UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

TRABALHO DE CONCLUSÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Projeto de uma Rede de Comunicação sem Fio Baseada no Transceptor nRF24L01+ Voltada para Sistemas de Automação Predial

Autor: Mateus Streit Giaretta

Orientador: Carlos Eduardo Pereira

Porto Alegre, julho de 14

Sumário

Resumo

Este trabalho propõe o desenvolvimento de uma estrutura de rede sem fio, composta por componentes de baixo custo, para aplicações de automação. As estratégias desenvolvidas têm como objetivo otimizar as características da rede para atendimento dos requisitos das aplicações alvo, em especial para a área de automação residencial. Elas se baseiam no uso e na expansão das funcionalidades do rádio transceptor nRF24L01+, por meio da implementação de funções em um microcontrolador externo, definido como sendo um Arduino Pro-mini.

Para evitar a ocorrência de colisão entre mensagens é determinada uma divisão hierárquica entre os nós da rede, com um mestre centralizando a comunicação com os demais dispositivos, tidos como escravos. Com o intuito de expandir o número de dispositivos e o alcance da rede, é proposta uma estrutura de rede dividida em clusters e o uso de um mecanismo de roteamento. Adicionalmente, são definidas as interfaces de troca de dados entre os dispositivos, por meio de comandos de leitura e escrita de variáveis compartilhadas. Ao final do trabalho são mostrados resultados de ensaios, que capacitam o sistema desenvolvido ao objetivo proposto de atuar em uma rede de automação residencial.

Palavras Chave: transceptor nRF24L01+, hierarquia mestre-escravo, divisão em clusters, automação residencial

Abstract

This project proposes the development of a framework for a wireless network, composed of low-cost components, for automation applications. The developed approaches are intended to optimize the network characteristics for home automation systems. The radio transceiver nRF24L01+ and an Arduino Pro-Mini were selected as target hardware and the required functions were developed in C.

To avoid messages collision, a hierarchical network strategy was developed, following a master-slave pattern. In order to expand the number and range of the network devices, a cluster-based network division and a routing mechanism were proposed. In addition, to allow data exchange between devices a shared memory interface was defined. The developed ideas were experimentally validated and the obtained results are shown in the last part of this report.

Keywords: nRF24L01 + transceiver, master-slave hierarchy, clustering division, home automation

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

SPI – Interface Periférica Serial

OSI - Open Systems Interconnection

ISO – International Organization for Standardization

PWM – Modulação por Largura de Pulso

GFSK - Gaussian Frequency Shift Keying

ISM – Industrial Scientific and Medical

TX – Transmissor

RX – Receptor

ACK – Pacote de Reconhecimento de Mensagem

CRC – Verificação Cíclica de Redundância

FLAGS – Bits de Configuração

RETR – Bit de Configuração de Retransmissão

MAP – Bit de Configuração de Mapeamento

RESP – Bit de Configuração de Resposta

DATA - Bit de Configuração de Troca de Dados

BOOL – Variável Booleana

INT – Variável Inteira

UINT – Variável Inteira sem Sinal

FLOAT – Variável de Ponto Flutuante

LEIT – Leitura

ESCR - Escrita

1 Introdução

Com uma presença já bem difundida em meios industriais, a automação tem sido uma área que tem recebido uma atenção cada vez maior nas aplicações residenciais. Segundo descrito em (Dias e Pizzolato, 2004) a automação predial e residencial, também chamada de domótica, é a integração dos serviços e tecnologias, aplicados a residências, flats, apartamentos, casas e pequenas construções, com o propósito de automatizá-los e obter uma melhora em relação à segurança e proteção, conforto, comunicação e gerenciamento técnico. Em seus diversos níveis de aplicação ela pode proporcionar o controle automático e remoto de uma grande variedade de dispositivos. Apesar de usarem técnicas e tecnologias semelhantes, a automação industrial necessita de um padrão mais elevado de responsabilidade e confiabilidade dos seus componentes e da estrutura de rede em si, quando comparado com a residencial.

Ainda de acordo com descrição contida em (Dias e Pizzolato, 2004), dentre os protocolos de comunicação sem fio utilizados na área da automação residencial e predial para interconexão de dispositivos sensores, atuadores e controladores, destacam-se os protocolos ZigBee e Bluetooth. Apesar de serem tecnologias consolidadas, os seus custos de implantação ainda limitam a sua aplicação em uma maior escala. Nesse contexto, este projeto tem como objetivo propor uma solução alternativa de baixo custo para o mesmo fim, com o uso do transceptor nrF24L01+ (Nordic, 2007). Este dispositivo funciona como um rádio transmissor e receptor que opera na banda de frequência livre em nível mundial ISM de 2,4GHz. Neste componente são implementadas as funções do protocolo de comunicação não padrão Enhanced ShockBurst. Este protocolo aplica funções básicas de comunicação para pequenas redes.

Para fins de comparação, em pesquisa realizada no mês de maio de 2014 em um dos maiores sites de revenda online do mundo (Aliexpress, 2014), o custo de um lote de 10 peças com frete internacional de um módulo nRF24L01+ tinha um custo aproximadamente 4,5 vezes menor do que o módulo mais barato do protocolo Bluetooth e cerca de 7 vezes menor do que um módulo ZigBee. Ambos os dispositivos citados não têm autonomia para trabalharem de maneira isolada, necessitando de um microcontrolador externo que comande suas ações. Tal fato dá o respaldo necessário para este trabalho, que pretende expandir as funcionalidades do transceptor nRF24L01+ através do uso de funções implementadas em um microcontrolador externo. Funções estas, otimizadas para o uso em um ambiente de automação residencial.

O microcontrolador externo é responsável por configurar e comandar as ações do rádio, o que é feito através de comandos pré-definidos enviados através de uma Interface Periférica Serial (conhecida pela sigla em inglês SPI). Neste trabalho foi escolhido o uso de um microcontrolador da plataforma Arduino. Escolha feita devido ao seu uso abrangente e pela facilidade proporcionada na criação de protótipos.

1.1 Objetivo Geral

Este projeto propõe o desenvolvimento de uma estrutura de rede de comunicação sem fio com base no radio transceptor nRF24L01+. Com o objetivo de investigar um conjunto de soluções que melhor se adapte aos pontos fortes e limitações dos componentes utilizados, ampliando suas funcionalidades por meio de funções implementadas em microcontrolador externo, de modo a melhor atender as necessidades de uma rede complexa de automação predial.

1.2 Objetivos Específicos

- O projeto deve priorizar o uso de hardware de baixo custo, de modo a proporcionar uma alternativa tecnológica com um padrão de custo benefício maior ao que é usado em aplicações semelhantes.

- Ampliar as possibilidades de uso do transceptor nRF24L01+ de modo a se adaptar às necessidades de uma rede de automação residencial.

- Desenvolver novas topologias de rede, proporcionando mecanismos capazes de expandir e garantir de maneira plena a cobertura da rede nos diversos espaços de um ambiente predial.

- Rastrear possíveis causas comuns de erros na comunicação e desenvolver estratégias para evitar as suas ocorrências.

- Analisar a capacidade do sistema de atender aos requisitos de uma comunicação de tempo determinístico.

- Realizar ensaios capazes de caracterizar o desempenho de diferentes funções e estruturas da rede desenvolvidas.

2 Revisão Bibliográfica

2.1 Análise do Desempenho do Transceptor nRF24L01+ em um Cenário com Múltiplas Transmissões

Um estudo proposto em (Christ et al., 2011) analisa exaustivamente a eficiência de uma rede de comunicação constituída somente por rádio transmissores nRF24L01+. Neste estudo foram realizados ao todo 500 ensaios com diferentes configurações de rede, onde componentes igualmente distribuídos transmitiam 50000 pacotes de dados em cada experimento. Em um primeiro teste foi colocado somente um receptor entre 14 transmissores, porém com somente um transmissor operando por vez. Ao longo dos ensaios variou-se o tamanho e taxa de envio da mensagem. Neste teste foi detectado um pequeno aumento na taxa de perda de pacotes, à medida que pacotes maiores eram enviados. Contudo a taxa média de perda de transmissão ficou na casa dos 0,5%.

Nos demais testes, adicionou-se um número maior de componentes transmitindo simultaneamente. Do mesmo modo, o tamanho do pacote de dados foi alternado desde 15 até 35 bytes. A frequência com que as mensagens eram transmitidas também foram alteradas no intervalor de 25 a 250 mensagens por segundo. Os resultados obtidos indicaram um aumento expressivo das perdas na transmissão, sendo que as perdas aumentam tanto com um aumento do tamanho dos pacotes, bem como com aumento da frequência de envio das mensagens e do número de componentes na rede. Uma versão gráfica dos resultados obtidos nestes experimentos pode ser vista na Figura 1.

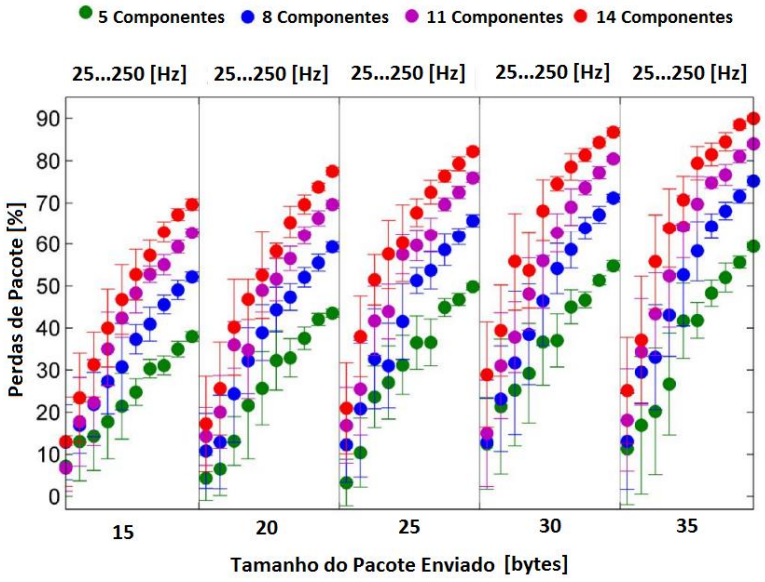


Figura 1 – Perda de Pacotes em Diferentes Cenários de Rede - Traduzido pelo autor de figura em (Christ et al., 2011)

Este estudo mostra claramente a ineficiência do sistema, da maneira como ele é originalmente proposto, para redes com um número maior de componentes e maiores taxas de transmissão. Portanto, mostra-se necessário o uso de técnicas mais avançadas na construção da rede a ser proposta, bem como estratégias de contingência para as perdas de pacotes, de forma a tornar a rede mais robusta.

2.2 Rede de Sensores Dividida em Clusters

O projeto de uma rede de sensores, com o objetivo de localizar vagas em um estacionamento é descrito em (Urdiain et al., 2012), usando como base rádios nRF24L01+ acoplados a microcontroladores da plataforma Arduino. Neste estudo propõem-se que uma rede complexa seja dividida em clusters de estruturas mais simples, onde cada cluster se comunica em uma frequência distinta. Para isso, cada cluster possui um nó mestre que coordena a comunicação interna de demais nós e envia os dados coletados para um nó central. Um esboço da configuração proposta é mostrado na Figura 2.

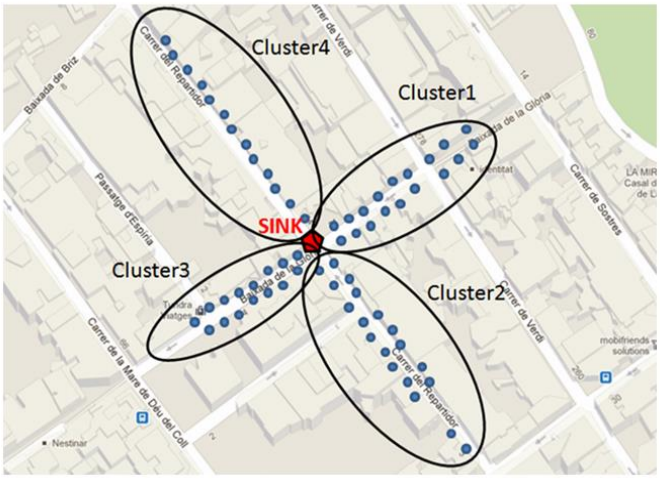


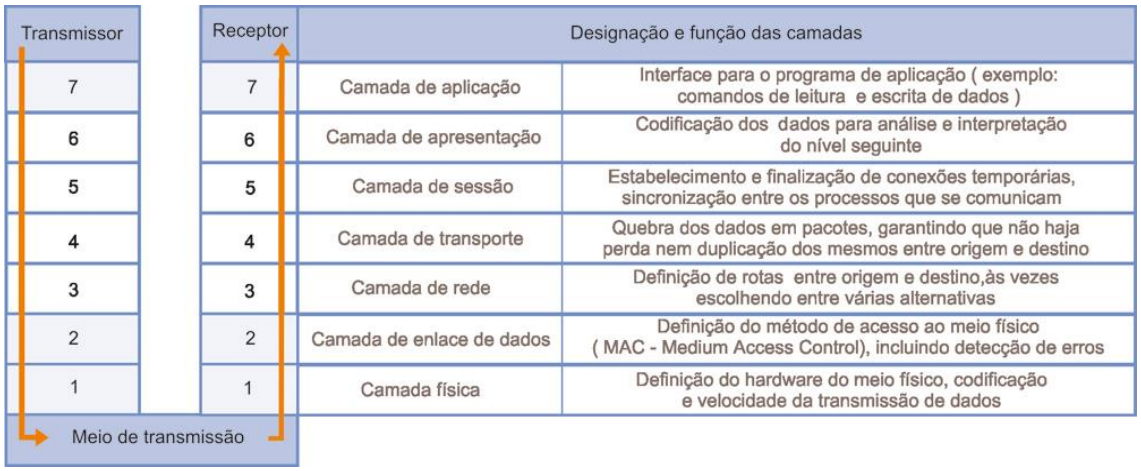
Figura 2 - Divisão em clusters - Fonte: (Urdiain *et al.*, 2012)

O desenvolvimento da rede proposta em (Urdiain et al., 2012) é baseado na biblioteca RF24 (Coliz, 2012), a qual implementa a interface entre o microcontrolador e o rádio transceptor por meio de funções definidas, sem a necessidade de acessar individualmente os registradores de configuração do nRF24L01+. A partir dessa ideia, foram desenvolvidas, através do microcontrolador Arduino, funções mais complexas para implementação da estrutura em clusters. O artigo detalha ensaios para validação experimental da proposta: (i) um teste de 90 minutos com 10 nós de comunicação em um mesmo cluster. Neste caso uma taxa de perda de transmissão de 71 % foi obtida; ii) um segundo ensaio de mesma duração, porém tendo os 10 nós divididos em dois clusters com 5 componentes cada. A divisão em clusters melhorou significativamente o desempenho da comunicação, fazendo a taxa de perdas cair para 41%. Apesar da redução obtida, uma taxa muito alta de perda de pacotes continua presente na comunicação. Os autores relatam como uma possível causa a dificuldade de sincronização entre os dispositivos.

2.3 Introdução ao Modelo de Referência OSI

Praticamente todas as redes em uso atualmente são baseadas de alguma forma no padrão comumente conhecido como ¨modelo OSI¨ de ¨Open Systems Interconnection ¨. O modelo de referência OSI, como descrito em (Day e Zimmermann, 1983), foi desenvolvido pela International Organization for Standardization (ISO), uma federação mundial de organismos nacionais de normalização que representa cerca de 130 países. O núcleo deste padrão é um conjunto de sete camadas que definem as diferentes etapas que os dados devem percorrer para transitar a partir de um dispositivo para outro através de uma rede. Esta divisão em camadas auxilia o entendimento do sistema e das tarefas

dentro de uma rede. Na Figura 3 encontra-se uma descrição breve sobre as sete camadas do Modelo de Referência OSI.



Transmissor	Receptor	Designação e função das camadas	
7	7	Camada de aplicação	Interface para o programa de aplicação (exemplo: comandos de leitura e escrita de dados)
6	6	Camada de apresentação	Codificação dos dados para análise e interpretação do nível seguinte
5	5	Camada de sessão	Estabelecimento e finalização de conexões temporárias, sincronização entre os processos que se comunicam
4	4	Camada de transporte	Quebra dos dados em pacotes, garantindo que não haja perda nem duplicação dos mesmos entre origem e destino
3	3	Camada de rede	Definição de rotas entre origem e destino,às vezes escolhendo entre várias alternativas
2	2	Camada de enlace de dados	Definição do método de acesso ao meio físico (MAC - Medium Access Control), incluindo detecção de erros
1	1	Camada física	Definição do hardware do meio físico, codificação e velocidade da transmissão de dados
Meio de transmissão			

Figura 3 – Camadas do modelo de Referência OSI – Fonte: (Redes da Organização Profibus, 2004)

Este trabalho está estruturado em capítulo de forma a apresentar suas funcionalidades separadas de acordo com a divisão em camadas do modelo OSI. A estrutura de rede na qual o transceptor nRF24L01+ é originalmente projetado para operar inclui somente funções de camada física e de enlace. Neste trabalho será feita uma análise das características do meio físico em que este sistema opera, uma expansão das funcionalidades de sua camada de enlace e a criação de funções de camada de rede e aplicação.

3 Descrição da Camada Física do Sistema Proposto

Este capítulo apresenta as características de camada física do sistema proposto. Este é o nível de hardware real. Ele define as características dos dispositivos usados e do meio em que eles atuam.

3.1 Descrição dos Componentes

3.1.1 Módulo Integrado para o Componente nRF24L01+

Módulo usado como base para o desenvolvimento deste trabalho, consiste em um circuito integrado ativo nRF24L01+, produzido pela empresa Nordic Semiconductor (Nordic, 2007). Neste mesmo módulo também são encapsulados uma antena, conectores e demais componentes passivos, tudo isso em uma placa com pouco mais de 3cm de comprimento. A Figura 4 mostra à esquerda o diagrama elétrico deste circuito, e à direita o esquemático da placa englobando todos estes componentes.

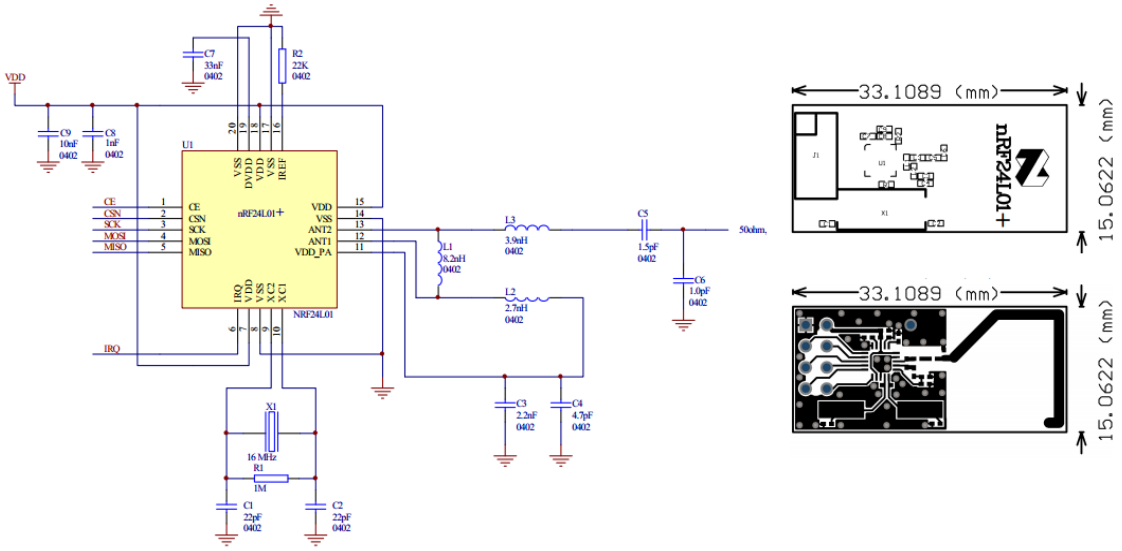


Figura 4 – Diagrama elétrico e placa do módulo nRF24L01+ - Fonte: (Nordic, 2007)

A interface do transceptor com o microcontrolador externo se dá por meio de comandos pré-definidos enviados através de um barramento. Associada a uma alimentação de até 3.3V, a interface do módulo consiste dos seguintes pinos:

- Pino CE (utilizado para ativar o chip)
- Pinos CSN, SCK, MOSI e MISO (pinos padrão da interface SPI)
- Pino IRQ (sinal que indica a ocorrência de três diferentes interrupções mascaráveis)

3.1.2 Microcontrolador Arduino Pro-Mini

Este microcontrolador é a versão mais minimalista entre todas as distribuições da plataforma Arduino. O modelo usado neste projeto, opera com um processador ATmega328, o mesmo usado pela versão mais conhecida da plataforma, o Arduino Uno. A placa possui 14 pinos digitais de entrada e saída (dos quais 6 podem ser usados como

saídas PWM), 8 entradas analógicas e furos para montagem de pinos ou soldagem direta de componentes. Em pesquisa de preço realizada (Aliexpress, 2014), o custo de uma versão não oficial do microcontrolador Arduino Pro-Mini em um lote de 10 peças foi comparada com um mesmo lote de Arduinos Uno. Nesta averiguação, uma mesma versão não oficial da plataforma Uno encontrava-se por um custo 4 vezes mais elevado. Esta diferença se deve ao fato de o modelo Uno ser otimizado para prototipagem, já o Pro-Mini é construído em uma placa compacta, com enfoque em aplicações diretas. Portanto, atendendo aos requisitos de redução de custo do projeto, neste trabalho serão usados somente microcontroladores do modelo Arduino Pro-Mini, com processador Atmega 328. Na Figura 5 é exibida uma representação deste microcontrolador.

Figura 5 – Placa Arduino Pro-Mini – Fonte: (Arduino, 2014)

3.1.3 Fontes de Alimentação

O conjunto Microcontrolador mais transceptor necessita de uma fonte de alimentação especial. O módulo nRF24L01+ deve ser alimentado com uma tensão entre 2.7V a 3.3V (Nordic, 2007), contudo seus pinos de comunicação com o microprocessador suportam até 5V, o tornando compatível com a saída do microcontrolador usado. Já a entrada de energia do Arduino deve ser entre a faixa de 5V a 12V. Associado a isso, foi observado em testes que o funcionamento do rádio está atrelado de maneira sensível a sua alimentação. Desníveis na tensão de alimentação, tanto do rádio, quanto do microcontrolador, impediram frequentemente o seu correto funcionamento.

Para tanto, foram desenvolvidas duas versões de fontes de alimentação. Uma delas usou uma fonte chaveada de 6V acoplada a um circuito regulador de tensão AMS1117 (Ams, 2014) de saída 3.3V. Para estabilizar a tensão foram usados um capacitor em paralelo à saída de 6V e outro à de 3.3 V. Os valores dos capacitores usados seguem o recomendado pelo fabricante. Este circuito não necessita de dispositivos extras de proteção, pois já possui um diodo interno com essa função. Na Figura 6 é mostrado o circuito regulador de tensão usado.

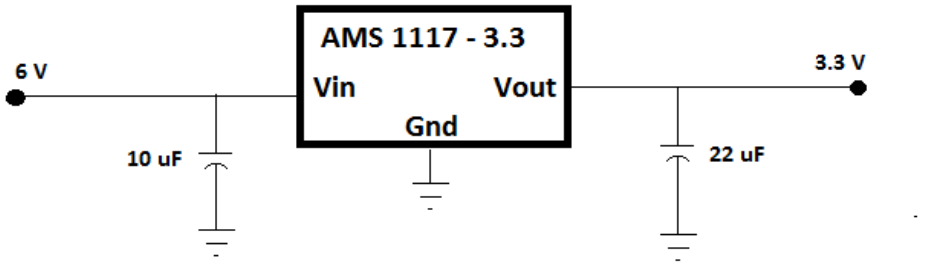


Figura 6 – Circuito Regulador de Tensão - Fonte: Autor

Outro conjunto mais simples de fontes foi desenvolvido com o uso de quatro baterias de 1.6V. Nesta configuração, duas baterias em série fornecem uma tensão de 3.2 V, enquanto as quatro ligadas em série fornecem 6.4 V. Esta ligação fornece um consumo desigual de energia das baterias e não é a melhor solução, porém devido ao baixo consumo energético dos componentes, após muitas horas de uso nenhum dos conjuntos de bateria montados chegou a perder a sua carga completamente.

3.2 Meio Físico de Transmissão

Como descrito em (Nordic, 2007), a camada física utilizada pelo nRF24L01+ transmite suas mensagens por meio de ondas de rádio com modulação em frequência GFSK (sigla do inglês Gaussian Frequency Shift Keying). Este tipo de modulação codifica os dados como uma série de alterações de frequência de uma onda moduladora. A forma dos impulsos da onda de rádio gerados usa uma banda espectral relativamente estreita e é, portanto, apropriado para pequenas redes sem fio que compartilham o mesmo meio com outros dispositivos.

3.2.1 Taxa de Transmissão e Uso de Banda de Frequência

Um parâmetro importante da camada física é a taxa de transmissão. No nRF24L01+, ela pode ser configurada para 250 Kbps, 1Mbps ou 2 Mbps (Nordic, 2007). Esta é uma grande vantagem deste dispositivo quando comparado com similares, como o Zigbee cuja taxa máxima é de 250 Kbps (Nanhao et al., 2008). Em aplicações comuns de redes de automação residencial normalmente não se necessita de alta taxa de transmissão de dados, entretanto uma taxa elevada também significa que a mensagem transmitida levará menos tempo para atingir o seu destinatário. Isso se traduz em uma menor probabilidade de colisão entre mensagens. Como ponto negativo, temos que, com uma mesma quantidade de energia empregada na transmissão, uma mensagem que trafega a uma taxa mais baixa tem uma maior capacidade de alcance do que a mesma mensagem a uma taxa mais alta.

O rádio nRF24L01+ é projetado para operar na banda ISM de frequência livre de 2,4 GHz, utilizando até 128 diferentes canais de 1 MHz de largura de banda cada. A Tabela 1 mostra as principais características de uso de banda de frequência deste dispositivo.

Tabela 1 – Uso de Banda de Frequência nRF24L01+ - Traduzido pelo autor a partir de tabela contida em (Nordic, 2007)

PARÂMETRO	Valor Típico	Unidade
Desvio de frequência a 250Kbps	± 160	kHz
Desvio de frequência a 1Mbps	± 160	kHz
Desvio de frequência a 2Mbps	± 320	kHz
Espaço para Canal sem Sobreposição a 250Kbps/1Mbps	1	MHz
Espaço para Canal sem Sobreposição a 2Mbps	2	MHz

Neste contexto, desvio de frequência é a medida da quantidade pela qual a frequência do sinal se difere da frequência de sua onda moduladora. Apesar de a onda se desviar desta frequência de maneira relativamente estreita, deve-se garantir um espaço mínimo muito maior, para evitar que efeitos de um sinal possam interferir na

comunicação de outros dispositivos na mesma rede. Para isso, foi definido pelo fabricante um intervalo mínimo para cada canal, que garanta a não sobreposição com os sinais de outros canais. Além disto, de acordo com a Tabela 1, ao se aumentar a taxa de transmissão, aumenta-se também a largura de banda necessária.

Neste projeto decidiu-se por fixar o uso de uma taxa de transmissão de 1Mbps em todos os testes e experimentos realizados. Esta decisão foi tomada pelo fato desta taxa combinar caraterísticas favoráveis de alcance, uso de banda de frequência e velocidade de transmissão. Ao contrário da indicação do fabricante, constatou-se que um canal tem sim a possibilidade de interferir na comunicação do canal vizinho. Isto foi constatado pelo recebimento espúrio de pacotes por rádios modulados em um canal imediatamente depois do qual a comunicação estava sendo feita. Diante disso, decidiu-se por manter um espaço de 1 MHz entre os canais usados para comunicação, buscando-se eliminar esta interferência indesejada.

3.2.2 Compartilhamento da Banda de Frequência

A banda ISM de frequência livre de 2,4 GHz, por ser de fato livre, é compartilhada com diversos outros componentes presentes no ambiente doméstico, como roteadores Wi-Fi, dispositivos Bluetouth e telefones sem fio. Esta coexistência de emissores dividindo o mesmo meio é uma fonte intrínseca de interferência e possível falha em transmissões. A Figura 7 mostra a largura de banda ocupada pelos principais dispositivos presentes nesta faixa de frequência.

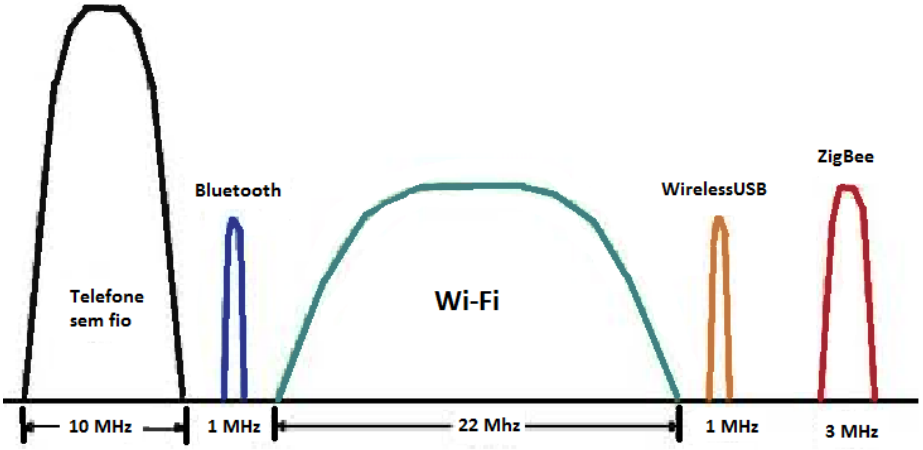


Figura 7 – Utilização de banda de sistemas sem fio operando na banda ISM de 2,4 GHz – Fonte: Tradução de figura contida em (Gerrior e Woodings, 2006)

É possível notar que as redes WI-FI ocupam um grande espectro de frequência. Isto se deve ao seu algoritmo usado para evitar colisões, que está sempre alternando sua frequência de transmissão a procura de canais com menor interferência. Essa abordagem de fazer uma leitura do nível de energia no ar, caso o componente possua essa capacidade, é citado por (Gerrior e Woodings, 2006) como uma alternativa para evitar colisões em um meio compartilhado como este. Cada protocolo e dispositivo ao seu modo possui uma maneira diferente de lidar com esse problema. A estratégia desenvolvida neste trabalho para uma rede usando o transmissor nRF24L01+ será abordada na seção 4.5.

3.2.3 Alcance de Transmissão

O nRF2401+ é classificado como um rádio de curto alcance de transmissão. Em (Nordic, 2007) é especificado que ele possui alcance de até 100m em espaço aberto, quando transmitindo a uma taxa de 250Kbps. Em diversos testes realizados em espaços fechados, a comunicação entre estes dispositivos ocorreu sem problemas de alcance quando em uma rede dentro de um mesmo ambiente aberto. Todavia, muitas vezes pode ser necessário que esta comunicação necessite atingir um alcance maior e atravessar obstáculos, como paredes e lajes, por exemplo. Neste caso, o seu alcance se limitou a uma média inferior a 10 metros, nos ensaios realizados. Este problema pode ser abordado em duas frentes. Uma delas é aumentando a capacidade de transmissão e recepção do sinal físico. Para isso existe um módulo comercial deste mesmo dispositivo, que além de aumentar a potência do sinal transmitido, é acompanhado de uma antena de tamanho maior e maior potência de irradiação. A outra abordagem é a de repassar a mensagem por meio de um roteamento entre os nós da rede, de modo a expandir o alcance de cada nó através de seus dispositivos vizinhos da mesma rede. Esta última abordagem será apresentada no Capítulo 5.

4 Camada de Enlace

Esta camada aborda as ligações entre os nós e mecanismos usados para acessar o meio físico e mover dados sobre a rede. O transceptor nRF24L01+ possui encapsulado o hardware que implementa o protocolo de enlace Enhanced Shockburst, principal responsável pelas funções de enlace do sistema. Neste capitulo também são apresentadas estratégias de acesso ao meio e de mitigação de erros.

4.1 Protocolo Enhanced Shockburst

O Enhanced Shockburst é um protocolo de camada de enlace de propriedade da Nordic Semiconductor (Nordic, 2007). Ele realiza funções automáticas de temporização, montagem, reconhecimento e retransmissão de pacotes de dados entre dois dispositivos nRF24L01+. Todo este manuseio é feito de maneira concorrente ao processamento do microcontrolador externo, sendo que este somente controla a comunicação por meio de comandos ao rádio, através da interface SPI. No Enhanced Shockburst a comunicação é feita pela troca de pacotes entre dois transceptores, um deles atuando como receptor (RX) e o outro como transmissor (TX), de acordo com o que segue:

- A troca de dados começa pela transmissão de um pacote de dados a partir da TX para o RX. Após a transmissão o TX é ajustado automaticamente no modo de recepção para esperar pelo reconhecimento da transmissão em forma de um pacote ACK.

- Se o pacote for recebido de maneira válida pelo RX, é transmitido um pacote de confirmação (ACK) para o TX antes de se retornar ao modo de recepção. É possível anexar dados ao pacote ACK, porém estes já devem estar previamente armazenados antes do recebimento da comunicação.

- Se o TX não recebe a mensagem de ACK, imediatamente ele retransmite o pacote de dados original e espera novamente por uma confirmação do receptor. É possível configurar parâmetros tais como o número máximo de retransmissões e o intervalo entre elas.

Em uma comunicação bilateral, quando se deseja uma resposta imediata a uma requisição enviada, pode-se programar os rádios para inverterem seu papel após o envio da mensagem. Neste caso, o receptor passa a atuar como transmissor, sendo que os mesmos mecanismos de transmissão continuam válidos. A Figura 8 descreve esta comunicação. Este caso mostra a ocorrência de 2 retransmissões no primeiro envio de mensagem, e outras 3 na transmissão da resposta.

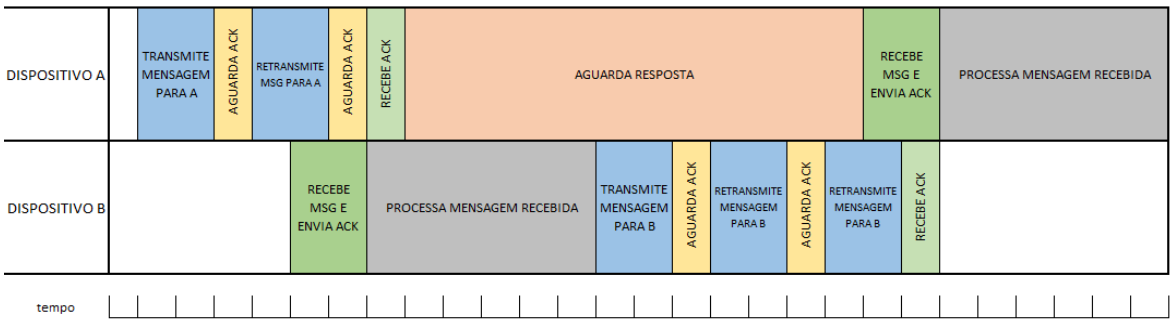


Figura 8 – Diagrama de tempo para transmissão de dados – Fonte Autor

Apesar de acrescentarem uma maior confiabilidade para em uma comunicação entre dois dispositivos, o uso do pacote de reconhecimento ACK e de retransmissões acarreta em um aumento no número de mensagens transmitidas. Em um cenário com múltiplos dispositivos compartilhando o mesmo canal, foi possível perceber, nos ensaios realizados, que estas configurações acabam por trazer um efeito negativo ao sistema, provocando um aumento expressivo na taxa de perda de pacotes.

O protocolo Enhanced ShockBurst também é responsável por montar o pacote de dados a ser enviado em cada transmissão. A Figura 9 mostra o formato do pacote de dados definido por ele.

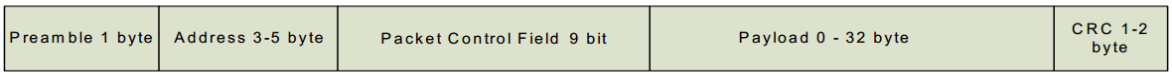


Figura 9 - Formato do Pacote – Fonte: (Nordic, 2007)

O Preamble é um byte previamente definido que serve para distinguir o início da mensagem de ruídos no canal. O próximo campo é composto pelo endereço do destinatário, que pode ter entre 3 e 5 bytes. O campo intitulado “Packet Control Field” contém o tamanho dos pacotes de dados e configurações de reconhecimento de pacotes e retransmissão. Finalmente, no campo Payload, encontra-se a real mensagem a ser transmitida, que pode possuir um tamanho entre 0 – 32 bytes. Por último encontra-se um campo para CRC (do inglês ¨Cyclic Redundancy Checking¨ ou Verificação de Redundância Cíclica), um mecanismo de detecção de erros.

O protocolo Enhanced Shockburst é concebido para uma topologia com até 7 dispositivos conectados em estrela. Esta configuração permite 2 cenários, um deles com um único receptor de diversos transmissores, e a outra com um transmissor e diversos receptores. Para o desenvolvimento de uma rede com uma topologia mais complexa e funções adicionais é necessário desenvolver um protocolo que expanda as funcionalidades do Enhanced ShockBurst. Isto será feito por meio de funções implementadas no microcontrolador Arduino. Neste caso, no entanto, estas funções adicionais passarão a exigir mais processamento do microcontrolador, pois não serão processadas dentro nRF24L01+, como é feito com as funções do Enhanced Shockburst.

4.2 Biblioteca RF24

A biblioteca RF24 (Coliz, 2012) é um driver para o transceptor nRF24L01+ desenvolvida especificamente para a plataforma Arduino. Conforme (Urdiain et al., 2012) esta biblioteca é responsável por fornecer uma interface para as funções implementadas pelo hardware deste dispositivo, de camada física e principalmente de camada de enlace, incluindo o protocolo Enhanced Shockburst. Isso é feito em um nível de abstração maior sem a necessidade de acessar diretamente os registradores do transceptor. Este software é distribuído de maneira livre sob os termos da GNU Public License versão 2. As funções desenvolvidas neste projeto usam como base esta biblioteca.

4.3 Recepção de Dados

De acordo com o protocolo Enhanced ShockBurst, as mensagens recebidas são armazenadas de maneira automática em um buffer de até 3 posições presente no

transceptor. Desta maneira, o microprocessador externo não necessita ter o seu hardware 100 % disponível para comunicação enquanto espera por uma mensagem. Porém, é necessário destinar um tempo específico do ciclo de processamento de cada dispositivo para checar no rádio se alguma mensagem foi recebida. A desvantagem deste tipo de alternativa é exigir um maior tempo de resposta para atender às solicitações de outros componentes.

Uma alternativa é o uso do pino IRQ do módulo nRF24L01+, um pino específico que gera um pulso na ocorrência de um evento relevante. A geração deste pulso ocorre quando o rádio recebeu um pacote, transmitiu com sucesso um pacote ou houve falha na transmissão. Este pino do rádio foi ligado então a um pino do processador com capacidade de acionar interrupções, de acordo com mudanças na sua entrada. No sistema proposto essa interrupção passou a ser usada somente para identificar a chegada de uma mensagem, pois já haviam funções implementadas para checar o sucesso ou não da transmissão de mensagens. Desta forma, o receptor está teoricamente sempre disponível para receber novos pacotes e atender prontamente suas solicitações, independente das instruções que o microprocessador esteja executando no momento.

A ressalva desta segunda alternativa é o fato de que na configuração padrão da plataforma Arduino não é permitida a ocorrência de interrupções simultâneas, ou seja, não pode haver uma segunda interrupção enquanto a função de tratamento à interrupção anterior esteja sendo executada. Esta configuração serve para limitar a ocorrência de erros, porém limita o uso desta função para aplicações que não dependam de outras interrupções.

4.4 Controle de Acesso ao Meio

No protocolo Enhanced ShockBurst não há nenhuma função que defina uma ordem de acesso ao meio. Os dispositivos teoricamente têm permissão de transmitir a qualquer momento uma mensagem, sem considerar o fato de que outros dispositivos podem estar transmitindo ao mesmo momento. Este cenário não é desejado, visto que ao se adicionar um número maior de nós na rede a tendência de colisões na transmissão pode aumentar consideravelmente, como apontado na seção 2.1.

Para organizar a comunicação e prevenir que dois rádios transmitam ao mesmo tempo, foi criada uma relação do tipo mestre e escravo entre os dispositivos da rede. Neste cenário, um nó é definindo como mestre, sendo responsável por centralizar a troca de informações com os demais componentes, tidos como escravos. Isto é feito de forma a criar um escalonamento do tempo de ciclo de troca de dados, onde cada escravo se comunica exclusivamente com o mestre central e somente quando requisitado. Esta divisão hierárquica age de forma a evitar possíveis colisões de pacotes. Uma representação desta topologia é mostrada na Figura 10.

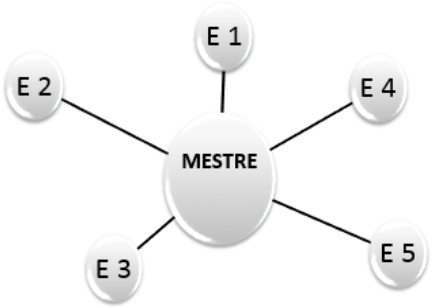


Figura 10 – Topologia da organização hierárquica de mestre e escravo – Fonte: Autor

Define-se ciclo de comunicação, como um período constante destinado para que o mestre troque dados com todos os seus escravos e execute demais tarefas de processamento a ele destinadas. A cada ciclo de troca de dados, uma sequência de passos pré-definida é executada. Inicialmente todos os escravos são definidos como receptores e o mestre como transmissor. O mestre inicia a comunicação enviando uma mensagem para um dos escravos. Ao receber o reconhecimento do escravo de que a mensagem foi recebida com sucesso, o mestre passa para o estado de receptor e aguarda pela resposta do escravo, que por sua vez passa a ser um transmissor. O mecanismo de transmissão da resposta para o mestre é idêntico ao do sentido reverso, contendo confirmação de recebimento por parte do receptor e possíveis retransmissões.

Para se garantir um determinismo entre todas as trocas de dados é preciso alocar uma quantidade de tempo fixa para a comunicação com cada componente. Caso ele não tenha recebido com sucesso a resposta do escravo neste período de tempo, uma nova tentativa só poderá ser feita no próximo ciclo. Esta sequência de operação é repetida para todos os escravos de maneira sequencial até fechar o ciclo. Na Figura 11 vemos uma representação de um ciclo de comunicação. Nela pode-se perceber que o escravo 2 não responde a solicitação enviada pelo mestre, entretanto isto não interfere no determinismo da comunicação com os demais dispositivos.

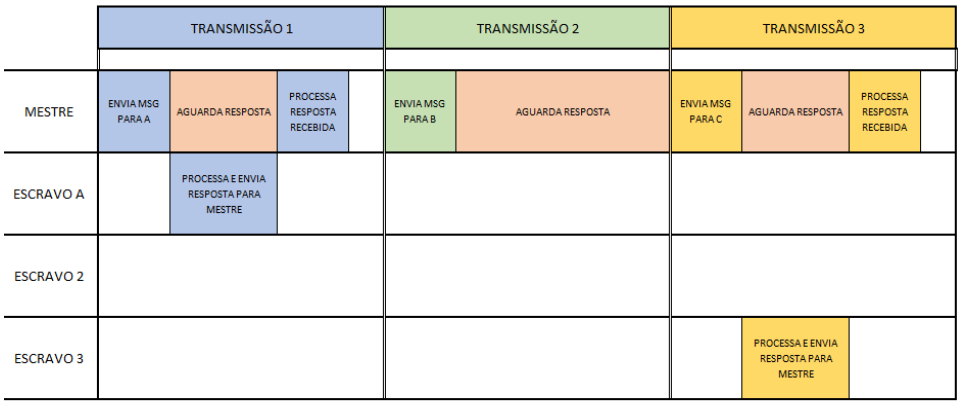


Figura 11 – Diagrama de tempo para um ciclo de comunicação – Fonte: Autor

O tempo de ciclo mínimo de troca de dados imposto pelo sistema é uma função da quantidade de componentes da rede e de quanto tempo é alocado para cada um deles, e dependerá da aplicação desenvolvida. Diversos fatores influenciam no tempo necessário para a comunicação com cada escravo. É preciso considerar o tempo máximo definido

para que ocorram retransmissões, nos dois sentidos de comunicação, e também o tempo necessário para que tanto o escravo como o mestre consigam processar as informações trocadas.

Do ponto de vista de comunicação, o controle é feito inteiramente pelo nó mestre. Cada escravo tem controle e acesso somente às suas variáveis, servindo como terminal remoto de entrada de sensores e saída de atuadores. Entretanto, é possível implementar-se funções dentro dos escravos para interpretação de comandos, tratamentos e filtragem de dados de sensores, condicionamento de atuadores e inclusive sistemas de controle discreto e contínuo. Desta forma, apesar de o controle das informações ser centralizado, o controle e a lógica do sistema de automação podem ser distribuídos entre todos os componentes da rede. Isto é benéfico do ponto de vista de que os Arduinos usados possuem capacidade limitada de processamento, o que para aplicações mais complexas inviabilizaria o uso de um processamento central. Outro ponto positivo desta abordagem é a robustez e segurança do sistema, pois, caso haja uma falha no nó mestre, os demais nós podem ter autonomia para levar o sistema para um estado de operação seguro.

4.5 Mecanismo para Evitar Colisões em Acesso ao Meio

Mesmo que sejam desenvolvidas estratégias que evitem que dois nós da rede transmitam ao mesmo tempo, não é possível prever quando outro dispositivo externo esteja acessando o mesmo meio, como visto na seção 3.2. Diante disso o nRF24L01+ possui uma funcionalidade específica, que retorna um bit informando se existe ou não algum sinal naquela faixa de frequência com potência maior que -64dBm (Nordic, 2007). Usando-se desta funcionalidade, foi desenvolvida uma função que escuta o canal a ser acessado por um tempo pseudoaleatório antes de efetuar qualquer transmissão. Foi definido experimentalmente, em diversos ensaios, um tempo de espera considerado suficiente que varia entre 100µs e 1000µs, podendo se estender até um limite de 4ms, caso encontre sinais sendo transmitidos no meio. Caso o tempo de espera ultrapasse os 4ms sem conseguir encontrar um espaço no meio, a transmissão não é efetuada e tenta-se novamente somente no próximo ciclo de troca de dados. Este tempo de espera aleatório interfere no determinismo do sistema, porém para tempos de ciclo da grandeza de décimos de segundo, o seu efeito é de baixa relevância.

O tempo de espera é determinado pelo gerador de números randômicos padrão das bibliotecas do Arduino. Porém os números ditos como aleatórios, fornecidos por esta função, na verdade possuem uma sequência pré-definida. O que foi feito para diferenciar esta sequência em cada dispositivo foi atrelar a inicialização desta geração por meio da leitura de uma das entradas analógicas não utilizadas do microcontrolador. Desta forma, a sequência de números gerados está atrelada ao ruído presente na entrada. Este ruído pode ser aproximado como um ruído branco e desta forma, de fato aleatório.

5 Camada de Rede

A camada de rede define processos utilizados para encaminhar os dados através da rede e a estrutura e da utilização de endereçamento lógico. Uma rede de automação residencial demanda funções mais complexas de rede, como um maior número de elementos, diferentes topologias e melhores estratégias de fluxo de dados. Porém funções desta camada não se encontram originalmente definidas para o sistema proposto. É neste ponto que se faz necessário o desenvolvimento de um protocolo específico para esta camada que atenda às necessidades mais complexas exigidas por uma rede de automação.

5.1 Definição de Cluster

Um cluster pode ser definido como a junção de dois ou mais componentes da rede em uma mesma estrutura de forma que, para os demais dispositivos da rede, ela se comporte como se fosse uma entidade única. Neste trabalho, esta estrutura é composta por um nó mestre centralizado que comanda a comunicação entre seus diversos escravos, implementando a proposta apresentada em seções anteriores. Este cluster é montado de forma que seus componentes compartilhem variáveis e tarefas.

Dentro desta hierarquia, o mestre do cluster é responsável por toda a interface de comunicação com os componentes externos da rede. Mesmo que algum dispositivo externo necessite de alguma informação presente em um dos escravos, essa requisição é primeiramente direcionada para o mestre, que então busca esta informação com o respectivo escravo e após responde para o solicitante.

5.2 Rede Dividida em Clusters

Seguindo a estratégia apresentada por (Urdiain et al., 2012), vide seção 2.2, foi desenvolvida uma rede que divide uma rede mais complexa em clusters de tamanhos menores. Na proposta apresentada pelo artigo, a estrutura de rede era voltada para uma rede de sensores, onde há somente uma direção no fluxo de dados. Para comunicações mais complexas porém, a mesma é falha, pois implica na necessidade de um nó central e em uma dificuldade para a transmissão de dados do nó central para os mestres de cada cluster, pois este era definido exclusivamente como receptor.

Neste trabalho foi desenvolvida uma abordagem diferente da proposta por (Urdiain et al., 2012), mas que aproveitasse suas principais características. Manteve-se a divisão da rede em clusters, com cada um destes operando internamente a uma frequência distinta. Esta divisão permite que a comunicação interna de cada cluster não interfira na troca de dados dos demais. Entretanto, devido às características de curto alcance do rádio e à necessidade de expandir a cobertura da rede para o maior espaço possível, optou-se por eliminar a existência de um nó central à rede. Desta forma, a troca de informações entre os clusters passa a ser direta e bidirecional entre os nós mestres. Como requisito para esta topologia, é necessário que cada mestre esteja ao alcance de todos os outros, ou pelo menos ao alcance dos mestres dos clusters com quem ele se comunica. Na Figura 12 é mostrado um esboço desta nova configuração.

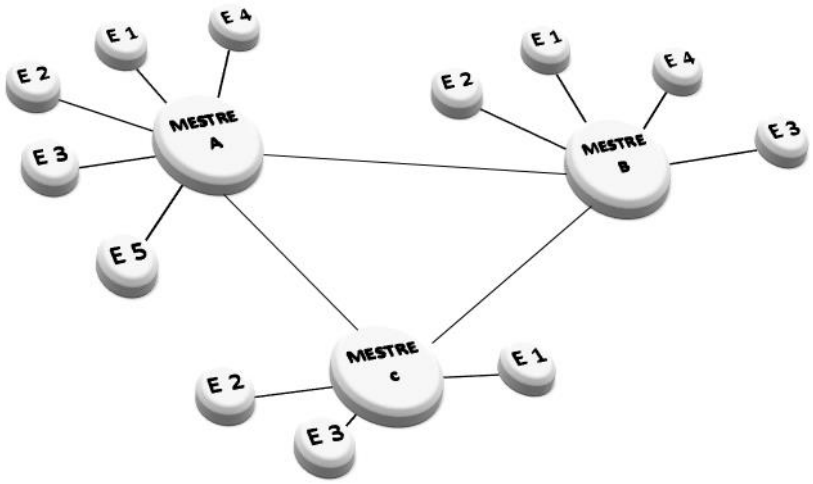


Figura 12 – Topologia de rede dividida em clusters – Fonte: Autor

Neste novo cenário, a rede passa a ter uma topologia semelhante à de uma árvore, aumentando o seu alcance e abrindo a possibilidade para o uso de um número muito maior de componentes. Isto é possível em função da responsabilidade do controle da comunicação ser dividida entre diversos mestres, ao contrário da topologia apresentada na seção anterior, onde um único componente gerenciava a comunicação de todos os dispositivos. Esta nova configuração, contudo, necessita de ajustes no modo como os nós trocam informações. (Urdiain et al., 2012) citaram problemas de sincronização entre os clusters com o uso de microcontroladores Arduino de baixo custo. Portanto, a estratégia a ser empregada deve levar em consideração uma possível falta de sincronia entre os nós mestres. Outro fator importante é de que a comunicação entre os clusters não possui a mesma hierarquia mestre/escravo como é feito na estrutura interna de cada cluster. Diante disso, é necessário prever que a troca de mensagens possa ser feita de maneira aleatória entre os mestres dos clusters.

A solução encontrada para atender as necessidades de troca de informações entre clusters foi a adição de um segundo transceptor nRF24L01+ a cada nó mestre. De modo que ambos os rádios compartilham o mesmo barramento SPI usado na comunicação com o microcontrolador. O rádio adicional atua somente como receptor aguardando exclusivamente as mensagens de outros mestres. Para evitar que as mensagens externas ao cluster interfiram na comunicação interna, foi designada uma segunda frequência, diferente da usada internamente, para recepção de mensagens externas. Portanto, nesta configuração, cada cluster é associado a dois canais de frequência distintas.

Em experimentos realizados, quando interrupções eram geradas para notificar o microcontrolador da chegada de novas mensagens, a adição de um segundo rádio aos nós mestres tornou o comportamento do sistema instável, aumentando de maneira considerável os erros de transmissão. Esta situação pode gerar a ocorrência de interrupções simultâneas, que como explicado anteriormente, não é permitida pela configuração padrão do Arduino. Para evitar que este problema aconteça, definiu-se que os nós mestre passariam a não usar mais deste recurso. Portanto, cada mestre deve escalonar um tempo do seu ciclo para a checagem e processamento de mensagens externas recebidas. Não é possível prever, no entanto, quantas mensagens externas podem chegar dentro do intervalo de um ciclo de comunicação. Este valor varia

dependendo da aplicação sendo executada. Em geral, a partir de considerações feitas por ensaios realizados, o período do ciclo de comunicação nesta estrutura de rede deve ser de no mínimo três vezes o tempo que inicialmente seria destinado para a comunicação somente com os escravos em uma topologia do tipo estrela.

Com a troca aleatória de mensagens entre os clusters, o determinismo entre uma mensagem e outra passa a ficar limitado para a comunicação entre componentes de mesmo cluster. Este fator altera a maneira de estruturar a rede ao se desenvolver uma aplicação. Para comunicações com clusters externos, deve-se prever que o mestre do cluster vizinho não esteja o sempre disponível para responder à solicitação enviada. Além disso, dependendo da solicitação, talvez o mestre tenha que buscar a informação almejada dentro de um de seus escravos. A Figura 13 mostra um diagrama de tempo para uma comunicação bidirecional entre dois clusters diferentes, onde este tipo de situação acontece.

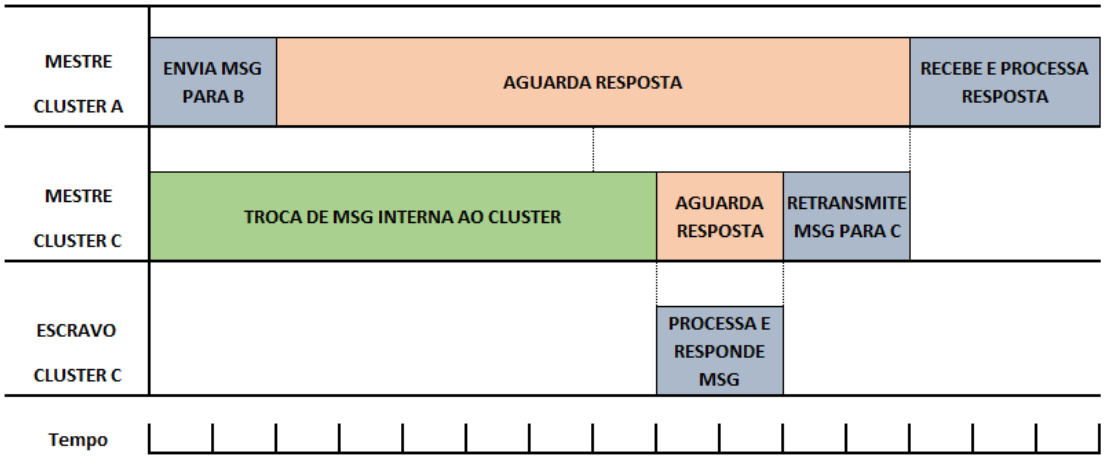


Figura 13 – Diagrama de tempo para comunicação entre clusters – Fonte: Autor

À medida do possível, deve-se dar preferência para manter no mesmo cluster nós com sensores e atuadores destinados a uma mesma função. Esta divisão dos clusters por função facilita a implementação e aumenta o desempenho do sistema de automação, uma vez que diminui a necessidade de troca de mensagens com componentes externos. Como exemplo, pode-se citar um sistema de automação hipotético de uma sala com piscina térmica. A rede deste sistema poderia ser dividida em três diferentes clusters, um para contar o número de pessoas que entram e saem do ambiente (com o uso de sensores de passagem presentes em cada entrada), o outro para controlar a temperatura da água (com sensores de temperatura e atuadores para bombear e aquecer a água) e um último para fornecer uma interface remota com o usuário (com o uso de um painel de comandos e uma interface ethernet). Neste caso, dentro do cluster de controle de temperatura poderia ser implementado um sistema de controle de laço fechado determinístico. Enquanto isso o outro cluster operaria os sensores de passagem para informar o número de pessoas presente no ambiente. O que poderia ser usado para informar um usuário remoto ou para desativar o aquecimento quando ninguém estivesse presente no ambiente. Dentre todas as informações de sensores e atuadores, as únicas variáveis trocadas entre os clusters se limitariam a temperatura da piscina, o número de pessoas da sala e a alguns poucos comandos esporádicos vindos da interface com o usuário.

5.3 Retransmissão de Mensagens entre Clusters

A topologia apresentada na seção anterior prevê como requisito que todos os mestres dos clusters com que se deseja comunicar estejam ao alcance de transmissão. Em ambientes prediais grandes, com diversos ambientes e obstáculos físicos entre o transmissor e o receptor, nem sempre isto pode ser garantido. Para ampliar esta capacidade de alcance, desenvolveu-se um mecanismo de retransmissão de mensagens entre os nós mestres.

Existem diversas estratégias desenvolvidas para o roteamento de mensagens entre os nós de uma rede. Porém muitas delas requerem um uso de processamento e memória muito elevados para o padrão limitado do microprocessador Arduino (que ainda precisa processar todas as outras funções de transmissão e recepção de mensagens). Além disso, estratégias que forcem um pacote de mensagem a trafegar entre um número elevado de nós, tendem a necessitar de um tempo maior de espera, o que não é desejável de uma maneira geral para um bom desempenho da rede. Portanto, o roteamento de mensagens foi limitado a somente um componente intermediário. Esta pequena alteração, aparentemente pouco significativa, incrementa consideravelmente o alcance total da rede e abrange ainda mais possibilidades de implementação de aplicações. A Figura 14 mostra a configuração desta nova topologia de rede. O mecanismo desenvolvido para atender este objetivo baseia-se no mapeamento dos dispositivos alcançados por meio de um mapa de rede, que será explicado a seguir. Antes disso, porém, é necessário prever qual será a capacidade máxima do sistema.

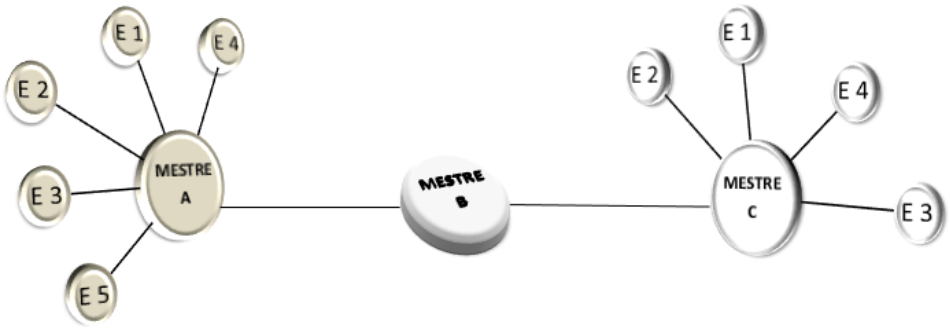


Figura 14 – Topologia de rede com mecanismo de retransmissão entre nós mestres – Fonte: Autor

5.3.1 Definição do Tamanho Máximo da Rede

Para determinar o espaço necessário pelo mapa de rede e o tamanho das mensagens de atualização deste mapa é preciso prever qual o conjunto máximo de clusters que o sistema irá comportar. Temos que cada cluster ocupa 2 canais de um total de 128, desta forma é possível haver até 64 diferentes clusters em uma rede. Levando em conta a possibilidade de interferência de canais vizinhos, relatada na seção 3.2, optou-se por limitar o número máximo de clusters para 32. Esta alteração também permite um mapeamento de rede mais rápido e eficiente, com mensagens de atualização menores e que ocupem menos da já limitada memória do microprocessador.

5.3.2 Vetor de Alcance e Mapa de Rede

Para o entendimento do mecanismo usado para retransmitir as mensagens na rede, primeiramente se faz necessário definir estes dois conceitos usados.

Vetor de Alcance é uma palavra de 32 bits, onde cada bit representa um dos 32 possíveis clusters da rede. Cada nó mestre da rede possui um vetor de alcance próprio, onde as posições binárias de valor 1 representam os clusters que estão dentro do alcance de transmissão deste dispositivo. Os bits de valor 0 significam clusters inativos ou fora de alcance. A linguagem de programação do Arduino permitiria a declaração de um vetor de variáveis booleanas, contudo, como ele trabalha com um processador que somente é capaz de executar instruções de 8 bits, cada posição deste vetor estaria de fato ocupando 8 bits de memória ao invés de 1 bit. Para otimizar o espaço de memória ocupado, este vetor é de fato armazenado como uma variável inteira de 32 bits, onde estratégias de manipulação binária são realizadas quando se deseja acessar ou modificar algum bit específico.

O Mapa de rede é o espaço destinado para armazenar os vetores de alcance de todos os possíveis 32 dispositivos da rede. Em termos lógicos, se trata de uma matriz binária de 32x32 bits. O mapa de rede de cada dispositivo mestre é composto pelo vetor de alcance do próprio dispositivo e pelos vetores de alcance dos dispositivos por ele alcançados. Na representação da linguagem de programação do Arduino ele, é definido como um vetor de 32 posições, cada uma com uma variável inteira de 32 bits, totalizando 128 bytes de memória ocupados.

5.3.3 Mecanismo de Busca e Retransmissão

Quando surge a necessidade de se transmitir uma mensagem para um dispositivo de um cluster externo, primeiramente é procurado no mapa de rede, dentro do vetor de alcance do próprio dispositivo, se aquele cluster está classificado como ativo e dentro do alcance. Caso o destinatário esteja nesta lista, a mensagem é enviada diretamente para ele. Para o caso contrário, o nó transmissor irá buscar pelo destinatário da mensagem no vetor de alcance de algum dos nós com quem ele consegue se comunicar. Para o caso de algum dos bits da coluna específica do destinatário, na matriz do mapa de rede, possuir valor diferente de 0, a mensagem será enviada para o dono do vetor de alcance no qual isto foi constatado, que deverá retransmiti-la para o destinatário final. Na mensagem enviada para o nó intermediário, deverá haver um campo informando que ela deve ser retransmitida e outros contendo informações sobre o canal e a identificação do destinatário final. Informações mais específicas sobre o formato da mensagem serão abordadas na seção 6.1. Se o cluster com o qual se deseja comunicar não estiver marcado como ativo em nenhuma das posições do mapa de rede, a mensagem não será enviada e um erro de comunicação será sinalizado.

Neste tipo de comunicação, a mensagem circula por um número maior de nós da rede, sendo que nenhum deles necessariamente estará disponível para responder à solicitação de maneira imediata. Isto leva a um aumento no intervalo necessário entre a transmissão da mensagem para o cluster intermediário e a chegada da reposta quando comparado com os casos anteriores. Além disso, a chance de ocorrência de erros na transmissão é múltipla da quantidade de nós por onde ela passa. A Figura 15 exemplifica com um diagrama de tempo uma situação onde o mestre do cluster A necessita de uma

informação contida em um escravo do cluster C. Para isso o cluster B serve como intermediário, retransmitindo a mensagem enviada por A.

Figura 15 – Diagrama de tempo de uma mensagem retransmitida por um cluster intermediário – Fonte: Autor

5.3.4 Construção e Atualização do Mapa de rede

Para se fazer uso desta estrutura de retransmissão de mensagens, é necessário que os nós possuam um mapa de rede atualizado de todos os dispositivos que estejam aptos à comunicação. Neste contexto, foi desenvolvido um mecanismo que atualiza as informações inerentes da rede de maneira dinâmica à medida que dispositivos são adicionados. Ao ser inicializado, cada dispositivo mestre manda uma mensagem para todos os canais ondem possam existir clusters.

A mensagem enviada para cada canal segue os mesmos passos de toda mensagem usual, porém com configurações de número e intervalo entre retransmissões aumentadas de modo a vasculhar de uma maneira mais persistente todos os possíveis canais da rede. Para os canais que responderem à mensagem enviada, o seu bit correspondente no vetor de alcance é atribuído o valor 1.

Um canal é tido como fora do alcance ou inativo se, ao final de todas as tentativas de retransmissão, nenhuma resposta for recebida do canal de destino. Isto vale também para o caso onde ao se executar o passo de escutar o canal de destino antes de transmitir, o canal permaneça ocupado por um tempo de espera maior que o definido. Isto indica a presença de ruído elevado no canal, impedindo qualquer comunicação. Para estes casos o canal é tido como não-alcançável e é marcado um 0 na sua respectiva posição do vetor de alcance.

Após checar todos os possíveis canais da rede, o dispositivo possui uma relação completa de todos os clusters que ele pode alcançar, armazenada no seu vetor de alcance. A partir disso, ele envia o seu vetor para cada dispositivo encontrado, para que os demais dispositivos saibam quem está ao seu alcance. Em resposta a esta mensagem os demais dispositivos também enviam o seu próprio vetor de alcance. A união destes vetores forma o mapa de rede.

6 Camada de Aplicação

A camada de aplicação é a interface mais direta da rede com o usuário. Neste trabalho, considera-se o usuário como sendo o projetista de um sistema de automação. Neste capítulo serão abordadas funções e estratégias voltadas para definir o modo como o sistema de automação gerencia os dados trocados.

6.1 Estrutura da Mensagem

O protocolo da camada de Enhanced ShockBurst define o formato e a estrutura do pacote de dados a ser enviado, como mostrado na seção 4.1. Dentre todos os campos contidos dentro deste pacote, o mais importante refere-se à mensagem de dados propriamente dita, nomeada pelo protocolo como Payload e delimitada pelo intervalo de tamanho de 1 a 32 bytes. O protocolo de enlace é responsável por fornecer o acesso deste pacote de dados no meio físico a ser transmitido, entretanto em nenhum momento ele se preocupa com o tipo de dados contido nele. Neste trabalho, o formato destes dados e a maneira como eles são usados é definido pela camada de aplicação. Onde ele é moldado para atender às necessidades de um sistema genérico de automação predial e a estrutura de rede proposta.

O espaço de 32 bytes destinado ao pacote de dados foi dividido em diferentes campos. Na linguagem de programação do microcontrolador Arduino, isto se fez com a montagem de uma estrutura de dados composta por diferentes tipos de variáveis, conhecida como Struct. As informações contidas nas mensagens são divididas em um cabeçalho, variáveis de 8 e 32 bits e comandos associados a cada variável. Uma representação desta estrutura é mostrada na Figura 16.

ESTRUTURA DA MENSAGEM DE DADOS																															
CABEÇALHO					COM. 8-BITS						VARIÁVEIS DE 8-BITS						COM. 32-BITS			VARIÁVEIS DE 32-BITS											
CANAL DEST	IDENT DEST	CANAL REMET	IDENT REMET	FLAGS	DCMD 0	DCMD 1	DCMD 2	DCMD 3	DCMD 4	DCMD 5	DVAR 0	DVAR 1	DVAR 2	DVAR 3	DVAR 4	DVAR 5	ACMD 0	ACMD 1	ACMD 2	AVAR 0				AVAR 1				AVAR 2			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Bytes (1 - 32)																															

Figura 16 – Estrutura da mensagem de dados – Fonte: Autor

6.1.1 Cabeçalho

Neste subespaço estão contidas informações sobre o destinatário e o remetente da mensagem, além de uma série de bits de configuração denominados FLAGS, totalizando um tamanho fixo de 5 bytes. As informações sobre o destinatário e o remetente da mensagem são divididas em dois campos para cada, um deles definindo o canal associado ao cluster em que o dispositivo está presente, e o outro a identidade que ele possui dentro deste cluster. Estes dois campos juntos definem a identidade única deste dispositivo dentro da rede. Apesar de o protocolo de enlace já apresentar estas informações no pacote que envolve estes dados, esta redundância é necessária para identificar o destinatário final da mensagem, quando ela passa por nós intermediários. Além disso, ela serve para evitar que rádios receptores acabem por ler e processar

mensagens que não estavam destinadas para seu endereço, falha esta que foi constatada de maneira recorrente nos testes realizados.

No espaço FLAGS, mostrado na Figura 17, são definidas algumas informações básicas sobre a o tipo de mensagem que está sendo enviada. Elas são usadas para identificar de maneira rápida o motivo da mensagem e desta forma direcionar o tratamento dado a ela. O campo RETR indica que o receptor da mensagem não é o destinatário final da comunicação, devendo este repassá-la ao destinatário presente na prévia identificação. O bit RESP é ativo quando o dispositivo que mandou a mensagem não está à espera de uma resposta do destinatário. Para o campo MAP, o valor 1 é atribuído quando a mensagem enviada é de atualização do mapa de rede, neste caso o receptor sabe que o único dado a ser trocado é o do vetor de alcance. Por último, o campo DATA indica troca de dados. Os demais bits do campo FLAGS encontram-se vazios e são destinados a futuras aplicações.

FLAGS							
MAP	RESP	RETR	DATA	VAZIO			
1 - Ativado 0 - Desativado	1 - Ativado 0 - Desativado	1 - Ativado 0 - Desativado	1 - Ativado 0 - Desativado				
7	6	5	4	3	2	1	0
bits (7 - 0)							

Figura 17 – Campo FLAGS – Fonte: Autor

6.1.2 Variáveis

Em um sistema de automação residencial, diferentes tipos de informação são trocados entre seus componentes. Em sua maior parte pode-se considerar que sejam variáveis de pequena dimensão, como status de atuadores/sensores, comandos de liga/desliga ou pequenos contadores. Porém, é necessário muitas vezes o envio de informações maiores ou com uma maior precisão numérica, tal como algumas leituras de sensores ou o vetor de alcance, descrito na seção 5.3.2. Para tanto, foram definidas duas categorias diferentes de variáveis a serem enviadas. Uma delas compreende variáveis de 8-bits, sendo destinado um espaço para que seis destas possam ser enviadas em uma mesma mensagem. A outra categoria está associada às variáveis de tamanho maior, cada uma ocupando 32 bits, onde é permitido o envio de três variáveis simultaneamente.

Em cada categoria de tamanho, as variáveis podem assumir três diferentes formas, sendo que cada formato possui uma extensão numérica diferente. Esta flexibilidade de representação tem o propósito de atingir as necessidades das mais variadas aplicações a serem desenvolvidas em um ambiente de automação. Esta implementação foi possível devido a uma estrutura da linguagem de programação do Arduino chamada Union. Com o uso dela, é possível definir um tipo de variável que possa assumir diferentes formas pré-definidas. A Tabela 2 mostra os diferentes tipos de variáveis permitidos, bem como a extensão que elas podem atingir.

Tabela 2 – Tipos de Variáveis a serem trocadas – Fonte (Brooks, 1999)

TAMANHO DA VARIÁVEL	QUANTIDADE	TIPO	EXTENSÃO NUMÉRICA		
8 - bits	6	BOOL	0	-	1
		UINT 8	0	-	255
		INT 8	-128	-	127
32-bits	3	FLOAT	3,4 E-38	-	3,4 E38
		UINT 32	0	-	4.294.967.295
		INT 32	-2.147.483.648	-	2.147.483.647

6.1.3 Comandos

Os comandos são campos de 8-bits de extensão, que estão associados a cada uma das variáveis contidas na mensagem, totalizando 6 comandos para variáveis de 8-bits e 3 comandos para as de 32 bits, como mostrado na Figura 16. A função do comando é orientar o receptor da mensagem sobre o que fazer com as variáveis enviadas ou qual variável está sendo requisitada. Na Figura 18 é mostrado o formato base de cada uma destas estruturas.

COMANDO							
LEIT/ESCR	TIPO DE VARIÁVEL		TAG DA VARIÁVEL				
0 - Leitura 1- Escrita	01 - BOOL / FLOAT 10 - UINT8 / UINT 32 11 - INT8 / INT32		VALOR ENTRE (0 - 35)				
7	6	5	4	3	2	1	0
bits (7 - 0)							

Figura 18 – Variável comando – Fonte: Autor

O primeiro campo de cada comando indica se a variável associada a este está sendo enviada para o destinatário (escrita), ou se está sendo feito uma requisição de leitura de alguma variável. Neste último caso, na resposta à solicitação, a leitura requerida é alocada no mesmo campo destinado à variável associada a este comando. Os próximos dois bits definem o tipo da variável que está sendo enviada ou requisitada. Apesar de possuírem a mesma estrutura de comando, os tipos associados às variáveis de 8-bits se diferem das de 32-bits. Os 5 bits restantes representam um valor inteiro, que varia de 0 a 35, associado à identificação da variável. Esta identificação está associada a um conjunto de variáveis compartilhadas pela rede. Para cada tipo, existe um vetor com 35 possibilidades distintas de variáveis a serem compartilhadas por cada dispositivo. Estas variáveis são de acesso global na rede e cabe a cada dispositivo definir quais das suas informações são compartilhadas com os outros por meio delas. Na Figura 19 é exemplificado a maneira com dois comandos distintos são interpretados.

Exemplo de Comandos																
COMANDO DCMD 2									COMANDO ACMD 1							
1	1	0	0	0	1	1	1		0	1	0	0	0	1	1	1
INTERPRETAÇÃO - Estou enviado uma variável do tipo uint8, presente no campo DVAR 2. Ela deve ser escrita na sua variável 7 do tipo uint8.									INTERPRETAÇÃO - Estou requisitando a sua variável 7 do tipo float. Ela deve ser escrita no campo AVAR1 da mensagem de resposta.							

Figura 19 – Exemplos de interpretação de comandos – Fonte: Autor

7 Resultados de Ensaios Realizados

7.1 Ensaio de Comunicação Bidirecional entre Dois Componentes

Neste experimento, dois componentes da rede foram dispostos de maneira a ficarem distanciados por aproximadamente 8 metros. As configurações da rede incluíam o uso de confirmação de recebimento de mensagens (ACK), e até três possíveis retransmissões em caso de falha. Estes dispositivos constituíam um par mestre/escravo, onde o primeiro enviava uma mensagem e aguardava pelo recebimento da resposta vinda do escravo. Este ciclo de troca de mensagens foi repetido a cada 30 ms, onde a cada vez era medido o tempo que levava para a mensagem enviada ser respondida.

Para medir a latência entre transmissão e recepção, foi enviada uma mensagem contendo dois comandos. O primeiro era de escrita, de uma variável de 32 bits contendo o tempo medido pelo transmissor pouco antes de enviar o pacote. Por sua vez, o segundo comando requisitava a leitura da mesma varável de 32-bits que havia sido escrita pelo comando anterior. Com a chegada da reposta, o valor recebido era subtraído do tempo atual.

Nesta configuração, foram testados três diferentes cenários. No primeiro, os dois componentes se comunicavam livremente sem interferências. A função que escuta por ruídos no canal antes de transmitir, descrita na seção4.5, estava ativa. No segundo teste foi adicionado à configuração prévia um terceiro elemento, cuja função era gerar interferência na rede. Este dispositivo foi programado para transmitir mensagens no mesmo canal, com um ciclo que variava randomicamente entre 10 e 30 ms. Em um último experimento a interferência foi mantida, mas desta vez a função que escuta o canal foi desativada. Os resultados destes ensaios podem ser vistos na Tabela 3.

Tabela 3 – Ensaio de Comunicação Bilateral – Fonte Autor

ENSAIO	TENTATIVAS DE COMUNICAÇÃO	MENSAGENS ENVIADAS COM SUCESSO	MENSAGENS RECEBIDAS COM SUCESSO	SUCESSO NA PRIMEIRA TRANSMISSÃO	SUCESSO NO ENVIO DA RESPOSTA	LATÊNCIA MÉDIA (µs)
COM função de ESCUTA e SEM INTERFERÊNCIA	1112	1112	1111	100.00%	99.91%	3905
COM função de ESCUTA e COM INTERFERÊNCIA randômica entre 10 e 30 ms	1217	1203	1153	98.85%	94.74%	5493
SEM função de ESCUTA e COM INTERFERÊNCIA randômica entre 10 e 30 ms	1182	1167	1019	98.73%	86.21%	4877

Como pode ser visto, na primeira situação, a comunicação obteve uma taxa inexpressiva de erros. Para os casos onde foi adicionada a interferência na rede, é possível analisar claramente o efeito da função de escuta. Apesar de ter adicionado uma latência maior para a rede, ela foi capaz de reduzir consideravelmente os efeitos da interferência. A latência do terceiro caso também aumentou, em comparação com o primeiro, devido ao maior número de retransmissões provocado pela interferência.

7.2 Ensaio de Comunicação Interna ao Cluster

O objetivo deste ensaio é analisar a comunicação entre um nó mestre e quatro escravos dispostos em uma topologia do tipo estrela, distanciados em 4 metros do nó central. A cada ciclo de comunicação, era enviada uma mensagem para cada nó escravo, com um espaçamento de 50 ms para cada uma. A configuração de confirmação de recepção de mensagens foi desativada, pois em experimentos foi detectado que, para

comunicação com múltiplos dispositivos, esta configuração acabava por provocar um aumento expressivo na taxa de perda de pacotes.

O mesmo mecanismo para medir a latência da rede do ensaio anterior foi novamente utilizado, porém desta vez os resultados são apresentados de maneira gráfica, como segue na Figura 20. Nestes gráficos, os pontos de valor de latência que cruzam a linha de 1,5 milissegundos (ms) significam perdas de pacote. A numeração dos escravos indica a ordem na qual a comunicação é feita dentro do ciclo de transmissão.

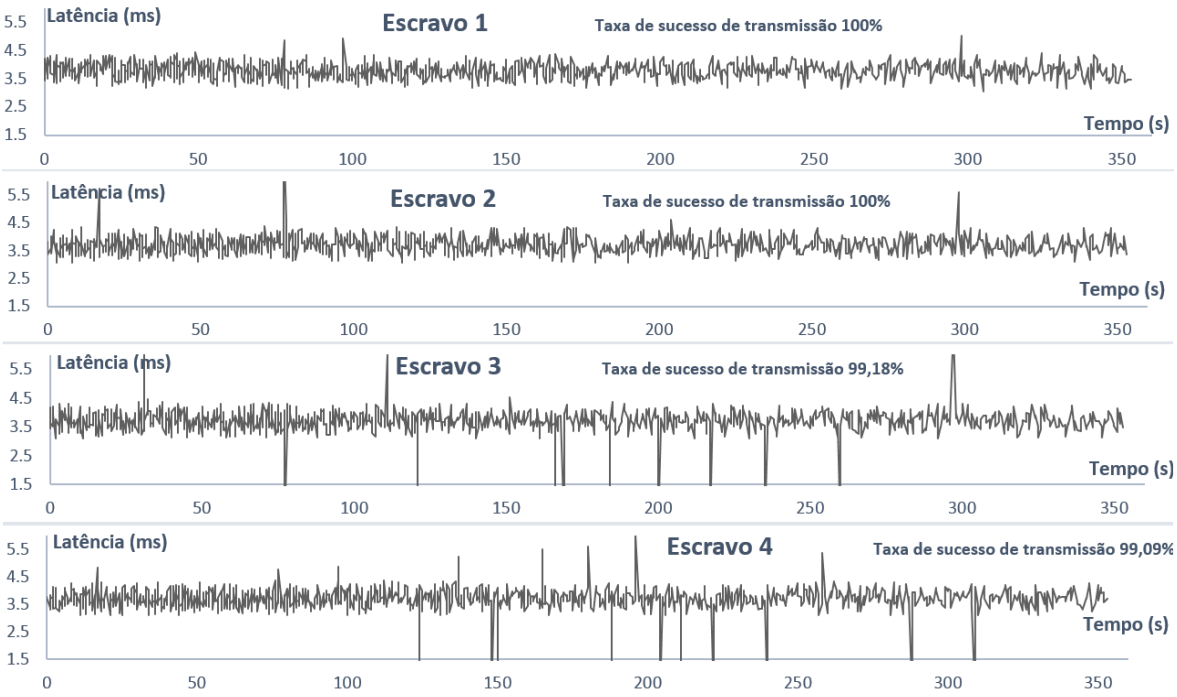


Figura 20 – Medida de latência para ensaio com múltiplos dispositivos – Fonte: Autor

Ao se analisar os resultados é possível acompanhar o efeito indesejado de outra configuração da rede, o uso de retransmissões. Nota-se uma perda de qualidade na comunicação dos últimos escravos em relação aos primeiros. Isto se explica pela possível ocorrência de retransmissões espúrias vindas das primeiras transmissões. O uso de retransmissões também afeta o determinismo do sistema.

7.3 Ensaio de Comunicação Entre Clusters

Neste ensaio, é testado o comportamento da rede ao se efetuar uma comunicação com um cluster externo. Nele, um mestre de um cluster se comunica com seus três escravos, com 30 ms de espaço para cada comunicação, em seguida envia uma mensagem para um cluster externo e aguarda pela resposta. O tempo de ciclo total deste mestre é composto por 90 ms de comunicação interna e outros 90 ms destinados a troca de mensagens externas, totalizando 180 ms. Já o outro cluster, destina 80 ms para troca de dados interna e também 80 para externa, num total de 160 ms de ciclo. Neste ensaio a configuração que permite retransmissões em caso de falha foi desativada. Os gráficos com a latência de cada mensagem são vistos na Figura 21.

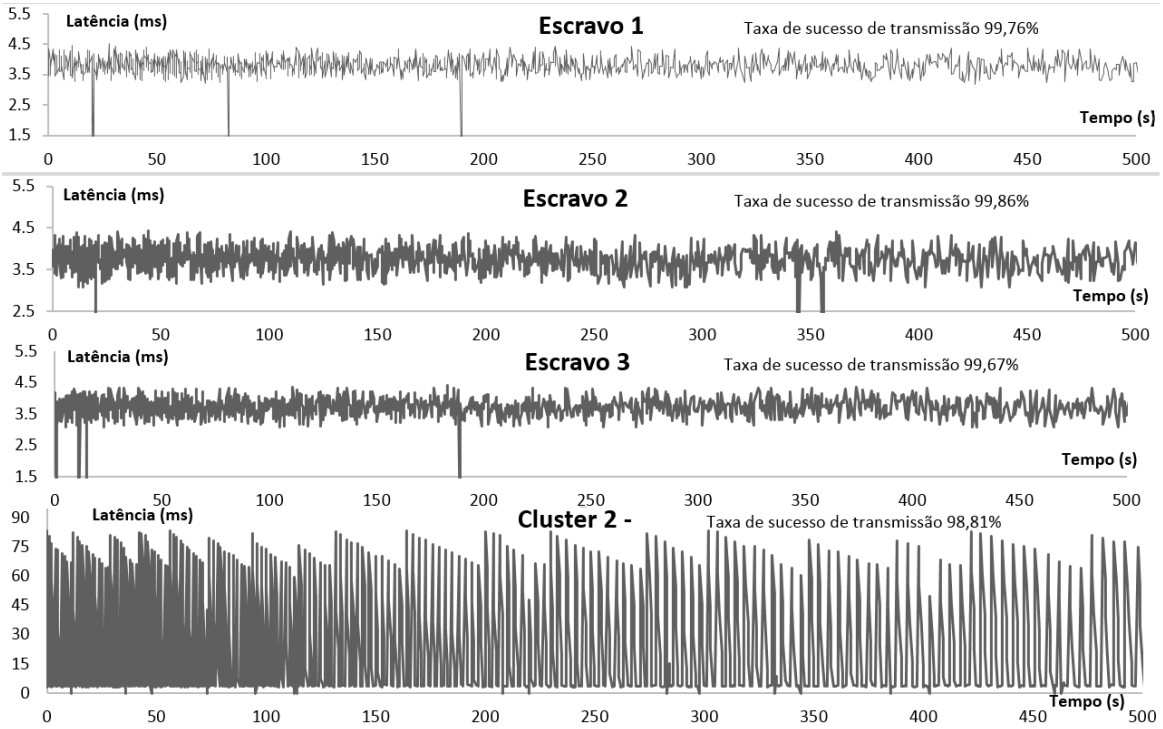


Figura 21 – Latência em comunicação interna e externa ao cluster – Fonte: Autor

Ao analisar os resultados, é notável o aumento expressivo na latência da mensagem trocada com o cluster externo. Isto se deve ao fato deste não estar sempre disponível para responder às mensagens recebidas, devido às suas tarefas de comunicação interna. É possível notar também um leve padrão de variação no tempo de espera das mensagens, isto se deve ao fato de ambos os mestres se comunicarem em um ciclo fixo, porém com tempos diferentes.

Nos gráficos da comunicação com os escravos, nota-se o efeito da desativação do uso de retransmissões. Esta nova configuração provocou uma leve piora da comunicação dos primeiros escravos, porém não é visto mais a mesma discrepância entre diferentes transmissões do mesmo ciclo, presente no ensaio anterior.

7.4 Ensaio com o uso de Retransmissão de Mensagens

Com este experimento, pretende-se analisar o modo como a rede se comporta com o uso de uma retransmissão de mensagem. Para isso, um nó mestre envia uma mensagem para o mestre de um cluster intermediário, requisitando que esta seja retransmitida para um terceiro destinatário final. Para a resposta, a mensagem atravessa o mesmo caminho. Cada cluster possui um tempo destinado para comunicação interna e outro para externa, sendo estes tempos diferentes para cada um. O resultado do ensaio é mostrado na Figura 22. Apesar do gráfico mostrar apenas os 5 primeiros minutos, a extensão real do ensaio foi de 191 minutos. O objetivo deste tempo prolongado foi confirmar a robustez do sistema para transmissões prolongadas, o que de fato pôde ser observado. No gráfico mostrado nota-se uma variação ainda maior na latência entre a mensagem enviada e a recebida.

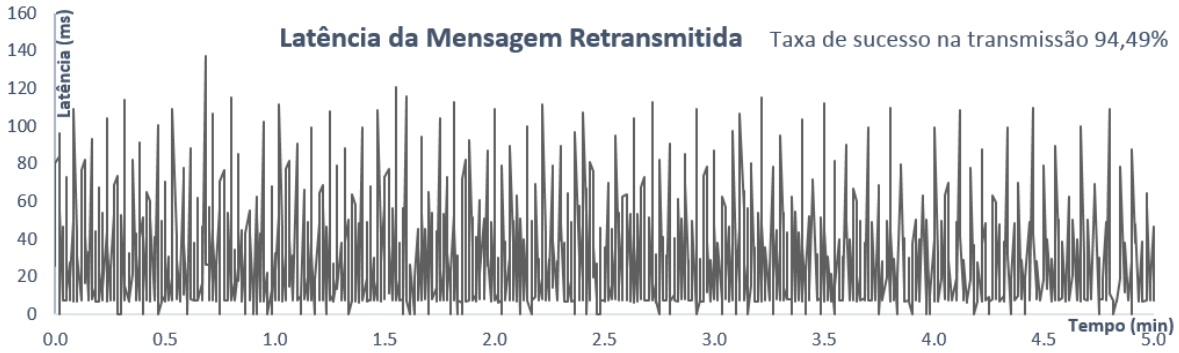


Figura 22 – Latência de uma mensagem retransmitida entre clusters – Fonte: Autor

7.5 Ensaio com o uso de componentes reais

Para testar o comportamento da rede em uma situação real, bem como a funcionalidade dos comandos de escrita e leitura de variáveis, foi montada sobre a rede uma pequena aplicação de automação residencial. Esta aplicação era composta por três diferentes nós escravos espalhados a uma distância de aproximadamente 5 metros de um nó mestre central. As mensagens eram trocadas com um tempo de ciclo de 300 ms.

Um dos nós era responsável pela leitura da temperatura e umidade do ambiente, provindas de um sensor DHT11, salvas em duas variáveis globais de 8 bits. Outro nó executava a leitura de um teclado numérico, armazenando a última tecla. O terceiro escravo possuía acoplado um display LCD de 16x2 linhas, que mostrava alternadamente o valor da temperatura ou da umidade do ambiente, de acordo com uma variável booleana enviada pelo nó mestre. Este último era encarregado de enviar comandos de leitura e escrita para todos os componentes e exportar os dados para um computador por meio de uma comunicação serial. O mestre também decidia o que seria mostrado no display LCD, de acordo com uma lógica baseada na última tecla lida pelo teclado numérico.

O sistema aparentemente funcionou corretamente, porém notou-se uma perda significativa da taxa de sucesso de transmissão. Para o escravo com o sensor a taxa foi de 96,51%, para o nó com teclado foi de 89,46% e para o display 80,03%. Este efeito explica-se pela interferência de funções das bibliotecas necessárias para acionar cada um destes componentes. Como cada biblioteca é desenvolvida separadamente e como todas elas compartilham um número limitado de recursos do processador, ao juntar-se várias dessas bibliotecas em uma mesma aplicação é quase impossível prever os conflitos entre as funções de cada uma delas. Para tanto, recomenda-se o uso de um segundo microprocessador, acoplado por uma ligação serial ou de outro barramento, exclusivo para o acionamento e leitura de sensores ou atuadores.

8 Conclusões e Trabalhos Futuros

Em um cenário onde a automação residencial ainda é vista como um artigo de luxo para muitos, devido ao seu alto custo, a proposta apresentada neste projeto mostra que o estado tecnológico atual já permite o desenvolvimento de sistemas que utilizem dispositivos de baixo custo, mas que atendam aos requisitos das aplicações. Neste contexto, o sistema proposto, apesar de suas limitações em termos de complexidade de funções, mostrou-se capaz de atuar de forma a suprir diversas necessidades presentes no ambiente de automação residencial de pequena escala.

A expansão das funcionalidades originais do transceptor nRF24L01+, permitiu o desenvolvimento de uma estrutura de rede mais complexa e adaptada a aplicações de automação. O sistema mostrou-se flexível de modo a acomodar troca de dados com tempo determinístico, e também aceitar mensagens vindas de maneira aleatória. A implementação de diferentes topologias permitiu a expansão da capacidade de alcance de sinal para diversos ambientes. Enquanto isso, a separação hierárquica dos nós em mestres e escravos e a criação de clusters, associada a estratégias de contenção de falhas, foram capazes de proporcionar resultados satisfatórios em termos de sucesso nas transmissões, quando comparados com estudos previamente realizados com o uso deste mesmo dispositivo.

Apesar de ser baseado em um protocolo considerado não padrão, o seu desenvolvimento em uma plataforma de compartilhamento de software livre, como é a Arduino, permite uma interligação deste trabalho com demais projetos futuros. Os resultados obtidos neste trabalho mostram um sistema promissor. Porém, como projeto, ele ainda não se encontra totalmente pronto para um possível uso comercial. O código desenvolvido ainda necessitaria de um maior amadurecimento, em termos de uma busca por possíveis erros e por uma maior eficiência de processamento e uso de memória. Funcionalidades também poderiam ser adicionadas, como o reconhecimento dinâmico das mensagens recebidas, sem a necessidade de reservar um tempo para esperar pela resposta das requisições enviadas. Adicionalmente, o projeto ainda carece de um número maior de testes com o uso de componentes reais, de forma a otimizar suas funções para uma gama maior de aplicações. Estas tarefas ficam destinadas a um possível trabalho futuro.

9 Referências

ALIEXPRESS. **aliexpress.com** 2014. Disponível em: < http://www.aliexpress.com >. Acesso em: 27 de maio de 2014.

AMS. Adjustable/Fixed Low Dropout Linear Regulator. 2014. Disponível em: < http://www.ams-semitech.com/attachments/File/AMS1117_20120314.pdf >. Acesso em: 27 de maio de 2014.

ARDUINO. Arduino Pro Mini. 2014. Disponível em: < http://arduino.cc/en/Main/ArduinoBoardProMini >. Acesso em: 27 de maio de 2014.

BROOKS, D. R. **C Programming: The Essentials for Engineering and Scientists**. Springer-Verlag New York, Inc., 1999. 496 ISBN 0387986324.

CHRIST, P. et al. Performance analysis of the nRF24L01 ultra-low-power transceiver in a multi-transmitter and multi-receiver scenario. Sensors, 2011 IEEE, 2011, 28-31 Oct. 2011. p.1205-1208.

COLIZ, J. Driver for nRF24L01(+) 2.4GHz Wireless Transceiver. 2012. Disponível em: < http://maniacbug.github.io/RF24/ >. Acesso em: 21/05/2014.

DAY, J. D.; ZIMMERMANN, H. The OSI reference model. **Proceedings of the IEEE,** v. 71, n. 12, p. 1334-1340, 1983. ISSN 0018-9219.

DIAS, C. L. D. A.; PIZZOLATO, N. D. DOMÓTICA: Aplicabilidade e Sistemas de Automação Residencial. **Vértices,** v. V. 6, n. 3, n. set/dez 2004, 2004. ISSN 1809-2667.

GERRIOR, M.; WOODINGS, R. W. Avoiding Interference in the 2.4-GHz ISM Band. 2006. Disponível em: < http://www.eetimes.com/document.asp?doc_id=1273359&page_number=1 >. Acesso em: 22/05/2014.

NANHAO, Z. et al. High Data Rate Wireless Sensor Networks Research. 2008.

NORDIC. **nRF24L01 Product Specification**. SEMICONDUCTOR, N.**:** 74 p. 2007.

Redes da Organização Profibus. Mecatrônica Atual 2004. Disponível em: < http://www.mecatronicaatual.com.br/educacao/1156-redes-da-organizao-profibus >. Acesso em: em: 22/05/2014.

URDIAIN, L. O. et al. **Wireless Sensor Network Protocol for Smart Parking Application Experimental Study on the Arduino Platform**. AMBIENT 2012: The Second International Conference on Ambient Computing, Applications, Services and Technologies. THINKMIND. Barcelona, Spain**:** 45 a 48 p. 2012.

10 Anexos

Nos seguintes anexos são apresentadas as principais funções de programação implementadas no microcontrolador Arduino durante o desenvolvimento deste trabalho.

10.1 Definições da Estrutura da Mensagem

```
//DEFINIÇÃO DE VALORES USADOS PARA CONFIGURAÇÕES
#define TYPE1 32
#define TYPE2 64
#define TYPE3 96
#define READ false
#define WRITE true

//ENUMERAÇÃO DE FLAGS
enum flag_enum{
  MAP, ANSWER, RET,DATA
};
typedef enum flag_enum flag_t;

//DEFINIÇÃO DO TIPO DA VARIÁVEL DE 8-BITS DA MENSAGEM
typedef union
{
bool BOOL;
uint8_t UINT8;
int8_t INT8;
} data0_t;

//DEFINIÇÃO DO TIPO DA VARIÁVEL DE 8-BITS DA MENSAGEM
typedef union
{
int32_t INT32;
uint32_t UINT32;
float FLOAT;
} data1_t; //4BYTES

//ESTRUTURA DA MENSAGEM
struct message_t {
uint8_t sourceID; //INDENTIFICAÇÃO DO REMETENTE
  uint8_t destID; // IDENTIFICAÇÃO DO DESTINATÁRIO
  uint8_t sourceCH; //CANAL DO REMETENTE
  uint8_t destCH; //CANAL DO DESTINATÁRIO
  uint8_t flags;//FLAGS
  uint8_t dcmd[6]; //COMANDOS ASSOCIADOS ÀS VARIÁVEIS DE 8-BITS
  data0_t dvar[6]; //VARIÁVEIS DE 8-BITS
  uint8_t acmd[3];//COMANDOS ASSOCIADOS ÀS VARIÁVEIS DE 32-BITS
  data1_t avar[3];//VARIÁVEIS DE 32-BITS
};
typedef struct message_t message_type; //DEFINIÇÃO DA VARIÁVEL DO TIPO MENSAGEM
```

10.2 Função que Escuta por Ruídos no Canal

```
bool listen(void)
 {
 radio.startListening(); //RADIO CONFIGURADO COMO RECEPTOR
 unsigned long started_listen = micros();//TEMPO INICIAL
 unsigned long time = started_listen;
 bool ok = false;
 bool air;
```

```
  unsigned long rand = random(100,800); //GERA UM VALOR ALEATÓRIO ENTRE 100 E 800 ms
  while (((micros () - time) < rand) | ((micros() - started_listen) < 4000))//EXECUTA CICLO
DURANTE TEMPO GERADO OU TIMEOUT
  {
   air=radio.testCarrier();//ESCUTA O CANAL
   if (air)//CASO ENCONTRE ALGO
   {
    rand = random(100,800); //GERA OUTRO NÚMERO ALEATÓRIO
    time = micros ();  //REINICIA A CONTAGEM
    ok = false; //
   }
   else
    ok = true;
  }
   radio.stopListening();//RÁDIO DEFINIDO COMO TRANSMISSOR
   return ok; //RETORNA ESTADO DO CANAL
}
```

10.3 Função que Checa o Recebimento de Mensagens

```
void check_radio()
{
if (radio.available())//CASO O RÁDIO TENHA RECEBIDO ALGUMA MENSAGEM
 {
  message_type rx;//DECLARAÇÃO DE VARIÁVEL DO TIPO MENSAGEM
  uint8_t siz = radio.getDynamicPayloadSize();//LÊ O TAMANHO DA MENSAGEM RECEBIDA
  radio.read(&rx, siz);//LÊ A MENSAGEM
  interpret_message (siz,rx);//INTERPRETA O CONTEÚDO DA MENSAGEM
  }
}
```

10.4 Função que Escreve a Mensagem e Aguarda pela Resposta

```
struct message_t write_and_ack (uint8_t siz , uint8_t time, struct message_t msg)
{
 bool ok = write_ch(siz, msg); //ESCREVE MENSAGEM
  if(ok)//CASO TRANSMISSÃO TENHA SUCESSO
  {
  radio.setChannel(MYCHANNEL);//DEFINE NOVAMENTE O RÁDIO NO SEU CANAL
  radio.startListening();//RECEPTOR
  unsigned long started_waiting_at = millis();
  bool timeout = false;
  while ( ! radio.available() && ! timeout )//ESPERA PELA RESPOSTA
    if (millis() - started_waiting_at > time )
      timeout = true;
    if ( timeout )
    Serial.print("FAIL");
    else
  {
    uint8_t siz = radio.getDynamicPayloadSize();
    radio.read( &msg, siz);//LÊ MENSAGEM RECEBIDA
   }
  }
   radio.stopListening();//TRANSMISOR
   return msg;//RETORNA MENSAGEM RECEBIDA
 }
```

10.5 Função que Escreve a Mensagem

```
bool write_ch(uint8_t siz, struct message_t msge)
 {
   bitWrite(msge.flags,DATA,true); //ATIVA FLAG DE TROCA DE DADOS
   radio.setChannel(msge.destCH); //DEFINE CANAL DE DESTINO
   radio.openWritingPipe(BASEADDRESS+msge.destID); //DEFINE ID DE DESTINO
   bool air = listen(); //ESCUTA CANAL ANTES DE TRANSMITIR
    if (!air)//CANAL SE CONDIÇÕES DE TRANSMISSÃO
    {
      radio.setChannel(MYCHANNEL);
      return false;//MENSAGEM NÃO ENVIADA
    }
    else
    {
     if (siz==0)//DEFINE TAMANHO DA MENSAGEM
        siz=11;
      else
        {
         if (siz==1) //MENSAGEM SOMENTE COM VAR DE 8-BITS
            siz=17;
         else
            siz=32; //MENSAGEM COMPLETA
        }
    radio.writeFast(&msge,siz);//ESCREVE MENSAGEM
    bool ok = radio.txStandBy();
    return ok;
   }
 }
```

10.6 Função que Escreve um Comando

```
uint8_t write_command(bool flow, uint8_t type, uint8_t tag)
  {
    uint8_t command=type; //TIPO DA VARIÁVEL - TYPE1-BOOL/FLOAT  TYPE2-UINT8/UINT32
TYPE3-INT8/INT32
    bitWrite(command,7,flow);//LEITURA OU ESCRITA
    command = command + tag;//TAG DA VARIÁVEL GLOBAL DO RECEPTOR
    return command;
  }
```

10.7 Função que inicializa e Rotula Mensagem a ser Enviada

```
 void lable_message(uint8_t mCH, uint8_t mID, uint8_t dCH, uint8_t dID, struct message_t *msg)
  {
  msg->sourceID=mID;
  msg->destID=dID;
  msg->sourceCH=mCH;
  msg->destCH=dCH;
  msg->flags=0;
  for (uint8_t i=0; i<6; i++)//ZERA VARÍAVEIS DE 8-BITS
  {
    msg->dcmd[i]=0;
    msg->dvar[i].UINT8=0;
  }
```

```
  for (uint8_t i=0; i<3; i++)//ZERA VARIÁVEIS DE 32-BITS
  {
    msg->acmd[i]=0;
    msg->avar[i].UINT32=0;
  }
 }
```

10.8 Função que Interpreta Mensagem Recebida

```
void interpret_message (uint8_t siz, struct message_t msg)
{

  if ((msg.destID == MYID) & (msg.destCH==MYCHANNEL))//MSG É PARA MIM?
  {
   message_type tx;//PREPARA MSG DE RESPOSTA
  lable_message(MYCHANNEL, MYID, msg.sourceCH, msg.sourceID, &tx);
  if (bitRead(msg.flags,DATA))//CASO MSG FOR DE TROCA DE DADOS
  {
    for (int8_t i=0; i<6; i++)//LÊ COMANDOS DE VAR DE 8-BITS
    {
    uint8_t type = msg.dcmd[i] & 112;
    type = type >> 5;
    if (type != 0) //CASO TIPO DA VAR FOR 0 O COMANDO É CONSIDERADO VAZIO
      {
       uint8_t tag = msg.dcmd[i] & 15;//LÊ TAG DA MSG
      if (bitRead(msg.dcmd[i],7)) // RECEBE VAR ENVIADAS
        {
         switch (type){
         case(1):
         var_bool[tag]=msg.dvar[i].BOOL;//ACESSA VAR GLOBAL REQUERIDA
         break;
         case(2):
         var_uint8[tag]=msg.dvar[i].UINT8;//ACESSA VAR GLOBAL REQUERIDA
         break;
         case(3):
         var_int8[tag]=msg.dvar[i].INT8;//ACESSA VAR GLOBAL REQUERIDA
         break;
         default:
         break;
         }
        }
       else// RESPONDE VAR REQUERIDAS
       {
           switch (type){//ACESSA VAR GLOBAL REQUERIDA
           case(1):
           tx.dvar[i].BOOL=var_bool[tag];
           break;
           case(2):
           tx.dvar[i].UINT8=var_uint8[tag];
           break;
           case(3):
           tx.dvar[i].INT8=var_int8[tag];
           break;
           default:
           break;
           }
       }
      }
    }
```

```
  if(siz>17) //TESTA SE HÁ VARIÁVEIS OU COMANDOS DE 32 BITS NA MENSAGEM
  {
    for (int8_t i=0; i<3; i++)//LÊ COMANDOS DE VAR DE 32-BITS
    {
      uint8_t type = msg.acmd[i] & 112;
      type = type >> 5;
      if (type != 0)
      {
       uint8_t tag = msg.acmd[i] & 15;
       if (bitRead(msg.acmd[i],7))// RESPONDE REQUISIÇÃO
       {
        switch (type){//RECEBE VAR ENVIADAS
        case(1):
        var_float[tag]=msg.avar[i].FLOAT;
        break;
        case(2):
        var_uint32[tag]=msg.avar[i].UINT32;
        break;
        case(3):
        var_int32[tag]=msg.avar[i].INT32;
        break;
        }
       }
       else
       {
         switch (type){//ENVIA VARIAVEIS REQUERIDAS
         case(1):
         tx.avar[i].FLOAT=var_float[tag];
         break;
         case(2):
         tx.avar[i].UINT32=var_uint32[tag];
         break;
         case(3):
         tx.avar[i].INT32=var_int32[tag];
         break;
         default:
         Serial.print("  NOT TYPE ");
         Serial.print(type);
         break;

        }
       }
      }
    }
  }
  }

  bitWrite(tx.flags,ANSWER,true);//FLAG DE RESPOSTA
  bool ok = write_ch(3, tx); //ENVIA RESPOSTA
 radio.startListening(); //RECEPTOR


 if (bitRead(msg.flags,MAP))//CASO MSG SEJA DE ATUALIZAÇÃO DE MAPA DE ALCANVE
 {
    bool flag = bitRead(msg.flags,ANSWER);
  if(!(bitRead(msg.flags,ANSWER)))
  {
    message_type tx;
    lable_message(MYCHANNEL, MYID, msg.sourceCH, msg.sourceID, &tx);
    bitWrite(channels[MYCHANNEL/2],(msg.sourceCH)/2, true); //registra canal na lista de
alcançaveis
    bitWrite(tx.avar[0].INT32, (msg.sourceCH)/2, true); //atualiza campo de mensagem
```

```
    tx.destCH = msg.sourceCH;  //destination channel
    tx.destID = 0x00; //destination ID
    bitWrite(tx.flags,MAP,true);
    bitWrite(tx.flags,ANSWER,true);
    bool ok = write_ch(3, tx);
    channels[(msg.sourceCH)/2] = msg.avar[0].INT32;//ATUALIZA MAPA DE ALCANCE
   for (int i=0; i<64; i++)
    {
     if (bitRead(channels[MYCHANNEL/2],i) > bitRead(msg.sourceCH,i))
      {
        tx.destCH = i*2;  //
        bool ok = write_ch(3, tx);
      }
    }
    //LIMPA FLAGS
    bitWrite(tx.flags,MAP,false);
    bitWrite(tx.flags,ANSWER,false);
   }
  }
}
else
{
  if (bitRead(msg.flags,RET)) //CASO MENSAGEM SEJA PARA RETRANSMISSÃO
  {
   uint8_t dest_id = msg.sourceID;//SALVA DADOS DO REMETENTE
   uint8_t dest_ch = msg.sourceCH;
   message_t tx = msg;
   lable_message(0,0,0,0,&msg); //LIMPA VARIAVÉL DE RECEBIMENTO
   tx.sourceID = MYID;//SE COLOCA COMO REMETENDE DA MSG RETRANSMITIDA
   tx.sourceCH = MYCHANNEL;
   bitWrite(tx.flags,RET,false); //DESATIVA FLAG DE RETRANSMISSÃO
   msg = write_and_ack (3, 60, tx); //RETRANSMITE MSG PARA DEST FINAL
      if ((msg.destID == MYID) & (msg.destCH==MYCHANNEL))//CASO RESPOSTA TENHA
CHEGADO
    {
      tx = msg;
      tx.destID = dest_id;//DEFINE DESTINATARIO
      tx.destCH = dest_ch;
      bitWrite(tx.flags,ANSWER,true); //DESATIVA FLAG DE RETRANSMISSAÕ
      bool ok = write_ch(3, tx);//ENVIA RESPOSTA
    }
     radio.startListening();//RECEPTOR
  }
 }
}
```

10.9 Função que Envia Mapa de Rede

```
 void send_map ()
 {
  //envia sua lista de dispositivos ao alcance para todos os dispositivos encontrados
    bitWrite(tx.flags,MAP,true);
    for (int i=0; i<64; i++)
   {
     if (bitRead(channels[MYCHANNEL/2],i))//envia somente para canais já mapeados
     {
     bool air = listen();
      if (!air)
      { //canal não está em condições de comunicação
```

```
      bitWrite(channels[MYCHANNEL/2], i, false); //elimina canal da lista
      bitWrite(tx.var3.INT64, i, false);
    }
    else
    {
    //envia mensagem e aguarda por resposta
     bool ok = radio.write( &tx, sizeof(message_t));
     if (!ok) //se não obtiver resposta
     bitWrite(channels[MYCHANNEL/2], i, false); //elimina canal da lista
     bitWrite(tx.var3.INT64, i, false);
    }
  }
  bitWrite(tx.flags,MAP,false); //desativa flag de envio de mapa de dispositivos
 }
}
```

10.10 Função que Mapeia Dispositivos ao Alcance

```
void channel_search ()
{
 //aumenta o número de tentativas de retransmissão e o intervalo entre eleas
 radio.setRetries(30,10);
 //escreve somente para os dispositivos mestres
 radio.openWritingPipe(BASEADDRESS);
 // tenta comunicação com todos os canais possíveis
 for (int i=0; i<64; i++)
 {
  //desloca um bit para a esquerda
  radio.setChannel(i*2);
  //escuta o canal antes de transmitir para não atrapalhar uma possível comunicação em curso
  bool air = listen();
  if (!air)
  {
    //canal não está em condições de comunicação
    bitWrite(channels[MYCHANNEL/2], i, false);
    }
  else
  {
  //envia mensagem e aguarda por ACK
  bool ok = radio.write( &tx, sizeof(message_t));
  if (ok)
    //dispositivo reconhecido
    bitWrite(channels[MYCHANNEL/2], i, true);
  else
    //canal não sendo usado
    bitWrite(channels[MYCHANNEL/2], i, false);
  }
  //imprime informações para debug
  Serial.print(i*2);
  Serial.print("-");
  bool a = bitRead(channels[MYCHANNEL/2],i);
  Serial.print(a);
  Serial.print("  ");
  }
  //salva o mapa de dispositivos ao alcance no seu respectivo campo da estrutura de mensagem
   tx.var3.INT64 = channels[MYCHANNEL/2];
   send_map(); //envia sua lista de dispositivos ao alcance para todos os dispositivos
 }
```