

Para o protótipo, usou-se todos os pinos de comunicação/alimentação e apenas o DIO0, configurado como RxDone. Trabalhando em conjunto com um pino de interrupção externa de qualquer plataforma, o DIO0 serve como gatilho para o acionamento das funções de leitura da mensagem.

4.1.1 Setando os registradores

A primeira tarefa a ser feita após o circuito ser energizado é configurar os registradores do módulo de acordo com os parâmetros de modulação a ser usado. A única forma de setar os registradores de configuração do chip é quando em modo de operação *sleep*, onde o chip entra em estado de baixa energia. Isso é feito alterando os três primeiros bits do registrador de modo de operação (RegOpMode) para o valor do modo *SLEEP* de acordo com a Tabela 4.3.

Tabela 4.3 - Mapa de modos de operação do registrador RegOpMode.

Modo de operação	Valor do RegOpMode(0x02)
SLEEP	000

STDBY	001
Frequency synthesis TX (FSTX)	010
Transmit (TX)	011
Frequency synthesis RX (FSRX)	100
Receive continuous (RXCONTINUOUS)	101
receive single (RXSINGLE)	110
Channel activity detection (CAD)	111

4.1.2 Transmitindo um pacote

Para habilitar o chip SX1278 a transmitir um pacote, uma sequencia de registradores devem ser corretamente iniciada. Primeiro o chip deverá estar em modo standby, para ter acesso a todos os registradores. Em modo *standby*, o ponteiro da fila de dados e o registrador informando o tamanho do *payload* deveram ser zerados, RegFifoAddrPtr e Reg*Payload*Length respectivamente. Feito isso o chip estará pronto para receber o *payload* em sua fila interna, pelo registrador RegFifo e atualiza o tamanho da fila pelo registrador Reg*Payload*Length. Com todo o *payload* escrito na fila basta alterar o modo de operação para modo transmissão e acompanhar o valor da flag RegIrqFlags que ira indicar o fim do envio do pacote.

4.1.3 Modo recepção

Em modo recepção, o chip irá escutar continuamente por um preâmbulo, parte inicial do pacote LoRa, os dados recebidos vão para o registrador de filas (RegFifo) e levanta-se uma flag indicando que um dado foi adquirido.

Para o projeto, um incremento no processo foi feito. Toda vez que um pacote é recebido, habilita uma interrupção no microcontrolador/computador embarcado através da programação do pino DIO1. Assim o algoritmo pode ser escrito de forma mais inteligente, evitando a contante leitura da *flag* de recepção no módulo LoRa.

4.2 Implementação do algoritmo CSMA/CA

Afim de evitar o problema do terminal oculto, a solução escolhida para contornar esse problema foi implementar a troca de pacotes RTS/CTS do padrao 802.11. Todas as transmissões em *unicast* são feitas usando a troca de pacotes RTS/CTS, já nas transmissões em *broadcast*, que serão utilizadas nos pacotes RREQ do protocolo de roteamento, o CSMA

puro é utilizado devido a complexidade de se implementar um algoritmo em handshake onde o destinatário da mensagem é todos os pontos dentro do raio de alcance do rádio. Usando apenas o CSMA puro, evita-se o excesso de *overhead* na comunicação, crucial para a implementação com *hardware* limitado.[19]

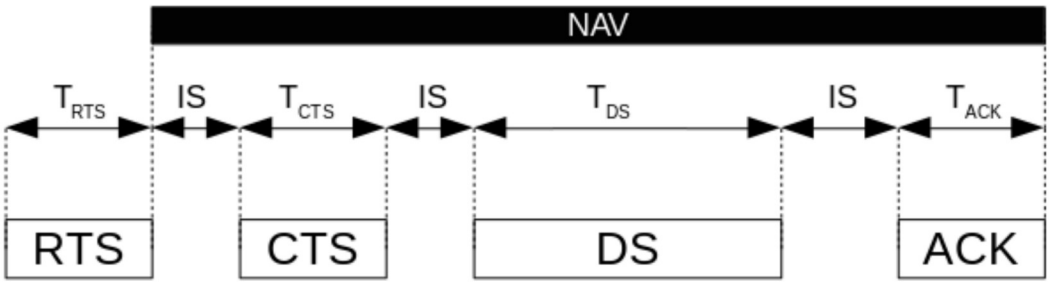
4.2.1 Tempo de espera aleatório

Um dos primeiros parâmetros específicos para essa implementação, é o tempo que o nó da rede irá esperar antes de começar a transmissão (mecanismo chave do CSMA puro). Um tempo muito curto pode causar colisões de pacotes e por outro lado, um tempo de espera muito elevado reduz a eficiência do algoritmo. A variância desse tempo é outro fator a se levar em conta, uma variação muito pequena pode aumentar a probabilidade de colisões se comparado com uma variação maior. Um dos problemas para se conseguir uma boa fonte de números aleatórios em um microcontrolador é a falta de entropia da fonte, os números são gerados em um processo inteiramente determinístico, principalmente se tiverem com um mesmo *firmware*. O problema direto de usar uma fonte pseudoaleatória para essa aplicação é que todos os nós da rede farão a mesma decisão, no mesmo tempo quando receberem um pacote de *broadcast*. A solução para aumentar a entropia dessas fonte é utilizar o conversor analógico/digital do microcontrolador com um pino flutuante. Com isso temos uma fonte com entropia suficiente para o funcionamento do CSMA/CA.

4.2.2 Cálculo do tempo de espera NAV

Outro parâmetro especifico na implementação desse algoritmo, é o tempo de *backoff*. Conforme a seção 2.2.2, é o tempo que nó da rede irá esperar antes de iniciar uma transmissão até o recebimento de uma confirmação de recebimento (ACK). Na seção 2.1.5 pudemos calcular o tempo que o pacote demora para ser transmitido de acordo com o manual técnico do fabricante, visto isso e sabendo o tamanho de todos os pacotes que compõe o restante da transmissão (RTS, CTS, DS e ACK) pode-se calcular o tempo de cada transmissão. Entretanto devido algumas variáveis não levadas em consideração para esse cálculo, como por exemplo o tempo de processamento do pacote, o tempo entre *frames* (Inter-frame Space). Na Figura 2.1 observa-se o período de todos os *frames*.

Figura 4.2 - Representação gráfica do período de uma transmissão.



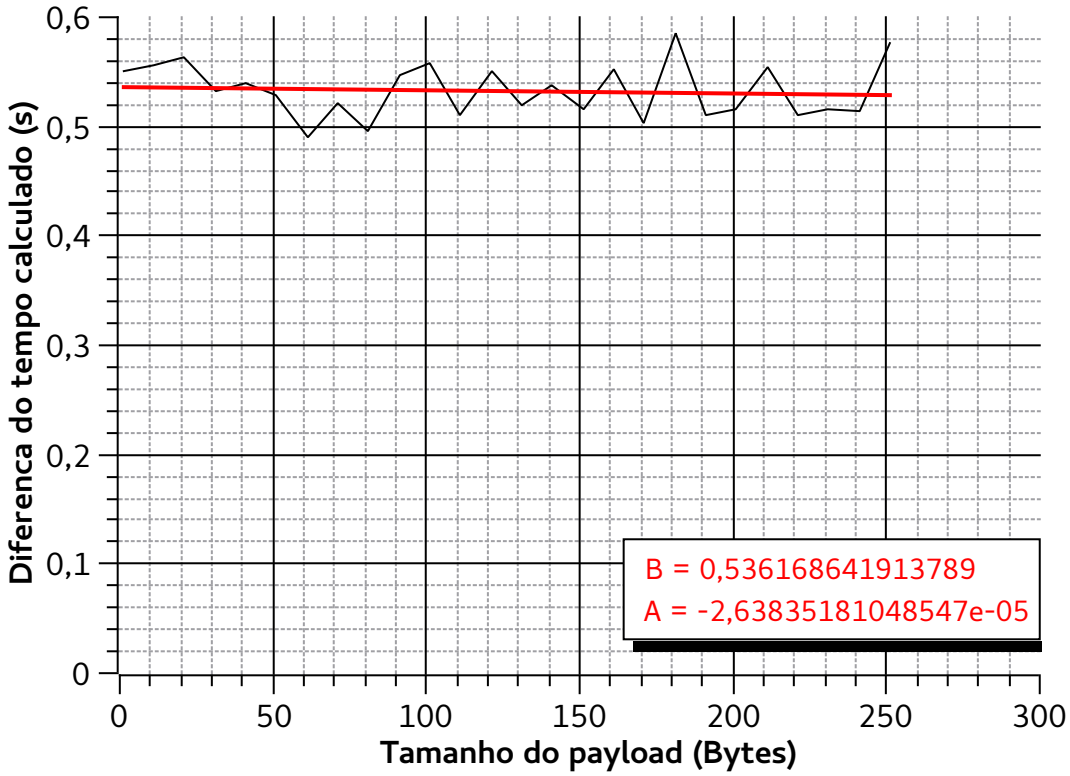
Fonte: autoria própria.

Sabendo que os frames CTS e ACK são fixos e o *frame* DS, onde carrega o *payload*, possui um tamanho variável e levando em consideração o calculo descrito na seção 2.1. Pode-se criar uma função que, dado o tamanho do *payload*, calcule o tempo estimado para o fim da transmissão. Restando apenas adequar esse cálculo inicial levando em conta a soma das distâncias inter-frames space (IFS).

Entretanto, é esperado que se tenha uma variação no tempo calculado com o tempo real em função da proposital omissão das distâncias entre *frames* no cálculo inicial. Desde que essa diferença seja previsível, a implementação um tempo de *backoff* funcional é possível com uma pequena correção.

Para verificar a diferença entre o tempo calculado inicialmente e o tempo real no envio de uma transmissão completa (considerando toda a troca de pacotes RTS/CTS) afim de definir um parâmetro a ser inserido no cálculo final, um experimento se encaixa bem para determinar o grau e a previsibilidade dessa variação. Para o experimento, um *script* que gera mensagens de 1 a 255 bytes com uma resolução de 10 bytes e 10 transmissões para cada tamanho de mensagem, totalizando 250 transmissões. O *script* pega o tamanho máximo do *payload* (255 bytes) e divide em X partes de acordo com a resolução predefinida e calcula a diferença dos tempos Y vezes, obtêm a média dessa diferença para cada tamanho de *payload* e salva tudo em um documento de texto. Para a análise dos resultados um gráfico com a diferença do tempo e o tamanho do *payload* para que, afim de definir um possível padrão de variação.

Figura 4.3 - Padrão de variação da diferença do tempo calculado e o tempo real em função do tamanho do payload.



Fonte: Autoria própria.

A Figura 4.3 mostra de forma visual os resultados obtidos pelo experimento. Aplicando uma regressão linear na função e observando os valores de coeficiente angular A e coeficiente linear B, podemos tirar algumas conclusões. Um valor do coeficiente angular tão próximo de zero indica uma invariância do tempo em questão e o tamanho do *payload*, o que é ótimo já que não tendo uma variação, tem-se apenas um valor constante a ser somado na conta final. Já com o coeficiente linear pode-se obter o valor médio dessa variação. Para a implementação, convêm utilizar um valor ligeiramente maior. Portanto para o cálculo final somamos todas as variáveis mostradas em (8). Onde T_{DS} é obtido em função do tamanho do *payload* e o tempo do CTS+ACK, por terem um tamanho fixo, simplificados em uma constante (0.1s).

$$T_{total} = T_{DS} + 0.1 + 0.6 \quad (8)$$

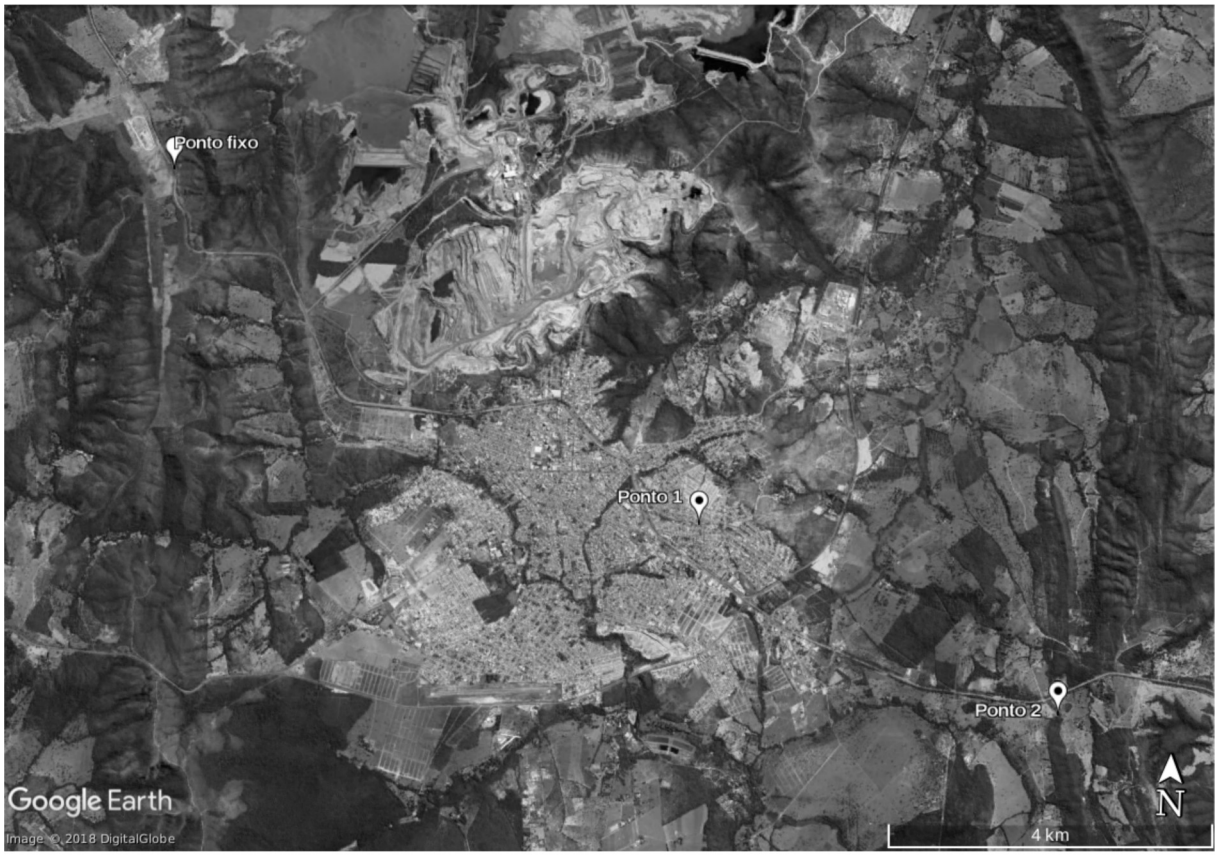
4.2.3 Resultados de desempenho em distâncias

Um experimento foi realizado para analisar os efeitos das longas distâncias entre dois dispositivos no algoritmo em questão. Usando o *script* do APÊNDICE A – SCRIPT E

REGISTROS DO TESTE CSMA/CA #1, que envia um número determinado de pacotes e salva em um aquivo de texto os dados de cada transmissão, que inclui o número do pacote, o *Received signal strength indication* (RSSI), estimativa do *signal-to-noise ratio* (SNR), tentativas e o tempo de transmissão. Foi usado como parâmetros o envio de 100 pacotes sem intervalo entre pacotes (intervalo igual a zero).

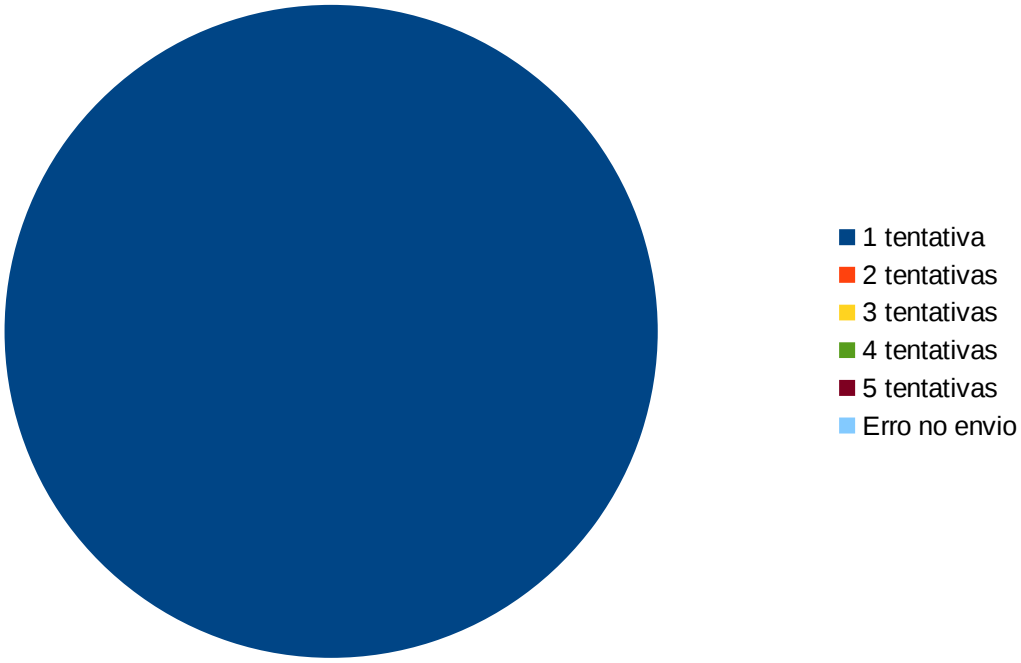
O experimento foi realizado na cidade de Paracatu - Minas Gerais aproveitando da topografia da região para obter grandes distâncias com visada direta. Com um protótipo fixado no pico de uma morro com vista para toda a cidade denominado de “Ponto fixo” pela Figura 4.4, o outro protótipo foi movido para os pontos 1 e 2 do mapa na Figura 4.4. Com isso tem-se distâncias de 8,26 Km entre o “Ponto fixo” e o “Ponto 1” e 13,27 Km entre o “Ponto fixo” e “Ponto 2” ambas distâncias calculadas pela linha reta entre os dois pontos considerando o deslocamento vertical (altitude).

Figura 4.4 - Mapa com a localização dos pontos onde foi realizado as medições.



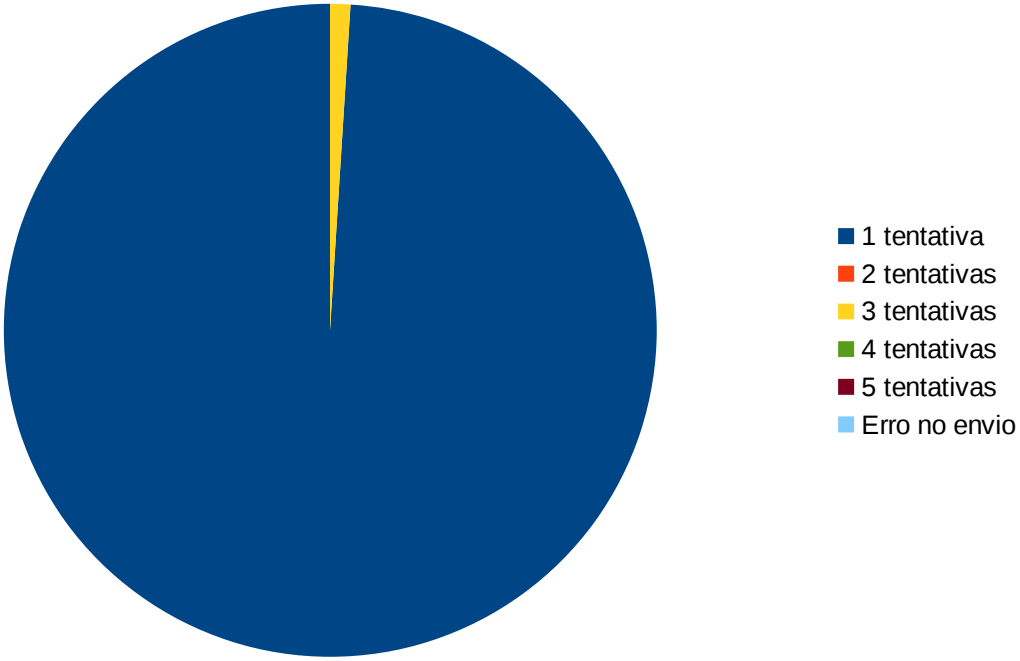
A análise gráfica entre a quantidade de pacotes e a quantidade de tentativas para estabelecer a comunicação se encontra na Figura 4.5, Figura 4.6 e Figura 4.7 com distâncias de 0 Km, 8,26 Km e 13,27 Km respectivamente.

Figura 4.5 - Desempenho do CSMA/CA para a distância de 0Km



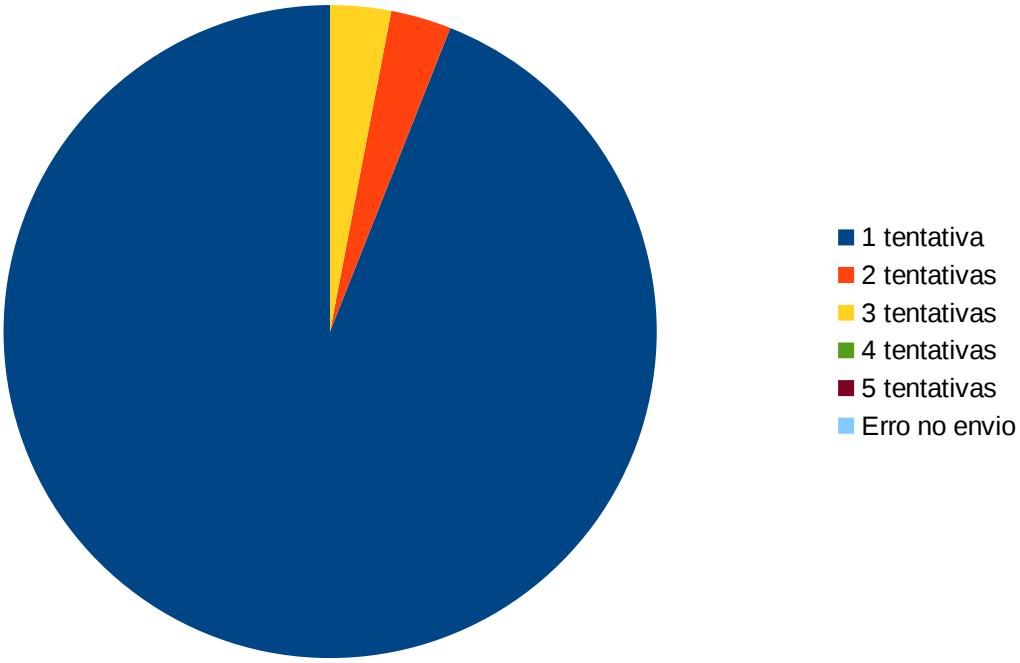
A Figura 4.5 é referente ao teste de controle, com os dois pontos da rede próximos tem-se condições ideais de potência de recepção e consequentemente uma ótima relação sinal/ruído (SNR). Para esse teste de controle, a potência de recepção média foi de -3,35 dBm, e 100% dos pacotes foram transmitidos uma única vez.

Figura 4.6 - Desempenho do CSMA/CA para a distância de 8.26Km



Na Figura 4.6 o a potência de recepção média (RSSI) foi de -108,07 dBm e 99% dos pacotes foram enviados uma única vez, enquanto 1% enviados três vezes.

Figura 4.7 - Desempenho do CSMA/CA para a distância de 13,27Km



Na Figura 4.7 a porcentagem de envios foram de 94%, 3% e 3% para 1 tentativa, 2 tentativas e 3 tentativas respectivamente.

Comparando os resultados dos testes com o teste de controle, observa-se que o algoritmo CSMA/CA em conjunto com a camada física do LoRa funcionam de forma satisfatória em grandes distâncias.

4.3 Implementação do AODV

Seguindo algumas otimizações do AODV para sistemas embarcados feitas por [16] apelidado como AODVjr, devido a simplificação e redução de alguns processos e variáveis. No trabalho em questão, foi implementado apenas os mecanismos chave para criação de rotas como o envio e o processamento dos pacotes de RREQ, RREP e UserMessege [12].

4.3.1 Adequação aos requisitos computacionais

Uma das dificuldades de se implementar um protocolo de roteamento em uma plataforma com memória dinâmica limitada é as tabelas. O número de elementos na rede geralmente depende da quantidade de memória e outros recursos computacionais.[11] As tabelas de roteamento exigem muito da memória já que, para cada elemento na tabela, um conjunto de variáveis são alocadas. Com as otimizações feitas por [16], cada elemento na tabela ocupa 19 Bytes. Conforme pode ser observado na Tabela 4.4. Um dos artifícios utilizados foi retirar a lista de precursores de cada rota.

Tabela 4.4 - Comparativo entre o tamanho de cada rota na memória.

Variável	**Tamanho AODVjr**	**AODV implementado**
Destination address	4Byte	1Bytes
Destination sequence number	4B	4Bytes
Destination sequence number flag	1B	1bit
Next hop address	4B	1Bytes
Hop count	1B	1Bytes
Route lifetime	4B	-
Route state	1B	1Byte
Total size	19B	9Bytes

No caso da implementação deste trabalho, não há necessidade de um número tão grande de dispositivos, então foi adotado apenas 8 bits (1 Byte) para identificar cada dispositivo. E como alguns mecanismos não foram implementados, a variável "*Route lifetime*", usada para definir uma validade da rota, foi removida da tabela. Restando apenas 9Bytes para cada rota.

Considerando o a memória dinâmica do ATmega328P e todo o programa já escrito, sobram 800 Bytes para variáveis globais e dividindo por 9, tem-se aproximadamente uma tabela de roteamento de no máximo 88 rotas, o suficiente para grande parte das aplicações em rede de sensores.

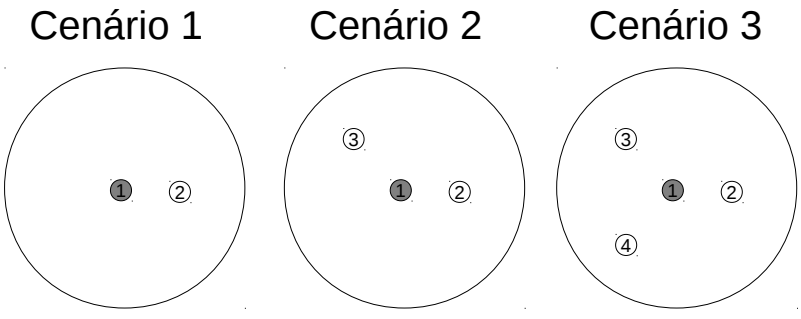
4.3.2 Resultados

Todos os experimentos são realizados por meio de *scripts* interpretados pelo dispositivo conectado a rede externa que age como concentrador de dados, no caso a plataforma raspberry pi. Todo os dispositivos da rede mantêm um registro, e por meio desse registro obtêm-se os dados brutos.

4.3.2.1 Experimento AODV #1

O primeiro experimento se consiste em todos os dispositivos em um mesmo raio de comunicação, situação ilustrada no Figura 4.8. Esse experimento tem como objetivo analisar o funcionamento do descarte de pacote RREQ duplicados, formação de rotas e avaliar se o tempo de descoberta tem uma forte ligação com o número de dispositivos em um mesmo raio de alcance. Já que com múltiplos dispositivos, tem-se múltiplas requisições de rota.

Figura 4.8 - Ilustração do experimento 1



Fonte: Autoria própria

O resultado desse teste é majoritariamente interpretativo, pois será analisado com mais enfase o registro de informações geradas pelas camadas CSMA/CA e AODV. O *script* e resultado pode ser encontrado no APÊNDICE B – EXPERIMENTO #1 DO AODV IMPLEMENTADO deste documento. O teste foi executando três vezes para cada cenário e com o resultado desse conjunto de três testes se obteve uma média que se encontra na Tabela 4.5.

Tabela 4.5 - Média de tempo para cada cenário.

cenário	**Media de tempo (s)**
1	1,5801
2	1,4488
3	1,5245

Analisando os resultados da Tabela 4.5 conclui-se que o número de dispositivos em um mesmo raio não influencia significativamente no tempo para a descoberta de rota, pelo menos para um pequeno número de dispositivos.

Nos primeiros registros do *log* observou-se o dispositivo fonte, dono do registro, enviando uma requisição de rota RREQ e depois descartando os *broadcast* duplicados enviado por outros dispositivos da rede, a quantidade de requisições descartadas foi proporcional ao número de dispositivos, na Tabela 4.6 pode-se ver essa relação. Então conclui-se que o mecanismo para descartar os *broadcasts* duplicados funcionou corretamente, recebeu, identificou e descartou todas as requisições duplicadas.

Tabela 4.6 - Relação de pacotes descartados e quantidades de dispositivos na rede.

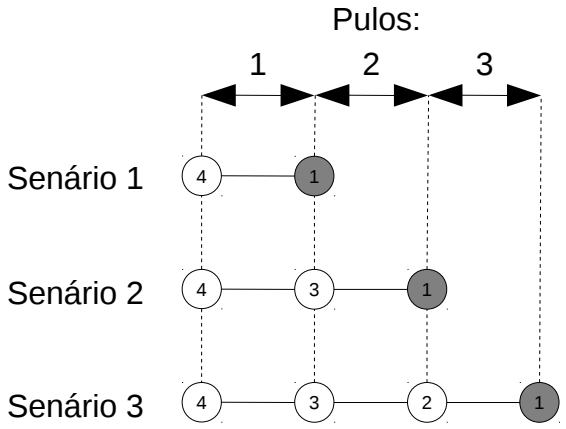
cenário	**Quantidade de dispositivos**	**Requisições descartadas**
1	2	1
2	3	2
3	4	3

4.3.2.2 Experimento AODV #2

Por meio do experimento 2 pode-se analisar vários cenários e tirar varias conclusões. Para o segundo experimento três cenários foram criados: Um utilizando apenas dois

dispositivos, três e quatro. Todos no raio de alcance apenas do vizinho, o experimento pode ser visualizado de forma intuitiva na Figura 4.9. Para cada cenário foi analisado o tempo para estabelecer uma rota nova, limpando a memória de todos os dispositivos da rede a cada novo cenário.

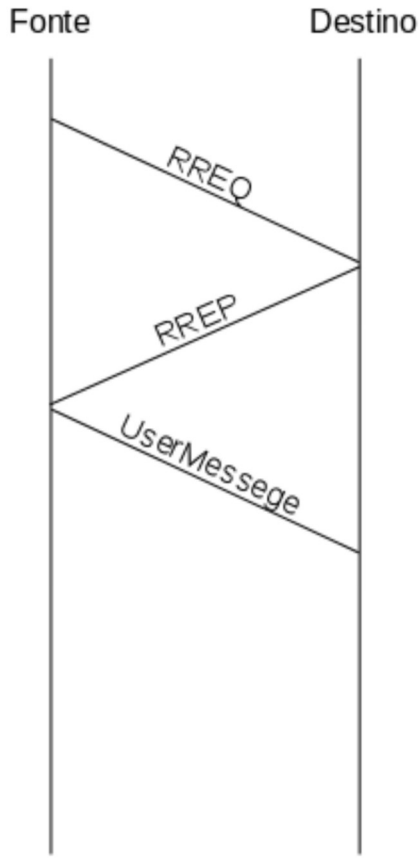
Figura 4.9 - Segundo Experimento AODV implementado



Fonte: Autoria própria

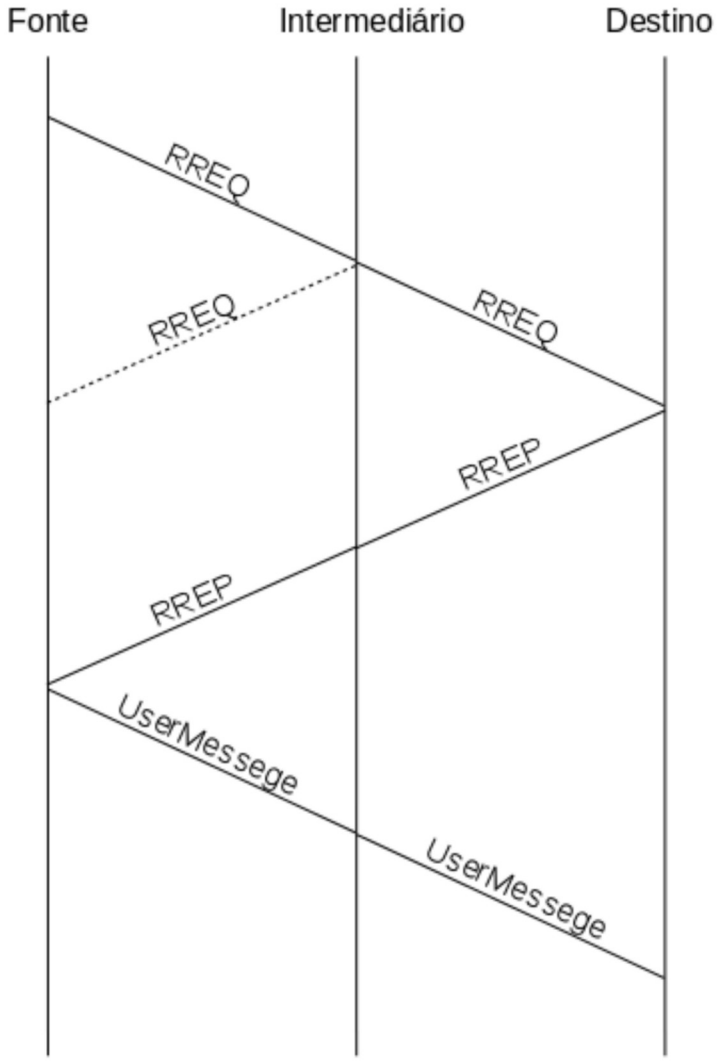
Com esse experimento pode-se analisar o tempo de formação de rota de acordo com o diâmetro da rede, o *Path discovery time*, um parâmetro importante para estimar o tempo máximo pra se enviar um novo *request* ou cancelar o processo do *path discovery*. No cenário 2 e 3 ilustrado no Figura 4.9 tem-se explicitamente o problema do terminal oculto, no qual dispositivos das extremidades não conseguem fazer o *carrie sense* entre si, aumentando as chances de causar interferência no dispositivo intermediário, discutido na seção 2.2.1. Parte do resultado obtido com esse experimento se encontra na Figura 4.13 e o *script* com os registros completo se encontra no APÊNDICE C – SCRIPT E REGISTROS DO EXPERIMENTO #2 DO AODV IMPLEMENTADO. Os eventos coletado pelos registros foram sintetizados em diagramas de sequencia contidos nas Figuras 4.10, 4.11 e 4.12.

Figura 4.10 - Diagrama de sequencia do cenário 1



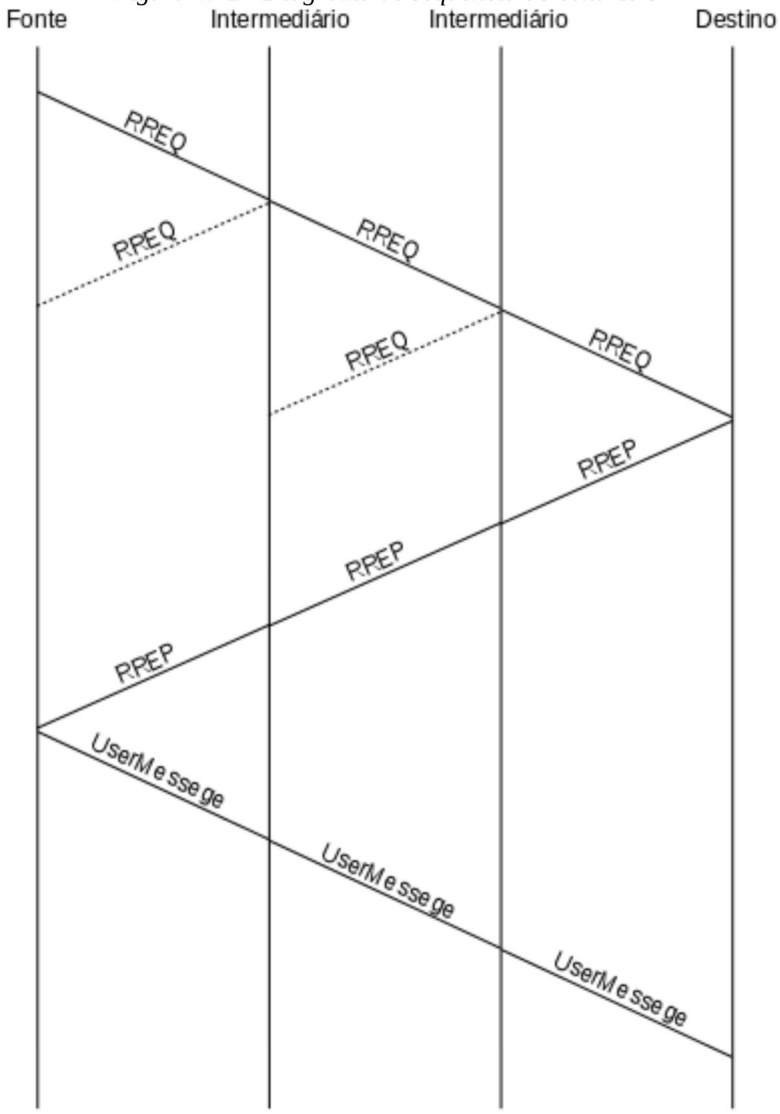
A Figura 4.10 ilustra o diagrama de sequencia do cenário 1, ou seja, apenas dois dispositivos na rede. O resultado é bem simples, a fonte manda uma requisição (RREQ), o destino recebe essa requisição e a processa. Como ele é o destinatário, não retransmite a requisição, manda um pacote de resposta (RREP) no lugar. Quando a resposta chega até a fonte, a rota completa é criada e a fonte manda a mensagem de usuário contendo o *payload*.

Figura 4.11 - Diagrama de sequencia do cenário 2



No segundo cenário, Figura 4.11, observa-se a adição de um dispositivo intermediário, assim como esperado, ele retransmite a requisição (em broadcast), a fonte descarta o *broadcast* duplicado. O destino recebe a requisição e, assim como no cenário 1, manda uma resposta e o dispositivo intermediário passa a atuar como intermediador da transmissão.

Figura 4.12 - Diagrama de sequencia do cenário 3

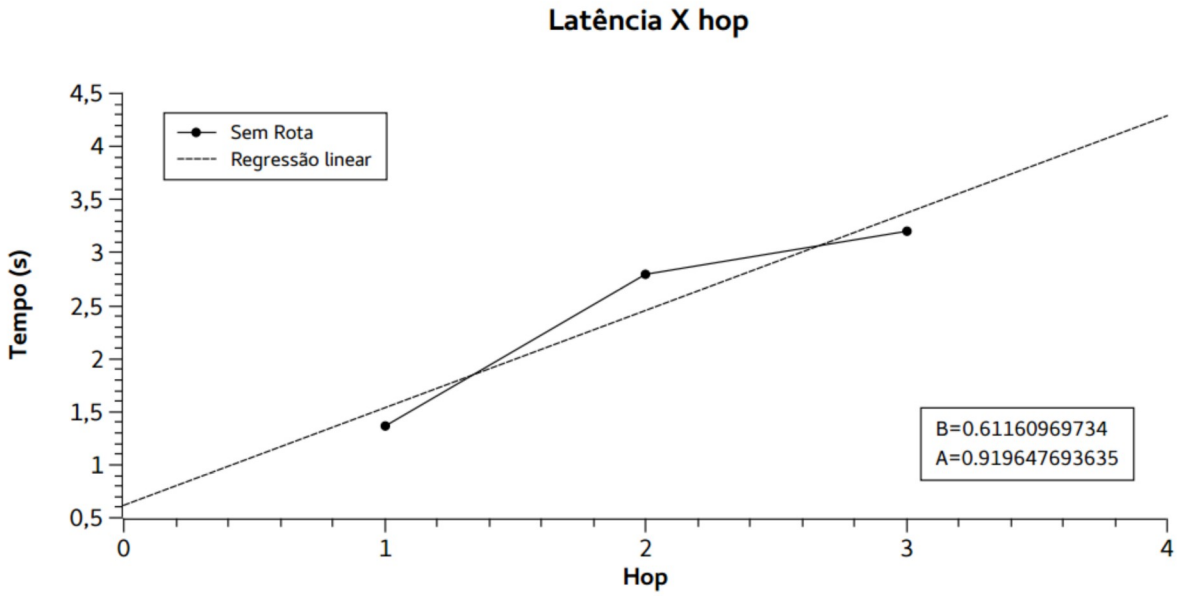


Com dois dispositivos intermediários, ilustrado na Figura 4.12, a mecânica é a mesma. Os dispositivos intermediários retransmitem a requisição em *broadcast* e descarta requisições duplicadas ate chegar ao destino.

O tempo para estabelecer uma rota completa nestes cenário foi incluído no gráfico da Figura 4.13 e estabelece a relação entre o tempo para que uma nova rota seja formada e a contagem de pulos. Por meio de uma regressão linear pode-se estimar o tempo de descoberta

de rota (ou *path discovery time)* para qualquer número de pulos *usando* os coeficientes coeficientes linear e angular. Com esses dados pode-se obter o tempo transversal da rede (*Network transversal time*), usado nos mecanismos do AODV.

Figura 4.13 - Gráfico com os dados obtidos no experimento



A tabela Tabela 4.7 confirma a disposição dos dispositivos na Figura 4.9, podendo deduzir que a tabela com a rota foi criada corretamente.

Tabela 4.7 - Tabela de rota criada em cada experimento.

cenário:	**ID destino**	**Próximo ID**	**Contagem de pulos**
1	4	4	1
2	4	3	2
3	4	2	3

5. Conclusão

Assim como em [16], esse projeto prova que é possível adaptar um protocolo de roteamento em sistemas computacionais simples como a de um microcontrolador de 8bis sem a necessidade de um sistema operacional. Sustentada por um pilha simples de protocolos, a combinação das características da morulação LoRa, sua alta eficiência energética e longo alcance com a flexibilidade de uma topologia em malha proporcionado pelo protocolo de roteamento baseado no AODV, pode se tornar uma nova solução no contexto das *sensor networks* e *IoT.*

No atual estagio o projeto, o protocolo implementado reduziu substancialmente o uso do memória se comparado com o protocolo AODV original e o implementado por [16]. Abrindo mão do tamanho da rede, limitada a 254 dispositivos, e simplificando algum mecanismos.

Um dos efeitos negativos observado que deve ser comentado nesse estudo é a latência para estabelecer uma rota comparando com o AODV original. A latência obtida nas simulações por [13] é, em média, de 206ms para uma média de 3.9 pulos. Comparado com o protocolo implementado nesse trabalho, pode-se usar a regressão linear obtida na seção 4.3.2.2 para encontrar a latência para a mesma média de pulos (3.9), chegando no valor de 4.198ms, aproximadamente 20 vezes mais de latência. Tornado o projeto inadequado para aplicações que requeiram baixa latência.

6. Planos futuros

A próxima etapa do projeto será dar continuidade aos testes do protocolo AODV que foi implementado e implementar os outros mecanismos de manutenção de rotas com o objetivo de deixar o protocolo plenamente funcional. Com melhorias no algoritmo aliado com a escolha de um *hardware* adequado, o resultado poderá ser interessante para ser aplicado em algum produto ou solução.

6.1.1 Melhorias no algoritmo

Para a melhoria do sistema como um todo, a base da pilha de protocolos deve ser estável, no entanto ainda há margem para a melhoria no algoritmo CSMA/CA implementado, como por exemplo a redução do espaço entre *frames* que causaria um melhor aproveitamento de banda e um aumento no *throughput*.

No AODV, a lista de melhorias é bem maior, já que não foi feito teste para todos os cenários. Uma melhoria pendente é a aplicação de filas no recebimento de mensagens, para que quando uma requisição de mensagem for enviada para o algoritmo, o mesmo já enfileira a mensagem e se não tiver uma rota válida, continua enfileirando as requisições ate que uma rota valida seja formada.

6.1.2 Possíveis aplicações do projeto

Uma aplicação bem conveniente nesse tipo de tecnologia é na área agrária pois com extensos campos e a necessidade de uma rede de sensores que abrandam toda a área, esse projeto pode ser uma alternativa barata se comparada com a aplicação de tecnologia de telefonia móvel para o mesmo cenário.

Outra aplicação é na rede de sensores para automação industrial. Nas indústrias geralmente contem toda uma rede estruturada de sensores, atuadores e controladores lógicos programáveis (PLC’s). Há no mercado equipamentos para a integração de sensores, localizados em áreas mais remotas, com a rede estruturada da indústria. A aplicação da tecnologia LoRa nesse ambiente seria uma nova abordagem para o problema.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, Marimuthu Palaniswamia, Internet of Things (IoT): A Vision, Architectural Elements, andFuture Directions, 2013

[2] , Sobre a Sigfox,Disponível em: <http://makers.sigfox.com/about/>,Acesso em: 10 nov. 2017

[3] Giedre Dregvaite, Robertas Damasevicius, Information and Software Technologies: 22nd International Conference, ICIST 2016, Druskininkai, Lithuania, October 13-15, 2016, Proceedings, 2016

[4] , Sigfox - The Global Communications Service Provider for the Internet of Things (IoT),Disponível em: <https://www.sigfox.com/en>,Acesso em: 29 nov. 2017

[5] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melià-Seguí, and Thomas Watteyne, Understanding the Limits of LoRaWAN, 2017

[6] , The alliance,https://www.lora-alliance.org/the-alliance,

[7] , LoRa | LoRa Wireless Technology | Semtech | Semtech,Disponível em: <http://www.semtech.com/wireless-rf/internet-of-things/what-is-lora/>,Acesso em 29 nov. 2017

[8] , LoRa Modulation Basics, 2015

[9] , Datasheet SX1276/77/78/79,

[10] Andrew S. Tanenbaum, REDE DE COMPUTADORES,

[11] Arun Mukhija, Reactive Routing Protocol for Mobile Ad-Hoc Networks,

[12] Mehran Abolhasan, Tadeusz Wysocki, Eryk Dutkiewicz, A review of routing protocols for mobile ad hoc networks, 2004

[13] Charles E. Perkins, Elizabeth M. Royer, Ad-hoc On-Demand Distance Vector Routing,

[14] , Part 11: Wireless LAN Medium AccessControl (MAC) and Physical Layer(PHY) speciÞcations, 1997

[15] Tetsuya Shigeyasu, Hiroshi Matsuno, Shozo Komaki, A New Broadcast Method to Prevent Colisions over Biased Terminal Arrangement, 2008

[16] Radek Salom, Petr Kaspar, Tomas Blecha, Jaroslav Freisleben, Jan Bartovsky, Petr Krist, Ales Hamacek, Implementation of AODV routing protocol in sensor wireless networks, 2012

[17] Radek Salom, Petr Kaspar, Tomas Blecha, Jaroslav Freisleben, Jan Bartovsky, Petr Krist, Ales Hamacek, Implementation of AODV Routing Protocol in Sensor Wireless Networks, 2012

[18] Ewa Niewiadomska-Szynkiewicz, Filip Nabrdalik, Secure Low Energy AODV Protocol for WirelessSensor Networks, 2017

[19] Tetsuya Shigeyasu, Hiroshi Matsuno, Shozo Komaki, A New Broadcast Method to Prevent Packet Collisions over Biased Terminal, 2008

APÊNDICE A – *SCRIPT* E REGISTROS DO TESTE CSMA/CA #1

Script:

```
1  import AODV
2  import time
3
4  def recebi(self, payload):
5     print('menssagem recebida: '+payload)
6
7  teste = AODV.AODV()
8  tic = time.time()
9  teste.send(2,"mensagem para 2")
10  toc = time.time()
11  tempoNovaRota = toc - tic
12  tic = time.time()
13  teste.send(2,"outra mensagem para 2")
14  toc = time.time()
15  tempoRotaEstabelecida = toc - tic
16  time.sleep(5)
17  print 'tempo sem rota pre estabelecida: '+str(tempoNovaRota)+'s'
18  print 'tempo com rota pre estabelecida: '+str(tempoRotaEstabelecida)+'s'
```

Registros:

2018-04-08 01:59:23,917 - CSMA - Enviado broadcast

2018-04-08 01:59:23,917 - AODV - RREQ enviado

2018-04-08 01:59:24,076 - AODV - Ignorando RREQ duplicado: id=1 broadcastID=1

2018-04-08 01:59:24,077 - CSMA - Broadcast recebido

2018-04-08 01:59:24,528 - CSMA - Enviado CTS para 2

2018-04-08 01:59:24,530 - CSMA - RTS recebido de 2

2018-04-08 01:59:24,679 - CSMA - Enviado ACK para 2

2018-04-08 01:59:24,680 - AODV - RREP recebido de 2 para 1 originado de 2

2018-04-08 01:59:24,680 - AODV - Tabela criada: {'desSeqNumFLAG': 0, 'desADDR': 2, 'routeState': 1, 'nextHopADDR': 2, 'hopCount': 1, 'desSeqNum': 1L}

2018-04-08 01:59:24,681 - CSMA - DS recebido: ### ##

2018-04-08 01:59:26,384 - CSMA - Enviando RTS para 2

2018-04-08 01:59:26,519 - CSMA - CTS recebido

2018-04-08 01:59:26,622 - CSMA - Enviado DS para 2 : mensagem para 2

2018-04-08 01:59:26,692 - CSMA - ACK recebido

2018-04-08 01:59:26,694 - AODV - User messege enviada

2018-04-08 01:59:26,947 - CSMA - Enviando RTS para 2

2018-04-08 01:59:27,080 - CSMA - CTS recebido

2018-04-08 01:59:27,204 - CSMA - Enviado DS para 2 : outra mensagem para 2

2018-04-08 01:59:27,277 - CSMA - ACK recebido

2018-04-08 01:59:27,278 - AODV - User messege enviada

tempo sem rota pre estabelecida: 2.87970304489s

tempo com rota pre estabelecida: 0.58447098732s

APÊNDICE B – EXPERIMENTO #1 DO AODV IMPLEMENTADO

Figura 1 - Dois dispositivos

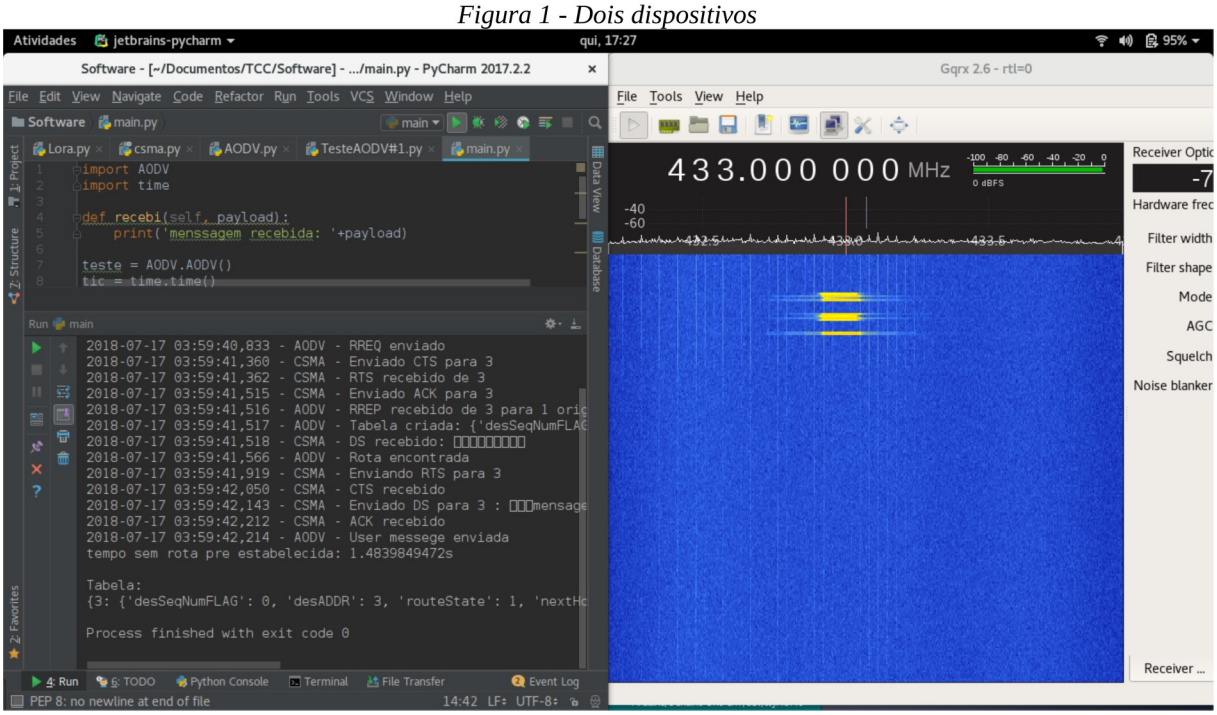


Figura 2 - Dois dispositivos

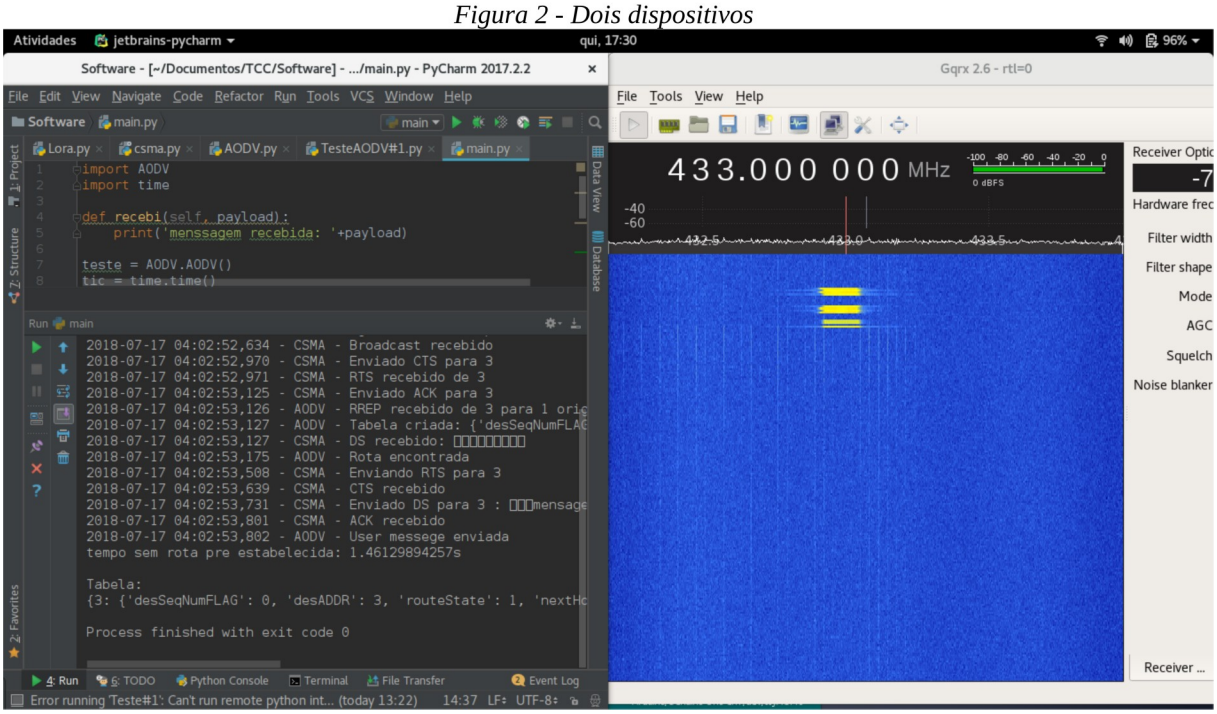
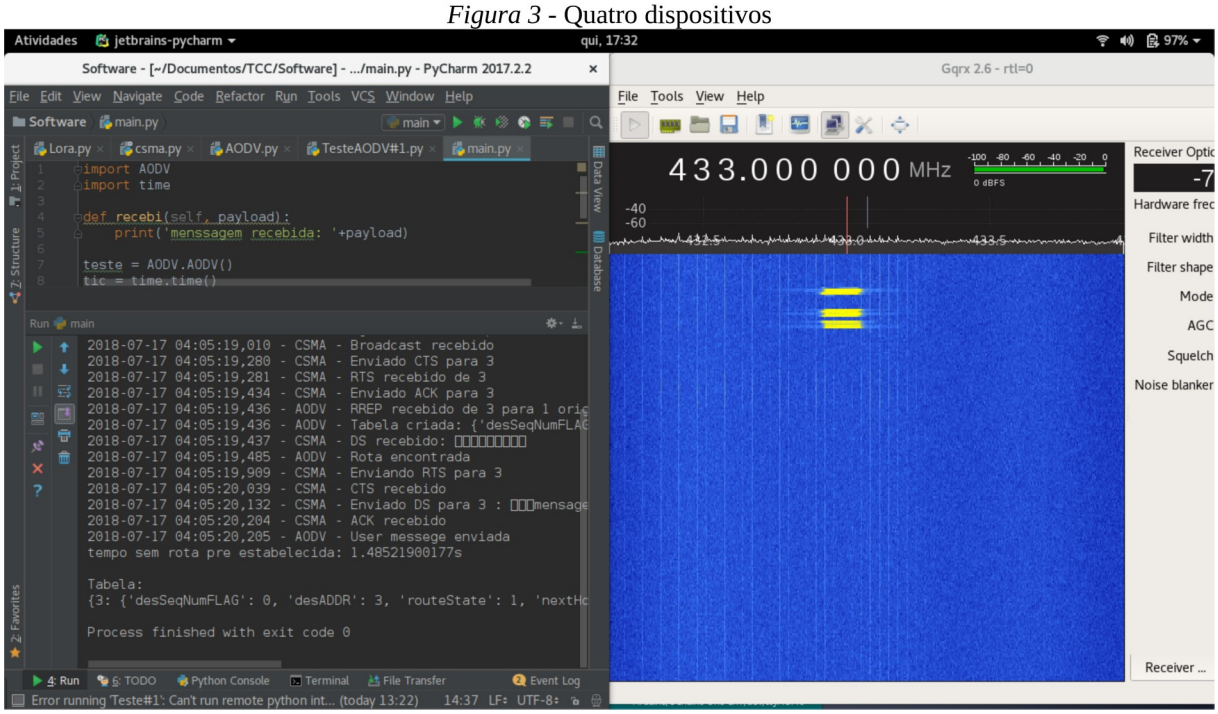


Figura 3 - Quatro dispositivos



APÊNDICE C – *SCRIPT* E REGISTROS DO EXPERIMENTO #2 DO AODV IMPLEMENTADO

Script:

```
import AODV
import time
def recebi(self, payload):
  print('menssagem recebida: '+payload)
teste = AODV.AODV()
tic = time.time()
teste.send(4,"mensagem")
toc = time.time()
tempoNovaRota = toc - tic
tic = time.time()
teste.send(4,"outra mensagem")
toc = time.time()
tempoRotaEstabelecida = toc - tic
time.sleep(5)
print 'tempo sem rota pre estabelecida: '+str(tempoNovaRota)
+'s'
print 'tempo com rota pre estabelecida:
'+str(tempoRotaEstabelecida)+'s'
print '\nTabela:\n'+str(teste.routeTable)
```

Registros:

cenário 1:

2018-07-17 03:42:55,264 - CSMA - Enviado broadcast
2018-07-17 03:42:55,265 - AODV - RREQ enviado
2018-07-17 03:42:55,694 - CSMA - Enviado CTS para 4
2018-07-17 03:42:55,695 - CSMA - RTS recebido de 4
2018-07-17 03:42:55,849 - CSMA - Enviado ACK para 4
2018-07-17 03:42:55,850 - AODV - RREP recebido de 4 para 1 originado de 4
2018-07-17 03:42:55,851 - AODV - Tabela criada: {'desSeqNumFLAG': 0, 'desADDR': 4, 'routeState': 1, 'nextHopADDR': 4, 'hopCount': 1, 'desSeqNum': 1L}
2018-07-17 03:42:55,851 - CSMA - DS recebido: ### ##
2018-07-17 03:42:55,899 - AODV - Rota encontrada
2018-07-17 03:42:56,222 - CSMA - Enviando RTS para 4
2018-07-17 03:42:56,356 - CSMA - CTS recebido
2018-07-17 03:42:56,449 - CSMA - Enviado DS para 4 : ###mensagem
2018-07-17 03:42:56,521 - CSMA - ACK recebido
2018-07-17 03:42:56,522 - AODV - User messege enviada
2018-07-17 03:42:56,523 - AODV - Rota encontrada
2018-07-17 03:42:57,008 - CSMA - Enviando RTS para 4

2018-07-17 03:42:57,143 - CSMA - CTS recebido
2018-07-17 03:42:57,246 - CSMA - Enviado DS para 4 : ###outra mensagem
2018-07-17 03:42:57,319 - CSMA - ACK recebido
2018-07-17 03:42:57,320 - AODV - User messege enviada
tempo sem rota pre estabelecida: 1.36044383049s
tempo com rota pre estabelecida: 0.798138856888s

Tabela:
{4: {'desSeqNumFLAG': 0, 'desADDR': 4, 'routeState': 1, 'nextHopADDR': 4, 'hopCount': 1, 'desSeqNum': 1L}}

cenário 2:

2018-07-17 03:31:46,490 - CSMA - Enviado broadcast
2018-07-17 03:31:46,491 - AODV - RREQ enviado
2018-07-17 03:31:46,665 - AODV - Ignorando RREQ duplicado: id=1 broadcastID=1
2018-07-17 03:31:46,667 - CSMA - Broadcast recebido
2018-07-17 03:31:47,348 - CSMA - CTS para 4 recebido, entrando em NAV: 0.74s
2018-07-17 03:31:47,492 - CSMA - ACK para 4 recebido
2018-07-17 03:31:48,168 - CSMA - Enviado CTS para 3
2018-07-17 03:31:48,169 - CSMA - RTS recebido de 3
2018-07-17 03:31:48,322 - CSMA - Enviado ACK para 3
2018-07-17 03:31:48,323 - AODV - RREP recebido de 3 para 1 originado de 4
2018-07-17 03:31:48,324 - AODV - Tabela criada: {'desSeqNumFLAG': 0, 'desADDR': 4, 'routeState': 1, 'nextHopADDR': 3, 'hopCount': 2, 'desSeqNum': 1L}
2018-07-17 03:31:48,325 - CSMA - DS recebido: ### ###
2018-07-17 03:31:48,373 - AODV - Rota encontrada
2018-07-17 03:31:48,879 - CSMA - Enviando RTS para 3
2018-07-17 03:31:49,013 - CSMA - CTS recebido
2018-07-17 03:31:49,106 - CSMA - Enviado DS para 3 : ###mensagem
2018-07-17 03:31:49,179 - CSMA - ACK recebido
2018-07-17 03:31:49,179 - AODV - User messege enviada
2018-07-17 03:31:49,181 - AODV - Rota encontrada
2018-07-17 03:31:49,571 - CSMA - RTS de 3 para 4 recebido , entrando em NAV: 0.74s
2018-07-17 03:31:49,682 - CSMA - Enviando RTS para 3
2018-07-17 03:31:51,246 - CSMA - Enviando RTS para 3
2018-07-17 03:31:51,381 - CSMA - CTS recebido
2018-07-17 03:31:51,484 - CSMA - Enviado DS para 3 : ###outra mensagem
2018-07-17 03:31:51,557 - CSMA - ACK recebido
2018-07-17 03:31:51,558 - AODV - User messege enviada
2018-07-17 03:31:52,021 - CSMA - RTS de 3 para 4 recebido , entrando em NAV: 0.74s
2018-07-17 03:31:52,185 - CSMA - DS para 4 recebido: ###outra mensage
tempo sem rota pre estabelecida: 2.79253220558s
tempo com rota pre estabelecida: 2.3780579567s

Tabela:
{4: {'desSeqNumFLAG': 0, 'desADDR': 4, 'routeState': 1, 'nextHopADDR': 3, 'hopCount': 2, 'desSeqNum': 1L}}

cenário 3:

2018-07-17 03:47:12,061 - CSMA - Enviado broadcast

2018-07-17 03:47:12,063 - AODV - RREQ enviado
2018-07-17 03:47:12,251 - AODV - Ignorando RREQ duplicado: id=1 broadcastID=1
2018-07-17 03:47:12,253 - CSMA - Broadcast recebido
2018-07-17 03:47:13,540 - CSMA - CTS para 3 recebido, entrando em NAV: 0.74s
2018-07-17 03:47:13,684 - CSMA - ACK para 3 recebido
2018-07-17 03:47:14,170 - CSMA - Enviado CTS para 2
2018-07-17 03:47:14,171 - CSMA - RTS recebido de 2
2018-07-17 03:47:14,324 - CSMA - Enviado ACK para 2
2018-07-17 03:47:14,326 - AODV - RREP recebido de 2 para 1 originado de 4
2018-07-17 03:47:14,327 - AODV - Tabela criada: {'desSeqNumFLAG': 0, 'desADDR': 4, 'routeState': 1, 'nextHopADDR': 2, 'hopCount': 3, 'desSeqNum': 1L}
2018-07-17 03:47:14,328 - CSMA - DS recebido: ### ###
2018-07-17 03:47:14,375 - AODV - Rota encontrada
2018-07-17 03:47:14,860 - CSMA - Enviando RTS para 2
2018-07-17 03:47:14,992 - CSMA - CTS recebido
2018-07-17 03:47:15,084 - CSMA - Enviado DS para 2 : ###mensagem
2018-07-17 03:47:15,157 - CSMA - ACK recebido
2018-07-17 03:47:15,158 - AODV - User messege enviada
2018-07-17 03:47:15,159 - AODV - Rota encontrada
2018-07-17 03:47:15,559 - CSMA - RTS de 2 para 3 recebido , entrando em NAV: 0.74s
2018-07-17 03:47:15,660 - CSMA - Enviando RTS para 2
2018-07-17 03:47:17,194 - CSMA - Enviando RTS para 2
2018-07-17 03:47:17,329 - CSMA - CTS recebido
2018-07-17 03:47:17,431 - CSMA - Enviado DS para 2 : ###outra mensagem
2018-07-17 03:47:17,505 - CSMA - ACK recebido
2018-07-17 03:47:17,505 - AODV - User messege enviada
2018-07-17 03:47:17,818 - CSMA - RTS de 2 para 3 recebido , entrando em NAV: 0.74s
2018-07-17 03:47:18,025 - CSMA - DS para 3 recebido: ###outra mensagemtar_pronto_mas..
tempo sem rota pre estabelecida: 3.19973921776s
tempo com rota pre estabelecida: 2.34751796722s

Tabela:
{4: {'desSeqNumFLAG': 0, 'desADDR': 4, 'routeState': 1, 'nextHopADDR': 2, 'hopCount': 3, 'desSeqNum': 1L}}