PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE TECNOLOGIA

LUCAS AUGUSTO DE ARAUJO MARQUES LEÃO

PROPOSTA DE UM ALGORITMO DE ROTEAMENTO BASEADO EM LÓGICA DIFUSA PARA RSSF EM AMBIENTES FECHADOS

PUC-Campinas
2015

LUCAS AUGUSTO DE ARAUJO MARQUES LEÃO

PROPOSTA DE UM ALGORITMO DE ROTEAMENTO BASEADO EM LÓGICA DIFUSA PARA RSSF EM AMBIENTES FECHADOS

Dissertação apresentada como exigência para obtenção do Título de Mestre em Engenharia Elétrica, ao Programa de Pós-graduação em Gestão de Redes de Telecomunicações, do Centro de Ciências Exatas, Ambientais e de Tecnologia da Pontifícia Universidade Católica de Campinas.

Orientador: Prof. Dr. David Bianchini

PUC-Campinas
2015

Ficha Catalográfica
Elaborada pelo Sistema de Bibliotecas e
Informação - SBI - PUC-Campinas

t621.3851
L437p

Leão, Lucas Augusto de Araujo Marques.
Proposta de um algoritmo de roteamento baseado em lógica difusa para RSSF em ambientes fechados / Lucas Augusto de Araujo Marques Leão. - Campinas: PUC-Campinas, 2015.
157p.

Orientador: David Bianchini.
Dissertação (mestrado) - Pontifícia Universidade Católica de Campinas, Centro de Ciências Exatas, Ambientais e de Tecnologias, Pós-Graduação em Engenharia Elétrica.
Inclui bibliografia.

1. Redes de sensores sem fio. 2. Algoritmos de computador. 3. Lógica difusa. 4. Sistemas de comunicação sem fio. I. Bianchini, David. II. Pontifícia Universidade Católica de Campinas. Centro de Ciências Exatas, Ambientais e de Tecnologias. Pós-Graduação em Engenharia Elétrica. III. Título.

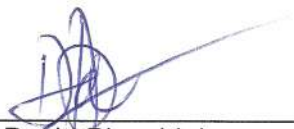
22.ed.CDD - t621.3851

LUCAS AUGUSTO DE ARAÚJO MARQUES LEÃO

PROPOSTA DE UM ALGORITMO DE ROTEAMENTO BASEADO EM LÓGICA DIFUSA PARA RSSF EM AMBIENTES FECHADOS

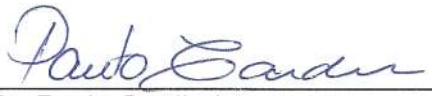
Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro de Ciências Exatas, Ambientais e de Tecnologias da Pontifícia Universidade Católica de Campinas como requisito parcial para obtenção do título de Mestre em Gestão de Redes de Telecomunicações.
Área de Concentração: Qualidade de Serviço de Teleinformática. Orientador: Prof. Dr. David Bianchini

Dissertação defendida e aprovada em 12 de maio de 2015 pela Comissão Examinadora constituída dos seguintes professores:



Prof. Dr. David Bianchini
Orientador da Dissertação e Presidente da Comissão Examinadora
Pontifícia Universidade Católica de Campinas

Prof. Dr. Omar Carvalho Branquinho
Pontifícia Universidade Católica de Campinas



Prof. Dr. Paulo Cardieri
Universidade Estadual de Campinas

Aos meus pais, aos meus irmãos e aos meus amigos, por todo apoio e paciência.

Agradecimentos

Aos meus queridos colegas de turma, que durante esta jornada deram o suporte e apoio para a realização desse trabalho. Pelos momentos divertidos, pelos momentos de tensão que passamos e pelas conquistas que obtivemos.

Aos meus caríssimos professores, que com paciência transmitiram o conhecimento e nos guiaram durante esse processo de crescimento. Em especial ao meu orientador Prof. Dr. David Bianchini pela paciência e ao meu sempre apoiador Prof. Dr. Omar Branquinho pela dedicação incondicional.

À Pontifícia Universidade Católica de Campinas pela Bolsa parcial concedida, viabilizando a realização desse curso.

Ao SRBR pelas horas que pude dedicar a este trabalho e que fizeram a diferença para a conclusão desse projeto.

Ao Vinicius, ao Gustavo, ao Kevin e ao Vitor pela inestimável ajuda durante a realização dos experimentos.

Aos meus estimados amigos, sempre presentes em minha vida, apoiando e incentivando todos meus sonhos e me dando forças para seguir em frente e superar cada obstáculo. Em especial ao Maurício pelos incentivos extras, à Ivelize e Vânia pelas valiosas dicas e à Lilian, Beatriz, Edgar e André pelo bom dia de cada dia.

Por fim, aos meus pais, a quem devo a vida e aos meus irmãos, que me ajudaram a ser quem sou. Sem vocês eu não existiria, sem vocês eu não teria chegado até aqui. Vocês são o meu porto seguro e a minha fonte de energia. Todas as dificuldades e desafios que já vivemos são a prova da força que temos juntos como família.

(...) o que dá o verdadeiro sentido ao encontro
é a busca e é preciso andar muito para se
alcançar o que está perto.

Todos os Nomes - José Saramago
(1922 - 2010)

Resumo

As Redes de Sensores Sem Fio (RSSF) têm sido uma solução amplamente utilizada no contexto de sistemas de gerenciamento de edificações. Os sensores são responsáveis por monitorar diversos aspectos do ambiente, como temperatura, iluminação e consumo de energia. Entretanto, os sensores estão expostos a condições adversas e mudanças constantes do ambiente, que podem afetar de maneira definitiva a comunicação e fluência dos dados. Neste sentido, este trabalho apresenta uma proposta de algoritmo de roteamento baseado em lógica difusa para identificação dos melhores caminhos em uma rede de sensores sem fio *indoor*. São apresentados os parâmetros utilizados (RSSI, Desvio Padrão do RSSI e Taxa de Erro de Pacote) para a definição do custo de cada caminho, a sequência de identificação de melhor caminho e os resultados obtidos em simulação e aplicação prática. A solução proposta agrega técnicas de roteamento em RSSF à utilização de lógica difusa para caracterização e definição dos custos dos enlaces entre os sensores. O algoritmo desenvolvido foi confrontado com uma solução de roteamento baseada em RSSI. Os experimentos demonstram que a solução permite a seleção de enlaces de melhor qualidade, reduzindo a probabilidade de perda de pacote em comparação ao algoritmo baseado apenas em RSSI.

Palavras-chave: RSSF, Redes de Sensores Sem Fio, Roteamento, Crosslayer, Lógica Difusa, RSSI, PER

Abstract

Wireless Sensor Networks (WNS) have been applied as monitoring solution for building management systems. The sensors are responsible for monitoring environment aspects such as temperature, lighting and energy consumption. However, the sensors are exposed to adverse conditions and frequent environment changes, which can dramatically affect communication and data flow. Thus, this work proposes a routing algorithm based on fuzzy logic to identify the best routes in an indoor wireless sensor network. The evaluated parameters are presented (RSSI, Standard Deviation and Packet Error Rate) along with the cost definition process for each route, the best route identification sequence and the results obtained in simulation and experimentation. The proposed solution mixes WSN routing techniques along with fuzzy logic to characterize and define the link cost. The developed algorithm was faced with a routing solution based on RSSI. The experiments demonstrate that the solution allows the selection of higher quality links, reducing the probability of packet loss in comparison to the algorithm based on RSSI.

Keywords: WNS, Wireless Sensor Networks, Routing, Crosslayer, Fuzzy Logic, RSSI, PER

Lista de Figuras

Lista de Tabelas

Lista de Equações

Lista de Abreviaturas e Siglas

BMS	-	Building Management Systems
CHEF	-	Cluster Head Election mechanism using Fuzzy logic
EB	-	Estação Base
FLBRA	-	Fuzzy Logic Based Routing Algorithm
GSM	-	Sistema Global para Comunicações Móveis
LEACH	-	Low Energy Adaptive Clustering Hierarchy
LEACH-C	-	Low Energy Adaptive Clustering Hierarchy Centralized
MANET	-	Mobile Ad Hoc Network
NS	-	Nó Sensor
PER	-	Packet Error Rate
RBF	-	RSSI-Based Forward
RFID	-	Radio-Frequency Identification
RSSF	-	Redes de Sensores Sem Fio
RSSI	-	Received Signal Strength Indicator
RSSR	-	Received Signal Strength Routing
SG	-	Software Gerente
SNR	-	Signal to Noise Ratio
TARF	-	Trust-Aware Routing Framework
WSN	-	Wireless Sensor Network

Sumário

1 Introdução

A literatura é rica em apresentar definições para Redes de Sensores Sem Fio (RSSF). As caracterizações mais recorrentes apontam para uma descrição conjunta de *hardware* e aplicação, sendo a RSSF definida como um agrupamento de sensores com recursos limitados de processamento, energia e alcance de rádio, distribuídos em uma rede sem fio ad-hoc, a fim de colaborar com uma ou mais aplicações de monitoramento (AL-KARAKI, 2004; KULKARNI, 2006; SUH, 2006; ORTIZ, 2012).

Devido à grande flexibilidade das RSSF, que permite a implantação de redes em diversos ambientes, desde florestas, desertos, áreas de conflito, até a ambientes fechados, como fábricas, escritórios e residências, as RSSF têm sido uma solução poderosa para o monitoramento de diversos aspectos do ambiente. Uma das aplicações atualmente pesquisadas é a do monitoramento de ambientes fechados, como nos sistemas de gerenciamento de edificações (BMS – *Building Management Systems*). Os sensores são responsáveis pelo monitoramento de aspectos do ambiente, capturando dados como temperatura, iluminação e consumo de energia (DE GUGLIELMO, 2012). Por meio desse monitoramento, estratégias de gerenciamento e controle da iluminação e climatização são aplicadas buscando maior eficiência na utilização dos recursos.

Aplicações de RSSF em ambientes dinâmicos, tais como escritórios e fábricas, estão sujeitas a constantes alterações físicas, sejam elas pela inclusão de novos obstáculos, como móveis e divisórias, bem como pela movimentação normal de pessoas ao longo do dia. Desta forma, o projeto da rede sem fio deve levar em consideração a possibilidade de constantes reconfigurações da rede. Essas reconfigurações são necessárias uma vez que as mudanças no ambiente geram impactos no desenho inicial da RSSF, que podem comprometer substancialmente a comunicação entre os sensores (ASLAM, 2011).

Neste cenário, identifica-se a importância de uma solução que atenda as demandas e características específicas de uma aplicação de RSSF em ambientes fechados. Para tanto, a fim de minimizar os impactos das alterações no ambiente, pensou-se na elaboração de uma estratégia de reconfiguração da rede, alterando

dinamicamente as rotas de comunicação entre os sensores, objetivando garantir a comunicação contínua entre os sensores e a base.

O algoritmo desenvolvido utiliza conceitos de Lógica Difusa para realizar a reconfiguração da rede, reconstruindo as tabelas de roteamento de acordo com o estado da RSSF. Por meio da avaliação de parâmetros de qualidade do sinal, atribuem-se pesos para cada enlace entre os nós sensores, seus vizinhos e a base. Uma vez definido o peso, avaliam-se os caminhos cuja rota final possua o menor peso resultante. A avaliação das possíveis rotas ocorre por meio da execução de um algoritmo de busca de melhor caminho. Dessa forma, escolhem-se os enlaces com as melhores características de qualidade de sinal. O peso de cada enlace é calculado através de regras definidas e que consideram os seguintes parâmetros: RSSI (*Received Signal Strength Indicator*), Desvio Padrão do RSSI e Taxa de Erro de Pacote (*Packet Error Rate* - PER). A utilização de tais parâmetros possibilita identificação dos efeitos das variações físicas no ambiente. A avaliação do desvio padrão, por exemplo, permite detectar a estabilidade de um dado sinal. A taxa de erro de pacote pode ser considerada como um parâmetro definitivo de qualidade, já que traduz a quantidade efetiva de informação entregue à base. Por fim, as medições RSSI podem revelar a possível existência de obstáculos ou linhas de visada, dependendo da intensidade do sinal.Este trabalho tem como foco desenvolver um protocolo de roteamento alternativo, utilizando Lógica Difusa, para aplicações de RSSF em ambientes fechados, a fim de verificar se é possível melhorar o desempenho da rede em que o ambiente esteja sujeito a uma variação frequente das características físicas. Sendo assim, buscando avaliar a aplicabilidade e efetividade da proposta, foram implementadas simulações e experimentos práticos que, por meio dos resultados, permitiram identificar ganhos consideráveis em comparação a outro protocolo adaptativo baseado somente em RSSI.

1.1 Objetivos e Contribuições

Embora se tenha como objetivo principal a proposta de um algoritmo de roteamento que permita a manutenção da comunicação da rede, mesmo quando

suscetível às inconstâncias de ambientes fechados dinâmicos, são também objetivos importantes: a definição de uma solução de complexidade reduzida e de alta aplicabilidade, a diminuição da taxa de perda de pacotes e a diminuição do número médio de saltos até que a informação de um nó sensor seja recebida pela base.

Neste contexto, este trabalho apresenta uma abordagem de baixa complexidade e objetiva, tal como protocolos de encaminhamento de pacote baseados em RSSI, contudo, permitindo uma maior assertividade na escolha das rotas, por meio do uso da taxa de erro de pacote e do desvio padrão das medições de RSSI como parâmetros para decisão de roteamento. O algoritmo proposto também se diferencia de outros protocolos existentes por utilizar Lógica Difusa para realizar o processo de avaliação de qualidade do enlace.

As contribuições deste trabalho podem ser sumarizadas em:

- Desenvolvimento de um novo protocolo de roteamento para aplicações de redes de sensores sem fio em ambientes fechados, junto com a avaliação dos resultados dos experimentos práticos, que contribuem para o desenvolvimento de aplicações reais de RSSF nas condições propostas pelo trabalho;
- Utilização de Lógica Difusa para avaliação da qualidade dos enlaces a fim de estabelecer um método de reconfiguração da rede a partir da situação da RSSF.

1.2 Organização do Trabalho

Este documento está organizado de forma a esclarecer e apresentar todas as etapas de pesquisa e desenvolvimento do projeto. Os tópicos abordados tratam da base teórica, ferramentas e processos utilizados para a elaboração e implementação do algoritmo proposto. A dissertação é composta pelos seguintes capítulos:

Capítulo 2: Apresenta conceitos sobre Lógica Difusa a fim de facilitar a compreensão dos métodos utilizados para avaliação dos enlaces;

Capítulo 3: Dedicado a uma breve contextualização sobre redes de sensores sem fio;

Capítulo 4: Trata do roteamento em redes de sensores sem fio, apresentando estratégias baseadas em RSSI e protocolos baseados em Lógica Difusa;

Capítulo 5: Apresenta a proposta do algoritmo de roteamento, detalhando o processo de avaliação das rotas e as etapas de funcionamento;

Capítulo 6: Descreve as atividades de implementação e execução dos testes de simulação e dos experimentos práticos;

Capítulo 7: Discute sobre os resultados obtidos, comparando o algoritmo proposto com o um protocolo baseado em RSSI;

Capítulo 8: Trata das considerações finais, apresentando uma conclusão do trabalho e listando perspectivas futuras.

2 Lógica Difusa

A lógica tradicional aristotélica, baseada em premissas e conclusões, não consegue representar fenômenos naturalmente imprecisos (ZADEH, 2005), pois trata as proposições como absolutamente verdadeiras ou absolutamente falsas, não havendo margem para o pensamento abstrato. Isso significa dizer que a lógica tradicional apenas trata de dados precisos e absolutos. Pode-se explicar a afirmação acima com o seguinte exemplo:

Premissa 1: Redes sem fio são vulneráveis.

Premissa 2: GSM é um sistema de rede sem fio.

Conclusão: Portanto, GSM é vulnerável.

Observa-se no exemplo acima que a lógica tradicional não admite imprecisão de dados e ideias vagas, pois leva a falsas conclusões. GSM é de fato um sistema de rede de transmissão sem fio, porém, não é um sistema totalmente vulnerável, uma vez que possui mecanismos de segurança (BROOKSON, 1994). Verifica-se que a “Premissa 1” está fundamentada em uma informação vaga, baseada em um histórico generalista e que portanto exprime imprecisão.

Como alternativa ao tratamento de problemas com dados imprecisos ou baseados em informações incompletas, podem ser citadas as técnicas de *Soft Computing*. Segundo Zadeh (1994), *Soft Computing* busca trabalhar as incertezas e imprecisões inerentes ao mundo real a fim de possibilitar a obtenção de resultados acurados. Entre as técnicas de Soft Computing, encontra-se a Lógica Difusa (ou Lógica Nebulosa), que visa abstrair conceitos concretos a fim de eliminar incertezas, avaliando as proposições de acordo com regras baseadas em experiência. Zadeh (1994) apresenta como objetivo da Lógica Difusa o tratamento de dados imprecisos, na tentativa de aproximação do pensamento e raciocínio humano. Essa definição permite comparar a capacidade humana de tomar decisões assertivas através de dados imprecisos com a técnica de Lógica Difusa, que traduz esse processo de tomada de decisão de forma a replicá-lo em equipamentos que seguem o modelo tradicional de lógica.

Zadeh (2010) afirma que a Lógica Difusa, ao contrário da lógica tradicional aristotélica, permite diferentes níveis de certeza. Isso significa dizer que é possível caracterizar variáveis através de conjuntos que se sobrepõem e que são influenciados por modificadores. Desta forma, é possível tratar conceitos abstratos de maneira semelhante ao pensamento humano, não considerando valores absolutos, mas sim a percepção gradativa em relação a uma determinada informação.

Esse conceito fica mais claro ao se tomar uma situação real, como a avaliação da intensidade de sinal de um determinado sensor. A seguir, é apresentado um exemplo para elucidar o conceito:

- Considerando dois nós sensores equipados com um rádio com potência de transmissão de até 10 dBm, e uma sensibilidade para recepção de até -90 dBm, deseja-se avaliar a intensidade do sinal recebido, a fim de realizar ajustes na potência inicial de transmissão.

Para que não seja necessário utilizar a potência máxima de transmissão do rádio, verifica-se a leitura do RSSI recebido para diferentes potências de transmissão. Seres humanos conseguem avaliar a intensidade do sinal através de conceitos abstratos, como “Forte” e “Fraco”, traduzindo valores absolutos, como -30 dBm ou -80 dBm. Pode-se, então, exemplificar melhor esse conceito tomando a variável “Intensidade do Sinal Recebido”, cujos conjuntos podem ser abstraídos como “Fraco”, “Médio” e “Forte” e utilizando modificadores como “Muito” e “Pouco”. A Tabela 1 apresenta os possíveis conjuntos da variável “Intensidade do Sinal Recebido”.

Tabela 1: Possíveis conjuntos da variável “Intensidade do Sinal Recebido”

Variável	Conjuntos
Intensidade do Sinal Recebido	Muito Fraco
	Fraco
	Pouco Fraco
	Médio
	Pouco Forte
	Forte
	Muito Forte

Um ser humano consegue através de sua experiência transformar valores absolutos em conceitos abstratos, facilmente compreendidos e compartilhados por outros, ou seja, se ao verificar uma leitura de RSSI se obtém a informação de que a intensidade do sinal recebido é de -50 dBm, consegue-se facilmente determinar que -50 dBm trata-se de um RSSI Médio (dentro dos limites definidos no exemplo). Dessa forma, é possível tomar a decisão sobre que tipo de ajuste deve ser feito no sensor transmissor, bem como compartilhar a informação a outros que desejam da mesma forma utilizar o dado para tomar algum tipo de decisão.

Para um computador, entretanto, essa não é uma tarefa simples, pois sua concepção está fundamentada na lógica tradicional bivalente (zero/um, ligado/desligado, verdadeiro/falso). Mesmo que dentro das estruturas de decisão se criem intervalos que ajudem a definir o conceito abstrato “Forte” e “Fraco”, ainda não será de fato possível simular o raciocínio humano, pois os valores limites continuam sendo fechados (ZADEH, 1994). Contudo, através da Lógica Difusa, é possível ter valores limites abertos e que se sobrepõem, retornando valores que representam a pertinência do item analisado em relação a cada conjunto (ZADEH, 1994). As Figura 1 e Figura 2 ilustram a diferença entre limites fechados não sobrepostos e limites abertos sobrepostos.

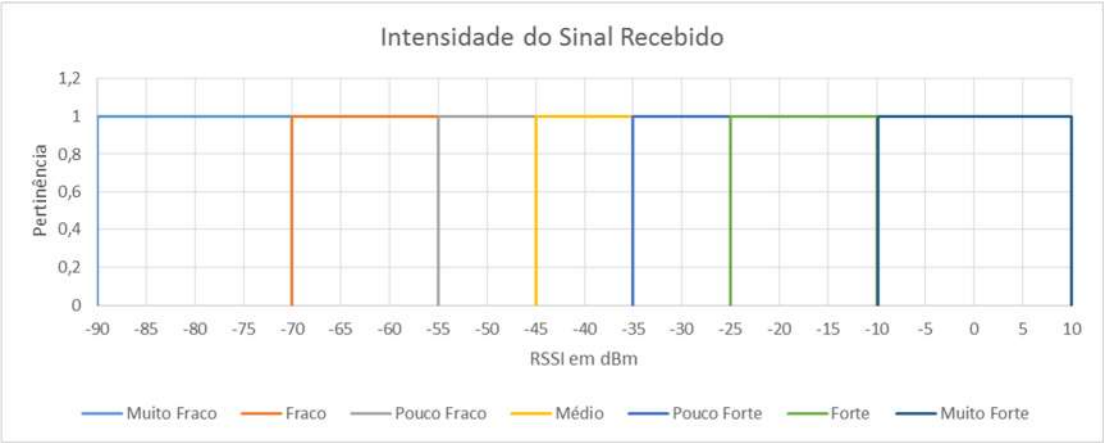


Figura 1: Conjuntos não sobrepostos da variável Intensidade do Sinal Recebido

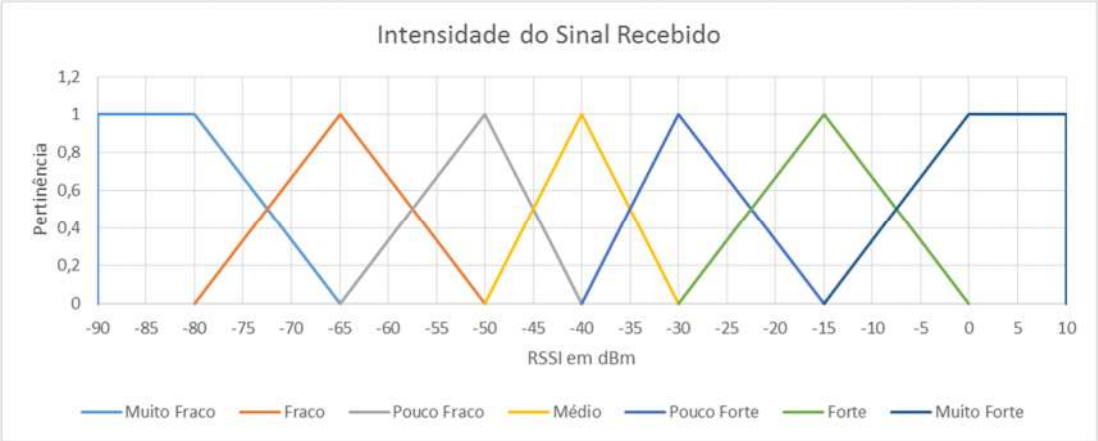


Figura 2: Conjuntos sobrepostos da variável Intensidade do Sinal Recebido

Observando as figuras Figura 1 e Figura 2 é possível verificar claramente a diferença em relação à sobreposição dos conjuntos. Em um sistema utilizando lógica convencional, não é possível que um mesmo valor possua correspondência simultânea em conjuntos diferentes, pois ou a Intensidade do Sinal Recebido está “Pouco Fraco” ou está “Médio”, nunca as duas ao mesmo tempo. Ao assumir o valor absoluto –60 dBm e utilizar a Figura 1 para identificar a qual conjunto pertence, verifica-se que o valor está inserido no conjunto “Fraco”, assim como o valor –70 dBm. Contudo, –70 dBm é quase –71 dBm, que já pertence ao conjunto “Muito Fraco”, sendo uma diferença muito pequena para ser considerada para tomada de decisão, mas suficiente para caracterizar uma mudança de conjunto. Nesse exemplo, consegue-se verificar a incapacidade da lógica convencional em tratar conceitos abstratos. Porém, se por outro lado, toma-se a Figura 2 para classificar os valores mencionados, perceber-se que é possível se aproximar do conceito abstrato utilizado pelo ser humano, pois há razões de pertinência tanto para o conjunto “Fraco” como para o conjunto “Muito Fraco”.

Desta forma, observa-se que há grande vantagem na utilização da Lógica Difusa para solução de problemas que exijam certo nível de abstração, pois através de uma classe de regras, é possível avaliar cada variável em relação a seus conjuntos e identificar padrões que colaborem com a tomada de decisão.

O processo de tomada de decisão por meio da execução da Lógica Difusa é chamado de Sistema Difuso. Um sistema difuso é dividio em três etapas:

Processo de Fuzzificação, Inferência e Defuzzificação. A Figura 3 apresenta um diagrama de funcionamento de um sistema difuso.

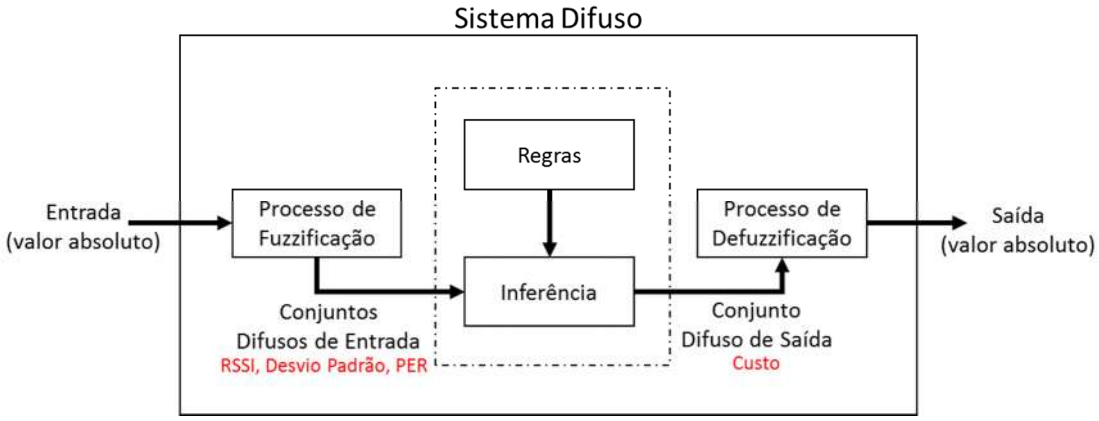


Figura 3: Representação de um Sistema Difuso. Adaptado de Mendel (1995).

O funcionamento completo de um sistema difuso é apresentado no Capítulo 5 em conjunto com a proposta de avaliação da qualidade do enlace. As etapas de identificação das variáveis e dos conjuntos difusos, bem como o processo de inferência e de obtenção do valor final são descritos em detalhe na seção 5.1.2 Avaliação.

3 Redes de Sensores Sem Fio

Uma RSSF é, em termos gerais, composta por duas entidades distintas: Estação Base (EB) e Nó Sensor (NS). O número de Estações Base e Nós Sensores podem variar de acordo com a aplicação, podendo haver redes de sensores sem fio com diversas estações base e diversos nós sensores.

A Estação Base funciona como um *Gateway* para os nós sensores, coletando e centralizando a informação monitorada. A EB pode estar conectada a uma outra rede, por exemplo uma rede IP com acesso à Internet, e funcionar como uma interface entre a RSSF e a rede externa, para onde os dados são encaminhados e tratados. As comunicações com os nós sensores é feita via rádio, com a Estação Base enviado e recebendo pacotes de dados de cada nó sensor. O número de EBs em uma aplicação varia de acordo com os objetivos, estrutura da rede e arquitetura do sistema (TOWNSEND, 2004). Aplicações de *Smart Cities* para o monitoramento do consumo de eletricidade em residências, por exemplo, podem demandar que diversas Estações Bases estejam espalhadas, cobrindo uma determinada região e centralizando a informação para o empacotamento e envio para uma central de processamento. Desta forma, espera-se da Estação Base uma capacidade melhor de processamento e maior autonomia em relação aos demais nós sensores.

Em relação aos Nós Sensores, pode-se dizer que, em geral, são dispositivos limitados, com fortes restrições em relação a memória e processamento. Uma RSSF pode conter de dezenas a centenas de nós sensores, trabalhando colaborativamente para o monitoramento de uma ou mais características do ambiente ou da aplicação. Os Nós Sensores podem comunicar-se entre si ou apenas com a Estação Base, dependendo da aplicação e da estrutura da RSSF. Desta forma, faz-se necessário e definição de um protocolo de roteamento, para que os pacotes de dados possam ser encaminhados corretamente dentro da rede (YICK, 2008).

As RSSF estão tipicamente estruturadas em uma pilha de cinco camadas, Aplicação, Transporte, Rede, Acesso ao Meio e Física (AKYILDIZ, 2002), como mostra a Figura 4.

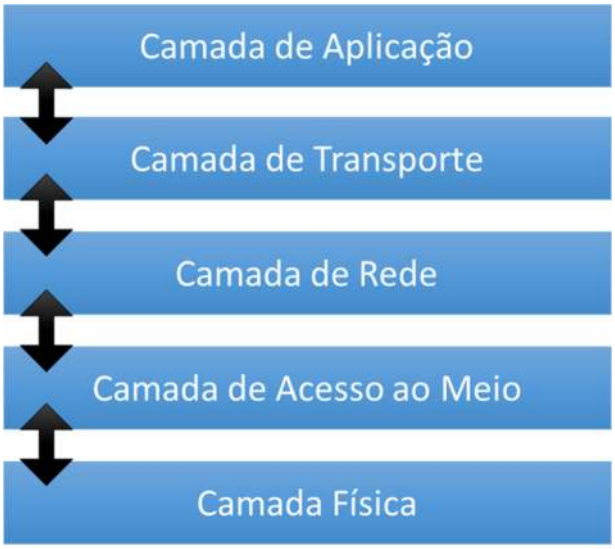


Figura 4: Pilha de Protocolos da RSSF

Nessa estrutura, cada camada desempenha um papel específico:

- Camada de Aplicação: responsável pelas informações monitoradas, provendo serviços de acordo com o tipo de aplicação;
- Camada de Transporte: responsável pelo controle da comunicação e manutenção do fluxo de dados;
- Camada de Rede: responsável pelo roteamento dos pacotes providos pela camada de transporte;
- Camada de Acesso ao Meio: responsável pelo controle de acesso ao meio e gerenciamento de energia;
- Camada Física: responsável pela modulação, canal e potência do sinal de rádio.

No entanto, é comum encontrar implementações *cross-layer* em Redes de Sensores sem Fio. Neste cenário, as camadas compartilham informações de maneira integrada, possibilitando que decisões de uma camada sejam influencidas por informações de outra camada. Yick (2008) afirma que em soluções *cross-layer* as camadas funcionam como um sistema, com informações disponíveis em todos os níveis, permitindo melhorias no desempenho e otimizando a interação entre as camadas.

Segundo Townsend (2004), as RSSF podem ser classificadas de acordo com a arquitetura da rede. As redes de sensores sem fio podem ser elaboradas seguindo diferentes topologias, como:

- Ponto-a-Ponto: os nós sensores se comunicam com a estação base por meio de uma cadeia de sensores. A estação base envia um pacote a um primeiro nó sensor, que encaminha a informação até que o nó sensor destino seja alcançado na corrente, como ilustra a Figura 5. A vantagem desta estratégia está ligada a simplicidade de implementação, no entanto, é pouco confiável, pois qualquer falha no caminho impede que os nós sensores subsequentes sejam alcançados.

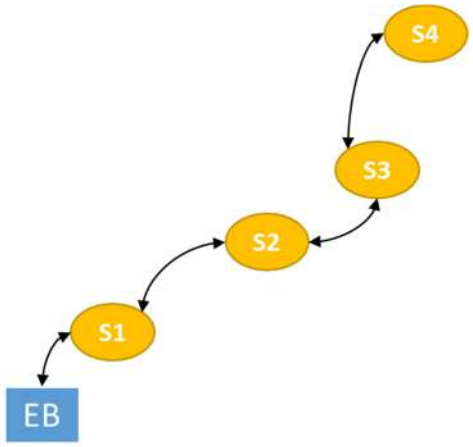


Figura 5: Topologia Ponta-a-Ponto

- Ponto-a-Multiponto: os nós sensores se comunicam com a estação base diretamente. A estação base pode enviar pacotes a cada um dos nós sensores, como ilustra a Figura 6. Contudo, os nós sensores não podem se comunicar diretamente com os demais nós da rede. A vantagem desta estratégia está ligada a economia de energia, já que não há a necessidade de mais nós sensores se envolverem na cadeia de encaminhamento. Contudo, o tamanho da rede fica limitado ao alcance de rádio da Estação Base e dos Nós Sensores.

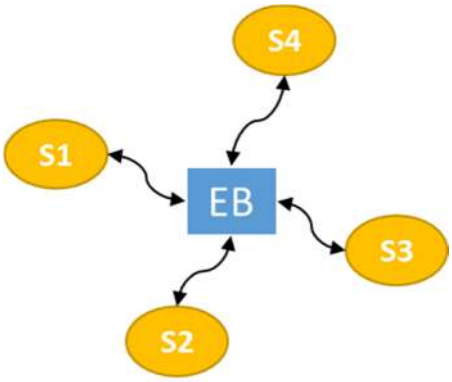


Figura 6: Topologia Ponto-a-Multiponto

- Rede de Malha (*Mesh)*: os nós sensores da rede podem se comunicar com a estação base ou qualquer outro nó sensor na rede diretamente, desde que estejam dentro da cobertura de rádio, como pode ser observado na Figura 7. Nesse tipo de rede a tolerância a falhas é maior, uma vez que se um nó sensor parar de transmitir, a rede poderá continuar funcionando com os demais nós sem prejuízo. A escalabilidade também é alta, favorecendo a implantação de redes de grande densidade de nós sensores. Contudo, a implementação torna-se mais complexa e o desafio para economia de energia é ainda maior.

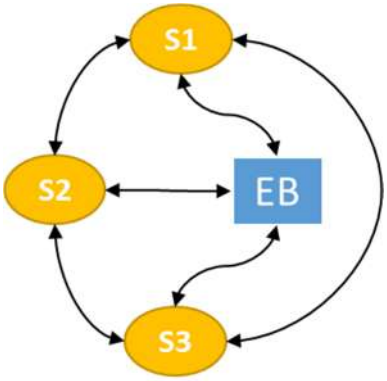


Figura 7: Topologia de Rede Mesh

- Rede Híbrida: os nós sensores da rede comunicam-se com a estação base diretamente ou com nós sensores centralizadores, que posteriormente encaminham a informação para a estação base.

Nesta configuração, há a definição de nós sensores líderes de grupos, que tomam a responsabilidade de fazer o encaminhamento de pacotes entre nós sensores mais distantes e a estação base, como mostra a Figura 8. Esse tipo de rede permite agregar vantagens da topologia Ponto-a-Multiponto e Rede *Mesh*, elevando a tolerância a falhas, buscando maior eficiência energética e viabilizando a implantação de redes de grande densidade. Entretanto, assim como na Rede *Mesh*, a implementação se torna mais complexa.

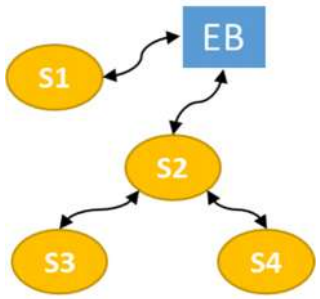


Figura 8: Topologia de Rede Híbrida

De acordo com Yick (2008), as aplicações de RSSF podem ser classificadas em duas categorias: Rastreamento e Monitoramento. Soluções de rastreamento em RSSF estão relacionadas a aplicações cujo objetivo é identificar objetos, animais ou pessoas em determinadas localidades. Por exemplo, uma solução de rastreamento de objetos em uma casa pode utilizar nós sensores espalhados por todos os cômodos a fim de detectar objetos. Os objetos precisam estar devidamente identificados com etiquetas RFID, para que possam ser localizados. A aplicação emite o comando para o rastreamento do objeto e os nós sensores próximos ao objeto etiquetado o identificam, informando a aplicação central o local onde a leitura da etiqueta do objeto foi detectada.

Já soluções de RSSF para monitoramento se caracterizam pelo acompanhamento de fenômenos, características ou eventos. As aplicações de monitoramento podem estar relacionadas ao acompanhamento de eventos naturais, como chuvas, atividades sísmicas e vulcânicas, bem como

características de ambientes fechados, tais como a temperatura, ventilação, luminosidade, detecção de fumaça, consumo energético etc. As aplicações de monitoramento podem ser classificadas também com relação a criticidade da informação e o tempo de resposta. Aplicações de tempo-real demandam soluções robustas de monitoramente, com tolerância a falhas e redundância nas medições. Entretanto, soluções de monitoramento de dados de baixa criticidade não necessitam de mecanismos de tolerância a falhas, pois a perda de um dado não compromete o funcionamento geral do sistema.

4 Roteamento em Redes de Sensores Sem Fio

Segundo Stankovic (2008), roteamento em redes de sensores sem fio é um serviço crítico, quando são necessários múltiplos saltos para que a informação possa fluir dos nós sensores para a estação base. São muitos os desafios a serem resolvidos, como a descoberta dos vizinhos e a identificação do melhor caminho até a estação base.

Ainda de acordo com Stankovic (2008), os protocolos existentes para redes Internet ou MANET não atendem às demandas e características inerentes das RSSF. Os protocolos existentes para a Internet assumem que os caminhos são normalmente cabeados e confiáveis, havendo pouca ou quase nenhuma perda de informação. Já os protocolos das redes MANET assumem a existência de uma simetria nos enlaces de subida e descida, o que nem sempre ocorre em RSSF.O objetivo de um protocolo de roteamento em RSSF é a entrega de pacotes, contudo, as estratégias de roteamento em redes de sensores sem fio variam de acordo com a aplicação e são dependentes do ambiente em que a RSSF está implementada (AL-KARAKI, 2004). Por essa razão, há um grande esforço de pesquisa para que soluções de RSSF sejam desenvolvidas de acordo com a necessidade de cada aplicação. Segundo Yick (2008), o desenvolvimento de um protocolo de roteamento para RSSF deve considerar as retrições de recursos características desse tipo de rede, como baixo poder de processamento, memória limitada e reduzida taxa de transferência.

4.1 Protocolos Baseados em RSSI

Soluções de roteamento baseadas em RSSI caracterizam-se pela avaliação da intensidade do sinal recebido. Utilizam-se as medições de RSSI entre a estação base e os nós sensores, e entre os próprios nós sensores, como forma de determinar as rotas para que os pacotes sejam encaminhados para a estação base (AWANG *et al.*, 2009b). Protocolos baseados em RSSI podem ainda considerar a distância física entre o nó sensor e a estação base, escolhendo o sensor mais próximo e elegendo-o como responsável por realizar o encaminhamento dos pacotes de nós sensores mais distantes (AWANG *et al.*,

2009a). Contudo, é arriscado supor que nós sensores com alto nível de RSSI estão próximos à estação base, uma vez que efeito de sombreamento (*shadowing effect*) pode gerar resultados imprevisíveis (AWANG *et al.*, 2009b).

A proposta apresentada por Awang *et al.* (2009a, 2009b) trata de um protocolo de encaminhamento de pacotes baseado em RSSI (RBF – *RSSI-Based Forward*) que combina decisões das camadas de enlace e roteamento no processo de avaliação e definição dos saltos a serem executados no encaminhamento dos pacotes até a estação base. É um protocolo orientado a eventos, com processamento distribuído e sem dependência do estado geral da rede, uma vez que não há necessidade do nó sensor emissor conhecer a topologia da rede ou a localização de seus vizinhos. Diferentemente dos protocolos baseados RSSI e localização, o protocolo RBF desconhece a localização da estação base e o encaminhamento é baseado nas medições de RSSI entre os nós sensores vizinhos e a estação base. Nesta proposta, supõe-se que a estação base é capaz de enviar anúncios (*beacon messages*) que podem ser recebidas por todos os nós sensores da RSSF. Com base nisso, os sensores utilizam o RSSI como um parâmetro para determinar o nó sensor mais próximo à estação base. A estratégia ocorre como uma seleção baseada na concorrência, em que o nó sensor com o mais alto nível de RSSI em relação à estação base é selecionado para encaminhar os pacotes para o próximo nó sensor. Contudo, como a taxa de sucesso na entrega de pacotes não é uma preocupação nesta estratégia, não há garantias de que o próximo salto é a melhor escolha e que irá efetivamente entregar o pacote.

Em Boukerche *et al.* (2008) temos uma outra solução de encaminhamento de pacotes descentralizada baseada em RSSI. É um algoritmo denominado “guloso” (ou cego), ou seja, que obtém a melhor opção local, sem avaliação geral das rotas, e não dependente da localização dos nós sensores e da estação base. O protocolo sugerido faz uso de mensagens em forma de anúncio (*broadcast messages*) para decidir o próximo salto durante o encaminhamento dos pacotes. Os autores descrevem duas abordagens diferentes, a saber: *RSSR Election* e *RSSR Selection*. A primeira abordagem, *RSSR Election*, é baseada na eleição de um líder, que irá realizar a tarefa de encaminhar os pacotes recebidos dos demais sensores. Nesta solução a estação base, que se assume ter um alcance de rádio

poderoso e capaz de atingir a totalidade da rede, envia uma requisição a um nó específico, sob a forma de uma mensagem de difusão (*broadcast*). Em seguida, todos os nós sensores da rede salvam a informação de RSSI em relação a estação base em um banco de dados local. Após verificar se a mensagem enviada pela estação base tem como destino a si própria, o nó sensor envia uma mensagem de difusão para todos os seus vizinhos com a informação solicitada e o valor da medição de RSSI que ele possui com a estação base. O vizinho cuja distância em relação a estação base for a menor (a distância é calculada a partir da RSSI) se elege como o líder e encaminha o pacote em forma de uma mensagem de difusão. Os outros nós que participam do processo de eleição recebem a mensagem e cancelam a sua participação. A segunda abordagem, *RSSR Selection*, é diferente por não iniciar uma eleição, mas desencadear uma fase de seleção. Depois de receber a mensagem da estação base, os nós sensores iniciam um processo de troca de anúncios. Dessa forma, cada nó sensor passa a conhecer a distância de seus vizinhos em relação a estação base. As informações são armazenadas em uma tabela de roteamento local, interna a cada um dos nós sensores, a fim de ser utilizada para a decisão dos próximos saltos durante o encaminhamento dos pacotes. Não obstante aos ganhos de uma solução distribuída, como a reação a mudanças na rede de forma implícita, as soluções propostas não abordam questões relacionadas a espaços isolados na rede, em que nós sensores apesar de aptos a receber pacotes da estação base, não conseguem alcançar outros nós sensores que possam encaminhar as respostas à estação base. Há também uma sobrecarga de pacotes na rede, uma vez que a cada nova solicitação de informação, todo o processo de seleção é refeito.

Kanzaki *et al.* (2011) apresenta uma solução de roteamento baseada nas características de propagação do sinal. A proposta é apresentada como um protocolo centralizado, que verifica o estado dos enlaces entre os nós sensores e constrói uma tabela de roteamento baseado nas informações de qualidade do enlace. A tabela de roteamento é armazenada dentro de cada nó sensor e consultada sempre que um pacote precisa ser encaminhado. A solução atua de maneira proativa, monitorando os enlaces e detectando possíveis degradações. Se existe um enlace na tabela de encaminhamento cujos parâmetros estão

abaixo de um limiar desejado, o algoritmo reconstrói a tabela de roteamento. A proposta utiliza a taxa de entrega de pacotes e relação sinal-ruído (SNR – *Signal to Noise Ratio*) de cada enlace como um parâmetro para avaliar a elegibilidade do percurso como uma entrada na tabela de rotas. Esta solução é um melhoramento de um trabalho anterior descrito em Nose (2010) que apresenta um protocolo de encaminhamento estático que utilizada a média de RSSI e a taxa de entrega do pacote como parâmetros para construir a tabela de roteamento. Em ambas as propostas, os autores exploram a relação entre a qualidade do enlace e a taxa de entrega de pacotes, a fim de classificar os enlaces de cada nó sensor e criar uma tabela de roteamento. Embora seja uma solução dinâmica, capaz de se adaptar a alterações da qualidade dos enlaces, a proposta pode ainda estar suscetível a buracos na rede, uma vez que estabelece limites para os parâmetros medidos. Isto significa que ocasionalmente os nós sensores podem estar fora do limite definido e, consequentemente, fora da tabela de roteamento, levando ao surgimento de espaços vazios na rede.

4.2 Protocolos Baseados em Lógica Difusa

O método sugerido por Dastgheib *et al.* (2011) faz uso de Lógica Difusa para selecionar os nós sensores que serão usados para a tarefa de encaminhamento de pacotes. Nesse método são consideradas duas variáveis principais a fim de selecionar o destino do próximo salto: o ângulo do nó sensor em relação aos seus vizinhos e o número de pacotes já encaminhados para o vizinho. Essas variáveis são analisadas através de um sistema difuso e o resultado é um número que representa a chance de um nó sensor ser selecionado como destino para o próximo salto. A decisão de encaminhamento é tomada avaliando os nós sensores que possuem a maior chance de transmitir o pacote. Embora seja uma solução distribuída, a localização da estação base deve ser conhecida pelos nós sensores a fim de selecionar corretamente o próximo salto. A localização é obtida comparando o ângulo do nó sensor em relação a estação base. Isso implica que a localização estação base não pode ser alterada, uma vez que isso implicaria na perda completa do referencial dos nós sensores. Além disso, não existe um método de controle para evitar que nós sensores com

altas taxas de perda de pacote sejam evitados durante o encaminhamento de pacotes.

Sakthidevi, Srievidhyajanani (2013) propõem um algoritmo descentralizado baseado em TARF (*Trust-Aware Routing Framework*) para determinar a confiabilidade de um sensor dentro da rede e verificar se determinadas características dos nós sensores são adequadas para agregação ou não, definindo assim as rotas para o encaminhamento de pacotes. A proposta faz uso de Lógica Difusa como uma ferramenta de apoio ao processo de decisão. Variáveis como energia despendida para o encaminhamento, distância em relação aos nós sensores vizinhos, RSSI e taxa de entrega de pacotes são utilizadas como entrada para a execução das regras de decisão. Os autores propõem também a criação de uma hierarquia, elegendo nós sensores com os melhores índices de conectividade para servirem de agregadores. Apesar de utilizar Lógica Difusa como forma de acelerar o processo de classificação dos nós sensores, a proposta torna-se complexa para implementações práticas, especialmente em ambientes fechados. Os autores apresentam resultados de simulação, sem indicativos das condições necessárias para a implementação em situações reais.

Outra proposta é apresentada por Babu (2012) e descreve um método para seleção de rotas usando Lógica Difusa e considerando a avaliação de três parâmetros: nível da bateria, confiabilidade do nó sensor e a distância em relação a estação base, considerando o nível de RSSI. O valor do nível de bateria residual é calculado tendo em conta a energia utilizada para a transmissão a uma determinada distância. O modelo proposto leva em conta uma solução descentralizada para selecionar os nós que devem servir como um caminho no processo de roteamento. Considera-se ainda a existência de um banco de dados, no próprio nó sensor, para o armazenamento de informações sobre o histórico dos dados dos nós sensores vizinhos. Os autores apresentam apenas resultados de simulação, considerando um único cenário centrado no processo de tomada de decisão. Não há evidências da aplicabilidade do método em um cenário real.

Baccour (2010) apresenta um estimador baseado em Lógica Difusa para avaliar a qualidade do enlace entre nós sensores. Parâmetros tais como a taxa de

entrega de pacotes, assimetria do enlace de subida e descida (*uplink* e *downlink*), a estabilidade do enlace (ocorrência de alterações nos parâmetros medidos) e da qualidade de canal (relação sinal-ruído) são utilizados como quantificadores para a avaliação da qualidade do enlace. Os autores justificam o uso da Lógica Difusa na estimativa de qualidade do enlace devido à imprecisão nas medidas dos fatores que afetam a qualidade e estabilidade da comunicação entre os nós sensores. A partir do estimador, estratégias de roteamento podem ser empregadas a fim de viabilizar a comunicação dentro da RSSF.

Em Lee, Cheng (2012) os autores comentam sobre a importância de se utilizar estratégias de agrupamento de sensores a fim de aumentar a eficiência energética da rede, resultando no aumento da vida útil da RSSF. São apontadas estratégias de agrupamento que levam em consideração a divisão do tráfego entre os sensores com maior nível de bateria e através da distribuição periódica do consumo de energia entre os sensores, elegendo diferentes líderes de grupos ao longo do tempo. Uma nova proposta de processamento distribuído é apresentada a fim de utilizar tanto a informação de nível de bateria atual, como o nível de bateria esperado ao final do ciclo. Essas variáveis são trabalhadas por um algoritmo de agrupamento de sensores utilizando Lógica Difusa. A cada iteração, os sensores se candidatam a líderes e calculam a probabilidade de serem o sensor líder. Os sensores comunicam suas chances aos demais por meio de mensagens de difusão e o sensor com a maior chance é escolhido como líder. A proposta foi validada através de simulação e comparada com outros protocolos existentes, como LEACH (processamento distribuído), CHEF (processamento distribuído) e LEACH-C (processamento centralizado). A proposta mostrou-se melhor em comparação aos demais protocolos de processamento distribuído, apresentando valores semelhantes ao protocolo de processamento centralizado.

5 Algoritmo Proposto

O algoritmo proposto faz uso de lógica difusa para avaliar parâmetros de qualidade de cada enlace na rede de sensores sem fio. Desta forma, analisa-se o nível médio de RSSI, o desvio padrão do RSSI medido, e a taxa de erro de pacote, a fim de classificar os enlaces, atribuindo custos que serão utilizados para construir a tabela de roteamento. Uma vez que o custo de cada caminho é determinado, o protocolo usa o algoritmo de Dijkstra (TANENBAUM, 2003) para realizar a busca pelas melhores rotas e, assim, reconfigurar a rede.

A estação base atua como uma entidade coordenadora, coletando as medidas de RSSI, o desvio padrão das medidas de RSSI e a taxa de erro de pacote de cada nó sensor. Cada sensor é responsável por armazenar os dados solicitados em sua própria memória e executar as medidas necessárias. No entanto, as medições ocorrem durante o tráfego normal da rede, não prejudicando o desempenho geral da RSSF com excessivos pacotes de controle. A estação base é também dotada de maior poder computacional e uma melhor cobertura de rádio.

5.1 Processo de Coleta, Avaliação e Definição de Rota

O processo de definição de rota ocorre em três etapas: coleta, avaliação e definição, como mostra a Figura 9. Inicialmente, é necessário obter as medidas das variáveis que são utilizadas para a tomada de decisão no processo de definição das rotas. Em seguida, analisam-se as variáveis a fim de classificar o enlace em relação a qualidade. Uma vez classificados, os enlaces são submetidos ao algoritmo de Dijkstra em busca das melhores rotas.

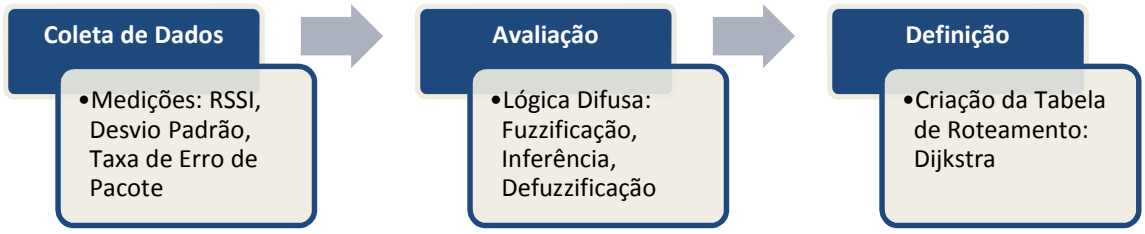


Figura 9: Etapas do Processo de Definição de Rota

5.1.1 Coleta

O processo de coleta de dados ocorre em duas etapas: medição dos parâmetros e envio dos dados. Antes de iniciar o processo de requisitar os dados de cada nó sensor, é necessário realizar medições dos parâmetros que serão analisados. As medições ocorrem de forma concomitante ao tráfego normal da rede, evitando, assim, o tráfego excessivo de pacotes de controle.

Quando um nó sensor envia um pacote de dados ou quando a base envia um pacote de requisição de dados, todos os nós sensores vizinhos detectam a transmissão. Nesse momento, após o nó sensor realizar o recebimento do pacote na primeira camada (camada física), verifica-se o destino do pacote para decidir se a mensagem será transferida para a próxima camada (camada de acesso ao meio). Uma vez que a mensagem já está disponível na camada física, a informação de remetente e o valor do RSSI da mensagem que foi recebida são armazenados em uma variável do tipo lista para compor o histórico de medições de RSSI. Esse processo permite avaliar continuamente a situação do ambiente em relação ao RSSI. Desta forma, não é necessário que os nós sensores enviem mensagens endereçadas especificamente a seus vizinhos para que o valor de RSSI seja avaliado. Uma vez constituído o histórico de medições de RSSI, é possível obter a média e o desvio padrão da amostra de dados.

Para a taxa de erro de pacote, utiliza-se um processo semelhante. Evita-se o envio de mensagens de confirmação de recebimento, caso o destino final esteja a dois ou mais saltos de distância do emissor. Como se deseja avaliar a taxa de perda de pacote entre os sensores vizinhos, a preocupação está em verificar se o pacote foi entregue para o nó sensor distante um salto do emissor. Desta-forma, cria-se uma cadeia de verificação, permitindo avaliar o caminho como um todo e identificar os melhores e piores enlaces.

A Figura 10 ilustra uma sequência de transmissão. Nesse exemplo, o nó sensor S6 necessita enviar um pacote para a estação base EB. Para tanto, recorre ao caminho pré-definido S6 -> S4 -> S2 -> S1 -> EB. A taxa de entrega de pacote deve ser verificada para cada um dos quatro enlaces. A fim de evitar que

quatro mensagens extras de controle sejam enviadas, utiliza-se o mesmo processo de detecção de transmissão aplicado para a verificação do RSSI. O nó sensor S6 marca o pacote com um identificador único e o transmite para o nó sensor S4 com uma etiqueta E {*id*, *fonte*, *destino*}, em que *id* representa o identificador único, *fonte* representa o nó sensor emissor e *destino* representa o nó sensor final da cadeia. O nó sensor S4 recebe o pacote e verifica que deve continuar a sequência de encaminhamento. O pacote é então transmitido com uma pequena modificação na etiqueta E, apenas alterando a variável *fonte* para o novo emissor. Quando o nó sensor S6 detecta a transmissão do pacote com a etiqueta E, e verifica que se trata do nó sensor S4 fazendo o encaminhamento, o processo é finalizado como sucesso. Se nenhuma transmissão for detectada até o próximo envio de pacote, o processo é finalizado como falha. Esse processo é repetido para cada enlace do caminho até o destino final.

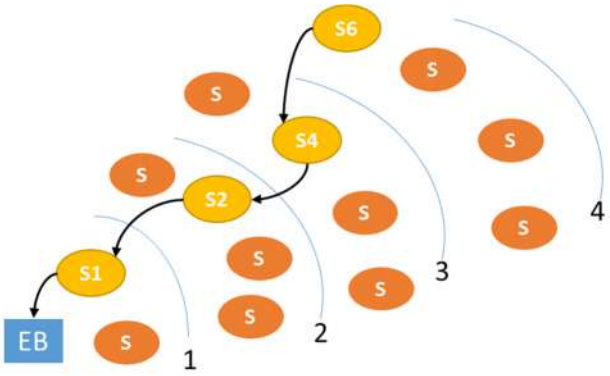


Figura 10: Sequência de transmissão

A confirmação de entrega de pacote ocorre de forma diferente apenas quando o destino final está a um salto de distância do nó sensor emissor, como é o caso do nó sensor S1 em relação a estação base EB. Nessa situação, recorre-se a utilização de uma mensagem de confirmação, apenas para notificar o nó sensor emissor de que houve sucesso no recebimento do pacote. Caso nenhuma confirmação seja recebida até a próxima transmissão, então o processo é finalizado com falha.

O processo de medição é uma atividade contínua e ocorre para todas as transmissões detectadas pelos nós sensores. Sendo assim, a estação base pode

a qualquer momento requisitar os dados de medições dos sensores. Definiu-se um agendamento para a requisição dos dados de medições. Esse agendamento pode ser variável e é controlado pela estação base. A etapa de coleta de dados tem seu ciclo finalizado após o envio dos dados à estação base.

5.1.2 Avaliação

A partir dos dados de medições recebidos durante a etapa de coleta, inicia-se a fase de avaliação e análise das medições. Esse processo ocorre por meio da aplicação dos conceitos de Lógica Difusa. Como já mencionado neste trabalho, Lógica Difusa permite tratar computacionalmente dados com diferentes níveis de certeza, como “Sinal Bom” e “Sinal Muito Bom”. A avaliação dos dados por meio de intervalos sobrepostos permite aplicar regras de definição mais flexíveis, que variam de acordo com o nível de certeza de uma determinada variável. Isso significa dizer que apesar de um determinado valor medido estar mais aderente a um conjunto específico, não ficam excluídas as pertinências desse valor a outros conjuntos. Dessa forma é possível criar regras de avaliação que combinam os diferenes níveis de certeza, fazendo com que a variável final de custo seja o resultado de uma análise mais ampla dos valores medidos.

A fim de possibilitar o uso de Lógica Difusa no processo de avaliação dos enlaces, é necessário estabelecer uma relação entre os dados a serem analisados para cada uma das variáveis e os conjuntos difusos. Para tanto, deve-se definir funções de pertinência, que suportam o processo de transformação dos valores absolutos em linguagem natural. Em Gomide (1995) temos que as funções de pertinência em lógica difusa representam conjuntos cujos elementos membros podem ter valores de pertinência em diferentes níveis. Pode-se utilizar funções triangulares e trapezoidais como forma de representação das diferenças de relevância. Neste trabalho, faz-se uso das funções triangulares e trapezoidais para representar cada uma das variáveis difusas e seus conjuntos. Sendo assim, têm-se os valores de RSSI divididos em conjuntos difusos e categorizadas em: Fraco, Médio e Forte, como descrito na Figura 11.

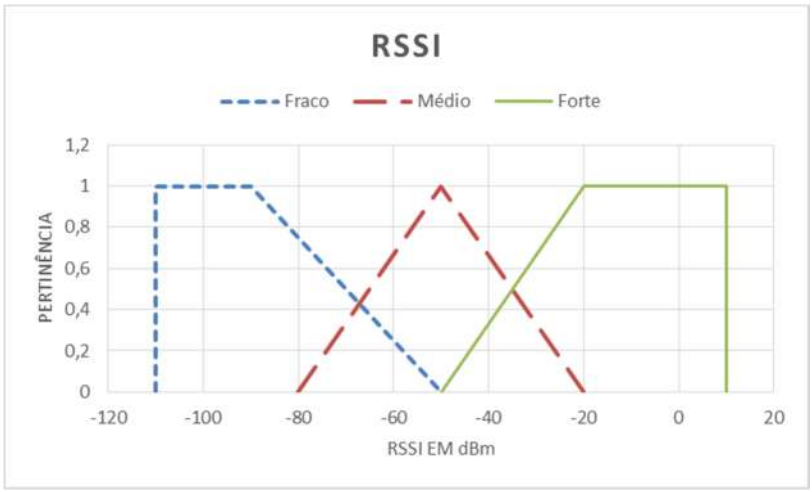


Figura 11: Conjuntos da variável RSSI

Observa-se na Figura 11 que existe uma área de intersecção entre os conjuntos Fraco e Médio, bem como nos conjuntos Médio e Forte. Isso significa, por exemplo, que valores entre -80 dBm e -50 dBm pertencem ao mesmo tempo ao conjunto Fraco e Médio, possuindo pertinências diferentes para cada um desses conjuntos.

O mesmo processo de definição de conjuntos difusos e funções de pertinência foi realizado para as demais variáveis do sistema. Os conjuntos do desvio padrão foram definidos em: Bom, Médio e Ruim, como apresenta a Figura 12.

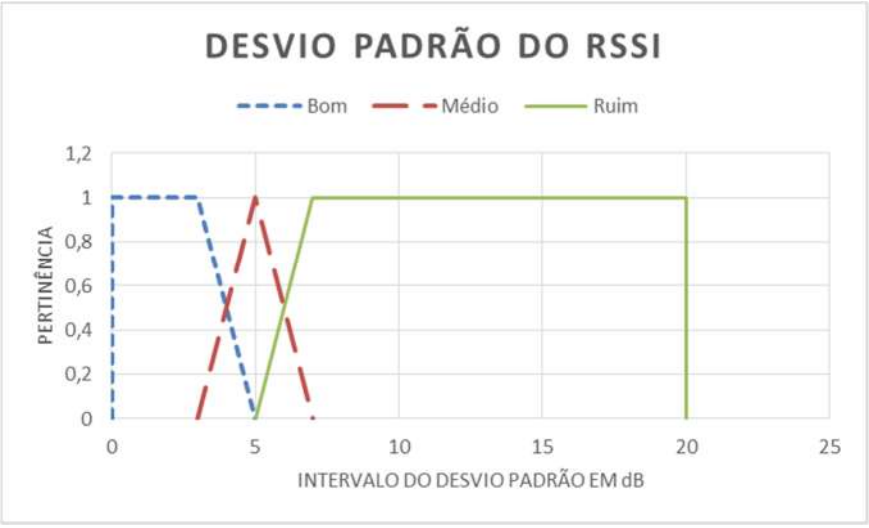


Figura 12: Conjuntos da variável Desvio Padrão

É importante ressaltar que os valores para os intervalos das variáveis RSSI e Desvio padrão podem variar de acordo com a potência mínima aceitável do rádio que está sendo usado. Para este trabalho, definiu-se uma sensibilidade de -90 dBm (@ 1.2 kBaud, PER 1%) e uma potência máxima de 10 dBm, uma vez que esses são os limites apresentados pela especificação do módulo de rádio dos sensores utilizados no experimento (SEEED TECHNOLOGY LIMITED, 2010).

Foram definidos também os conjuntos da variável Taxa de Perda de Pacote, representada por Baixa, Média e Alta conforme descrito na Figura 13.

Figura 13: Conjuntos da variável Taxa de Erro de Pacote

Contudo, precisou-se ainda definir um conjunto de saída, capaz de representar de forma abstrata o resultado da avaliação. Desta forma, especificou-se a variável "Custo" para representar a abstração dos intervalos de custo dos enlaces. Os conjuntos que compõem a variável "Custo" são: Alto, Médio e Baixo como representado na Figura 14.

Figura 14: Conjuntos da variável Custo

Uma vez definidos os conjuntos das variáveis que devem ser analisadas durante a identificação dos melhores caminhos na RSSF, é preciso que se identifique a pertinência de cada valor lido em relação a esses conjuntos. A pertinência é calculada seguindo uma regra simples de semelhança de triângulos, como mostra a Figura 15.

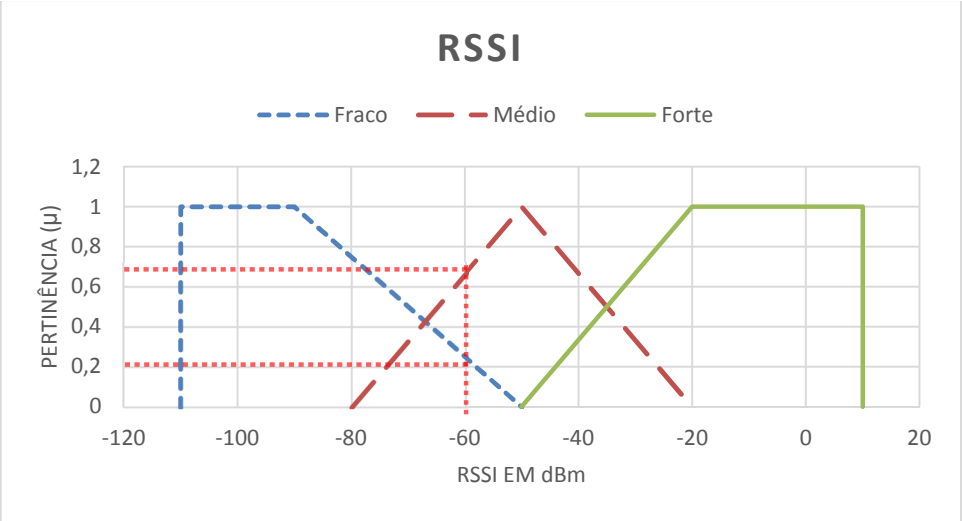


Figura 15: Pertinência do valor lido -60 dBm

Observando a Figura 15 temos que para um valor de RSSI de -60 dBm a pertinência ao conjunto Forte é 0, ao conjunto Médio é 0,67 e ao conjunto Fraco é 0,25 Isso significa dizer que o valor de RSSI avaliado possui relevância tanto para

o conjunto Médio como para o conjunto Fraco, porém em uma gradação diferente. As pertinências aos conjuntos definidos devem ser encontradas para todas as variáveis que estão sendo analisadas e para todos os valores que se deseja avaliar.

Uma vez definidas as funções de pertinência das variáveis a serem analisadas, é necessário identificar a relevância de cada valor obtido a partir das medidas em relação aos conjuntos difusos. A pertinência encontrada para cada medida de cada conjunto difuso é utilizada como entrada para a execução da inferência a partir de regras. A Tabela 2 representa as regras definidas para este trabalho. As regras são criadas a partir da observação dos fenômenos, da avaliação de especialistas e delimitadas pelo escopo da aplicação. A execução das regras permite a identificação de subgrupos da variável "Custo", em outras palavras, é possível verificar as áreas de influência de cada regra para cada parâmetro na definição dos custos do enlace. A definição de regras é uma etapa importante do processo, uma vez que ela reflete a análise empírica do sistema estudado. É possível observar em Mamdani (1974) o método de inferência que foi utilizado neste trabalho, que é basicamente a execução de um processo de implicação SE...ENTÃO.

Tabela 2: Lista de Regras para Inferência

Parâmetro 1	Parâmetro 2	Parâmetro 3	Consequência
PER Alta	PER Alta	PER Alta	Custo Alto
PER Média	PER Média	PER Média	Custo Alto
PER Baixa	RSSI Fraco	Desvio Padrão Ruim	Custo Alto
PER Baixa	RSSI Fraco	Desvio Padrão Médio	Custo Médio
PER Baixa	RSSI Fraco	Desvio Padrão Bom	Custo Baixo
PER Baixa	RSSI Médio	Desvio Padrão Ruim	Custo Alto
PER Baixa	RSSI Médio	Desvio Padrão Médio	Custo Médio
PER Baixa	RSSI Médio	Desvio Padrão Bom	Custo Baixo
PER Baixa	RSSI Forte	Desvio Padrão Ruim	Custo Alto
PER Baixa	RSSI Forte	Desvio Padrão Médio	Custo Baixo
PER Baixa	RSSI Forte	Desvio Padrão Bom	Custo Baixo

Como já mencionado, a execução das regras permite construir a relação de cortes para os custos de cada enlace, ou seja, cada linha da tabela de regras resulta em um valor de corte para um conjunto da variável “Custo”. Diz-se que se trata de um corte, pois ainda não se tem identificado um único valor de custo resultante de todo o processo.

A execução respeita a função:

$$Cn = min(P1, P2, P3) \quad (1)$$

Em que Cn é o resultado da enésima regra, P1 é o valor de pertinência para o primeiro parâmetro, P2 é o valor de pertinência para o segundo parâmetro e P3 é o valor de pertinência para o terceiro parâmetro. A função “min()” (mínimo) retorna o menor valor de pertinência entre P1, P2 e P3, denotando o conceito de intersecção e do operador “E”.

Como podemos observar na tabela de regras, temos diferentes combinações resultando em um mesmo conjunto da variável “Custo”, por exemplo: “PER Baixa, RSSI Fraco e Desvio Padrão Ruim” resulta no mesmo que “PER Alta”, ou seja, tem como consequência o mesmo conjunto da variável “Custo”. Neste caso é necessário utilizar o operador “OU”, através da função “max()” (máximo), assumindo sempre o maior valor entre os cortes para um mesmo conjunto, representando o conceito de união. A função (2) demonstra o conceito:

$$Corte\ Resultante = max(C1, C2) \quad (2)$$

A Figura 16 apresenta os cortes da execução de duas regras que resultam em um mesmo conjunto.

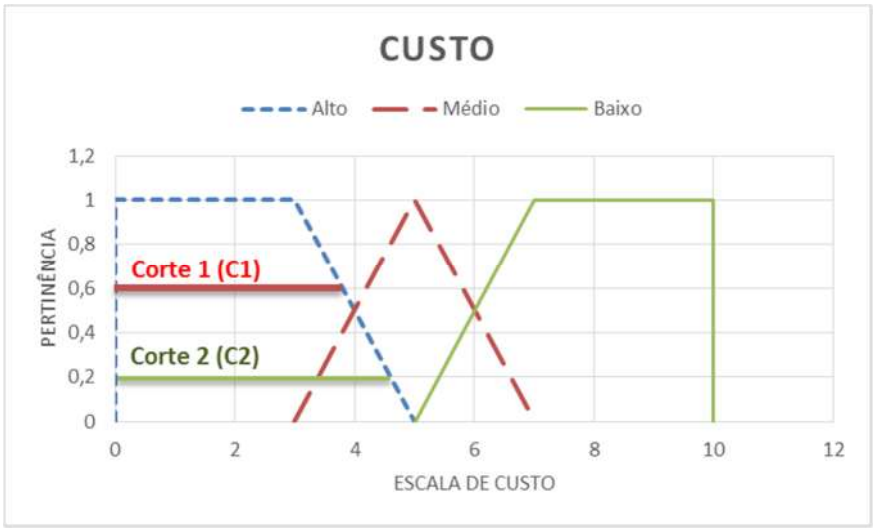


Figura 16: Cortes resultantes das regras de mesmo conjunto

Ao final da execução de todas as regras, teremos um gráfico semelhante ao da Figura 16, porém com um corte diferente para cada um dos conjuntos. O novo gráfico, resultante dos cortes identificados, representa os limites determinados pelas regras e pelas pertinências aos conjuntos de cada variável analisada, como mostra a Figura 17. Através dessa forma geométrica, é possível obter o valor de potencial de melhor vizinho.

Figura 17: Gráfico resultantes da execução de todas as regras

Os métodos utilizados para determinar o valor absoluto de potencial são chamados de defuzzificação. São várias as estratégias de defuzzificação que podem ser empregadas para a obtenção desse valor. A escolha de uma estratégia ou outra deve depender da aplicação final e do rigor necessário para a identificação do valor absoluto (JIANG, 1996). Para este trabalho, utilizou-se o método centroide, que calcula o centro de massa da forma geométrica identificada, resultando no valor absoluto do custo para um determinado enlace. O método centroide foi selecionado uma vez que apresenta erros menores para a média dos quadrados quando comparado a outros métodos, como o método da média dos máximos (LEE, 1990).

O método centroide utiliza a equação (3) para o cálculo do valor absoluto:

$$\boldsymbol{X} = \frac{\sum_{i=1}^{n} \mu_i x_i}{\sum_{i=1}^{n} \mu_i} \tag{3}$$

Em que x_i é o passo desejado para partição da forma geométrica resultante e μ_i representa o maior valor entre as pertinências do valor x_i aos conjuntos do potencial de melhor vizinho, limitados pelos cortes 1, 2 e 3.

Em Siqueira (2013) é possível encontrar o conceito do algoritmo utilizado para a execução da lógica difusa. Nesse método as informações referentes às variáveis difusas, conjuntos difusos e regras de inferência estão organizadas de forma matricial, possibilitando a generalização da execução. Isso significa que alterações em quaisquer um dos parâmetros não resultam em modificações no algoritmo. Todos os ajustes podem ser realizados nos dados inseridos nas matrizes. Essa flexibilidade permite que adaptações aos parâmetros definidos ocorram de maneira rápida e transparente para a função de execução da lógica difusa.

A etapa de avalição é finalizada com a construção de um grafo representando cada possível enlace na rede e o custo de cada salto. Contudo, no simulador e no experimento o grafo é representado por meio de uma matriz, como sugere a Figura 18. A matriz “Origem-Destino” contém o custo verificado de cada

enlace existente na rede. Por exemplo, o nó sensor S2 (origem) possui como vizinhos a Estação Base (EB) com um custo do enlace de 3, os nós sensores (destino) S1 com um custo do enlace de 1, S3 com custo 3, S4 com custo 2, S5 com custo 4 e S6 com custo 4. Os nós sensores S7, S8, S9 e S10 estçao fora do alcance de S2, então o custo é representado como um valor infinito (inf).

Destino

Origem	EB	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
EB	-	1	3	4	4	5	5	6	6	8	8
S1	1	-	1	2	1	3	3	inf	inf	inf	inf
S2	3	1	-	3	2	4	4	inf	inf	inf	inf
S3	inf	2	3	-	1	2	2	3	3	4	4
S4	inf	1	2	1	-	1	1	2	2	3	3
S5	inf	3	4	2	1	-	1	2	2	3	4
S6	inf	3	4	2	1	1	-	1	3	3	4
S7	inf	inf	inf	3	2	2	1	-	1	2	2
S8	inf	inf	inf	3	2	2	3	1	-	1	2
S9	inf	inf	inf	4	3	3	3	2	1	-	2
S10	inf	inf	inf	4	3	4	4	2	2	2	-

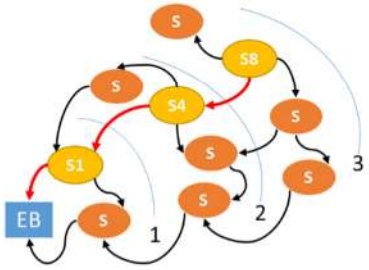


Figura 18: Representação da tabela de custos

5.1.3 Definição

Uma vez identificado o valor absoluto do custo do enlace para cada nó sensor, o algoritmo de Dijkstra é executado a fim de encontrar as melhores rotas para o encaminhamento de pacotes. Neste ponto, a tabela de encaminhamento de cada sensor é definida com a identificação da sequência de saltos a serem executados. A solução prevê a identificação dos caminhos de descida e subida dos pacotes para cada um dos nós sensores, uma vez que podem existir assimetrias no enlace (BACCOUR, 2010), levando a custos diferentes para o caminho de subida em relação ao caminho de descida.

A definição dos caminhos considera a criação de uma tabela de roteamento que pode ser representada através de um grafo, como mostra o exemplo da Figura 19. Pode-se notar que os nós sensores conhecem apenas o nó seguinte na cadeia de saltos. O caminho completo é conhecido apenas pelo software gerente, que repassa aos nós sensores somente a informação

necessária para o salto seguinte. O objetivo dessa estratégia está relacionado a simplificação do processo de alteração da tabela de rotas, uma vez que alterações na rede podem ser propagadas por meio de poucas modificações na tabela. A inserção de um novo nó sensor na rede pode gerar apenas mais uma entrada na tabela de rotas de cada sensor, sem a necessidade de alterar todas as informações já existentes.

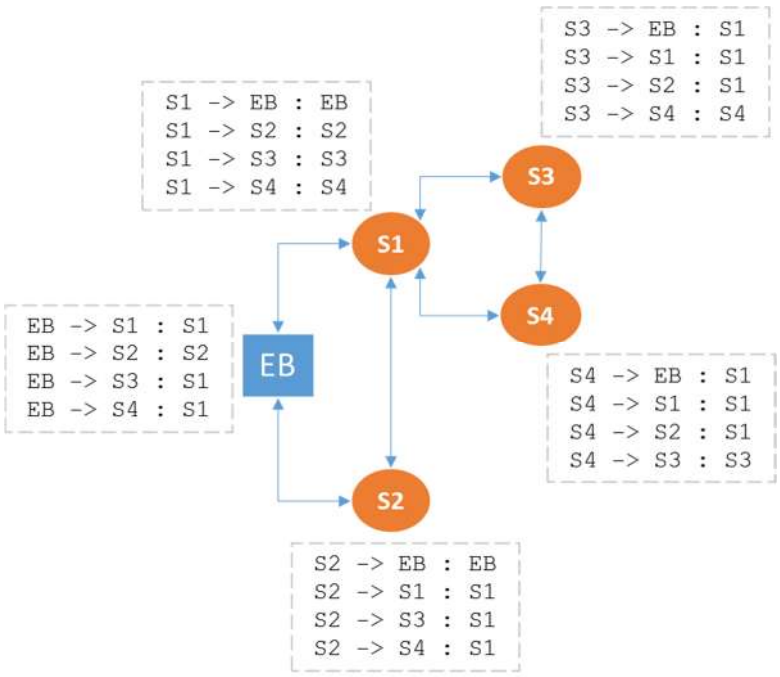


Figura 19: Representação da tabela de roteamento resultante

Uma vez identificados todos os melhores caminhos, constrói-se a tabela de roteamento de cada nó sensor, que recebe uma mensagem de alteração de rota com os novos caminhos a serem utilizados. A partir do momento do recebimento do pacote de alteração de rota, os nós sensores passam a fazer uso da nova tabela.

5.2 Execução do Algoritmo

A execução do algoritmo proposto ocorre em duas instâncias. A primeira se refere a unidade central de processamento e a segunda aos processos executados por cada nó sensor. Cada instância é responsável pela execução de

uma etapa do algoritmo. Os nós sensores executam as medições das variáveis analisadas de maneira autônoma e sem a interferência da unidade central de processamento. Contudo, o controle geral da execução, bem como as solicitações de medições e o envio de pacotes de alteração de rota são tarefas realizadas pela unidade de processamento central por meio da estação base.

Há duas fases distintas na execução do algoritmo. Primeiro é necessário realizar a inicialização da rede, com a descoberta dos nós sensores e a criação da primeira tabela de roteamento. Esse processo ocorre de forma iterativa, até que todos os nós sensores sejam descobertos. Em seguida, inicia-se a fase de operação, com o monitoramento da rede e a reconfiguração das rotas, caso sejam detectadas degradações na qualidade dos parâmetros avaliados. O fluxograma da Figura 20 demonstra o funcionamento geral do algoritmo.

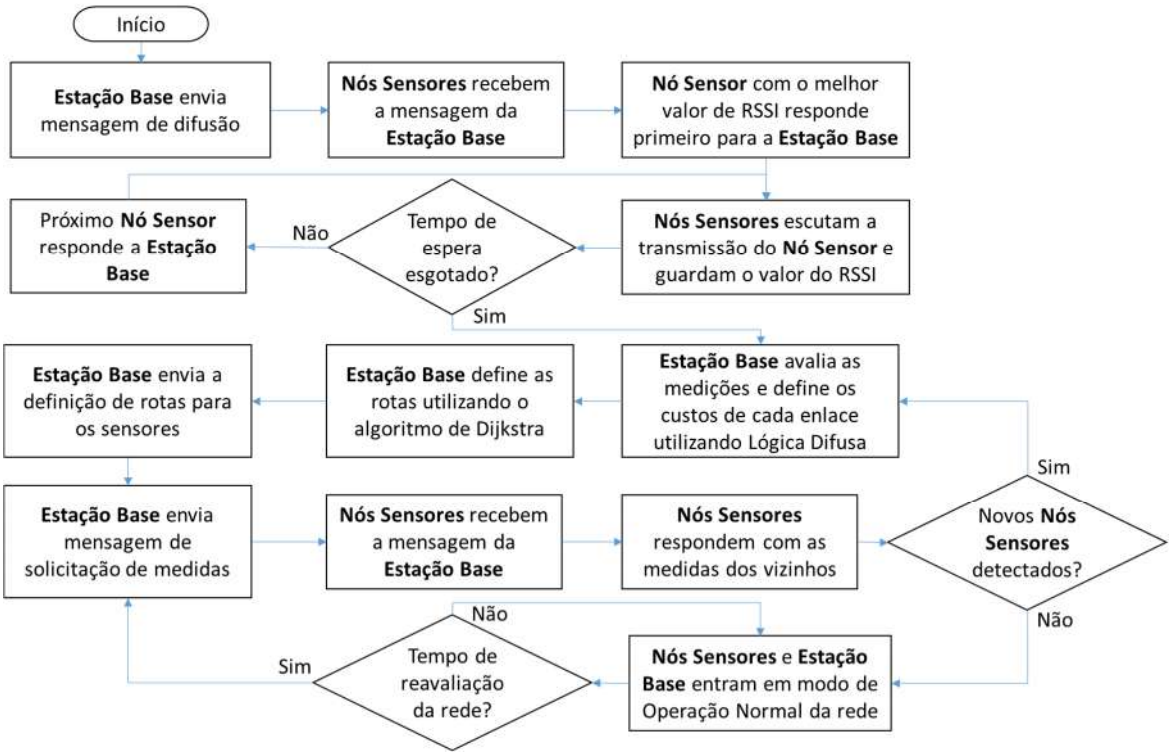


Figura 20: Fluxograma do funcionamento geral do algoritmo

5.2.1 Inicialização e Configuração

Fase de Inicialização e Configuração - a estação base envia uma mensagem de difusão para todos os nós sensores na rede. Os nós sensores

utilizam esta mensagem para avaliar a sua RSSI com a estação base. Cada nó sensor perto da estação base, e com melhor RSSI, responde à mensagem. A fim de evitar colisões, os nós sensores esperam durante um intervalo de tempo com base na RSSI medida em relação a estação base antes de enviar a resposta. Os nós sensores vizinhos escutam as respostas e mantêm os valores de RSSI de outros nós sensores em um banco de dados local. A estação base aguarda a resposta de cada nó sensor, até que um tempo limite expire. Em seguida, a estação base envia uma nova mensagem de difusão solicitando as medições de RSSI dos vizinhos de cada nó sensor. Os nós sensores respondem a estação base com as informações de seus vizinhos. Nesse momento, a estação base identifica a existência de outros nós sensores, cujas mensagens de resposta não haviam sido recebidas. Ao utilizar os dados recebidos, a estação base executa o procedimento de avaliação dos enlaces utilizando a lógica difusa e a etapa de definição através do algoritmo de Dijkstra, a fim de criar a tabela de roteamento. A estação base envia as informações de roteamento para cada nó sensor, que iniciam encaminhamento de pacotes de acordo com as novas rotas. Uma nova mensagem de difusão solicitando informações RSSI de vizinhos é enviada a fim de identificar novos nós sensores. Este processo é repetido até que nenhum novo sensor seja identificado. Observando a Figura 21 pode-se identificar a necessidade de seguidas iterações até que todos os sensores da rede tenham sido detectados, uma vez que a cada novo ciclo, os vizinhos da camada mais distante são reconhecidos.

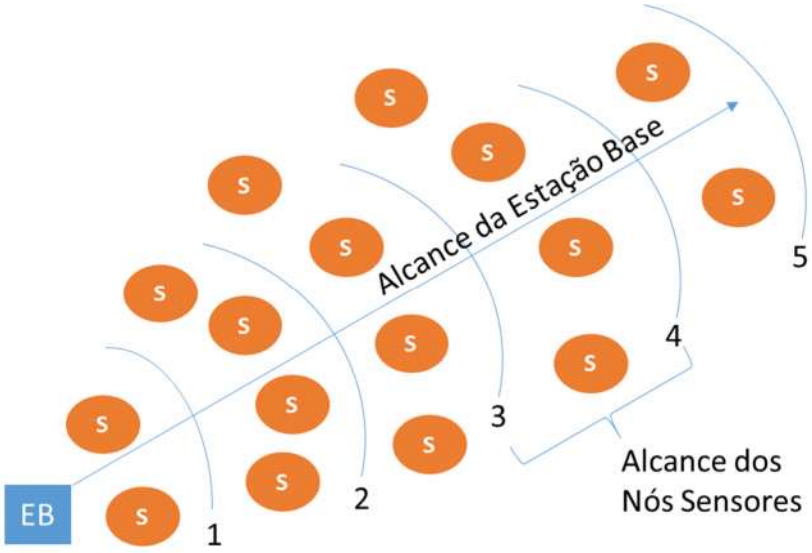


Figura 21: Alcance rádio dos nós sensores e da estação base

O Algoritmo 1 apresenta os procedimentos gerais da fase de configuração.

Algoritmo 1 CONFIGURAÇÃO DA REDE

1: **Função** configuração()
2: envia_mensagem(*msg_difusão*) //*envia mensagem para todos os nós sensores*
3: *tempo_limite* = determina_tempo_limite(*padrão*) //*inicia contagem para espera da resposta dos nós sensores*
4: [*info_rede, contador_sensores , sensores_vizinhos*] = espera_resposta(*tempo_limite*) //*recebe resposta dos nós sensores*
5: envia_mensagem (*requisição_rssi, contador_sensores*) //*envia requisição de leituras de RSSI dos sensores identificados*
6: *tempo_limite* = determina_tempo_limite(*contador_sensores*) //*inicia contagem para espera da resposta dos nós sensores, baseado no número de nós sensores identificados*
7: [*info_rede, contador_sensores , sensores_vizinhos*] = espera_resposta(*tempo_limite*) //*recebe resposta dos nós sensores*
8: define_rota(*info_rede, contador_sensores*) //*define as rotas baseado nas informações recebidas*
9: **enquanto** *senores_vizinhos<>nulo* **faça**
10: via_mensagem (*requisição_rssi, sensores_vizinhos*) // *envia requisição de leituras de RSSI dos sensores vizinhos identificados*
11: *tempo_limite* = determina_tempo_limite(*sensores_vizinhos*) //*inicia contagem para espera da resposta dos nós sensores, baseado no número de nós sensores identificados*
12: [*info_rede, contador_sensores , sensores_vizinhos*] = espera_resposta(*tempo_limite*) //*recebe resposta dos nós sensores*
13: define_rota(*info_rede, contador_sensores*) //*define as rotas baseado nas informações recebidas*
14: **Fim**

```
15: retorna (nulo)
16:
17: Função espera_resposta(tempo_limite)
18: enquanto não expirado(tempo_limite) faça  //espera pelas respostas
    enquanto não acaba o tempo
19:     info_sensor = recebe_resposta()  //recebe resposta de cada sensor
20:     tempo_limite = atualiza_tempo_limite()  //atualiza tempo limite
21:     [info_rede, contador_sensores , sensores_vizinhos] =
    verifica_novo_sensor(info_sensor) //verifica a existência de novos sensores e
    separa dados recebidos
22: Fim
23: retorna (info_rede, contador_sensores , sensores_vizinhos)
24:
25: Função define_rota(info_rede, contador_sensores)
26: info_caminho = lógica_difusa(info_rede)  //executa os procedimentos da lógica
    difusa
27: tabela_rotas = dijkstra(info_caminho)        //executa o algoritmo de Dijkstra
28: para cada sensor em contador_sensores //altera a tabela de rotas de cada
    sensor na rede
29:     muda_rota(sensor, tabela_rotas)  //envia mensagem de alteração de rotas
30: Fim
31: retorna(nulo)
```

5.2.2 Operação

Fase de operação - uma vez que a tabela de roteamento inicial está definida, a rede começa a fase de operação. Os nós sensores encaminham seus dados até a estação base seguindo as regras definidas na tabela de roteamento. Ao mesmo tempo, a informação de RSSI dos vizinhos e a taxa de erro de pacote são recolhidos e armazenados para uma verificação futura. Após um tempo predefinido, a estação base envia uma nova mensagem de difusão solicitando novas leituras. Os nós sensores transmitem para a estação base as informações de seus vizinhos que haviam sido previamente armazenadas. A estação base analisa as informações recebidas para identificar se é necessária uma mudança na tabela de roteamento. Caso exista a necessidade de adaptações na configuração inicial, seja pela inserção de novos sensores ou pela degradação da qualidade dos enlaces, uma etapa de redefinição é iniciada, seguindo os mesmos procedimentos da etapa de reconfiguração.

O Algoritmo 2 representa a fase de operação do protocolo.

Algoritmo 2 OPERAÇÃO DA REDE

```
1:  Função operação()
2:  situação_rede = OPERACIONAL
3:  tempo_limite = LIMIAR
4:  enquanto situação_rede = OPERACIONAL faça
5:      [info_rede, dados] = espera_dados_sensor(tempo_limite) //espera pelos
    dados dos sensores até que o tempo limite definido termine
6:      se dados <> nulo então  //encaminha pacotes para a camada de
    aplicação se houver
7:          camada_aplicação(dados)
8:      fim
9:      se expirado(tempo_limite) então  //verifica se o tempo limite definido
    terminou para verificação a situação da rede
10:         situação_rede = verifica_rede()
11:         se situação_rede = FALHA então //verifica se a rede está operacional
12:             configuração()      //caso exista falhas na rede e uma
    reconfiguração seja necessária
13:         fim
14:         tempo_limite = LIMIAR //atualzia variável de tempo limite
15:     fim
16: fim
17: retorna (nulo)
18:
19: Função verifica_rede()
20: envia_mensagem (requisição_rssi, contador_sensores)   //envia requisição de
        leituras de RSSI dos sensores
21: tempo_limite = determina_tempo_limite(contador_sensores)  //inicia
    contagem para espera da resposta dos nós sensores, baseado no número de nós
    sensores identificados
22: [info_rede, contador_sensores , sensores_vizinhos] =
    espera_resposta(tempo_limite) //recebe resposta dos nós sensores
23: se sensors_vizinhos > 0 então  //verifica se novos sensores foram adicionados
24:     retorna(FALHA)
25: fim
26: nova_info_caminho = lógica_fuzzy(info_rede)
27: se nova_info_caminho <> info_caminho then
28:     retorna(FALHA)
29: fim
30: retorna(OPERACIONAL)
```

6 Implementação e Execução

A implementação do algoritmo foi dividida em etapas a fim de possibilitar um desenvolvimento evolutivo da proposta. Inicialmente, definiu-se as diretrizes básicas de funcionamento do protocolo, identificando aspectos chaves que deveriam ser tratados, como a preocupação com a taxa de entrega de pacote, a reação a eventos na rede de sensores sem fio que influenciassem no desempenho geral da rede e a redução de complexidade do protocolo de roteamento, a fim de possibilitar a implementação em aplicações práticas.

A elaboração de um simulador permitiu a avaliação da proposta de forma dinâmica, uma vez que adaptações às diretrizes iniciais puderam ser desenvolvidas e testadas rapidamente, até que os resultados se apresentassem satisfatórios. Desta forma, criou-se uma espécie de crivo, que permitiu construir uma solução fundamentada em análise de resultados, sendo necessária, posteriormente, apenas a comprovação dos dados de simulação por meio dos experimentos.

Uma vez desenvolvido e avaliado por meio de simulação, o protocolo foi submetido a testes práticos. Foram desenvolvidas adaptações no *firmware* da plataforma Radiuino e elaborado um software controlador responsável pela gerência da rede. Os testes práticos ocorreram em diversas iterações, a fim de verificar o desempenho do algoritmo em situações reais.

6.1 Simulação

Para avaliar o desempenho do algoritmo proposto em comparação com um protocolo baseado em RSSI foi desenvolvido um simulador utilizando o software SciLab (ENTERPRISES, 2012). Os valores de RSSI foram gerados considerando o fenômeno de *shadowing* (sombreamento), baseado na distância entre os nós sensores em cada um dos cenários. Utilizou-se também um valor aleatório X_σ seguindo uma distribuição normal de média nula e desvio padrão σ igual a sete (ANDERSEN, 1995), a fim de simular a variação do ambiente. Dessa forma, o

simulador respeitou o modelo descrito por Andersen (1995), através da Equação (4) que representa a média da potência de sinal recebido em função da distância.

$$\mathrm{P(d)} = \mathrm{P(d_0)} - 10\eta \log\left(\frac{\mathrm{d}}{\mathrm{d_0}}\right) + \mathrm{X_\sigma} \quad (4)$$

Em que:

- ***P(d)*** representa a média da potência recebida a uma distância *d*;
- ***P(d_0)*** representa a média da potência recebida a uma distância de referência conhecida d_0;
- $\boldsymbol{\eta}$ representa um coeficiente de perda, obtido de acordo com o tipo de ambiente;
- $\boldsymbol{X_\sigma}$ representa uma variável aleatória de distribuição normal de média nula e desvio padrão σ;

Os valores de η e σ podem ser obtidos por meio do estudo do ambiente ou pela utilização de valores predefinidos de acordo com o tipo de ambiente (ANDERSEN, 1995). A critério de simulação, optou-se por utilizar os valores obtidos de estudos prévios, como os apresentados por Seidel (1992) e Andersen (1995):

$$\eta = 4 \quad e \quad \sigma = 7 \text{ dB}$$

O desvio padrão foi calculado com base nos valores de RSSI gerados. A taxa de erro de pacote foi introduzida de forma aleatória e independente para cada enlace. A simulação contemplou todos os passos de execução do algoritmo: coleta, avaliação e definição. Para tanto, foram desenvolvidas diversas funções, que estão disponíveis para consulta no Anexo A – Implementação do Simulador.

Para a execução da simulação, foram definidos seis cenários, variando apenas a quantidade total de nós sensores e o tamanho total do ambiente. A distância entre os nós sensores foi mantida como uma constante e fixada em 3 m. A escolha dos cenários de teste teve como objetivo identificar o comportamento do algoritmo de acordo com a quantidade de nós sensores na rede. A Tabela 3 apresenta as características de cada cenário.

Tabela 3: Cenários de Teste

Cenários	Quantidade de Nós Sensores	Área do Ambiente
S01	**8**	**36 m²**
S02	**24**	**144 m²**
S03	**48**	**324 m²**
S04	**80**	**576 m²**
S05	**120**	**900 m²**
S06	**160**	**1296 m²**

Os valores obtidos para todas as variáveis analisadas foram utilizados ao mesmo tempo para o algoritmo proposto e para a solução baseada em RSSI. Dessa forma, foi possível observar o comportamento das duas soluções em situações idênticas. Para estabelecer um fator de comparação entre o algoritmo proposto e a solução baseada em RSSI (RBF) foi definido o processo apresentado na equação (5):

$$F = \frac{\sum_{\mathrm{i}}^{n} 1-(S_{Fuzzy_i} - S_{RBF_i})}{n} \ , \ [-1 \leq F \leq 1] \quad (5)$$

Em que:

- **F** representa o parâmetro de comparação;
- **n** representa o número de nós sensores;
- $\mathbf{S_{Fuzzy}}$ representa a taxa de sucesso de um caminho específico para a solução proposta;
- $\mathbf{S_{RBF}}$ representa a taxa de sucesso de um caminho específico para o algoritmo baseado em RSSI.

As taxas de sucesso podem ser descritas como apresentado em (6) e (7):

$$S_{Fuzzy} = 1 - PEP_{Fuzzy} \ , \ [0 \leq PEP_{Fuzzy} \leq 1] \quad (6)$$

$$S_{\mathrm{RBF}} = 1 - PEP_{\mathrm{RBF}} \ , \ [0 \leq PEP_{\mathrm{RBF}} \leq 1] \quad (7)$$

Em que:

- **PEP_{Fuzzy}** representa a taxa de erro de pacote de um caminho específico para a solução baseada em lógica difusa.
- **PEP_{RBF}** representa a taxa de erro de pacote de um caminho específico para a solução baseada em RSSI.

A taxa de erro de pacote de um caminho específico é verificada por meio do acumulo de erros de todos os enlaces que compõem o caminho até o fim da rota, tal como descrito em (8):

$$PEP = 1 - [\prod_{i=1}^{n}(1 - PER_i)] \quad (8)$$

Em que:

- **PEP** representa a taxa de erro de pacote de um caminho específico;
- **n** representa o número de enlaces que compõe o caminho;
- **PER** representa a taxa de erro de pacote de um enlace específico.

O parâmetro **F** representa um fator de comparação entre a estratégia de roteamento baseada em lógica difusa e a solução baseada em RSSI. Os valores de F entre [-1,0[demonstram que a solução baseada em RSSI é mais eficiente. Já para F = 0, temos que as soluções são semelhantes e apresentam os mesmos resultados. Por fim, para valores de F entre]0,1], temos que a solução baseada em lógica difusa tem um desempenho melhor em comparação ao roteamento baseado em RSSI.

Para cada cenário definido, foram executadas 100 iterações da simulação a fim de gerar os valores do fator F e avaliar a consistência da solução proposta. A Figura 22: Curva de desempenho apresenta os resultados finais para cada cenário após as 100 iterações. As curvas θ1 e θ2 são, respectivamente, os valores inferiores e superiores do intervalo de confiança e FM é a média de F para as 100 iterações em cada cenário.

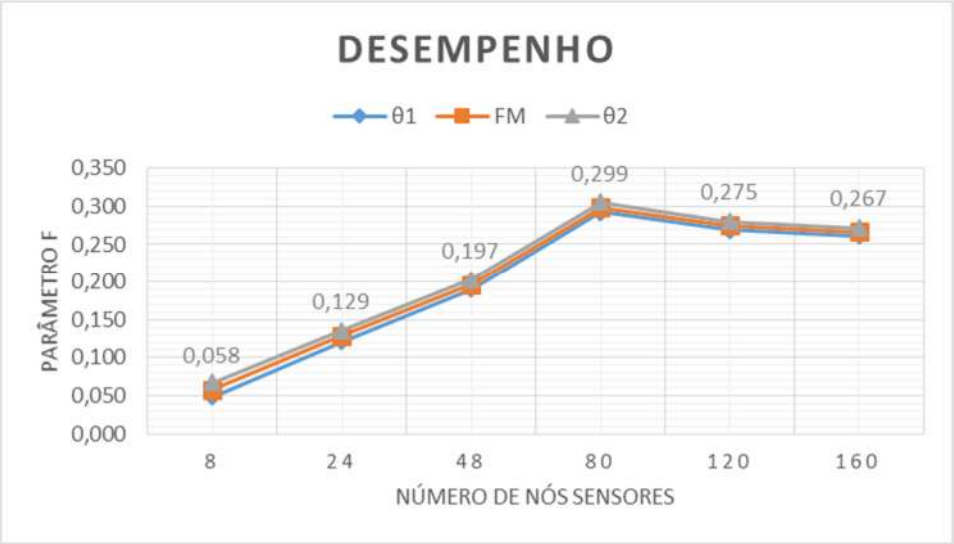


Figura 22: Curva de desempenho

Verifica-se que o algoritmo proposto apresenta sempre um desempenho melhor do que a estratégia baseada unicamente nas medições de RSSI, uma vez que F > 0 em todos os cenários. Isto significa que os enlaces selecionados através da avaliação por meio da lógica difusa das variáveis de RSSI, desvio padrão e taxa de erro de pacote apresentavam menos erros, resultando em uma maior confiabilidade da rede. Observa-se também que à medida que se eleva o número de nós sensores na rede, obtêm-se resultados ainda melhores para o algoritmo proposto. No entanto, quando o número de nós sensores sobe acima de 100 nós, há uma queda no desempenho da solução. Isto ocorre devido ao aumento da quantidade de rotas de múltiplos saltos. Essa necessidade de múltiplos saltos deve-se a existência de mais nós sensores e ao aumento da área total do cenário de teste, implicando na degradação do desempenho da rede. O aumento da incidência de múltiplos saltos atua negativamente na probabilidade de erro de pacote, como descrito na equação 8 acima.

O algoritmo proposto também apresenta melhor desempenho em termos de número total de saltos para o nó sensor mais distante. Como evidenciado pela Figura 23, a solução baseada em lógica difusa usa menos saltos para chegar à estação base. Isto também é observado na Figura 24, que representa o número médio de saltos em cada cenário tanto para o algoritmo proposto, como para a solução baseada em RSSI.

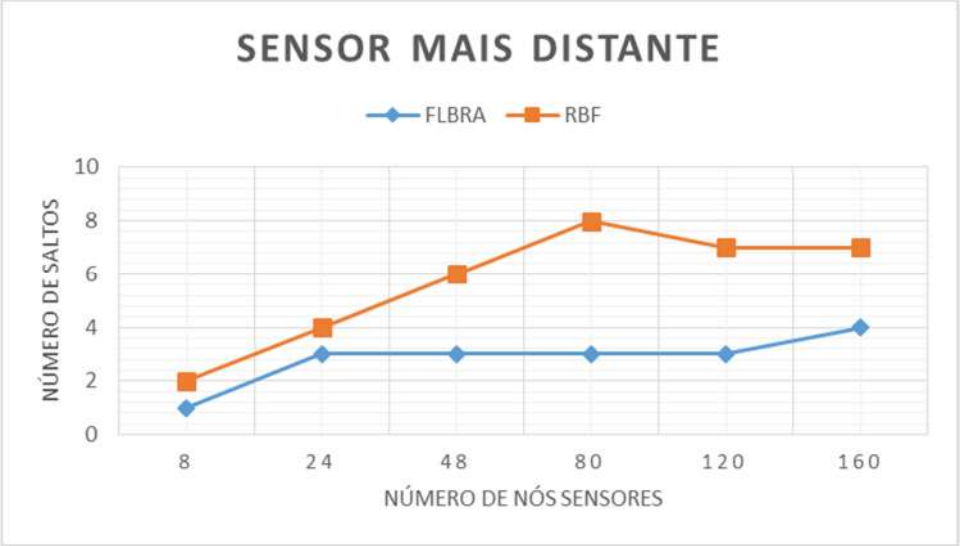


Figura 23: Número de saltos até o nó sensor mais distante

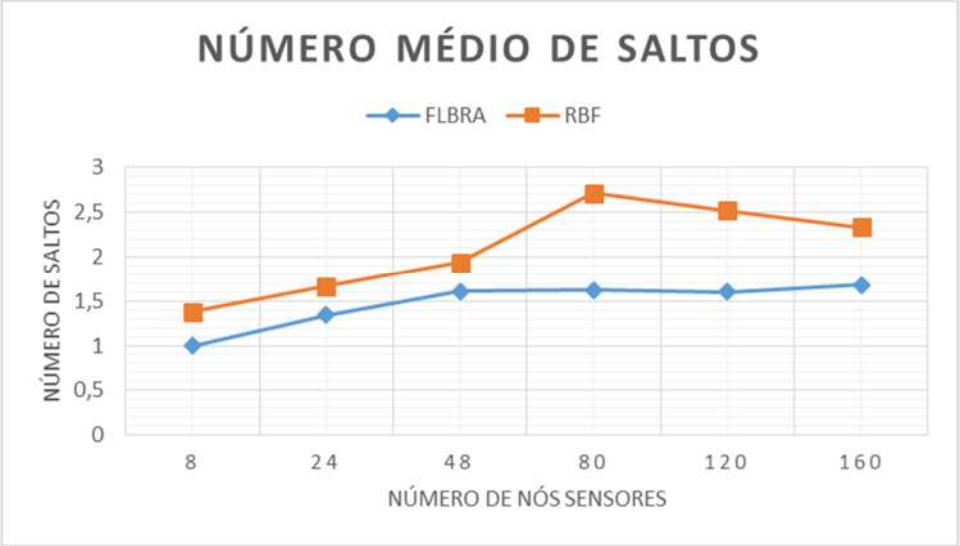


Figura 24: Número médio de saltos

Ao buscar as melhores rotas para o encaminhamento de pacotes, o algoritmo proposto também evita o excesso de saltos, uma vez que a cada novo enlace agregado ao caminho, aumenta-se a probabilidade de erro.

6.2 Experimento

A fim de possibilitar a execução de testes práticos, foram desenvolvidas adaptações no *firmware* da plataforma Radiuino. Precisou-se elaborar pacotes de controle com funções específicas que permitissem a obtenção das informações necessárias das camadas de acesso e rede, bem como alterações em parâmetros da camada de rede. Fez-se necessário também a implementação de adaptações na camada física que permitissem o monitoramento das trocas de pacotes dos nós sensores vizinhos. A aplicação resultante caracteriza-se por uma solução *crosslayer*, uma vez que a tabela de roteamento é diretamente influenciada por informações obtidas de camadas inferiores.

Desenvolveu-se também um software gerente responsável pela supervisão da rede e definição da tabela de roteamento, seguindo tanto a solução baseada em RSSI (RBF), como a proposta apresentada nesse trabalho. Dessa forma, possibilitou-se a avaliação dos resultados práticos de maneira comparativa, seguindo o mesmo modelo adotado na simulação.

6.2.1 Plataforma Radiuino

Radiuino é uma plataforma de código aberto para prototipação de Redes de Sensores Sem Fio baseada em Arduino. Trata-se de um conjunto de *Hardware* e *Software*, implementados sob um microcontrolador MegaAVR da Atmel e um transceptor CC1101 da Texas Instrument, podendo ainda incorporar outros módulos de *hardware* de acordo com a aplicação, como sensores de presença, luminosidade, temperatura, distância etc (BRANQUINHO, 2014). Um Kit de desenvolvimento Radiuino é composto por duas placas BE900, operando em 915 MHz, uma placa base UartSBee, para conexão com computador, uma placa sensor DK101, uma fonte de alimentação e um Cabo USB-Mini, como apresentado na Figura 25.

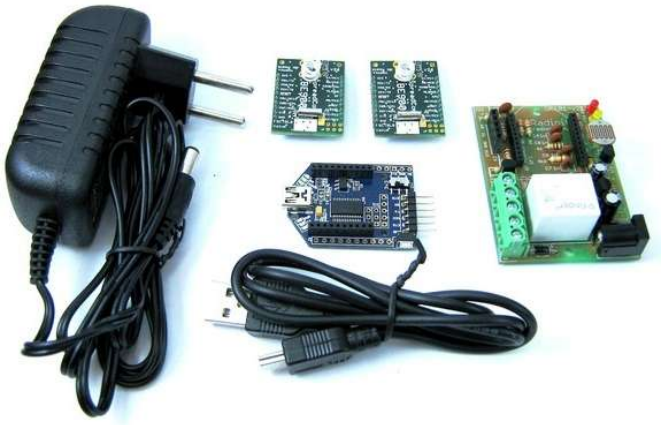


Figura 25: Kit de desenvolvimento Radiuino.

Fonte: BRANQUINHO, 2014

O Radiuino possui uma implementação de *Software* baseada em um modelo de cinco camadas: Física, Acesso, Rede, Transporte e Aplicação. É possível ter acesso e modificar o código em qualquer uma das camadas que compõe a plataforma. A flexibilidade do Radiuino permite a implementação de protótipos de RSSF de aplicações diversas, tornando-se ideal para o desenvolvimento de novas soluções de roteamento.

6.2.2 Adaptações do firmware

O *firmware* da plataforma Radiuino está estruturado em um modelo de cinco camadas (física, acesso, rede, transporte e aplicação), o que possibilita uma melhor organização e coesão dos módulos. Pressman (2006) afirma que as funcionalidades implementadas dentro de um módulo devem ser limitadas pelas responsabilidades que o módulo exerce no conjunto do software, facilitando reuso e manutenção das funções desenvolvidas. Contudo, a aplicação do conceito de *cross-layer* faz uma redefinição das responsabilidades de cada camada afetada, uma vez que permite a atuação de camadas inferiores na tomada de decisão de camadas superiores. Como apresentado por Srivastava (2005), soluções *cross-layer* tendem a ser exclusivas, desenvolvidas para resolver problemas específicos.

As adaptações desenvolvidas no *firmware* básico do Radiuno tiveram como objetivo permitir a avaliação de parâmetros específicos e a execução de novas rotinas a partir dos resultados obtidos dessas avaliações. Foram criadas novas variáveis de controle e uma divisão da memória EEPROM para o armazenamento da tabela de rotas dos sensores, como mostra a Figura 26.

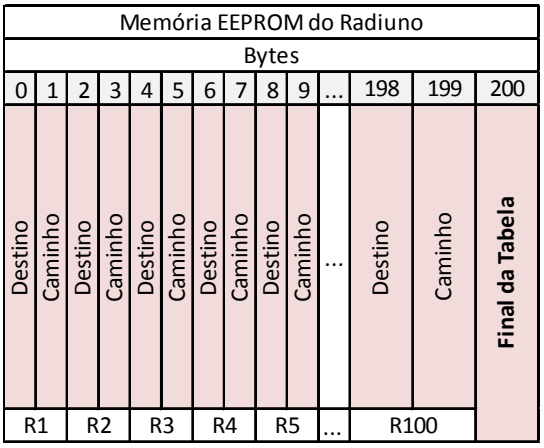


Figura 26: Mapeamento da Memória EEPROM

Também foram definidos novos pacotes de controle, a fim de permitirem a obtenção dos dados armazenados nos sensores e a reconfiguração das tabelas de roteamento. Os novos pacotes de controle desenvolvidos foram:

a) Pacote de Alteração de Rota: PAR

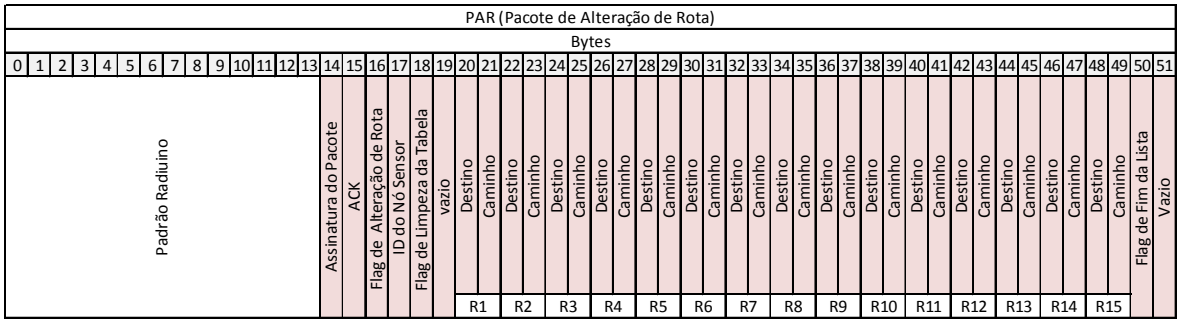


Figura 27: Pacote de Alteração de Rota

b) Pacote de Leitura da Tabela de Rotas: PLTR

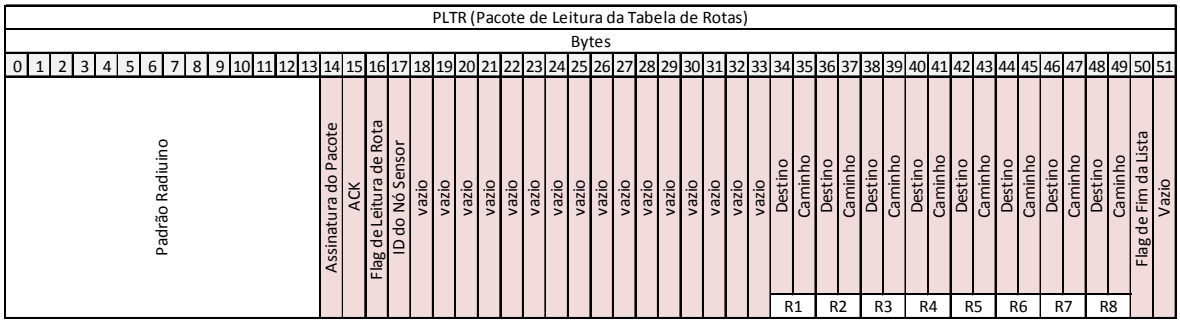


Figura 28: Pacote de Leitura da Tabela de Rotas

c) Pacote de Leitura de Dados da Rota Pré-definida: PLDRP

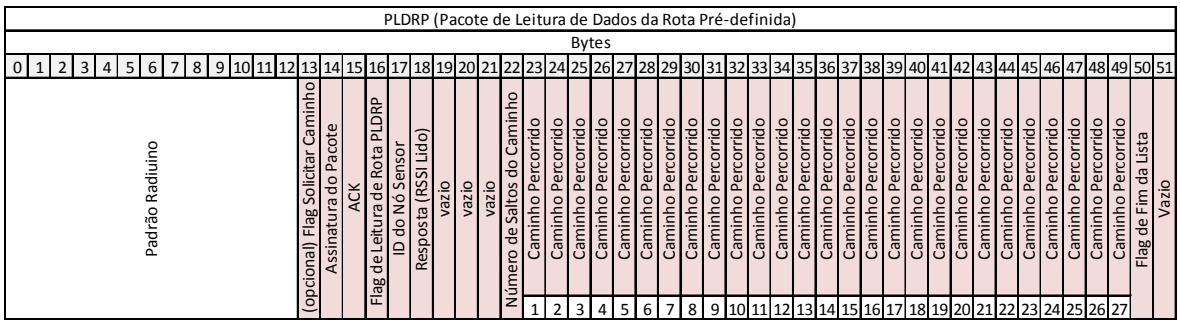


Figura 29: Pacote de Leitura de Dados da Rota Pré-definida

d) Pacote de Leitura de Dados de Rota Específica: PLDRE

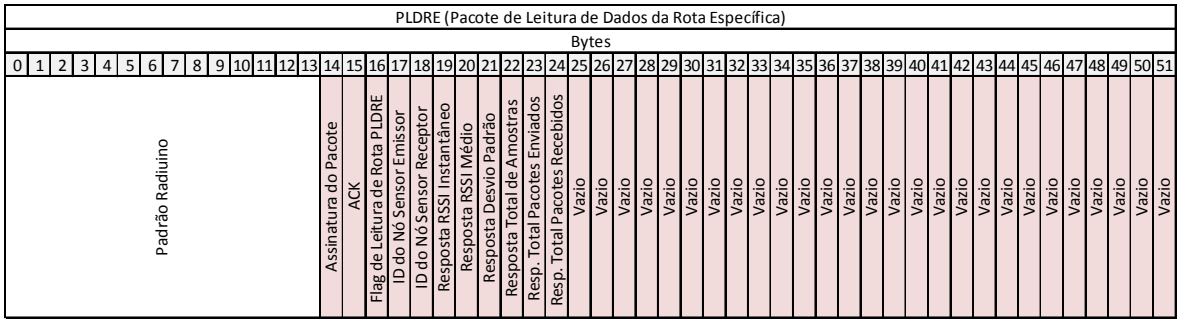


Figura 30: Pacote de Leitura de Dados da Rota Específica

Para todos os pacotes transmitidos, reserva-se o *byte* 14 para a Assinatura do Pacote. Todos os pacotes recebem um número aleatório que o identifica na rede. Essa assinatura é utilizada para verificar que o pacote foi corretamente transmitido e recebido.

As leituras de RSSI entre os nós sensores ocorrem na camada física. Quando o nó sensor detecta uma transmissão, a informação do pacote é obtida, a fim de verificar a quem se destina. Caso o pacote tenha como destino o próprio nó sensor, a informação é elevada para a camada superior, para que seja devidamente tratada. Caso contrário, o nó sensor apenas armazena a leitura de RSSI, para que possa manter um histórico de leituras de RSSI entre ele e o nó sensor emissor. Observa-se ainda se o pacote detectado possui uma assinatura conhecida. Caso o nó sensor reconheça a assinatura, significa que o pacote foi emitido por ele próprio em um momento anterior. Dessa forma, o nó sensor confirma que a transmissão da resposta obteve sucesso e marca a transmissão como concluída.

Por se tratar de uma solução centralizada, o desenvolvimento das adaptações procurou tornar o Radiuino independente da proposta de roteamento, de forma que fosse possível executar tanto a solução baseada em RSSI, quanto a solução baseada em Lógica Difusa, sem a necessidade de alteração do *firmware*, para troca de solução. A implementação completa das modificações realizadas no *firmware* da plataforma Radiuino podem ser encontradas no Anexo B – Modificações do Firmware Radiuino.

6.2.3 Software Gerente

Por se tratar de uma solução centralizada, a inteligência do sistema concentra-se na Estação Base, que contém o Software Gerente (SG) implementado em um computador e conectado a um nó sensor que realiza a interface com os demais nós sensores. A Figura 31 exemplifica a arquitetura da Estação Base.

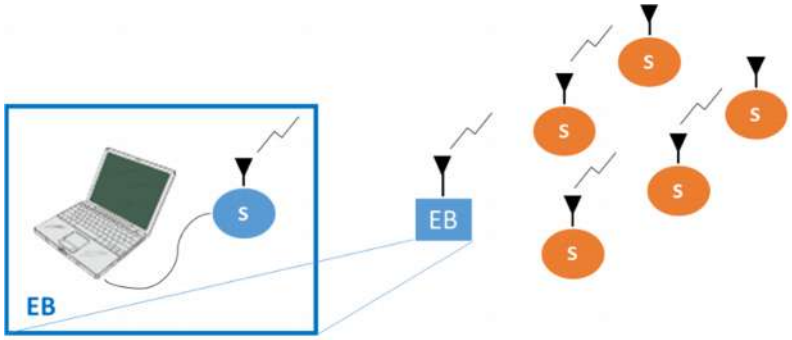


Figura 31: Composição da Estação Base

O Software Gerente é responsável pela gestão da rede. Os processos de inicialização e manutenção são executados a partir do Software Gerente. Para o experimento foram desenvolvidas as duas soluções de roteamento, tanto a proposta baseada em Lógica Difusa, quanto a proposta baseada em RSSI. Dessa forma, foi possível realizar a execução das duas soluções nos mesmos cenários e avaliar os resultados de maneira comparativa.

Antes do início do experimento, o SG permite escolha de qual proposta deve ser executada, a fim de possibilitar a troca de soluções de maneira simplificada, sem a necessidade de alteração do *firmware* dos nós sensores. Permite-se também que sejam definidos limites para a quantidade de nós sensores na rede. Dessa forma é possível realizar testes com quantidades diferentes de nós sensores, sem a necessidade de liga-los ou desliga-los fisicamente. É possível ainda definir intervalos diferentes para a etapa de verificação, que precede a reavaliação da rede, o que permite reduzir o tempo de execução dos testes.

É por meio do SG que todas as coletas de dados são realizadas. O programa envia os pacotes de controle solicitando os dados e os trata de acordo com as configurações iniciais. Uma vez descobertas as melhores rotas, o SG envia novos pacotes de controle a fim de alterar as tabelas de roteamento de cada sensor. Esse processo é realizado tanto para a solução baseada em RSSI, quanto para a solução baseada em Lógica Difusa. A implementação completa do Software Gerente encontra-se no Anexo C – Implementação do Software Gerente.

6.2.4 Execução do Experimento

A execução do experimento considerou a definição de quatro cenários de teste. Os cenários foram realizados tanto para a solução baseada em RSSI, quanto para a proposta de solução baseada em lógica difusa. Os testes realizados procuraram identificar uma tendência de comportamento, respeitando os limites físicos e de recursos disponíveis.

A descrição dos cenários pode ser encontrada na Tabela 4.

Tabela 4: Cenários de Teste do Experimento

Cenário	Quantidade de Nós Sensores	Tempo de Execução	Solução
S01	4	4h	RBF e Lógica Difusa
S02	6	4h	RBF e Lógica Difusa
S03	8	4h	RBF e Lógica Difusa
S06	14	4h	RBF e Lógica Difusa

Os cenários foram construídos em um dos laboratórios da PUC Campinas, por se tratar de um ambiente com grande movimentação de pessoas e alterações físicas regulares. Os sensores foram distribuídos entre os dois pisos do laboratório, cobrindo todo o espaço útil disponível, como mostram as figuras 30 a 34.

Figura 32: Foto do Ambiente de Teste - Composição da Estação Base e Nó Sensor

Figura 33: Foto do Ambiente de Teste – Distribuição dos Nós Sensores no Piso 1

Figura 34: Foto do Ambiente de Teste – Nó Sensor na escada de acesso ao Piso 2

Figura 35: Foto do Ambiente de Teste – Distribuição dos NS no Piso 2, parte A

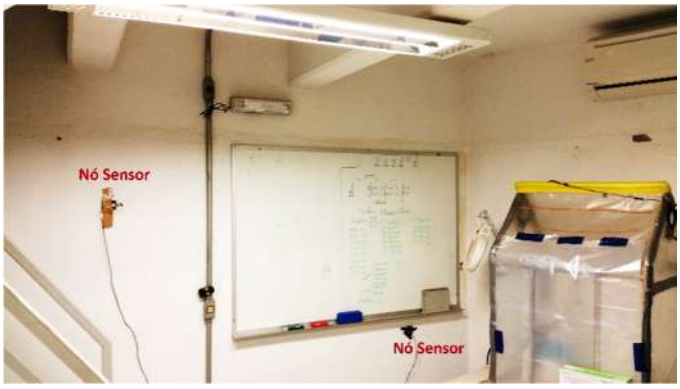


Figura 36: Foto do Ambiente de Teste – Distribuição dos NS no Piso 2, parte B

Cada cenário foi testados por um período de 4 horas, com requisições de novas leituras a cada 2 segundos e reavaliação da situação da rede a cada 10 minutos. Optou-se pela reavaliação a cada 10 minutos, para que fosse possível observar os novos caminhos criados a cada modificação no ambiente. Por se tratar de um laboratório de uso comum entre os alunos da graduação e pós-graduação, é intensa a movimentação de pessoas, com constantes mudanças no número total de indivíduos no ambiente. A espera de 10 minutos antes da reavaliação permitiu coletar dados sobre o comportamento da rede e as trocas de pacotes durante o período que seria dedicado ao tráfego normal (dados de medições dos sensores: luminosidade, temperatura, umidade etc).

Para a avaliação dos resultados, definiu-se um modelo de coleta de dados em arquivo texto em um formato que permitisse a leitura dos dados por uma rotina em SciLab, possibilitando que a mesma análise feita para a simulação pudesse ser realizada para os dados experimentais. Abaixo é possível verificar uma entrada do arquivo de *log* do experimento:

11/12/2014,10:20:34; RM; 0.0->0.0:0;0.0->1.0:2;0.0->2.0:2;0.0->3.0:4;0.0->4.0:4;1.0->0.0:2;1.0->1.0:1;1.0->2.0:2;1.0->3.0:2;1.0->4.0:2;2.0->0.0:0;2.0->1.0:1;2.0->2.0:2;2.0->3.0:4;2.0->4.0:4;3.0->0.0:4;3.0->1.0:4;3.0->2.0:4;3.0->3.0:3;3.0->4.0:4;4.0->0.0:0;4.0->1.0:2;4.0->2.0:2;4.0->3.0:3;4.0->4.0:4; 0.0->0.0: 0.0 | 0.0 | 0.0 | 0.0 | 0;0.0->1.0: -59.5 | -59.5 | 3.6162028534 | 0.978524405054 | 0;0.0->2.0: -26.5 | -26.5 | 3.33541601603 | 0.92513368984 | 0;0.0->3.0: -68.5 | -68.5 | 3.65718470958 | 1.0 | 0;0.0->4.0: -52.5 | -52.5 | 4.51386752132 | 0.0422535211268 | 0;1.0->0.0: -60.5 | -60.0 | 3.6162028534 | 0.952162095379 | 0;1.0->1.0: 0.0 | 0.0 | 0.0 | 0.0 | 0;1.0->2.0: 47.5 | 47.0 | 4.41588043316 | 0.431924882629 | 0;1.0->3.0: RSSI | MEDIA | DP | PER | SALTOS;1.0->4.0: RSSI | MEDIA | DP | PER | SALTOS;2.0->0.0: -26.5 | -27.0 | 3.4278273002 |

0.915789473684 | 0;2.0->1.0: 50.5 | 49.0 | 3.7621597725 | 0.713147410359 | 0;2.0->2.0: 0.0 | 0.0 | 0.0 | 0.0 | 0;2.0->3.0: RSSI | MEDIA | DP | PER | SALTOS;2.0->4.0: -67.5 | -68.5 | 4.16833300013 | 0.0 | 0;3.0->0.0: -67.5 | -68.0 | 4.03112887415 | 0.563271395633 | 0;3.0->1.0: RSSI | MEDIA | DP | PER | SALTOS;3.0->2.0: RSSI | MEDIA | DP | PER | SALTOS;3.0->3.0: 0.0 | 0.0 | 0.0 | 0.0 | 0;3.0->4.0: 48.0 | 48.0 | 5.0 | 0.544554455446 | 0;4.0->0.0: -52.0 | -52.5 | 4.25734659148 | 0.041095890411 | 0;4.0->1.0: RSSI | MEDIA | DP | PER | SALTOS;4.0->2.0: -69.0 | -69.5 | 4.70372193056 | 0.478873239437 | 0;4.0->3.0: RSSI | MEDIA | DP | PER | SALTOS;4.0->4.0: 0.0 | 0.0 | 0.0 | 0.0 | 0;

Uma entrada do arquivo de *log* pode ser dividida em quatro blocos de dados. A primeira parte, destacada em verde, representa a data e horário da realização da medida; a segunda parte, destacada em amarelo, representa se houve ou não alteração de rota, sendo “RM” para Rota Mantida e “RA” para Rota Alterada; a terceira parte, em destaque azul, representa o caminho a ser realizado pelo pacote a partir de cada nó sensor, respeitando o formato [Origem]:[Destino]->[Caminho]; a quarta parte, destacada em cinza, representa a leitura dos valores de RSSI instantâneo (valor da última leitura feita), média dos valores de RSSI lidos, desvio padrão dos valores RSSI, taxa de erro de pacote e número de saltos realizados até a estação base (enlaces diretos com a estação base contam como um). Os dados de um enlace, por exemplo, entre o Nó Sensor 1 e o Nó Sensor 2, são representados de acordo com o formato a seguir: [Nó Sensor 1]->[Nó Sensor 2]:RSSI | MEDIA | DP | PER | SALTOS. Quando não há valores no *log*, mas apenas uma indicação do campo, significa que o enlace em questão é inexistente.

Os resultados obtidos após a execução dos cenários e a análise realizada de acordo com os parâmetros definidos durante as simulações, permitiu identificar que a proposta de roteamento baseada em Lógica Difusa obteve desempenho superior a solução baseada em RSSI também nos testes em ambiente real. Os detalhes da implementação da rotina de análise dos dados experimentais podem ser encontrados no Anexo D – Implementação da Rotina de Análise de Dados.

7 Análise de Resultados

Como já apresentado na seção 6.1, a avaliação dos resultados foi realizada de maneira comparativa entre a solução proposta baseada em lógica difusa e a solução baseada em RSSI. Seguindo o mesmo processo de análise, realizou-se testes experimentais, a fim de identificar o comportamento da solução em cenários reais, bem como confrontar os resultados obtidos em simulação.

Com relação ao desempenho do protocolo proposto, pode-se verificar na Figura 37 que F > 0 para todos os cenários executados. Como visto anteriormente, valores de F entre]0,1] representam um desempenho melhor para o protocolo proposto em relação ao protocolo baseado em RSSI. Percebe-se ainda que o desempenho do protocolo proposto melhora a medida que se aumenta a quantidade de nós sensores na rede. Esse fenômeno ocorre pois eleva-se também o número de enlaces e as possibilidades de escolher caminhos melhores, com relação aos parâmetros analisados. O protocolo baseado em RSSI verifica e escolhe somente os enlaces com os melhores valores de RSSI, não observando fatores importantes, como a variação do sinal – um enlace instável com grande variação de sinal, ou seja, com desvio padrão é elevado, pode estar oculto por um RSSI médio aceitável – no protocolo RBF não há preocupação também com a taxa de perda de pacote, fazendo com que os caminhos com número elevado de perdas continuem sendo utilizados.

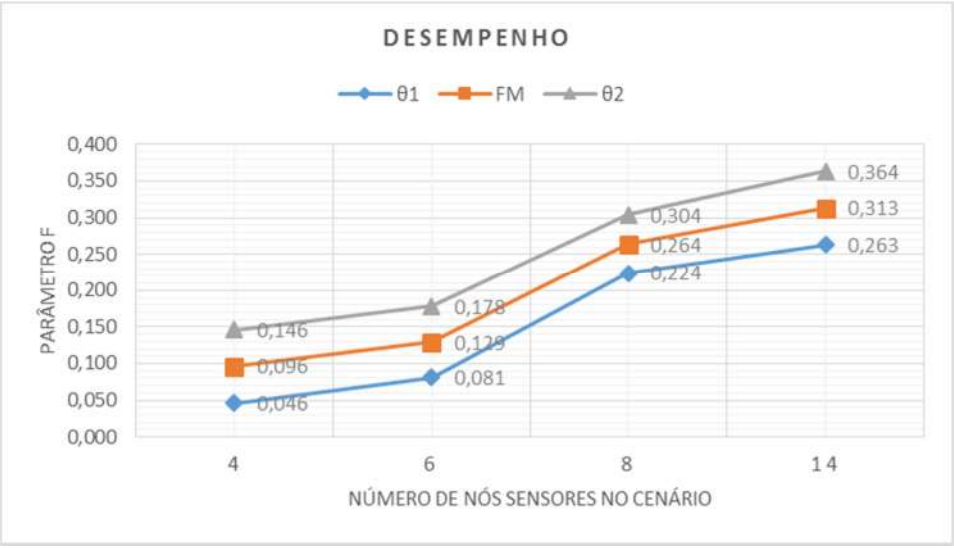


Figura 37: Curva de desempenho – Parâmetro F Médio e limites do intervalo de confiança.

Com relação ao número médio de saltos, pode-se observar na Figura 38 que o valor permaneceu praticamente constante para o protocolo proposto (FLBRA), utilizando menos saltos do que a proposta baseada em RSSI (RBF). Como já verificado nos resultados de simulação, ao se elevar o número de saltos em uma rota até a Estação Base, eleva-se também a probabilidade de perda de pacote, já que a probabilidade de sucesso diminui a cada novo salto.

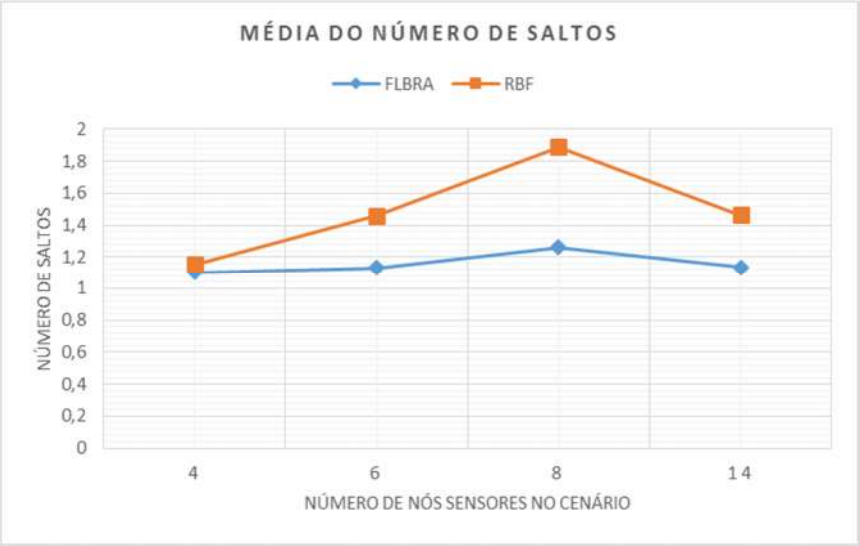


Figura 38: Número médio de saltos até a Estação Base.

Ainda com relação ao número de saltos, pode-se verificar na Figura 39 o número de saltos do nó sensor mais distante na rede. Percebe-se que para o protocolo proposto o número permaneceu em três saltos em todos os cenários. Isso significa que na pior situação de rede o algoritmo precisa de apenas três saltos para chegar até a Estação Base. Esse valor representa o pico de distância de um nó sensor em um dado momento da rede. O número de saltos necessários para chegar à Estação Base varia ao longo do tempo, de acordo com as reavaliações da rede.

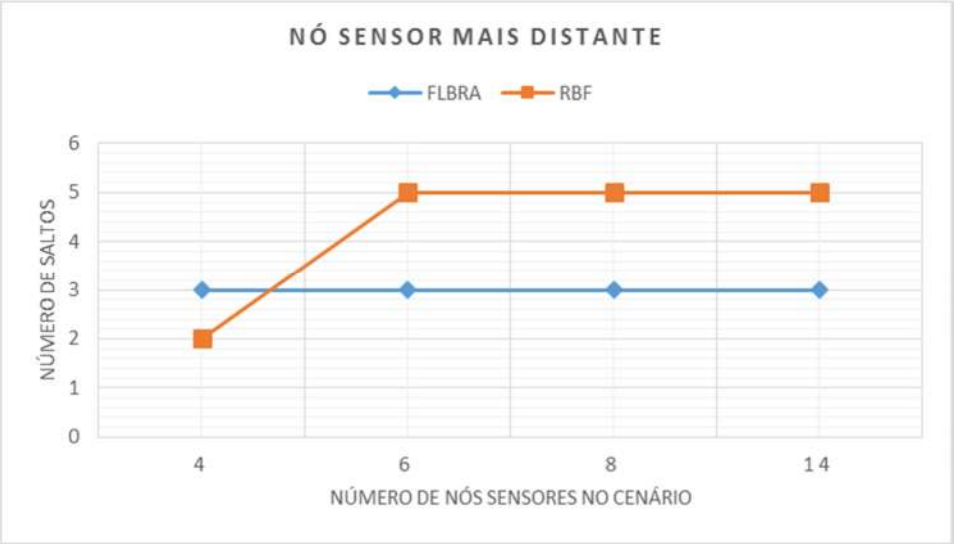


Figura 39: Número de saltos do nó sensor mais distante.

Em uma análise comparativa entre Experimento e Simulação, pode-se verificar na Figura 40 que o protocolo proposto apresenta um resultado ainda mais satisfatório em condições reais. Isso ocorre pois durante os testes em simulação o ambiente foi modelado apenas considerando o nível de RSSI em função da distância, um desvio padrão pré-definido e um valor aleatório, como visto na seção 6.1. Não foram considerados outros fatores que pudessem influenciar o resultado, como períodos de estabilidade na rede ou a disposição dos sensores em relação a estação base. Dessa forma, o distanciamento dos resultados deve-se às diferenças de condições dos testes simulados e experimentais.

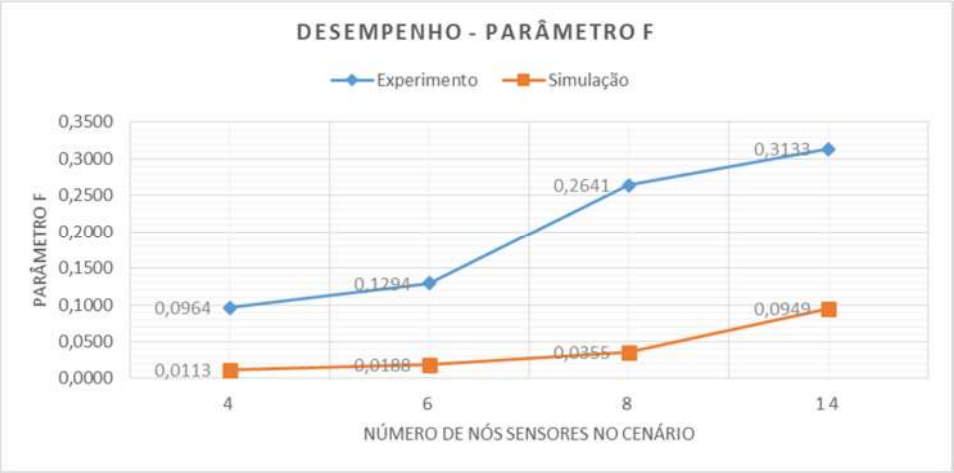


Figura 40: Número de saltos do nó sensor mais distante.

O desvio obtido na análise dos resultados experimentais e de simulação possui um valor médio de 82%, como pode ser observado na Tabela 5. Contudo, o valor alto para o desvio não invalida a aplicação, uma vez que a simulação foi desenvolvida para verificar a viabilidade e desempenho da proposta em relação a solução baseada em RSSI.

Tabela 5: Desvio entre Dados Experimentais e de Simulação

Cenários	Número de Nós Sensores	Parâmetro F		
		Experimento	Simulação	Desvio em %
S01	4	0,0964	0,0113	88,23%
S02	6	0,1294	0,0188	85,45%
S03	8	0,2641	0,0355	86,57%
S06	14	0,3133	0,0949	69,71%

Para que o mesmo desempenho dos experimentos fosse observado também durante a simulação, seria necessário desenvolver um estudo mais aprofundado sobre o ambiente de teste, coletando todas as informações necessárias para que fosse possível reproduzir as mesmas condições durante a simulação. No entanto, o foco do trabalho seria desviado, já que o objetivo não é desenvolver um simulador do ambiente utilizado para o teste.

8 Conclusão

Este trabalho propôs o desenvolvimento de um novo protocolo de roteamento para redes de sensores sem fio em ambientes fechados, utilizando Lógica Difusa como ferramenta para determinar o custo de cada enlace, com base na qualidade do sinal, buscando identificar as melhores rotas para o encaminhamento de dados, com o objetivo de melhorar o desempenho da rede em relação a taxa de perda de pacotes.

Pode-se observar a partir dos dados de simulação que o algoritmo proposto apresentou melhores taxas de sucesso para entrega de pacotes em relação à solução baseada somente na avaliação do RSSI. Foi possível verificar que a medida que se eleva o número de nós sensores na rede, a proposta baseada em Lógica Difusa apresenta resultados ainda mais satisfatórios.

Ainda como parte da validação da proposta, pode-se verificar o desempenho do algoritmo baseado em Lógica Difusa em um ambiente real, por meio da execução de experimentos. Foram implementadas adaptações no *firmware* da plataforma Radiuino e o desenvolvimento de um Softwre Gerente, que juntos permitiram a execução de testes em cenários reais. Os resultados obtidos após a execução dos experimentos demonstraram que o algoritmo proposto possui um desempenho superior ao protocolo baseado em RSSI. Foi possível verificar de forma dinâmica o funcionamento do protocolo, com as reavalizações da rede e as trocas de rotas a medida que a qualidade dos enlaces se alteravam.

Como trabalho futuro, sugere-se modificar o algoritmo de modo que os parâmetros possam ser auto-ajustáveis, variando dinamicamente de acordo com as características físicas do ambiente, ou seja, sugere-se utilizar as próprias medidas para se determinar os intervalos das funções de pertinência das variáveis de RSSI, desvio padrão e PER, a fim de buscar uma adaptação aos diferentes ambientes em tempo de execução, seguindo critérios de qualidade pré-especificados.

Referências

AKYILDIZ, Ian F. et al. Wireless sensor networks: a survey. **Computer networks**, v. 38, n. 4, p. 393-422, 2002.

AL-KARAKI, Jamal N.; KAMAL, Ahmed E. Routing techniques in wireless sensor networks: a survey. **Wireless communications, IEEE**, v. 11, n. 6, p. 6-28, 2004.

ANDERSEN, Jorgen Bach; RAPPAPORT, Theodore S.; YOSHIDA, Susumu. Propagation measurements and models for wireless communications channels. **Communications Magazine, IEEE**, v. 33, n. 1, p. 42-49, 1995.

ASLAM, M. S. et al. Wi-design, Wi-manage, why bother? **2011 Ifip/ieee International Symposium On Integrated Network Management (im)**, Dublin, n., p.730-744, 23-27 maio 2011.

AWANG, Azlan; LAGRANGE, Xavier; ROS, David. A cross-layer medium access control and routing protocol for wireless sensor networks. **Proc. 10emes Journées Doctorales en Informatique et Réseaux (JDIR 2009)**, 2009a.

AWANG, Azlan; LAGRANGE, Xavier; ROS, David. **RSSI-based forwarding for multihop wireless sensor networks**. Springer Berlin Heidelberg, 2009b.

BABU, Shaik Sahil et al. Fuzzy Logic Election of Node for Routing in WSNs. Trust, **Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on**. IEEE, 2012. p. 1279-1284.

BACCOUR, Nouha et al. F-lqe: A fuzzy link quality estimator for wireless sensor networks. **Wireless Sensor Networks**. Springer Berlin Heidelberg, 2010. p. 240-255.

BOUKERCHE, Azzedine, et al. A novel location-free greedy forward algorithm for wireless sensor networks**. Communications, 2008. ICC'08. IEEE International Conference on**. IEEE, 2008.

BRANQUINHO, Omar. **Plataforma Radiuino para Estudos em Redes de Sensores Sem Fio**, Disponível em: http://www.radiuino.cc, Acessado em: 2014

BROOKSON, Charles. GSM security: A description of the reasons for security and the techniques. **Security and Cryptography Applications to Radio Systems, IEE Colloquium on**. IET, 1994. p. 2/1-2/4.

DASTGHEIB, Seyyed Jalaleddin et al. A new method for flat routing in wireless sensor networks using fuzzy logic**. Computer Science and Network Technology (ICCSNT), 2011 International Conference on. IEEE, 2011**. p. 2112-2116.

DE GUGLIELMO, Domenico; ANASTASI, Giuseppe. Wireless sensor and actuator networks for energy efficiency in buildings. **Sustainable Internet and ICT for Sustainability (SustainIT),** 2012. IEEE, 2012. p. 1-3.

ENTERPRISES, Scilab. Scilab: Free and Open Source software for numerical computation. **Scilab Enterprises**, Orsay, France, 2012.

GOMIDE, Fernando; GUDWIN, Ricardo; TANSCHEIT, Ricardo. Conceitos fundamentais da teoria de conjuntos fuzzy, lógica fuzzy e aplicações. **Proc. 6 th IFSA Congress-Tutorials**. 1995. p. 1-38.

JIANG, Tao; LI, Yao. Generalized Defuzzification Strategies and Their Parameter Learning Procedures. **IEEE Transactions On Fuzzy Systems**, Washington, v. 4, n. 1, p.64-71, fev. 1996.

KANZAKI, Akimitsu et al. A Dynamic Route Construction Method Based on Measured Characteristics of Radio Propagation in Wireless Sensor Networks. **Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on**. IEEE, 2011. p. 30-37.

KULKARNI, Sunil; IYER, Aravind; ROSENBERG, Catherine. An address-light, integrated MAC and routing protocol for wireless sensor networks.**Networking, IEEE/ACM Transactions on**, v. 14, n. 4, p. 793-806, 2006.

LEE, Chuen-Chien. Fuzzy logic in control systems: fuzzy logic controller. II.Systems, **Man and Cybernetics, IEEE Transactions on**, v. 20, n. 2, p. 419-435, 1990.

LEE, Jin-Shyan; CHENG, Wei-Liang. Fuzzy-logic-based clustering approach for wireless sensor networks using energy predication. **Sensors Journal, IEEE**, v. 12, n. 9, p. 2891-2897, 2012.

MAMDANI, Ebrahim H. Application of fuzzy algorithms for control of simple dynamic plant. **Proceedings of the Institution of Electrical Engineers**. IET Digital Library, 1974. p. 1585-1588.

MENDEL, Jerry M. Fuzzy logic systems for engineering: a tutorial. **Proceedings of the IEEE**, v. 83, n. 3, p. 345-377, 1995.

NOSE, Yasuhiro et al. A Route Construction Based on Received Signal Strength in Wireless Sensor Networks, (2010) **International Conference on Computers and Their Applications (CATA 2010),** USA, p. 127-130.S.

ORTIZ, Antonio M.; OLIVARES, Teresa. Fuzzy Logic Applied to Decision Making in Wireless Sensor Networks. **Fuzzy Logic–Emerging Technologies and Applications**, p. 221-240, 2012.

PRESSMAN, Roger S. **Engenharia de Software**. Tradução: Rosângela Delloso Penteado. 2006.

SAKTHIDEVI, I.; SRIEVIDHYAJANANI, E. Secured Fuzzy Based Routing Framework for dynamic wireless sensor networks. **Circuits, Power and Computing Technologies (ICCPCT), 2013 International Conference on**. IEEE, 2013. p. 1041-1046.

SEEED TECHNOLOGY LIMITED (China) (Org.). **RFBee User Manual v1.1**. 2010. Disponível em: <http://www.seeedstudio.com/depot/datasheet/RFBee%20User%20Manual%20v1.1.pdf>. Acesso em: 04 fev. 2015.

SEIDEL, Scott Y.; RAPPAPORT, Theodore S. 914 MHz path loss prediction models for indoor wireless communications in multifloored buildings. **Antennas and Propagation, IEEE Transactions on**, v. 40, n. 2, p. 207-217, 1992.

SIQUEIRA, Ana Raquel Calais. **Análise matricial nebulosa de indicadores para apoio a tomada de decisão na governança corporativa de TIC**. 2013. 150 f. Dissertação (Mestrado) - Curso de Mestrado em Engenharia Elétrica, Pontifícia Universidade Católica de Campinas, Campinas, 2013.

SRIVASTAVA, Vineet; MOTANI, Mehul. Cross-layer design: a survey and the road ahead. **Communications Magazine, IEEE**, v. 43, n. 12, p. 112-119, 2005.

STANKOVIC, John A. Wireless Sensor Networks. IEEE Computer, v. 41, n. 10, p. 92-95, 2008.

SUH, Changsu; KO, Young-Bae; SON, Dong-Min. An energy efficient cross-layer MAC protocol for wireless sensor networks. In: **Advanced Web and Network Technologies, and Applications**. Springer Berlin Heidelberg, 2006. p. 410-419.

TANENBAUM, Andrew S.; DE COMPUTADORES, Redes. 4ª Edição. **Rio de Janeiro: Editora Campus**, 2003.

TOWNSEND, Chris; ARMS, Steven. Wireless Sensor Networks: Principles and Applications. In: WILSON, John S.. **Sensor Technology Handbook**. 2004. p. 439-449.

YICK, Jennifer; MUKHERJEE, Biswanath; GHOSAL, Dipak. Wireless sensor network survey. **Computer networks**, v. 52, n. 12, p. 2292-2330, 2008.

ZADEH, Lotfi A.. The concept of a generalized constraint: a bridge from natural languages to mathematics. **2005 Annual Meeting Of The North American Fuzzy Information Processing Society: NAFIPS 2005**, Ann Arbor, p.1-6, 25 mar. 2005. Anual.

ZADEH, Lotfi A.. Soft computing and fuzzy logic. **IEEE Software**, Washington, p. 48-56. nov. 1994.

ZADEH, Lotfi A.. A Summary and Update of “Fuzzy Logic”. **2010 IEEE International Conference On Granular Computing**, San Jose, p.42-44, 14-16 ago. 2010.

Anexo A – Implementação do Simulador

Arquivo: Simulador.sci

```
//*********************************************************************************//
// Simulador                                                                       //
// *******************************************************************************//
// Lucas Augusto de Araujo Marques Leão
// Programa de Mestrado em Engenharia Elétrica
// PUC Campinas
//
// *******************************************************************************//

clearglobal

EXIBE_GRAFICO = 0
ITERACOES = 10
CENARIO = 6

//*********************************************************************************//
// Configurações
// *******************************************************************************//

//-- Inicialização de parâmetros
valor_Beta = 4 //Use -1 se quiser um beta aleatório entre beta_low e beta_high, caso contrário, indique um valor fixo
beta_low = 2
beta_high = 3
sig_dBm = 7
potenciaTx = 0 // dBm
ganhoTx = 0 // dB
ganhoRx = 0 // dB
frequencia = 915000000
c=((3*10^8)/frequencia)
sensibilidade = -90 //sensibilidade do radio

//*********************************************************************************//
// Leitura de dados dos arquivos
// *******************************************************************************//

//--Carrega as funções utilizadas
exec('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Simulador\Lucas\ListaDeFuncoes.sci');

//--Carrega a função de Lógica Nebulosa
exec('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Simulador\Lucas\fuzzy\FuzzyLogic.sce');

//--Abre o arquivo Excel e carrega as informações das variáveis
[fd,SST,Sheetnames,Sheetpos] = xls_open('C:\Users\leao.lucas\Google Drive\PUC
Mestrado\Projeto\Simulador\Lucas\cenario3.xls')

//--lê os valores da Sheet
[vCenario,TextInd] = xls_read(fd,Sheetpos(CENARIO))

//--fecha o arquivo Excel
mclose(fd)

//Abre o arquivo Excel e carrega as informações das variáveis
[fd,SST,Sheetnames,Sheetpos] = xls_open('C:\Users\leao.lucas\Google Drive\PUC
Mestrado\Projeto\Simulador\Lucas\fuzzy\matriz.xls')

//lê os valores da primeira Sheet
//-- Matriz C
//------ [a   b   c   d   u]
[vMatC_lido,TextInd] = xls_read(fd,Sheetpos(1))

//-- Matriz E
//----- [Crisp NumLinhas]
[vMatE_lido,TextInd] = xls_read(fd,Sheetpos(2))
```

```
//-- Matriz R
//----- [Crisp NumLinhas]
[vMatR_lido,TextInd] = xls_read(fd,Sheetpos(3))

//fecha o arquivo Excel
mclose(fd)

//*************************************************************************//
// Construção do Cenário - Base e Sensores
// *************************************************************************//

   //--Matriz com o cenário da Base: PosX, PosY
   cenarioBase = [0,0]

   //--Matriz com o cenário: PosX, PosY
   cenarioSensores  =  vCenario

  //-------------------------------------------------------------------
  // Construção da REDE
  //-------------------------------------------------------------------

   media_saltos_f = []
   media_saltos_r = []
   itera_void_fuzzy = []
   itera_void_rssi = []
   prob_fuzzy = []
   prob_rssi = []
   itera = []
   dif = []
   saltos_f = []
   saltos_r = []
   saltos_f_max = 0
   saltos_r_max = 0
   saltos_f_min = 100000
   saltos_r_min = 100000

   for(g=1:ITERACOES)

   rssf = list()

   //--Cria a variável estruturada para a base
   rssf($+1) = struct('id',1,'tipo',0,'x',cenarioBase([1],[1]),'y',cenarioBase([1],[2]),'lider_f',0,'lider_r',0,'vizinho',list())
   tabelaID(1) = 1

   //--Verifica o tamanho da Matriz de Sensores
   tam = size(cenarioSensores)

   //--Cria a variável estruturada para os Sensores
   for (i=1:tam(1))
      rssf($+1)     =     struct('id',i+1,'tipo',1,'x',cenarioSensores([i],[1]),'y',cenarioSensores([i],[2]),'lider_f',-1,'lider_r',-
1,'vizinho',list())
      tabelaID(i+1) = i+1
   end

if(EXIBE_GRAFICO)
   //Exibe representação gráfica da rede
   exibeGrafico(rssf,1)
end

//*************************************************************************//
// Rotinas para construção da hierarquia da rede
// *************************************************************************//

   printf("\n\n--------- Iteracao: %d ----------\n",g)

   //--Chama rotina que constrói a hirarquia
   rssf = identificaTodosVizinhos(rssf, valor_Beta)
```

```
if(EXIBE_GRAFICO)
   //Representação gráfica da comunicação dos sensores
   tam=size(rssf)
   k=tam(1)
   while(k>=1)
      if rssf(k).lider_f>0  then
         x=[rssf(k).x rssf(rssf(k).lider_f).x]
         y=[rssf(k).y rssf(rssf(k).lider_f).y]
         xset("color",5)
         xpoly(x,y,"lines",1)
         p=get("hdl");
         p.polyline_style=4;
         if(k>rssf(k).lider_f)
            index = rssf(k).lider_f
         else
            index = rssf(k).lider_f - 1
         end
         if (k==1) then
            str = "Base Station"
         else
            str = "Sensor " + string(k-1)
         end
//"RSSI:  "  +  string(rssf(k).vizinho(index).rssi_media)  +  "  |  Desvio  Padrão:  "  +
string(rssf(k).vizinho(index).desvio_padrao) + " | PER: " + string(rssf(k).vizinho(index).per)
         xstring(rssf(k).x,rssf(k).y,str,0,0)
      end
      k=k-1
    end

   //Exibe representação gráfica da rede
   exibeGrafico(rssf,2)

   //Representação gráfica da comunicação dos sensores
   tam=size(rssf)
   k=tam(1)
   while(k>1)
      if rssf(k).lider_r>0  then
         x=[rssf(k).x rssf(rssf(k).lider_r).x]
         y=[rssf(k).y rssf(rssf(k).lider_r).y]
         xset("color",5)
         xpoly(x,y,"lines",1)
         p=get("hdl");
         p.polyline_style=4;
         if(k>rssf(k).lider_f)
            index = rssf(k).lider_f
         else
            index = rssf(k).lider_f - 1
         end
         if (k==1) then
            str = "Base Station"
         else
            str = "Sensor " + string(k-1)
         end
//"RSSI:  "  +  string(rssf(k).vizinho(index).rssi_media)  +  "  |  Desvio  Padrão:  "  +
string(rssf(k).vizinho(index).desvio_padrao) + " | PER: " + string(rssf(k).vizinho(index).per)
         xstring(rssf(k).x,rssf(k).y,str,0,0)
      end
      k=k-1
    end
end
   //Cálculo da Sucesso acumulada para Fuzzy
   tam=size(rssf)
   k=tam(1)
   prob_fuzzy(1) = 1
   saltos_f(1) = 0
   void_fuzzy=0
```

```
for (i=2:k)
   j=i
   if(rssf(j).lider_r==0)
      prob_fuzzy(i) = 0
      saltos_f(i) = 0
      void_fuzzy=void_fuzzy+1
   else
      prob_fuzzy(i) = 1 - rssf(j).vizinho(rssf(j).lider_f).per
      saltos_f(i) = 1
      j=rssf(j).lider_f
      while(1)
         if (j==1) then
            break
         else
            prob_fuzzy(i) = prob_fuzzy(i) * (1-rssf(j).vizinho(rssf(j).lider_f).per)
            j=rssf(j).lider_f
            saltos_f(i) = saltos_f(i) + 1
         end
      end
   end
end

//Cálculo da Sucesso acumulada para RSSI
tam=size(rssf)
k2=tam(1)
prob_rssi(1) = 1
saltos_r(1) = 0
void_rssi=0
for (i=2:k2)
   j=i
   if(rssf(j).lider_r==0)
      prob_rssi(i) = 0
      saltos_r(i) = 0
      void_rssi=void_rssi+1
   else
      prob_rssi(i) = 1 - rssf(j).vizinho(rssf(j).lider_r).per
      saltos_r(i) = 1
      j=rssf(j).lider_r
      while(1)
         if (j==1) then
            break
         else
            prob_rssi(i) = prob_rssi(i) * (1-rssf(j).vizinho(rssf(j).lider_r).per)
            j=rssf(j).lider_r
            saltos_r(i) = saltos_r(i) + 1
         end
      end
   end
end

//printf("Taxa de Entrega de Pacote para método Fuzzy: %f \n Taxa de Entrega de Pacote para método RSSI: %f
\n\n",PD_fuzzy,PD_rssi)

tam=size(rssf)
k3=tam(1)
for (i=2:k3)
   dif(i)=(prob_fuzzy(i)-prob_rssi(i))
   //printf("Sensor %2.0f) -- Fuzzy: %2.2f | RSSI: %2.2f | Diff: %2.2f\n",i,prob_fuzzy(i),prob_rssi(i),dif(i))
end

if(saltos_f_max<max(saltos_f))
   saltos_f_max = max(saltos_f)
end
if(saltos_r_max<max(saltos_r))
   saltos_r_max = max(saltos_r)
end
```

```
    if(saltos_f_min>min(saltos_f))
       saltos_f_min = min(saltos_f)
    end
    if(saltos_r_min>min(saltos_r))
       saltos_r_min = min(saltos_r)
    end

    itera(g) = mean(dif)
    aux = saltos_f(2:k3,1)
    media_saltos_f(g) = mean(aux)
    aux = saltos_r(2:k3,1)
    media_saltos_r(g) = mean(aux)
    itera_void_fuzzy(g) = void_fuzzy
    itera_void_rssi(g) = void_rssi
    itera_fuzzy(g) = mean(prob_fuzzy)
    itera_rssi(g) = mean(prob_rssi)

    printf("\nFator F: %f\n",itera(g))

    printf("\nVoids no Fuzzy: %f\n",itera_void_fuzzy(g))

    printf("\nVoids no RSSI: %f\n",itera_void_rssi(g))

    printf("\nMedia de Saltos Fuzzy: %f\n", media_saltos_f(g))

    printf("\nMedia de Saltos RBF: %f\n\n", media_saltos_r(g))

    end

    printf("Fator F  |  Saltos Fuzzy  |  Saltos RBF\n")

    tam_itera = size(itera)
    for i=1:tam_itera(1)
       printf("% .3f  |  % .3f          |  % .3f\n",itera(i), media_saltos_f(i), media_saltos_r(i))
    end

    printf("\n\n\nMax Saltos FUZZY: %f\n Max Saltos RBF: %f\n Min Saltos FUZZY: %f\n Min Saltos RBF:
%f",saltos_f_max,saltos_r_max,saltos_f_min,saltos_r_min)

    disp(mean(itera))

    disp("Simulação finalizada!")
```

Arquivo: ListaDeFuncoes.sci

```
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\//
// Funções                                                                    //
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\//

//*****************************************************************************//
// Vizinhos
// Identifica todos os sensores vizinhos
// *****************************************************************************//

function Mrssf=identificaTodosVizinhos(rssf_temp, valorBeta)

    //--Potência na distância d0
    prd0=fprd0()

    //--Verifica o tamanho da rede
    tamRede = size(rssf_temp)

    RSSImwMax = 0
    RSSImwMin = 100000

    printf("Verificando: \n")
```

```
    //--Percorre a lista de sensores
    for(i=1:tamRede)
        //--Percorre a rede em busca dos sensores vizinhos
        printf("%3.0f ",i)
        if (modulo(i,10)==0) then
            printf("\n")
        end
        for(j=1:tamRede)
            vMatCusto(i,j)=1000
            vMatRSSImw(i,j)=1000
            //--Pula sensores iguais
            if (rssf_temp(i).id<>rssf_temp(j).id) then
                //printf("Sensor %d com o Sensor %d\n",i,j)
                //--Calcula distância euclidiana entre o sensor na área de sombrea e os demais sensores
                dist_euclid = sqrt(((rssf_temp(j).x-rssf_temp(i).x)^2)+((rssf_temp(j).y-rssf_temp(i).y)^2))
                //--Calcula um valor para RSSI
                rssi_sim = PotenciaRxShadowing(prd0,dist_euclid,valorBeta)
                if (rssi_sim([1],[1])>sensibilidade) then
                    //--Calculo um valor aproximado para o Beta
                    beta_estimado = calculaBeta(rssi_sim([1],[1]), dist_euclid)
                    //Gera uma PER
                    xPER = grand(1,1,"def")//(grand(1,1,'poi',2)/10)+(grand(1,1,'poi',1)/100)
                    if(xPER>1)
                        xPER = 1
                    end
                    //Atualiza Matriz E
                    vMatE_mod = vMatE_lido
                    vMatE_mod([1],[1]) = rssi_sim([1],[1])
                    vMatE_mod([2],[1]) = rssi_sim([1],[2])
                    vMatE_mod([3],[1]) = xPER
                    //vMatE_mod([4],[1]) = beta_estimado
                    //--Verifica sensores com Potencial bom
                    valor_potencial = 10-execFuzzy(vMatE_mod, vMatC_lido, vMatR_lido)
                else
                    beta_estimado = 1000
                    xPER = 1
                    valor_potencial = 1000
                end
                //--Armazenando informações do vizinho
                rssf_temp(i).vizinho($+1)                                                                                    =
struct('id',j,'beta',beta_estimado,'desvio_padrao',rssi_sim([1],[2]),'rssi_media',rssi_sim([1],[1]),'per',xPER,'custo',valor_po
tencial)
                vMatCusto(j,i)=valor_potencial

                if (rssi_sim([1],[1])>sensibilidade) then
                    vMatRSSImw(j,i)=10^(rssi_sim([1],[1])/10)
                    if (RSSImwMax<vMatRSSImw(j,i)) then
                        RSSImwMax=vMatRSSImw(j,i)
                    end
                    if (RSSImwMin>vMatRSSImw(j,i)) then
                        RSSImwMin=vMatRSSImw(j,i)
                    end
                else
                    vMatRSSImw(j,i)=-1
                end
            end
        end
    end

    vMatPesoRSSI = 11 - ((9*vMatRSSImw)+(RSSImwMax-(10*RSSImwMin)))/(RSSImwMax-RSSImwMin)

    vMatPesoRSSI = abs(vMatPesoRSSI)

    printf("\nVerificação do melhor caminho para:\n")
    //--Percorre a lista de sensores criando os caminhos
    printf("- Fuzzy\n")
```

```
    rssf_temp = Dijkstra(vMatCusto, rssf_temp, 1)
    printf("- RSSI\n")
    rssf_temp = Dijkstra(vMatPesoRSSI, rssf_temp, 2)

    Mrssf = rssf_temp

endfunction

//*************************************************************************//
// Gráfico da RSSF
// Exibe o gráfico que representa a posição para cada sensor da rede
// *************************************************************************//

function exibeGrafico(rssf_temp, janela)

    cenBase([1],[1]) = rssf_temp(1).x
    cenBase([1],[2]) = rssf_temp(1).y

    tamRede = size(rssf)

    for (i=2:tamRede(1))
        cenSensores([i],[1]) = rssf_temp(i).x
        cenSensores([i],[2]) = rssf_temp(i).y
    end

    scf(janela);
    title('WSN Layout');
    //Limites do gráfico
    MaxX = max(cenSensores([:],[1]))+5
    MinX = min(cenSensores([:],[1]))-5
    MaxY = max(cenSensores([:],[2]))+5
    MinY = min(cenSensores([:],[2]))-5
    plot2d(0,0,1,rect=[MinX,MinY,MaxX,MaxY], frameflag=3)

    //--Base
    xpoly(cenBase(:,[1]),cenBase(:,[2]),"marks")
    aeBase=gce();
    set(aeBase,"mark_style",1);


    //--Sensores
    for (i=2:tamRede(1))
        //--Exibe o gráfico
        xpoly(rssf(i).x,rssf(i).y,"marks")
        aeSens=gce();
        set(aeSens,"mark_style",2);
    end

endfunction

//*************************************************************************//
// Shadowing
// Cálculo do RSSI em função da distância e de um Beta
// *************************************************************************//
function y=PotenciaRxShadowing(prd0, distanc, betaShadowing)

  resultado = []

   //Simula 100 leituras
  if (betaShadowing<=0) then
    betaShadowing = grand(100,1,'uin',beta_low,beta_high)
    resultado                =                potenciaTx+ganhoTx+ganhoRx-10*log10((((4*(%pi)*distanc)/c))^2)-
(10*betaShadowing*log10(distanc/1))
  else
    x = grand(100,1,'nor',0,sig_dBm)
    resultado                =                potenciaTx+ganhoTx+ganhoRx-10*log10((((4*(%pi)*distanc)/c))^2)-
(10*betaShadowing*log10(distanc/1)+x)
```

```
  end

  y = [mean(resultado),stdev(resultado)]

endfunction

//*************************************************************************//
// Potência na Distância d0
// Cálculo da potência na distância d0
// *************************************************************************//

function pot=fprd0()

    d0 = 1
    ld0=10*log10((4*%pi*d0/c)^2)
    pot=potenciaTx+ganhoTx+ganhoRx-ld0

endfunction

//*************************************************************************//
// Cálculo do Beta
// Com base nos valores de RSSI, distância rádio e atenuação, verifica o beta
// *************************************************************************//

function Beta=calculaBeta(vRssi, distancia)

    //--Potência na distância d0
    prd0=fprd0()

    beta_est = (prd0-vRssi)/(10*log10(distancia))

    Beta = beta_est

endfunction

//*************************************************************************//
// GeraCaminho (Guloso)
// *************************************************************************//

function res=GeraCaminho(vMatCusto, rssf_temp, tipo)

    tamCusto = size(vMatCusto)
    vMatSombra = ones(1,tamCusto(1))
    tamSombra = size(vMatSombra)
    vMatPeso = vMatCusto
    vMatSombra(1)=0
    rssf_temp(1).lider = 0
    vMatPeso([:],[1]) = 1000
    while(max(vMatSombra)==1)
        peso = 1000
        for(i=1:tamSombra(2))
            if(vMatSombra(i)==0)
                [m,k] = min(vMatPeso([i],[:]))
                if(peso>m)
                    lider = i
                    liderado = k
                    peso = m
                end
            end
        end
        if (tipo==1) then
            rssf_temp(liderado).lider_f = lider
        else
            rssf_temp(liderado).lider_r = lider
        end
        vMatSombra(liderado)=0
        vMatPeso(liderado,lider)=10000
```

```
    vMatPeso([:],[liderado])=10000
  end

  res = rssf_temp

endfunction

//*************************************************************************//
// Dijkstra (melhor caminho)
// *************************************************************************//

function res=Dijkstra(vMatCusto, rssf_temp, tipo)

  tamRede = size(rssf_temp)
  grafo = struct('vertice',1,'perm',1,'dist',0,'custo',0,'path',0)
  for(k=2:tamRede)
    grafo($+1) = struct('vertice',k,'perm',0,'dist',10000,'custo',10000,'path',0)
  end

  VertMin=1

  while(1)

    if(0)
      for(k=1:tamRede)
        //disp(grafo(k))
        printf("vertice: %d | perm: %d | dist:%f | custo:%f |
path=%d\n",grafo(k).vertice,grafo(k).perm,grafo(k).dist,grafo(k).custo,grafo(k).path)
      end
      disp("-----------------------------------------------")
    end

    tamRedeVizinho = size(rssf_temp(VertMin).vizinho)
    for(j=1:tamRedeVizinho)
      if (grafo(rssf_temp(VertMin).vizinho(j).id).dist > (grafo(VertMin).dist +
vMatCusto(VertMin,rssf_temp(VertMin).vizinho(j).id))) then
        grafo(rssf_temp(VertMin).vizinho(j).id).dist = grafo(VertMin).dist +
vMatCusto(VertMin,rssf_temp(VertMin).vizinho(j).id)
        grafo(rssf_temp(VertMin).vizinho(j).id).path = VertMin
        grafo(rssf_temp(VertMin).vizinho(j).id).custo = vMatCusto(VertMin,rssf_temp(VertMin).vizinho(j).id)
      end
    end

    ValorVertMin = 10000
    VertMin = 0
    for(k=1:tamRede)
      if(grafo(k).perm==0 & ValorVertMin>grafo(k).dist)
        ValorVertMin = grafo(k).dist
        VertMin = k
      end
    end

    if(VertMin==0)
      break
    end
    grafo(VertMin).perm = 1
  end

  for(k=2:tamRede)
    if (tipo==1) then
      rssf_temp(k).lider_f = grafo(k).path
    else
      rssf_temp(k).lider_r = grafo(k).path
    end
  end

  if(1)
```

```
        for(k=1:tamRede)
            //disp(grafo(k))
            printf("vertice: %d | perm: %d | dist:%f | custo:%f |
path=%d\n",grafo(k).vertice,grafo(k).perm,grafo(k).dist,grafo(k).custo,grafo(k).path)
        end
        disp("-------------------------------------------------")
    end

    res = rssf_temp

endfunction
```

Arquivo: FuzzyLogic.sci

```
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\//
// Lógica Nebulosa                                                    //
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\//

function resultado=execFuzzy(vMatE, vMatC, vMatR)

//--- Preenche a Matriz com os valores de pertinência -------------------------
//verifica o tamanho da Matriz E (descobre número de variáveis nebulosas)
tamMatE=size(vMatE)

//posição inicial da linha para a Matriz C
pos=1
tamMatConj = 0
//percorre a Matriz E
for(j=1:tamMatE(1))
    //percorre a Matriz C
    for(i=1:vMatE([j],[2]))
        //preenchimento da pertinência para cada conjunto nebuloso
        vMatC([pos],[5]) = geraFuzzy(vMatC([pos],[1]), vMatC([pos],[2]), vMatC([pos],[3]),
vMatC([pos],[4]),vMatE([j],[1]))
        pos=pos+1

        //printf("\n geraFuzzy(%f,%f,%f,%f,%f) \r",vMatC([pos],[1]), vMatC([pos],[2]), vMatC([pos],[3]),
vMatC([pos],[4]),vMatE([j],[1]))
        //disp(vMatE)

    end
    tamMatConj = tamMatConj + vMatE([j],[2])
end
//-------------------------------------------------------------------------------

//--- Preenche a Matriz de acordo com as Regras ------------------------------
//verifica o tamanho da Matriz R (descobre número de regras)
tamMatR=size(vMatR)

//percorre a Matriz R
if tamMatR(2)==3 then
    for(j=1:tamMatR(1))
        temp = min(vMatC([vMatR([j],[1])],[5]),vMatC([vMatR([j],[2])],[5]))
        vMatC([vMatR([j],[3])],[5]) = max(vMatC([vMatR([j],[3])],[5]),temp)
    end
elseif tamMatR(2)==4 then
    for(j=1:tamMatR(1))
        temp = min(vMatC([vMatR([j],[1])],[5]),vMatC([vMatR([j],[2])],[5]),vMatC([vMatR([j],[3])],[5]))
        vMatC([vMatR([j],[4])],[5]) = max(vMatC([vMatR([j],[4])],[5]),temp)
    end
elseif tamMatR(2)==5 then
    for(j=1:tamMatR(1))
        temp =
min(vMatC([vMatR([j],[1])],[5]),vMatC([vMatR([j],[2])],[5]),vMatC([vMatR([j],[3])],[5]),vMatC([vMatR([j],[4])],[
5]))
        vMatC([vMatR([j],[5])],[5]) = max(vMatC([vMatR([j],[5])],[5]),temp)
```

```
    end
end
//-------------------------------------------------------------------------

//--- Recorte da Matriz C -----------------------------------------------
linIniC = tamMatConj+1
temp = size(vMatC)
linFimC = temp(1)
vMatCsub = vMatC([linIniC:linFimC],[1:5])
//-------------------------------------------------------------------------

//--- Defuzzificação ----------------------------------------------
X = [0:0.1:10]
tamX = size(X)
tamMatCsub=size(vMatCsub)
SumXiMii=0
SumMii=0

for(i=1:tamX(2))
    Mip=[]
    for(pos=1:tamMatCsub(1))
        Mip(pos) = min(geraFuzzy(vMatCsub([pos],[1]), vMatCsub([pos],[2]), vMatCsub([pos],[3]),
vMatCsub([pos],[4]),X(i)),vMatCsub([pos],[5]))
    end
    Mii(i) = max(Mip)
    SumXiMii = SumXiMii + X(i)*Mii(i)
    SumMii = SumMii + Mii(i)
end

resultado = SumXiMii/SumMii

endfunction

//*************************************************************************//
// Gera Fuzzy
// Função de que calcula o valor de pertinência
// *************************************************************************//
function resp=geraFuzzy(a, b, c, d, x)
    if(b==x | c==x | (b<x & x<c))
        resp = 1
    elseif(x<=a | x>=d)
        resp = 0
    elseif(a<x & x<b)
        resp = (x-a)/(b-a)
    elseif(c<x & x<d)
        resp = (d-x)/(d-c)
    else
        resp = 100000
        disp("ERRO")
    end
endfunction
```

Anexo B – Modificações do *Firmware* Radiuino

Arquivo: Radiuino_Sensor_PRGF_V3.ino

```
// Radiuino_Sensor : firmware para os nós Sensores da rede

// Mais informações em www.radiuino.cc
// Copyright (c) 2011
// Author: Pedro Henrique Gomes e Omar C. Branquinho
// Versão 1.0: 12/09/2011

// Modificado por: Lucas Augusto de Araujo Marques Leão
// Nova versão 2.0: 01/02/2014
// Modificações para o protocolo FLBRA

// Este arquivo é parte da plataforma Radiuino
// Este programa é um software livre; você pode redistribui-lo e/ou modifica-lo dentro dos termos da Licença Pública Geral Menor GNU
// como publicada pela Fundação do Software Livre (FSF); na versão 2 da Licença, ou (na sua opnião) qualquer futura versão.
// Este programa é distribuido na esperança que possa ser  util, mas SEM NENHUMA GARANTIA; sem uma garantia implicita
// de ADEQUAÇÂO a qualquer MERCADO ou APLICAÇÃO EM PARTICULAR. Veja a Licença Pública Geral Menor GNU para maiores detalhes.
// Você deve ter recebido uma cópia da Licença Pública Geral Menor GNU junto com este programa, se não, escreva para a Fundação
// do Software Livre(FSF) Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

// This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library
// is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
// or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public License for more details. You should have received a copy
// of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St,
// Fifth Floor, Boston, MA  02110-1301  USA

#include <RADIUINO.h>
#include <EEPROM.h>
#include <SPI.h>

#include "Headers.h"

#define FIRMWARE_VERSION   1.0     /* Versão 1.0 Radiuino */
byte int_rx = 0;                   /* Inicialização da interrupção de recepção de pacotes - O hardware gera uma interrupção no começo que não deve ser tratada */
byte int_buff = 0;                 /* Inicialização da interrupção de buffer overflow - O
```

```
hardware gera uma interrupção no começo que não deve ser tratada */

/**
 * Configura o Arduino. É executado uma única vez no início do firmware.
 */
void setup()
{
  /* Inicialização da camada Física */
  Phy.initialize();

  /* Inicialização da camada de Controle de Acesso ao Meio */
  Mac.initialize();

  /* Inicialização da camada de Rede */
  Net.initialize();

  /* Inicialização da camada de Transporte */
  Transp.initialize();

  /* Inicialização da camada de Aplicação */
  App.initialize();

  /*Inicializa separador da EEPROM*/
  EEPROM.write(RT_DATA_SEPARATOR,255);

  /* Anexa as funções de interrupção de recepção de pacotes e de estouro de
buffer de recepção */
  attachInterrupt(0, IntReceiveData, RISING);
  attachInterrupt(1, IntBufferOverflow, RISING);
  pinMode(GDO0, INPUT);

  /* Inicializa o Timer1 e configura o período para 1 segundo */
  Timer1.initialize(1000000);

  /* Anexa a função de interrupção de estouro do Timer1 */
  Timer1.attachInterrupt(IntTimer1);

  /* Escreve mensagem de inicialização */
  //Serial.print("Radiuino! Sensor");

 /* AJUDA A DEBUGAR AS VARIÁVEIS ----------------------
  Serial.println();
  int count=0;
  int value=0;
  while(count<=510){
      value = EEPROM.read(count);
      Serial.print(count);
      Serial.print("\t");
      Serial.print(value, DEC);
      Serial.println();
```

```
      count++;
  }
 ----------------------------------------------------*/

}

/**
 * Laço de execução do Arduino. É executado continuamente.
 */
void loop()
{
  /* Verifica se deve entrar em modo Sleep */
  verifySleepEntering();
}

/**
 * Trata o pacote recebido pela rede sem fio.
 */
void IntReceiveData()
{

  /* Se for a primeira vez que a interrupção é executada */
  if (int_rx == 0)
  {
    int_rx = 1;
    return;
  }

  /* Reenvia toda informação recebida da rede para o PC */
  if ( digitalRead(GDO0) == HIGH )
  {
    /* Recebe os dados do RF */
    Phy.receive(&g_pkt);
  }

  return;
}

/**
 * Trata o estouro do buffer de recepção.
 */
void IntBufferOverflow()
{
  /* Se for a primeira vez que a interrupção é executada */
  if (int_buff == 0) {
    int_buff = 1;
    return;
  }

  /* Esvazia o buffer de recepção e vai para o estado de RX */
```

```
  cc1101.Strobe(CC1101_SFRX);
  cc1101.Strobe(CC1101_SRX);

  return;
}

/**
 * Trata o estouro do período do Timer1.
 */
void IntTimer1()
{
  /* Aqui podem ser colocadas tarefas periódicas. O período do Timer1 é de 1 segundo por padrão.
   * Mas pode ser mudado no setup() */

  return;
}

/**
 * Realiza uma operação Atômica para verificar se deve entrar no modo Sleep ou se deve sair do modo Sleep
 */
void verifySleepEntering ( void ) {
  ATOMIC_BLOCK(ATOMIC_FORCEON)
  {
    if (Mac.time_to_sleep > 0) {

      /* Aguarda o fim da tranmissão do ACK SLEEP */
      delayMicroseconds(500);
      while(Phy.txFifoFree() != CC1101_FIFO_SIZE);

      /* Entra em modo Sleep */
      Phy.lowPowerOn();

    }
    else if (Mac.time_to_sleep == 0) {

      /* Saindo do modo de Sleep */
      cc1101.Strobe(CC1101_SIDLE);
      delay(1);
      cc1101.Strobe(CC1101_SFTX);
      cc1101.Strobe(CC1101_SFRX);
      cc1101.Strobe(CC1101_SRX);

      Mac.time_to_sleep = -1;
    }
  }
}

/**
```

```
 * Interrupção de WatchDog. É executada sempre que o contador de WatchDog é estourado
 */
ISR(WDT_vect) {

  /* Verifica se precisamos decrementar o contador de Sleep */
  if (Mac.time_to_sleep > 0) {
    Mac.time_to_sleep--;
    return;
  }

}
```

Arquivo: Headers.h

```
// Header.h : cabeçalhos de classes

// Mais informações em www.radiuino.cc
// Copyright (c) 2011
// Author: Pedro Henrique Gomes e Omar C. Branquinho
// Versão 1.0: 12/09/2011

// Modificado por: Lucas Augusto de Araujo Marques Leão
// Nova versão 2.0: 01/02/2014
// Modificações para o protocolo FLBRA

// Este arquivo é parte da plataforma Radiuino
// Este programa é um software livre; você pode redistribui-lo e/ou modifica-lo dentro dos termos da Licença Pública Geral Menor GNU
// como publicada pela Fundação do Software Livre (FSF); na versão 2 da Licença, ou (na sua opnião) qualquer futura versão.
// Este programa é distribuido na esperança que possa ser  util, mas SEM NENHUMA GARANTIA; sem uma garantia implicita
// de ADEQUAÇÂO a qualquer MERCADO ou APLICAÇÃO EM PARTICULAR. Veja a Licença Pública Geral Menor GNU para maiores detalhes.
// Você deve ter recebido uma cópia da Licença Pública Geral Menor GNU junto com este programa, se não, escreva para a Fundação
// do Software Livre(FSF) Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301 USA

// This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library
// is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
// or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public License for more details. You should have received a copy
// of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St,
// Fifth Floor, Boston, MA  02110-1301  USA
```

```
#ifndef HEADERS_H
#define HEADERS_H 1

/**
 * Estrutura do pacote a ser transmitido e recebido. O pacote possui 52 bytes, sendo 4 bytes de cabeçalho de camada física (Phy), 4 bytes de cabeçalho
 * de camada de Controle de Acesso ao Meio (MAC), 4 bytes de cabeçalho de camada de Rede (Net), 4 bytes de cabeçalho de camada de Transporte (Transp) e
 * e 4 bytes de cabeçalho de camada de Aplicação (App). Os 36 bytes restantes são reservados para payload de AD e IO.
 */
typedef struct
{
  /* Cabeçalho da camada Física (Phy) */
  byte PhyHdr[4];

  /* Cabeçalho da camada de Controle de Acesso ao Meio (MAC) */
  byte MACHdr[4];

  /* Cabeçalho da camada de Rede (Net) */
  byte NetHdr[4];

  /* Cabeçalho da camada de Transporte (Transp) */
  byte TranspHdr[4];

  // FOI INCLUÍDO UM VETOR COM 36 POSIÇÕES - aqui no Radiuino tinha os ADs e IOs
  /* Bytes o payload de AD */
  byte Data[36];

} packet;

packet g_pkt;

/**
 * Divisão da memória EEPROM para os dados do Sensor
 */
#define ROUTE_TABLE_INI 0
#define ROUTE_TABLE_END 199
#define RT_DATA_SEPARATOR 200
#define DATA_INI 201
#define DATA_END 511

/**
 * Número de Sensores na Rede, incluindo a base
 */
#define NETWORK_SIZE 21
```

```
/**
 * Número de Amostras para coleta
 */
#define NUMBER_SAMPLES 15

/**
 * Número de amostras que devem ser salvas
 */
#define PKT_SENT 0

/**
 * Númro de amostras que devem ser salvas
 */
#define PKT_RECEIVED 1

/**
 * Classe de camada de Aplicação
 */

#define AD0_PIN  0
#define AD1_PIN  1
#define AD2_PIN  2
#define AD3_PIN  3
#define AD4_PIN  4
#define AD5_PIN  5

#define IO0_PIN  4
#define IO1_PIN  5
#define IO2_PIN  6
#define IO3_PIN  7
#define IO4_PIN  8
#define IO5_PIN  9

class APP
{
  public:
    APP(void);
    inline void initialize(void);
    inline void send(packet * pkt);
    inline void receive(packet * pkt);

  private:
    int IO_0, IO_1, IO_2, IO_3, IO_4, IO_5;

};

/* Objeto de acesso à classe da camada de Aplicação */
extern APP App;

/**
```

```
 * Classe de camada de Transporte
 */
class TRANSP
{
  public:
    TRANSP(void);
    inline void initialize(void);
    inline void send(packet * pkt);
    inline void receive(packet * pkt);

    //Definição da variável info de pacotes
    int pkt_info[2];

    //-- Definição da variável de histórico de PER
    byte hist_per[NETWORK_SIZE][2]; //1 - numero de pacotes efetivamente recebidos no destino .. 0 -  numero de pacotes enviados

  private:

};

/* Objeto de acesso à classe da camada de Transporte */
extern TRANSP Transp;

/**
 * Classe de camada de Rede
 */
class NET
{
  public:
    NET(void);
    inline void initialize(void);
    inline void send(packet * pkt);
    inline void receive(packet * pkt);
    void swapAddresses(packet * pkt, int addr_dest);
    void routeChange(packet * pkt);
    void infoCheck(packet * pkt, int pkt_type);
    void changePosition(packet * pkt);

    byte my_addr;
    //byte my_dest;
    //byte my_neighbor;
  private:

};

/* Objeto de acesso à classe da camada de Rede */
extern NET Net;

/**
```

```
 * Classe de camada de Controle de Acesso ao Meio
 */
#define  SLEEP_MSG  1
#define  SLEEP_ACK  2
class MAC
{
  public:
    MAC(void);
    inline void initialize(void);
    inline void send(packet * pkt);
    inline void receive(packet * pkt);

    volatile int time_to_sleep;
  private:

};

/* Objeto de acesso à classe da camada de Controle de Acesso ao Meio */
extern MAC Mac;

/**
 * Classe de camada Física
 */
#define BUFFLEN  CC1101_PACKT_LEN
byte serialData[BUFFLEN + 1];

class PHY
{
  public:
    PHY(void);
    inline void initialize();
    inline void send(packet * pkt);
    inline int receive(packet * pkt);

    void sendSerial(packet * pkt);
    void receiveSerial(void);

    byte txFifoFree(void);
    void setChannel(byte channel);
    void lowPowerOn(void);
    void setPower(byte power);
    void setFreqOffset(byte freq_offset);

    boolean carrierSense(void);

    byte power;               /* Potência */
    byte channel;             /* Canal */
    byte freq_offset;         /* Offset de frequencia */
    int serial_baudrate;      /* Serial baudrate */
```

```
    //-- Definição da variável de histórico de leituras de RSSI
    byte hist_rssi[NETWORK_SIZE][NUMBER_SAMPLES];

  private:
    int initCC1101Config(void);
    void configWatchdog(int time);

};

/* Objeto de acesso à classe da camada Física */
extern PHY Phy;

/* Configuração de registradores do CC1101. Obtidos através do SmartRF Studio
7 */
// Deviation = 4.760742
// Base frequency = 914.999969
// Carrier frequency = 914.999969
// Channel number = 0
// Carrier frequency = 914.999969
// Modulated = true
// Modulation format = GFSK
// Manchester enable = false
// Sync word qualifier mode = 30/32 sync word bits detected
// Preamble count = 4
// Channel spacing = 199.951172
// Carrier frequency = 914.999969
// Data rate = 9.59587
// RX filter BW = 203.125000
// Data format = Normal mode
// CRC enable = true
// Device address = 0
// Address config = No address check
// CRC autoflush = false
// PA ramping = false
// TX power = 10
const                                                                    byte
CC1101_registerSettings[CC1101_NR_OF_CONFIGS][CC1101_NR_OF_REGIS
TERS] PROGMEM = {
{
    0x04,  // IOCFG2       GDO2 Output Pin Configuration
    0x07,  // IOCFG0       GDO0 Output Pin Configuration
    0x07,  // FIFOTHR       RX FIFO and TX FIFO Thresholds
    0x34,  // PKTLEN        Packet Length
    0x04,  // PKTCTRL1      Packet Automation Control
    0x04,  // PKTCTRL0      Packet Automation Control
    0x00,  // ADDR         Device Address
    0x00,  // CHANNR        Channel Number
    0x06,  // FSCTRL1       Frequency Synthesizer Control
    0x00,  // FSCTRL0       Frequency Synthesizer Control
    0x23,  // FREQ2        Frequency Control Word, High Byte
```

```
    0x31,  // FREQ1        Frequency Control Word, Middle Byte
    0x3B,  // FREQ0        Frequency Control Word, Low Byte
    0x4A,  // MDMCFG4      Modem Configuration
    0x83,  // MDMCFG3      Modem Configuration
    0x13,  // MDMCFG2      Modem Configuration
    0x22,  // MDMCFG1      Modem Configuration
    0xF8,  // MDMCFG0      Modem Configuration
    0x60,  // DEVIATN      Modem Deviation Setting
    0x18,  // MCSM0        Main Radio Control State Machine Configuration
    0x16,  // FOCCFG       Frequency Offset Compensation Configuration
    0x1C,  // BSCFG        Bit Synchronization Configuration
    0x7B,  // AGCCTRL2     AGC Control
    0x00,  // AGCCTRL1     AGC Control
    0xB0,  // AGCCTRL0     AGC Control
    0xB6,  // FREND1       Front End RX Configuration
    0x10,  // FREND0       Front End TX Configuration
    0xE9,  // FSCAL3       Frequency Synthesizer Calibration
    0x2A,  // FSCAL2       Frequency Synthesizer Calibration
    0x00,  // FSCAL1       Frequency Synthesizer Calibration
    0x1F,  // FSCAL0       Frequency Synthesizer Calibration
    0x59,  // FSTEST       Frequency Synthesizer Calibration Control
    0x88,  // TEST2        Various Test Settings
    0x31,  // TEST1        Various Test Settings
    0x09,  // TEST0        Various Test Settings
}
};

const                                                                    byte
CC1101_paTable[CC1101_NR_OF_CONFIGS][CC1101_PA_TABLESIZE]
PROGMEM ={
// -30  -20   -15  -10    0   5    7   10
  {0x03,0x0E,0x1E,0x27,0x8E,0x84,0xCC,0xC3},    // Configuração 0
};

#endif
```

Arquivo: _1_Phy.ino

```
// Phy : classe da camada Física

// Mais informações em www.radiuino.cc
// Copyright (c) 2011
// Author: Pedro Henrique Gomes e Omar C. Branquinho
// Versão 1.0: 12/09/2011

// Modificado por: Lucas Augusto de Araujo Marques Leão
// Nova versão 2.0: 01/02/2014
// Modificações para o protocolo FLBRA

// Este arquivo é parte da plataforma Radiuino
// Este programa é um software livre; você pode redistribui-lo e/ou modifica-lo
```

```
dentro dos termos da Licença Pública Geral Menor GNU
// como publicada pela Fundação do Software Livre (FSF); na versão 2 da Licença, ou (na sua opnião) qualquer futura versão.
// Este programa é distribuido na esperança que possa ser  util, mas SEM NENHUMA GARANTIA; sem uma garantia implicita
// de ADEQUAÇÂO a qualquer MERCADO ou APLICAÇÃO EM PARTICULAR. Veja a Licença Pública Geral Menor GNU para maiores detalhes.
// Você deve ter recebido uma cópia da Licença Pública Geral Menor GNU junto com este programa, se não, escreva para a Fundação
// do Software Livre(FSF) Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301 USA

// This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library
// is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
// or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public License for more details. You should have received a copy
// of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St,
// Fifth Floor, Boston, MA  02110-1301  USA

#include "Headers.h"

/**
 * Construtor da camada Física.
 */
PHY::PHY()
{


//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
  //                  AJUSTE DE POTÊNCIA

//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
  // -30  -20   -15  -10    0    5    7   10    - Potência em dBm
  //   0     1     2    3    4    5    6    7    - Número que deve ser colocado na potência
  power = 4;                  /* Potência */


//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
  //                  SELEÇÃO DO CANAL DE OPERAÇÃO

//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

```
  channel = 13;              /* Canal */


//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
 //               DIFERENÇA DE FREQUÊNCIA A SER AJUSTA EM FUNÇÃO DA
PRECISÃO DO CRISTAL

//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
  freq_offset = 0xe7;        /* Offset de Frequencia */




//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
 //          TAXA DA COMUNICAÇÃO USB ENTRE O COMPUTADOR E A BASE
DO RADIUINO

//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
  serial_baudrate = 9600;    /* Serial baudrate */

}

/**
 * Inicializa a camada Física.
 */
void PHY::initialize(void)
{
  //-- Inicializa a variável de histórico de RSSI
  int i,j;
  for(i=0;i<NETWORK_SIZE;i++)
    for(j=0;j<NUMBER_SAMPLES;j++)
      hist_rssi[i][j]=255;

  /* Configurando o serial baudrate */
  Serial.begin(serial_baudrate);

  //* Inicializa o transceptor CC1101 */
  cc1101.PowerOnStartUp();

  /* Inicializa a configuração do transceptor CC1101 */
  initCC1101Config();

  /* Configura o canal a ser usada */
  setChannel(channel);

  /* COnfigura a potência a ser usada */
```

```
  setPower(power);

  /* Configura o offset de frequência */
  setFreqOffset(freq_offset);

  /* Configura o timeout do WatchDog para 1 segundo */
  configWatchdog(6);
}

/**
 * Recebe dados da porta serial.
 */
void PHY::receiveSerial(void)
{
  byte len;            /* Tamanho do dado recebido pela porta serial */
  byte fifoSize = 0;    /* Tamanho do espaço livre no FIFO de TX */

  static byte pos = 0;  /* Total de bytes recebidos pela porta serial */

  /* Le a porta serial e incrementa o total de bytes recebidos */
  len = Serial.available() + pos;

  /* Processa no máximo BUFFLEN bytes */
  if (len > BUFFLEN )
  {
    len = BUFFLEN;
  }

  /* Verifica quanto espaço temos no FIFO de TX */
  fifoSize = Phy.txFifoFree();      /* O fifoSize deve ter o número de bytes
atualmeente livre no FIFO de TX */

  /* Reinicializa as variáveis e sai da função */
  if ( fifoSize <= 0)
  {
    Serial.flush();
    pos = 0;
    return;
  }

  /* Evita estourar o FIFO de TX */
  if (len > fifoSize)
  {
    len = fifoSize;
  }

  /* Finalmente escreve os dados lidos da serial no FIFO de TX */
  for (byte i = pos; i < len; i++)
  {
    serialData[i] = Serial.read();  /* serialData é o nosso buffer global */
```

```
  }

  /* Atraso de 1 milissegundo */
  delayMicroseconds(1000);

  /* Verifica se existem mais dados para receber */
  if ((Serial.available() > 0)  && (len < CC1101_PACKT_LEN))
  {
    pos = len;  /* Mantem a quantidade de bytes já recebidos e espera pela próxima
entrada nessa função */
    return;
  }

  if (len == sizeof(packet))
  {
    /* Envia a mensagem recebida pelo RF */
    Phy.send((packet *)serialData);
    /* O buffer da serial está livre novamente */
    pos = 0;
  }
  else
  {
    Serial.flush();
    pos = 0;
    return;
  }
}

/**
 * Transmite dados pela porta serial.
 */
void PHY::sendSerial(packet * pkt)
{
  /* Escreve o pacote inteiro na porta serial */
  Serial.write((byte *)pkt, sizeof(packet));
}

/**
 * Le o espaço disponível no FIFO de TX.
 */
byte PHY::txFifoFree(void)
{
  byte size;

  cc1101.Read(CC1101_TXBYTES, &size);

  /* Trata um possível underflow to FIFO de TX */
  if (size >= 64)
  {
    cc1101.Strobe(CC1101_SFTX);
```

```
    cc1101.Read(CC1101_TXBYTES,&size);
  }

  return (CC1101_FIFO_SIZE - size);
}

/**
 * Ajusta o canal a ser usado.
 */
void PHY::setChannel(byte channel)
{
  cc1101.Write(CC1101_CHANNR, channel);
}

/**
 * Ajusta a potência a ser usada.
 */
void PHY::setPower(byte power)
{
  cc1101.setPA(0, power);
}

/**
 * Ajusta o offset de frequencia.
 */
void PHY::setFreqOffset(byte freq_offset)
{
  cc1101.Write(CC1101_FSCTRL0, freq_offset);
}

/**
 * Inicializa a configuração do CC1101.
 */
int PHY::initCC1101Config(void)
{
  /* Carrega a primeira configuração (pode ser inserida mais de uma configuração
no código) */
  cc1101.Setup(0);

  /* Configura o endereço do rádio */
  cc1101.Write(CC1101_ADDR, Net.my_addr);

  /* Configura a potência para a máxima possível (7) */
  cc1101.setPA(0, 7);

  /* Coloca o CC1101 no estado de RX */
  cc1101.Strobe(CC1101_SIDLE);
  delay(1);
  cc1101.Write(CC1101_MCSM1 ,   0x0F );
  cc1101.Strobe(CC1101_SFTX);
```

```
  cc1101.Strobe(CC1101_SFRX);
  cc1101.Strobe(CC1101_SRX);

  return OK;
}

/**
 * Entra em estado de Low Power mode
 */
void PHY::lowPowerOn(void){

  /* Colocando o CC1101 no estado de SLEEP */
  cc1101.Strobe(CC1101_SIDLE);
  cc1101.Strobe(CC1101_SPWD);

  /* Desabilita o conversor AD */
  _SFR_BYTE(ADCSRA) &= ~_BV(ADEN);

  /* Configura as portas de saída como entrada */
  pinMode (IO0_PIN, INPUT);
  pinMode (IO1_PIN, INPUT);
  pinMode (IO2_PIN, INPUT);
  pinMode (IO3_PIN, INPUT);
  pinMode (IO4_PIN, INPUT);
  pinMode (IO5_PIN, INPUT);

  /* Configura o modo Sleep do ATMega */
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);

  cli();
  sleep_enable();         /* Habilita o bit de Sleep no registrador MCUCR */

  /* Desliga os módulos do ATMega */
  power_adc_disable();
  power_spi_disable();
  power_timer0_disable();
  power_usart0_disable();
  power_timer0_disable();
  power_timer1_disable();
  power_timer2_disable();
  power_twi_disable();

  /* Coloca o ATMega em modo Sleep efetivamente */
  sei();
  sleep_mode();

  /**** O PROGRAMA CONTINUA AQUI DEPOIS DE ACORDAR ****/
  sleep_disable();        /* Desabilita o bit de Sleep */

  /* Configura as portas de saída como saída */
```

```
  pinMode (IO0_PIN, OUTPUT);
  pinMode (IO1_PIN, OUTPUT);
  pinMode (IO2_PIN, OUTPUT);
  pinMode (IO3_PIN, OUTPUT);
  pinMode (IO4_PIN, OUTPUT);
  pinMode (IO5_PIN, OUTPUT);

  /* Habilita o conversor AD */
  _SFR_BYTE(ADCSRA) |= _BV(ADEN);

  /* Liga todos os modulos do ATMega */
  power_all_enable();

}

/**
 * Retorna o status de presença de portadora no canal (Carrier Sense)
 * return   1 se o canal está ocupado, 0 se o canal está livre
 */
boolean PHY::carrierSense(void)
{
  byte cs;

  /* O status da portadora é lido no registrador PKTSTATUS */
  cc1101.Read(CC1101_PKTSTATUS,&cs);
  /* O bit de Carrier Sense é o bit 6 */
  cs &= 0x40;

  if (cs)
    return true;
  else
    return false;

}

/**
 * Configura o valor do timeout do WatchDog
 *  0=16ms,  1=32ms,  2=64ms,  3=128ms,  4=250ms,  5=500ms,  6=1sec,  7=2sec,
8=4sec, 9=8sec
 */
void PHY::configWatchdog(int time) {

  byte value;

  if (time > 9 ) time = 9;
  value = time & 7;
  if (time > 7) value |= (1<<5);
  value |= (1<<WDCE);

  /* Habilita a interrupção de WatchDog no Satus Register */
```

```
  MCUSR &= ~(1<<WDRF);

  /* Configura as flags do registrador de WatchDog */
  WDTCSR |= (1<<WDCE) | (1<<WDE);

  /* Configura o valor do timeout */
  WDTCSR = value;

  /* Habilita a interrupção do WatchDog */
  WDTCSR |= _BV(WDIE);

}

/**
 * Envia dados pelo RF.
 */
inline void PHY::send(packet * pkt)
{
  byte *txData = (byte *)pkt;

  digitalWrite (4, HIGH);

  /* Coloca o CC1101 no estado IDLE */
  cc1101.Strobe(CC1101_SIDLE);

  /* Escreve uma rajada com os dados a serem transmitidos */
  cc1101.WriteBurst(CC1101_TXFIFO, txData, sizeof(packet));

  /* Vai para o estado TX */
  cc1101.Strobe(CC1101_STX);

  /* Aguarda enquanto todos os bytes estão sendo transmitidos */
  while(1)
  {
    byte size;
    cc1101.Read(CC1101_TXBYTES, &size);
    if( size == 0 )
    {
      break;
    }
    else
    {
      cc1101.Strobe(CC1101_STX);
    }
  }

  digitalWrite (4, LOW);
}

/**
```

```
 * Recebe dados pelo RF.
 */
inline int PHY::receive(packet * pkt){

  byte stat, rssi, lqi;

  digitalWrite (6, HIGH);

  /* Le uma rajada de dados do FIFO de RX */
  cc1101.ReadBurst(CC1101_RXFIFO, (byte *)pkt, sizeof(packet));

  /* Le o byte de RSSI */
  cc1101.Read(CC1101_RXFIFO, &rssi);

  /* Le o byte de LQI */
  stat = cc1101.Read(CC1101_RXFIFO, &lqi);

  //-- Rotina para guardar histórico de RSSI e PER ----------------------------------------
------------
  int i;
  int value;
  for(i=0;i<NUMBER_SAMPLES;i++){
    if(hist_rssi[pkt->NetHdr[2]][i]==255){
      hist_rssi[pkt->NetHdr[2]][i]=rssi;
      if(i<(NUMBER_SAMPLES-1))
        hist_rssi[pkt->NetHdr[2]][i+1]=255;
      else
        hist_rssi[pkt->NetHdr[2]][0]=255;
      break;
    }

    if(Transp.pkt_info[0]==pkt->TranspHdr[2]        &&        Transp.pkt_info[1]==pkt-
>NetHdr[2]){
      Transp.hist_per[Transp.pkt_info[1]][PKT_RECEIVED]                                =
Transp.hist_per[Transp.pkt_info[1]][PKT_RECEIVED] + 1;
      //Limpa a variável de info de Pacote
      Transp.pkt_info[0] = 0;
      Transp.pkt_info[1] = 255;
    }

  }
  //-------------------------------------------------------------------------------------------

  /* Verifica se o endereço destino é o meu - o pacote é para mim */
  if (pkt->NetHdr[0] != Net.my_addr)
  {
    //delay(30000);
    //digitalWrite (IO4_PIN, HIGH);
    //Mac.send(pkt);
    //delay(30000);
```

```
    //digitalWrite (IO4_PIN, LOW);

    digitalWrite (6, LOW);
    return ERR;
  }

  /* Trata um possível overflow do FIFO de RX */
  if ((stat & 0xF0) == 0x60){
    cc1101.Strobe(CC1101_SFRX); /* Descarta o FIFO de RX */

    digitalWrite (6, LOW);
    return ERR;
  }

  /* Coloca a informação de RSSI e LQI no cabeçalho da camada física */
  pkt->PhyHdr[0] = rssi;
  pkt->PhyHdr[1] = lqi;

  /* Envia o pacote para a camada superior */
  Mac.receive(pkt);

  digitalWrite (6, LOW);

  return OK;
}

/* Instanciação do objeto de acesso à classe da camada Física */
PHY Phy = PHY();
```

Arquivo: _2_MAC.ino

```
// MAC : classe da camada de Controle de Acesso ao Meio

// Mais informações em www.radiuino.cc
// Copyright (c) 2011
// Author: Pedro Henrique Gomes e Omar C. Branquinho
// Versão 1.0: 12/09/2011

// Modificado por: Lucas Augusto de Araujo Marques Leão
// Nova versão 2.0: 01/02/2014
// Modificações para o protocolo FLBRA

// Este arquivo é parte da plataforma Radiuino
// Este programa é um software livre; você pode redistribui-lo e/ou modifica-lo dentro dos termos da Licença Pública Geral Menor GNU
// como publicada pela Fundação do Software Livre (FSF); na versão 2 da Licença, ou (na sua opnião) qualquer futura versão.
```

```
// Este programa é distribuido na esperança que possa ser  util, mas SEM NENHUMA GARANTIA; sem uma garantia implicita
// de ADEQUAÇÂO a qualquer MERCADO ou APLICAÇÃO EM PARTICULAR. Veja a Licença Pública Geral Menor GNU para maiores detalhes.
// Você deve ter recebido uma cópia da Licença Pública Geral Menor GNU junto com este programa, se não, escreva para a Fundação
// do Software Livre(FSF) Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

// This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library
// is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
// or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public License for more details. You should have received a copy
// of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St,
// Fifth Floor, Boston, MA  02110-1301  USA

#include "Headers.h"

/**
 * Construtor da camada de Controle de Acesso ao Meio.
 */
MAC::MAC()
{
}

/**
 * Inicializa a camada de Controle de Acesso ao Meio.
 */
void MAC::initialize(void)
{
  time_to_sleep = -1;
}

/**
 * Envia o pacote para a camada inferior
 */
inline void MAC::send(packet * pkt)
{
  unsigned long starttime = millis();

  /* Aguarda enquanto o canal está ocupado. Espera no máximo 100 ms */
  while(Phy.carrierSense() && ((millis() - starttime) < 100));

  /* Envia para a camada inferior */
  Phy.send(pkt);
```

```
  return;
}

/**
 * Recebe o pacote da camada inferior
 */
inline void MAC::receive(packet * pkt)
{
  /* Se a mensagem é do tipo SLEEP */
  if (pkt->MACHdr[0] == SLEEP_MSG) {

    /* Calcula o tempo total para dormir */
    time_to_sleep = 256 * (pkt->MACHdr[1] & 0x7F) + pkt->MACHdr[2];

    /* Retorna um pacote de SLEEP ACK para o emissor */
    pkt->MACHdr[0] = SLEEP_ACK;

    /* Troca os endereços de Origem e Destino */
    Net.swapAddresses(pkt, 255);

    /* Send SLEEP ACK packet */
    Phy.send(pkt);
  }
  else {
    Net.receive(pkt);
  }

  return;
}

/* Instanciação do objeto de acesso à classe da camada de Controle de Acesso
ao Meio */
MAC Mac = MAC();
```

Arquivo: _3_Net.ino

```
// NET : classe da camada de Rede

// Mais informações em www.radiuino.cc
// Copyright (c) 2011
// Author: Pedro Henrique Gomes e Omar C. Branquinho
// Versão 1.0: 12/09/2011

// Modificado por: Lucas Augusto de Araujo Marques Leão
// Nova versão 2.0: 01/02/2014
// Modificações para o protocolo FLBRA
```

```
// Este arquivo é parte da plataforma Radiuino
// Este programa é um software livre; você pode redistribui-lo e/ou modifica-lo dentro dos termos da Licença Pública Geral Menor GNU
// como publicada pela Fundação do Software Livre (FSF); na versão 2 da Licença, ou (na sua opnião) qualquer futura versão.
// Este programa é distribuido na esperança que possa ser  util, mas SEM NENHUMA GARANTIA; sem uma garantia implicita
// de ADEQUAÇÂO a qualquer MERCADO ou APLICAÇÃO EM PARTICULAR. Veja a Licença Pública Geral Menor GNU para maiores detalhes.
// Você deve ter recebido uma cópia da Licença Pública Geral Menor GNU junto com este programa, se não, escreva para a Fundação
// do Software Livre(FSF) Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

// This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library
// is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
// or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public License for more details. You should have received a copy
// of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St,
// Fifth Floor, Boston, MA  02110-1301  USA

#include "Headers.h"

/**
 * Construtor da camada de Rede.
 */
NET::NET()
{


//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
  //                  ENDEREÇO DO SENSOR

//+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
  my_addr = 13;   /* Endereço do próprio sensor */

  //my_dest = 3;   // destino - é o endereço de subida. Ou seja, quando chega um pacote que vem do vizinho envia para este endereço
  //my_neighbor = 3;   //vizinho - é o endereço de descida. Ou seja, é o vizinho para o qual é enviado o pacote na descida
}
```

```
/**
 * Inicializa a camada de Controle de Acesso ao Meio.
 */
void NET::initialize(void)
{
}

/**
 * Realiza a troca de endereços Origem e Destino
 */
void NET::swapAddresses(packet * pkt, int addr_dest)
{
  //verifica se não é um swap simples (legado do código anterior)
  if(addr_dest!=255){
    //conta o tamanho da EEPROM
    int count=0;
    //recebe o valor gravado na EEPROM
    int value=0;
    //percorre a EEPROM para verificar o endereço de destino
    while(value!=255 && count<=ROUTE_TABLE_END){
      value = EEPROM.read(count);
      if(value==addr_dest){
        value = EEPROM.read(count+1);
        pkt->NetHdr[0] = value;
        pkt->NetHdr[2] = Net.my_addr;
        return;
      }
      else
        count = count + 2;
    }
    return;
  }
  else{
    /* Troca os endereços de destino e origem para a retranmissão dos pacotes */
    pkt->NetHdr[0] = pkt->NetHdr[2];
    pkt->NetHdr[2] = Net.my_addr;
  }
}

/**
 * Envia o pacote para a camada inferior
 */
inline void NET::send(packet * pkt)
{
  /* Envia para a camada inferior */
  digitalWrite (IO4_PIN, HIGH);
  Mac.send(pkt);
  delay(20000);
  digitalWrite (IO4_PIN, LOW);
```

```
  return;
}


/**
 * Rotina para configuração da tabela de roteamento
 */
void NET::routeChange(packet * pkt)
{
  //variável que armazena valor lido da EEPROM
  int value;
  //contador do endereçamento da EEPROM
  int count_eeprom;

  //verifica se a tabela de roteamento deve ser completamente apagada
  if(pkt->Data[2]==100)

for(count_eeprom=ROUTE_TABLE_INI;count_eeprom<=ROUTE_TABLE_END;co
unt_eeprom++)
     EEPROM.write(count_eeprom,255);

  //contador do endereçamento do pacote
  int count_pkt=4;
  //flag que indica se um sensor já consta na tabela de roteamento (0 - não, 1 -
sim)
  int SensorIsRegistered = 0;

  value=0;
  count_eeprom=0;

  //loop que percorre os dados do pacote
  while(pkt->Data[count_pkt]<100){
    SensorIsRegistered = 0;
    //loop que percorre os dados da EEPROM
    value = EEPROM.read(count_eeprom);
    while(value!=255 && count_eeprom<=ROUTE_TABLE_END){
      //verifica se o sensor indicado no pacote existe na tabela da EEPROM
      if(value==pkt->Data[count_pkt]){
        //caso exista, então atualiza novo valor
        EEPROM.write(count_eeprom+1,pkt->Data[count_pkt+1]);
        SensorIsRegistered = 1;
        break;
      }
      count_eeprom = count_eeprom + 2;
      value = EEPROM.read(count_eeprom);
    }
    //caso sensor não exista na tabela da EEPROM, adiciona como nova entrada
    if(SensorIsRegistered==0){
      EEPROM.write(count_eeprom,pkt->Data[count_pkt]);
      EEPROM.write(count_eeprom+1,pkt->Data[count_pkt+1]);
```

```
      //incremento para o próximo endereço da EEPROM
      count_eeprom = count_eeprom + 2;
      //marca o final da lista da EEPROM
      EEPROM.write(count_eeprom,255);
    }
    //incremento para o próximo endereço do pacote
    count_pkt = count_pkt + 2;
  }

  //marca que a configuração foi realizada
  pkt->TranspHdr[3]=100;

  //inverte os endereços para que o pacote seja devolvido (ACK)
  swapAddresses(pkt, 0);
}

/**
 * Rotina para coleta de dados (RSSI, Route Table)
 */
void NET::infoCheck(packet * pkt, int pkt_type)
{
  int count=0;
  int value=0;
  int num_amostra=0;

  switch (pkt_type){
  case 1:
    //coleta a leitura de RSSI
    pkt->Data[2]=pkt->PhyHdr[0];

    //marca que aleitura foi realizada
    pkt->TranspHdr[3]=100;

    //inverte os endereços para que o pacote seja devolvido (ACK)
    swapAddresses(pkt, 0);
    break;
  case 2:
    count=0;
    while(count<=ROUTE_TABLE_END){
      value = EEPROM.read(count);
      if(value==255)
        break;
      else
        pkt->Data[count+18]=value;
      count++;
    }

    //marca que aleitura foi realizada
    pkt->TranspHdr[3]=100;
```

```
    //inverte os endereços para que o pacote seja devolvido (ACK)
    swapAddresses(pkt, 0);
    break;
  case 3:
    if(pkt->NetHdr[0]==Net.my_addr && pkt->NetHdr[2]==0 && (pkt->Data[1]==0 ||
pkt->Data[1]==Net.my_addr) && (pkt->Data[2]==0 || pkt->Data[2]==Net.my_addr))
    {
      //segundo e primeiro sensores ok
      pkt->TranspHdr[3]=100;

      //coleta a leitura de RSSI instantâneo
      pkt->Data[3]=pkt->PhyHdr[0];

      //cálculo da média do RSSI histórico
      for(count=0;count<NUMBER_SAMPLES;count++)
        if(Phy.hist_rssi[0][count]!=255){
          value = value + Phy.hist_rssi[0][count];
          num_amostra = num_amostra + 1;
        }
      //coleta da média do RSSI
      pkt->Data[4] = value/num_amostra;

      //cálculo do desvio padrão do RSSI histórico
      for(count=0;count<NUMBER_SAMPLES;count++)
        if(Phy.hist_rssi[0][count]!=255){
          value = value + pow((Phy.hist_rssi[0][count]-pkt->Data[4]),2);
        }
      //coleta desvio padrão do RSSI
      pkt->Data[5] = value;

      //número de amostras
      pkt->Data[6] = num_amostra;

      //coleta o dado de PER - Pacotes Enviados
      pkt->Data[7]                =                Transp.hist_per[0][PKT_SENT];//(1-
(Transp.hist_per[0][PKT_RECEIVED]/Transp.hist_per[0][PKT_SENT]))*100;

      //coleta o dado de PER - Pacotes Recebidos
      pkt->Data[8]=Transp.hist_per[0][PKT_RECEIVED];

      //altera o endereço para devolução do pacote
      swapAddresses(pkt, 0);
    }
    else{
      if(pkt->TranspHdr[3]==0 && pkt->Data[1]!=Net.my_addr){
        //inverte os endereços para que seja feita a medida
        pkt->NetHdr[0] = pkt->Data[1];
        pkt->NetHdr[2] = Net.my_addr;

        //marca que o setou o primeiro sensor
```

```
      pkt->TranspHdr[3]=1;
    }
    else{
      if(pkt->TranspHdr[3]==1 || (pkt->TranspHdr[3]==0 && pkt->Data[1]==Net.my_addr)){
        //inverte os endereços para que seja feita a medida
        pkt->NetHdr[0] = pkt->Data[2];
        pkt->NetHdr[2] = Net.my_addr;

        //marca que o setou o segundo sensor
        pkt->TranspHdr[3]=2;
      }
      else{
        if(pkt->TranspHdr[3]==2 && pkt->NetHdr[0]==pkt->Data[2] && pkt->NetHdr[2]==pkt->Data[1]){
          //coleta a leitura de RSSI
          pkt->Data[3]=pkt->PhyHdr[0];

          //cálculo da média do RSSI histórico
          for(count=0;count<NUMBER_SAMPLES;count++)
            if(Phy.hist_rssi[pkt->Data[1]][count]!=255){
              value = value + Phy.hist_rssi[pkt->Data[1]][count];
              num_amostra = num_amostra + 1;
            }
          //coleta da média do RSSI
          pkt->Data[4] = value/num_amostra;

          //cálculo do desvio padrão do RSSI histórico
          for(count=0;count<NUMBER_SAMPLES;count++)
            if(Phy.hist_rssi[pkt->Data[1]][count]!=255)
              value = value + pow((Phy.hist_rssi[pkt->Data[1]][count]-pkt->Data[4]),2);
          //coleta desvio padrão do RSSI
          pkt->Data[5] = value;

          //número de amostras
          pkt->Data[6] = num_amostra;

          //coleta o dado de PER - Pacotes Enviados
          pkt->Data[7] = Transp.hist_per[pkt->Data[1]][PKT_SENT];//(1-(Transp.hist_per[pkt->Data[1]][PKT_RECEIVED]/Transp.hist_per[pkt->Data[1]][PKT_SENT]))*100;

          //coleta o dado de PER - Pacotes Recebidos
          pkt->Data[8]=Transp.hist_per[pkt->Data[1]][PKT_RECEIVED];

          //marca que a leitura foi realizada
          pkt->TranspHdr[3]=100;

          //altera o endereço para devolução do pacote
          swapAddresses(pkt, 0);
```

```
        }
       }
      }
    }
    break;
  }
  return;
}

/**
 * Recebe o pacote da camada inferior
 */
inline void NET::receive(packet * pkt) {

  //rotina para debug - mostra o caminho que o pacote fez//////////////////
  if(pkt->TranspHdr[1]==1 && pkt->Data[0]==20){
    int contador = 0;
    contador = pkt->Data[6];
    pkt->Data[contador+7]=Net.my_addr;
    contador++;
    pkt->Data[6] = contador;
  }
  ////////////////////////////////////

  //--Verifica se é um pacote de configuração de roteamento
  if(pkt->Data[0]==10 && pkt->Data[1]==Net.my_addr && pkt->TranspHdr[3]==0){
    routeChange(pkt);
  }
  //--Verifica se é um pacote de leitura de RSSI
  else if(pkt->Data[0]==20 && pkt->Data[1]==Net.my_addr && pkt->TranspHdr[3]==0){
    infoCheck(pkt,1);
  }
  //-- Verifica se é um pacote de retorno do conteudo da EEPROM (tabela de roteamento)
  else if(pkt->Data[0]==30 && pkt->Data[1]==Net.my_addr && pkt->TranspHdr[3]==0){
    infoCheck(pkt,2);
  }
  //--Verifica se é um pacote de leitura de RSSI entre dois sensores
  else if(pkt->Data[0]==40 && pkt->TranspHdr[3]!=100){
    infoCheck(pkt,3);
  }
  //-- Verifica se o pacote não foi utilizado e faz o Forward
  else if(pkt->TranspHdr[3]==0){
    //faz forward do pacote
    swapAddresses(pkt,pkt->Data[1]);
  }
  //-- Se o pacote já foi utilizado, apenas segue a devolução
  else{
```

```
    //altera o endereço para devolução do pacote
    swapAddresses(pkt, 0);
  }

  //envia o pacote para a camada superior
  Transp.receive(pkt);
  return;
}

/* Instanciação do objeto de acesso à classe da camada de Rede */
NET Net = NET();
```

Arquivo: _4_Transp.ino

```
// Tansp : classe da camada de Transporte

// Mais informações em www.radiuino.cc
// Copyright (c) 2011
// Author: Pedro Henrique Gomes e Omar C. Branquinho
// Versão 1.0: 12/09/2011

// Modificado por: Lucas Augusto de Araujo Marques Leão
// Nova versão 2.0: 01/02/2014
// Modificações para o protocolo FLBRA

// Este arquivo é parte da plataforma Radiuino
// Este programa é um software livre; você pode redistribui-lo e/ou modifica-lo dentro dos termos da Licença Pública Geral Menor GNU
// como publicada pela Fundação do Software Livre (FSF); na versão 2 da Licença, ou (na sua opnião) qualquer futura versão.
// Este programa é distribuido na esperança que possa ser  util, mas SEM NENHUMA GARANTIA; sem uma garantia implicita
// de ADEQUAÇÂO a qualquer MERCADO ou APLICAÇÃO EM PARTICULAR. Veja a Licença Pública Geral Menor GNU para maiores detalhes.
// Você deve ter recebido uma cópia da Licença Pública Geral Menor GNU junto com este programa, se não, escreva para a Fundação
// do Software Livre(FSF) Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
```

```
// This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library
// is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
// or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public License for more details. You should have received a copy
// of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St,
// Fifth Floor, Boston, MA  02110-1301  USA

#include "Headers.h"

/**
 * Construtor da camada de Transporte.
 */
TRANSP::TRANSP()
{

}

/**
 * Inicializa a camada de Transporte.
 */
void TRANSP::initialize(void)
{
  int i;
  for(i=0;i<NETWORK_SIZE;i++){
    hist_per[i][PKT_SENT] = 0;
    hist_per[i][PKT_RECEIVED] = 0;
  }
  pkt_info[0] = 0;
  pkt_info[1] = 255;
}

/**
 * Envia o pacote para a camada inferior
 */
inline void TRANSP::send(packet * pkt)
{
  //Sinaliza o ID do pacote que está sendo enviado
  if(pkt_info[0]==0 && pkt_info[1]==255){
    pkt_info[0] = pkt->TranspHdr[2];
    pkt_info[1] = pkt->NetHdr[0];
    hist_per[pkt_info[1]][PKT_SENT] = hist_per[pkt_info[1]][PKT_SENT] + 1;
//pacotes enviados
  }

  /* Envia para a camada inferior */
```

```
  Net.send(pkt);

  return;
}

/**
 * Recebe o pacote da camada inferior
 */
inline void TRANSP::receive(packet * pkt)
{
  static byte counter = 0;

  pkt_info[0] = 0;
  pkt_info[1] = 255;

  /* Insere um contado no cabeçalho de transporte do pacote */
  pkt->TranspHdr[0] = counter++;

  /* Envia para a camada superior */
  App.receive(pkt);

  return;
}

/* Instanciação do objeto de acesso à classe da camada de Transporte */
TRANSP Transp = TRANSP();
```

Arquivo: _5_App.ino

```
// APP : classe da camada de Aplicação

// Mais informações em www.radiuino.cc
// Copyright (c) 2011
// Author: Pedro Henrique Gomes e Omar C. Branquinho
// Versão 1.0: 12/09/2011

// Modificado por: Lucas Augusto de Araujo Marques Leão
// Nova versão 2.0: 01/02/2014
// Modificações para o protocolo FLBRA

// Este arquivo é parte da plataforma Radiuino
// Este programa é um software livre; você pode redistribui-lo e/ou modifica-lo dentro dos termos da Licença Pública Geral Menor GNU
// como publicada pela Fundação do Software Livre (FSF); na versão 2 da Licença, ou (na sua opnião) qualquer futura versão.
```

```
// Este programa é distribuido na esperança que possa ser  util, mas SEM NENHUMA GARANTIA; sem uma garantia implicita
// de ADEQUAÇÂO a qualquer MERCADO ou APLICAÇÃO EM PARTICULAR. Veja a Licença Pública Geral Menor GNU para maiores detalhes.
// Você deve ter recebido uma cópia da Licença Pública Geral Menor GNU junto com este programa, se não, escreva para a Fundação
// do Software Livre(FSF) Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

// This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library
// is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
// or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public License for more details. You should have received a copy
// of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St,
// Fifth Floor, Boston, MA  02110-1301  USA

#include "Headers.h"

/**
 * Construtor da camada de Aplicação.
 */
APP::APP()
{
}

/**
 * Inicializa a camada de Aplicação.
 */
void APP::initialize(void)
{
  /* Configura Port C Pin 0 como Output */

  pinMode (IO0_PIN, OUTPUT);
  pinMode (IO1_PIN, OUTPUT);
  pinMode (IO2_PIN, OUTPUT);
  pinMode (IO3_PIN, OUTPUT);
  pinMode (IO4_PIN, OUTPUT);
  pinMode (IO5_PIN, OUTPUT);

  IO_2 = 1; /* Enable LDR */
  IO_0 = IO_1 = IO_3 = IO_4 = IO_5 = 0;

}

/**
```

```
 * Envia o pacote para a camada inferior
 */
inline void APP::send(packet * pkt)
{
  return;
}

/**
 * Recebe o pacote da camada inferior
 */
inline void APP::receive(packet * pkt)
{
  int AD0, AD1, AD2, AD3, AD4, AD5;
  if(pkt->MACHdr[3]==2 & 0)
  {
    digitalWrite (IO4_PIN, HIGH);
    delay(50000);
    digitalWrite (IO4_PIN, LOW);
    delay(50000);
    digitalWrite (IO4_PIN, HIGH);
    delay(50000);
    digitalWrite (IO4_PIN, LOW);
    pkt->MACHdr[3]= 222;
  }

   /* Envia para a camada inferior */
  Transp.send(pkt);
  //Phy.send(pkt);

  return;
}

/* Instanciação do objeto de acesso à classe da camada de Aplicação */
APP App = APP();
```

Anexo C – Implementação do Software Gerente

Arquivo: Principal_Fuzzy.py

```
import serial
import numpy
import math
import time
import struct
import itertools
import time
import random
import radiuino_Fuzzy
#-------------------------------------------------------------------------------------------------------------
#Inicializa COM.
ser = radiuino_Fuzzy.fInicializaSerialCOM()
print("COM conectada: " + radiuino_Fuzzy.fCOMConectada(ser) + "\n")
#-------------------------------------------------------------------------------------------------------------
# Requisições iniciais.
n_sensores = raw_input("Digite o número de sensores que estão na rede:\n")
Tempo = raw_input("Digite o tempo, em minutos, para que a rota seja reavaliada:\n")
#-------------------------------------------------------------------------------------------------------------
# Declaração de matrizes e listas.
size = int(n_sensores)
SCusto = numpy.zeros((size,size+1))
Custo = numpy.zeros((size,size))
S = list()
dados = []
t = time.clock()
temporal = int(Tempo)
name = "DADOS"
tamanho = []
rota = 'RM;'
#-------------------------------------------------------------------------------------------------------------
# Cria as matrizes, com o tamanho dado pela matriz PATH.
RSSI = numpy.zeros((size,size)) # Matriz de RSSI.
MEDIA_RSSI = numpy.zeros((size,size)) # Matriz de Desvio Padrão da RSSI.
PER = numpy.zeros((size,size)) # PER.
DP = numpy.zeros((size,size)) # Desvio padrão.
#-------------------------------------------------------------------------------------------------------------
# Inicializa Matriz SCusto.
for i in range(0,int(n_sensores)):
        SCusto[i][0] = i
#-------------------------------------------------------------------------------------------------------------
# Rota inicial.
```

```
PATH = radiuino_Fuzzy.PrimeiraRota(ser,int(n_sensores)) # Define a primeira rota. Tem, inicialmente, todos os sensores da rede
size = len(PATH)
#-------------------------------------------------------------------------------------------------------------------
while True: #Mantém o programa sempre rodando.
#-------------------------------------------------------------------------------------------------------------------
# Determina para qual sensor a base deve enviar o pacote, para sua leitura.
            for i in range(0,len(PATH)):
                  tamanho.append(str(i))
                  tamanho.append(str(PATH[i][0]))
#-------------------------------------------------------------------------------------------------------------------
# Leitura de RSSI entre sensores.
            for i in range(0,size):
               for j in range(0,size):
                  if i != j:
                     time.sleep(1)
                     dados = radiuino_Fuzzy.fLeituraDeRSSI(ser,i,j,tamanho)
                     print("-------------------------------------------------------------------------------------------------------")
#-------------------------------------------------------------------------------------------------------------------
# Calculo dos parâmetros.
                        if len(dados) > 0:
                           x = radiuino_Fuzzy.CalculaRSSI(dados[20])
                           MEDIA_RSSI[i][j] = x
                           media_rssi = x
                           rssi = radiuino_Fuzzy.CalculaRSSI(dados[19])
                           RSSI[i][j] = rssi
                           if dados[21] > 0:
                                 dp = math.sqrt((1.0/(float(dados[22])-1.0))*float(dados[21]))
                           else:
                                 dp = 10
                           DP[i][j] = dp
                           if dados[24]>=dados[23]:
                              PER[i][j] = 0
                           else:
                              PER[i][j] = float(1 - float(float(dados[24])/float(dados[23])))
                        else:
                           MEDIA_RSSI[i][j] = numpy.inf
                           media_rssi = numpy.inf
                           RSSI[i][j] = numpy.inf
                           rssi = numpy.inf
                           DP[i][j] = numpy.inf
                           dp = numpy.inf
                           PER[i][j] = 1
                        per = PER[i][j]
```

```
                    Custo[i][j] = radiuino_Fuzzy.Fuzzy(rssi,dp,per)
            #radiuino_Fuzzy.EscreveMatriz(PER) #Teste
            SALTOS = radiuino_Fuzzy.Saltos(PATH,SCusto)
            Pera = radiuino_Fuzzy.PER_A(PATH,PER,SCusto)

#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Arquiva os parâmetros.
            radiuino_Fuzzy.ArquivaPATH(name,PATH,rota,SCusto)

radiuino_Fuzzy.ArquivaMatriz(name,RSSI,MEDIA_RSSI,DP,SALTOS,Pera,SCust
o)
            rota = 'RM;'
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Condição de disparo.
            if  time.clock() - t > 60*temporal: #Parâmetros temporal de disparo
                print("Tempo decorrido = " + str(time.clock() - t))
                #temporal += 1
                t = time.clock()
                rota = 'RA;'
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Recalcula a rota.
                for i in range(0,size):
                    S.append(i)
                #radiuino_Fuzzy.EscreveVetor(S)
                #print"Custo"
                #radiuino_Fuzzy.EscreveMatriz(Custo)
                SCusto = radiuino_Fuzzy.Shadows(Custo,S)
                S = []
                PATH = radiuino_Fuzzy.DefineRota(ser,SCusto)
                print"PATH"
                radiuino_Fuzzy.EscreveMatriz(PATH)
                print "\n"
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
```

Arquivo: radiuino_Fuzzy.py

```
import serial
import numpy
import math
import time
import struct
import itertools
import time
import random
#####################################################################
#####################################################################
```

```
########################################
pkt = {}
########################################################################
########################################################################
########################################
def fInicializaSerialCOM(): #Inicializa COM.
#Cria instancia da porta serial.
    ser = serial.Serial(port='COM17',
                        baudrate=9600,
                        parity=serial.PARITY_NONE,
                        timeout=1)
    ser.write(hex(255))
    time.sleep(2)
    ser.read(52)
    return ser
########################################################################
########################################################################
########################################
#Fecha COM.
def fFechaCOM(ser):
    ser = fInicializaSerialCOM()
    ser.close()
    return
########################################################################
########################################################################
########################################
# Verifica se a porta esta conectada e retorna qual COM.
def fCOMConectada(ser):
    com = ser.portstr
    return com
########################################################################
########################################################################
########################################
def fLeituraDeRSSI(ser,sensor_ini,sensor_fim,tamanho): #Envia pacote de leitura
de RSSI, Desvio Padrao e PER.
    for i in range(0,52): #Inicializa o pacote com ZEROS.
        pkt[i] = 0

    pkt[14] = int(random.uniform(1, 255)) #ID unico do Pacote.
    pkt[16] = 40 #Indica que é um pacote de leitura de RSSI entre dois sensores.

    pkt[8] = 255
    if(sensor_ini==0):
        pkt[8] = sensor_fim
    else:
        while i+1<len(tamanho):
            if(tamanho[i]==sensor_ini):
                pkt[8] = tamanho[i+1]
            i=i+2
        if(pkt[8]==255):
```

```
            pkt[8] = sensor_ini

        #Rotina desnecessaria, pois já está indicando o sensor inicial
        #else:
        #    pkt[8] = sensor_ini

#Sensores que terao o RSSI verificados.
    pkt[17] = sensor_ini
    pkt[18] = sensor_fim

#Limpa o buffer da serial.
    ser.flushOutput()
    lineteste=""
    TXbyte = ''
    for k in range(0,52): #Transmite o pacote.
        TXbyte = TXbyte+chr(pkt[k])
        lineteste = lineteste + str(pkt[k]) + " "
    ser.write(TXbyte)
    time.sleep(1)
    line = ser.read(52)
    lineteste2=""
    line_t={}
    for i in range(0,len(line)):
        line_t[i] = ord(line[i])
        lineteste2 = lineteste2 + str(ord(line[i])) + " "
    print("Envia para %s, leitura entre %s e %s"%(pkt[8],sensor_ini,sensor_fim))
    print("Enviado : " + lineteste )
    print("Recebido: " + lineteste2)
    return line_t
#########################################################################
#########################################################################
########################################
def Rota_Sensor(ser,Sensor,Destino,Caminho,LimparTabela):
    for i in range(0,52): #Inicializa todo o pacote com ZEROS.
        pkt[i] = 0
    print("Sensor = %s, Destino = %s, Caminho = %s, Limpar Tabela: %s
\n"%(Sensor,Destino,Caminho,LimparTabela))
    pkt[14] = int(random.uniform(1, 255)) #ID único do pacote.
    pkt[8] = int(Sensor) #Sensor que deve receber a alteração de rota.Mandamos o
pacote para este sensor.
    pkt[16] = 10 #Indica que é um pacote de alteracao de rota.
    pkt[17] = int(Sensor) #Sensor que tera rota alterada.
    pkt[20] = int(Destino) #Sensor destino.
    pkt[21] = int(Caminho) #Sensor caminho.
    pkt[22] = 100 #Fim da Rota
    if(LimparTabela): #Indica se a tabela de rota deve ser apagada completamente.
        pkt[18] = 100
    else:
        pkt[18] = 0
    ser.flushOutput() #Limpa o buffer da serial.
```

```
    TXbyte = ''
    for k in range(0,52): #Transmite o pacote.
        TXbyte = TXbyte+chr(pkt[k])
    ser.write(TXbyte)
    time.sleep(1)
    line = ser.read(52)
    line_t={}
    for i in range(0,len(line)):
        line_t[i] = ord(line[i])
    return line_t
#########################################################################
#########################################################################
########################################
def Dijkstra(SCusto,destino):
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Listas e variáveis necessárias para o programa.
    rede = len(SCusto) # Numero de rede na rede.
    inf = numpy.inf
    Custo = numpy.zeros((rede,rede))
    CT = list() # Custo.
    S = list() # Sensores disponíveis.
    aux = list() # lista auxiliar, nela estarão todos os sensores da rede.
    caminho = list() # Contém o destino e o caminho de todos os sensores até ele.
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Inicialização da matriz principal.
    for linha in range(0,rede):
        for coluna in range(0,rede):
            Custo[linha][coluna] = SCusto[linha][coluna+1]
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Inicialização das listas.
    for i in range(0,rede):
        S.append(SCusto[i][0])
        CT.append(numpy.inf)
        aux.append(SCusto[i][0])
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Cria, agora, a matriz principal.
    C = numpy.zeros((rede,4)) #Primeira Coluna: sensor, Segunda Coluna:
Permissão, Terceira Coluna: Custo, Quarta Coluna: Caminho.
    for i in range(0,rede): # Copia a tabela inicial.
        C[i][0] =  S[i]
        C[i][1] = True
        C[i][2] = numpy.inf
        C[i][3] = destino # Inicializa-se com o caminho direto.
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Caso a base não seja o destino.
```

```
    if destino != 0: # A base só é levada em conta se ela for o destino.
        for i in range(1,rede):
            Custo[0][i] = inf
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Caso o destino seja uma sombra.
    if destino not in S:
        return 'erroooooooooooou'
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Programa Principal_Fuzzy.
    C[aux.index(destino)][2] = 0 #Custo do destino ao destino é zero.
    CT[aux.index(destino)] = 0 #Custo do destino ao destino é zero.
    C[aux.index(destino)][1] = False #Permissão negada.
    inter = aux.index(destino)
    proximo = destino

    while len(S) > 0:
        for sensor in range(0,rede):
            if C[sensor][2] > C[inter][2] + Custo[inter][sensor]:
                C[sensor][2] =  C[inter][2] + Custo[inter][sensor]
                C[sensor][3] = proximo

        C[inter][1] = False
        apaga = S.index(proximo) # Índice referente ao sensor que será apagado.
        S.pop(apaga) # Apaga o sensor usado.
        CT.pop(apaga) # Apaga o custo referente ao sensor usado.

        for i in range(0,len(S)): # Escreve o custo referente aos sensores.
            CT[i] = C[aux.index(S[i])][2]

        if len(CT) > 0: # Define qual é o próximo sensor a ser utilizado na lista
            indice = CT.index(min(CT))
            proximo = S[indice]
            #print ("Sensor intermediário = %s"%proximo)
            inter = aux.index(proximo)
    caminho.append(destino)
    #radiuino.ImprimeMatriz(Custo)
    for i in range(0,rede):
        caminho.append(int(C[i][3]))
    #print "Matriz C"
    #EscreveMatriz(C)
    return caminho
#########################################################################
#########################################################################
########################################
def DefineRota(ser,SCusto):
    size = len(SCusto)
    caminho_sensor = []
    S = list()
```

```
    PATH = numpy.zeros((size,size))
    limpa_tabela = True
    for i in range(0,size):
        S.append(SCusto[i][0])
    for x in range(0,size): # Percorre todos os sensores.
        caminho_sensor = Dijkstra(SCusto,S[x])
        if type(caminho_sensor) != str: # Se o sensor destino está funcionando.
            aux = caminho_sensor[0]
            caminho_sensor.pop(0)
            for y in range(0,len(PATH)):
                PATH[S.index(aux)][y] = caminho_sensor[y]
                if (y != x and y != 0):
                    Rota_Sensor(ser,y,int(aux),int(caminho_sensor[y]),limpa_tabela)
        caminho_sensor = []
        limpa_tabela = False
    #print "PATH"
    #EscreveMatriz(PATH)
    #print "\n"
    return PATH
##########################################################################
##########################################################################
##########################################
def Shadows(MatrizCusto,Sensores):
    inf = numpy.inf
    rede = len(MatrizCusto)
    SCusto = numpy.zeros((rede,rede+1))
    sombra = 0
    linha = 0
    coluna = 0
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Programa principal.
    for linha in range(0,rede):
        for coluna in range(0,rede):
            if MatrizCusto[linha][coluna] == inf:
                sombra += 1
        if sombra == rede - 1 and linha != 0:
            MatrizCusto = numpy.delete(MatrizCusto, (linha), axis=0) # Exclui linha
referente ao sensor sombreado.
            MatrizCusto = numpy.delete(MatrizCusto, (linha), axis=1) # Exclui coluna
referente ao sensor sombreado.
            Sensores.pop(linha) # Atualiza a lista de sensores funcionais da rede.
            return Shadows(MatrizCusto,Sensores)
        sombra = 0
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Criação da matriz SCusto.
    for j in range(0,rede):
        for k in range(0,rede):
            SCusto[j][0] = Sensores[j] # Lista de sensores funcionais da rede.
```

```
            SCusto[j][k+1] = MatrizCusto[j][k]
    #print "SCusto"
    #EscreveMatriz(SCusto)
    #print "\n"
    return SCusto
#########################################################################
#########################################################################
########################################
def PrimeiraRota(ser,n_sensores):
    SCusto = numpy.zeros((int(n_sensores),int(n_sensores)+1))
    for i in range(0,int(n_sensores)):
        SCusto[i][0] = i
    PATH = DefineRota(ser,SCusto)
    return PATH
#########################################################################
#########################################################################
########################################
def EscreveVetor(vetor):
    for i in range(0,len(vetor)):
        print vetor[i]
#########################################################################
#########################################################################
########################################
def EscreveMatriz(Matriz):
    Linha = len(Matriz)
    Coluna = len(Matriz[0])
    linha = {}
    for i in range(0,Linha):
        for j in range(0,Coluna):
            linha = str(linha) + " " + str(Matriz[i][j])
        print linha
        linha = {}
#########################################################################
#########################################################################
########################################
def ArquivaMatriz(nome,RSSI,MEDIA,DP,SALTOS,PER,SCusto):
    name = nome + ".txt"
    file = open(name, 'a')
    S = list()
    i = len(SCusto)
    j = len(SCusto)
    for l in range(0,len(SCusto)):
        S.append(SCusto[l][0])
    for a in range(0,len(S)):
        for b in range(0,len(S)):
            if RSSI[a][b] < 1e+45:
                #file.write(str(S[a]) + ";" + str(S[b]) + ";" + str(RSSI[a][b])  + ";" +
str(MEDIA[a][b]) + ";" + str(DP[a][b]) + ";" + str(PER[a][b]) + ";" +
str(int(SALTOS[a][b])) + ",")
                file.write(str(S[a]) + "->" + str(S[b]) + ": " + str(RSSI[a][b])  + " | " +
```

```
str(MEDIA[a][b]) + " | " + str(DP[a][b]) + " | " + str(PER[a][b]) + " | " +
str(int(SALTOS[a][b])) + ";")
            else:
                #file.write(str(S[a]) + "," + str(S[b]) + "," + "ERRO;")
                file.write(str(S[a]) + "->" + str(S[b]) + ": RSSI | MEDIA | DP | PER |
SALTOS;")
    file.write("\n")
    file.close()
    fil = open("PROGRAMA.txt", 'a')
    i = len(SCusto)
    j = len(SCusto)
    a = 0
    b = 0
    for a in range(0,i):
        for b in range(0,j):
            if RSSI[a][b] < 1e+45:
                #fil.write(str(S[a]) + ";" + str(S[b]) + ";" + str(RSSI[a][b]) + ";" +
str(MEDIA[a][b]) + ";" + str(DP[a][b]) + ";" + str(PER[a][b]) + ";" +
str(int(SALTOS[a][b])) + "\n")
                fil.write(str(S[a]) + "->" + str(S[b]) + ": " + str(RSSI[a][b]) + " | " +
str(MEDIA[a][b]) + " | " + str(DP[a][b]) + " | " + str(PER[a][b]) + " | " +
str(int(SALTOS[a][b])) + "\n")
            else:
                #fil.write(str(S[a]) + ";" + str(S[b]) + ";" + "ERRO\n")
                fil.write(str(S[a]) + "->" + str(S[b]) + ": RSSI | MEDIA | DP | PER |
SALTOS\n")
    fil.close()
##########################################################################
##########################################################################
#########################################
def ArquivaPATH(nome,PATH,rota,SCusto):
    S = list()
    name = nome + ".txt"
    file = open(name, 'a')
    string = time.strftime("%d/%m/%Y")
    string2 = time.strftime("%H:%M:%S")
    file.write(string +"," + string2 + ";")
    file.write(rota)
    i = len(SCusto)
    j = i
    for l in range(0,i):
        S.append(SCusto[l][0])
    for a in range(0,i):
        for b in range(0,j):
            #file.write(str(S[a]) + ";" + str(S[b]) + ";" + str(int(PATH[b][a])) + ",")
            file.write(str(S[a]) + "->" + str(S[b]) + ":" + str(int(PATH[b][a])) + ";")
    file.close()
    fil = open('Programa.txt', 'a')
    i = len(SCusto)
    j = i
```

```
    a = 0
    b = 0
    for a in range(0,i):
        for b in range(0,j):
            #fil.write(str(S[a]) + "," + str(S[b]) + "," + str(int(PATH[b][a])) + "\n")
            fil.write(str(S[a]) + "->" + str(S[b]) + ":" + str(int(PATH[b][a])) + "\n")
    fil.close()
##########################################################################
##########################################################################
##########################################
def CalculaRSSI(dados):
    if dados >= 128:
       x=((dados-256)/2.0)-74
    if dados == 0:
       x = 0
    else:
       x=(dados/2.0)-74
    return x
##########################################################################
##########################################################################
##########################################
def Saltos(PATH,SCusto):
    salto = 1
    S = list()
    SALTOS = numpy.zeros((len(PATH),len(PATH)))
    for i in range(0,len(SCusto)):
        S.append(SCusto[i][0])
#-------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------
# Programa principal.
    for a in range(0,len(PATH)):
        for b in range(0,len(PATH)):
            dest = S[a]
            orig = S[b]
            origem = orig
            while PATH[S.index(dest)][S.index(orig)] != orig and orig != dest:
                salto = salto + 1
                orig = PATH[S.index(dest)][S.index(orig)]
                if PATH[S.index(dest)][S.index(orig)] == orig:
                    return SALTOS
            if orig == dest:
                salto = 0
            SALTOS[S.index(dest)][S.index(origem)] = salto
            salto = 1
    return SALTOS
##########################################################################
##########################################################################
##########################################
def PER_A(PATH,PER,SCusto):
    Acerto = 1
```

```
    n =0
    S = list()
    rede = int(len(PATH))
    for i in range(0,len(SCusto)):
        S.append(SCusto[i][0])
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Programa principal
    PERA = numpy.zeros((rede,rede))
    for a in range(0,rede):
        for b in range(0,rede):
            dest = S[a]
            orig = S[b]
            origem = orig
            if PATH[S.index(dest)][S.index(orig)] == dest:
                PERA[S.index(dest)][S.index(orig)] = PER[S.index(dest)][S.index(orig)]
            else:
                while PATH[S.index(dest)][S.index(orig)] != dest and orig != dest:
                    x = PATH[S.index(dest)][S.index(orig)]
                    right = 1 - PER[S.index(x)][S.index(orig)]
                    Acerto = Acerto*right
                    orig = x
                    n += 1
                if n > 0:
                    x = PATH[S.index(dest)][S.index(orig)]
                    right = 1 - PER[S.index(x)][S.index(orig)]
                    Acerto = Acerto*right
                    n = 0
                PERA[S.index(dest)][S.index(origem)] = 1 - Acerto
                Acerto = 1
    print "PER"
    EscreveMatriz(PERA)
    print "\n"
    return PERA
##########################################################################
##########################################################################
##########################################
def geraPert(a,b,c,d,x):
        if b==x or c==x or (b < x and x < c):
            return 1
        if x <= a or x >= d:
            return 0
        if a < x and x < b:
            valor = float(float(x-a)/float(b-a))
            return valor
        if c < x and x < d:
            valor = float(float(d-x)/float(d-c))
            return valor
        else:
            print "ERRO"
```

```
        return 100000
#########################################################################
#########################################################################
########################################
def Defuzzifica(pa,pm,pb):
#pa = pertinencia custo alto, pm = pertinencia custo medio, pb = pertinencia custo
baixo.
    X = list()
    aux = list()
    aux2 = list()
    aux2.append(pa)
    aux2.append(pm)
    aux2.append(pb)
    X1 = 0
    X2 = 0
    MatrizE = ([0,0,3,5],[3,5,5,7],[5,7,10,10])
    for a in range(0,100):
        X.append(0.1*a)
    for j in range(0,len(X)):
        for i in range(0,len(MatrizE)):
            a = geraPert(MatrizE[i][0],MatrizE[i][1],MatrizE[i][2],MatrizE[i][3],X[j])
            aux.append(min(a,aux2[i]))
        pertinencia = max(aux)
        aux = []
        X1 = X1 + X[j]*pertinencia
        X2 = X2 + pertinencia
    if X2 > 0:
        Valor = float(X1/X2)
    else:
        return "ERRO"
    return Valor
#########################################################################
#########################################################################
########################################
def Fuzzy(rssi,dp,per):
    dados = list()
    dados.append(rssi)
    dados.append(dp)
    dados.append(per)
    j =0
    k =0
    if rssi == numpy.inf:
        return numpy.inf
    MatrizC = ([-50,-30,10,10,0],[-80,-50,-50,-30,0],[-180,-180,-70,-
50,0],[0,0,4,5,0],[4,5,5,6,0],[5,6,20,20,0],[0,0,0.1,0.3,0],[0.1,0.3,0.3,0.5,0],[0.3,0.5,1
,1,0],[0,0,0,0,0])
# Primeiro parâmetro: RSSI, segundo parâmetro: DP, terceiro parâmetro: PER.
# Quinta coluna: pertinência relativa à linha.
    MatrizR =
([8,8,8,0],[7,7,7,0],[6,2,5,0],[6,2,4,1],[6,2,3,2],[6,1,5,0],[6,1,4,1],[6,1,3,2],[6,0,5,0],[6,
```

```
0,4,2],[6,0,3,2])
# Última coluna: referente à coluna que será colocado o valor na nona linha.
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------#
# Programa principal.
        for i in range(0,9): # Gera as pertinências iniciais.
            if j > 2:
                k +=1
                j = 0
            a = float(dados[k])
            c = float(geraPert(MatrizC[i][0],MatrizC[i][1],MatrizC[i][2],MatrizC[i][3],a))
            MatrizC[i][4] = c
            j += 1
        for k in range(0,11): # Aplica a regra à MatrizC.
            MatrizC[9][MatrizR[k][3]]                                                                    =
max(MatrizC[9][MatrizR[k][3]],min(MatrizC[MatrizR[k][0]][4],MatrizC[MatrizR[k][1]][
4],MatrizC[MatrizR[k][2]][4]))
        valor = Defuzzifica(MatrizC[9][0],MatrizC[9][1],MatrizC[9][2])
        if type(valor) != str:
            custo = float(1/float(valor))
        else:
            return numpy.inf #estava como "return valor", mas troquei para retornar
valor infinito
        return custo
#########################################################################
#########################################################################
#########################################
```

Arquivo: Principal_RBF.py

```
import serial
import numpy
import math
import time
import struct
import itertools
import time
import random
import radiuino_RBF
#-------------------------------------------------------------------------------------------------------------
------------
#Inicializa COM.
ser = radiuino_RBF.fInicializaSerialCOM()
print("COM conectada: " + radiuino_RBF.fCOMConectada(ser) + "\n")
#-------------------------------------------------------------------------------------------------------------
------------
# Requisições iniciais.
n_sensores = raw_input("Digite o número de sensores que estão na rede:\n")
Tempo = raw_input("Digite o tempo, em minutos, para que a rota seja
reavaliada:\n")
#-------------------------------------------------------------------------------------------------------------
```

```
------------
# Declaração de matrizes e listas.
size = int(n_sensores)
SCusto = numpy.zeros((size,size+1))
Custo = numpy.zeros((size,size))
S = list()
dados = []
t = time.clock()
temporal = int(Tempo)
name = "DADOS"
tamanho = []
rota = 'RM;'
#-------------------------------------------------------------------------------------------------------------
------------
# Cria as matrizes, com o tamanho dado pela matriz PATH.
RSSI_INV = numpy.zeros((size,size)) # Matriz de RSSI invertida.
RSSI = numpy.zeros((size,size)) # Matriz de RSSI.
MEDIA_RSSI = numpy.zeros((size,size)) # Matriz de Desvio Padrão da RSSI.
PER = numpy.zeros((size,size)) # PER.
DP = numpy.zeros((size,size)) # Desvio padrão.
#-------------------------------------------------------------------------------------------------------------
------------
# Inicializa Matriz SCusto.
for i in range(0,int(n_sensores)):
        SCusto[i][0] = i
#-------------------------------------------------------------------------------------------------------------
------------
# Rota inicial.
PATH = radiuino_RBF.PrimeiraRota(ser,int(n_sensores)) # Define a primeira rota.
Tem, inicialmente, todos os sensores da rede
size = len(PATH)
#-------------------------------------------------------------------------------------------------------------
------------
while True: #Mantém o programa sempre rodando.
#-------------------------------------------------------------------------------------------------------------
------------
# Determina para qual sensor a base deve enviar o pacote, para sua leitura.
            for i in range(0,len(PATH)):
                    tamanho.append(str(i))
                    tamanho.append(str(PATH[i][0]))
#-------------------------------------------------------------------------------------------------------------
------------
# Leitura de RSSI entre sensores.
            for i in range(0,size):
                for j in range(0,size):
                    if i != j:
                        time.sleep(1)
                        dados = radiuino_RBF.fLeituraDeRSSI(ser,i,j,tamanho)
                        print("-----------------------------------------------------------------------
-------------------------")
```

```
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------
# Calculo dos parâmetros.
                        if len(dados) > 0:
                            x = radiuino_RBF.CalculaRSSI(dados[20])
                            MEDIA_RSSI[i][j] = x
                            media_rssi = x
                            if type(x) == int or type(x) == float:
                                x = float(x/10)
                                x = float(math.pow(10,x))
                                x = 0.001*x
                                x = 1/x
                                RSSI_INV[i][j] = x
                            rssi = radiuino_RBF.CalculaRSSI(dados[19])
                            RSSI[i][j] = rssi
                            if dados[21] > 0:
                                  dp              =              math.sqrt((1.0/(float(dados[22])-
1.0))*float(dados[21]))
                            else:
                                  dp = 10
                            DP[i][j] = dp
                            if dados[24]>=dados[23]:
                                PER[i][j] = 0
                            else:
                                PER[i][j] = float(1 - float(float(dados[24])/float(dados[23])))
                        else:
                            MEDIA_RSSI[i][j] = numpy.inf
                            RSSI_INV[i][j]  = numpy.inf
                            media_rssi = numpy.inf
                            RSSI[i][j] = numpy.inf
                            rssi = numpy.inf
                            DP[i][j] = numpy.inf
                            dp = numpy.inf
                            PER[i][j] = 1
                #radiuino_RBF.EscreveMatriz(PER) #Teste
                SALTOS = radiuino_RBF.Saltos(PATH,SCusto)
                Pera = radiuino_RBF.PER_A(PATH,PER,SCusto)

#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------
# Arquiva os parâmetros.
                radiuino_RBF.ArquivaPATH(name,PATH,rota,SCusto)

radiuino_RBF.ArquivaMatriz(name,RSSI,MEDIA_RSSI,DP,SALTOS,Pera,SCusto)
                rota = 'RM;'
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------
# Condição de disparo.
                if  time.clock() - t > 60*temporal: #Parâmetros temporal de disparo
                      print("Tempo decorrido = " + str(time.clock() - t))
```

```
                    #temporal += 1
                    t = time.clock()
                    rota = 'RA;'
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------
# Recalcula a rota.
                    for i in range(0,size):
                          S.append(i)
                    SCusto = radiuino_RBF.Shadows(RSSI_INV,S)
                    print"Custo"
                    radiuino_RBF.EscreveMatriz(SCusto)
                    S = []
                    PATH = radiuino_RBF.DefineRota(ser,SCusto)
                    print"PATH"
                    radiuino_RBF.EscreveMatriz(PATH)
                    print "\n"
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------
```

Arquivo: radiuino_RBF.py

```
import serial
import numpy
import math
import time
import struct
import itertools
import time
import random
#########################################################################
#########################################################################
########################################
pkt = {}
#########################################################################
#########################################################################
########################################
def fInicializaSerialCOM(): #Inicializa COM.
#Cria instancia da porta serial.
   ser = serial.Serial(port='COM17',
                    baudrate=9600,
                    parity=serial.PARITY_NONE,
                    timeout=1)
   ser.write(hex(255))
   time.sleep(2)
   ser.read(52)
   return ser
#########################################################################
#########################################################################
########################################
#Fecha COM.
```

```
def fFechaCOM(ser):
    ser = fInicializaSerialCOM()
    ser.close()
    return
#########################################################################
#########################################################################
########################################
# Verifica se a porta esta conectada e retorna qual COM.
def fCOMConectada(ser):
    com = ser.portstr
    return com
#########################################################################
#########################################################################
########################################
def fLeituraDeRSSI(ser,sensor_ini,sensor_fim,tamanho): #Envia pacote de leitura de RSSI, Desvio Padrao e PER.
    for i in range(0,52): #Inicializa o pacote com ZEROS.
        pkt[i] = 0

    pkt[14] = int(random.uniform(1, 255)) #ID unico do Pacote.
    pkt[16] = 40 #Indica que é um pacote de leitura de RSSI entre dois sensores.

    pkt[8] = 255
    if(sensor_ini==0):
        pkt[8] = sensor_fim
    else:
        while i+1<len(tamanho):
            if(tamanho[i]==sensor_ini):
                pkt[8] = tamanho[i+1]
            i=i+2
        if(pkt[8]==255):
            pkt[8] = sensor_ini

        #Rotina desnecessaria, pois já está indicando o sensor inicial
        #else:
        #    pkt[8] = sensor_ini

#Sensores que terao o RSSI verificados.
    pkt[17] = sensor_ini
    pkt[18] = sensor_fim

#Limpa o buffer da serial.
    ser.flushOutput()
    lineteste=""
    TXbyte = ''
    for k in range(0,52): #Transmite o pacote.
        TXbyte = TXbyte+chr(pkt[k])
        lineteste = lineteste + str(pkt[k]) + " "
    ser.write(TXbyte)
    time.sleep(1)
```

```
    line = ser.read(52)
    lineteste2=""
    line_t={}
    for i in range(0,len(line)):
        line_t[i] = ord(line[i])
        lineteste2 = lineteste2 + str(ord(line[i])) + " "
    print("Envia para %s, leitura entre %s e %s"%(pkt[8],sensor_ini,sensor_fim))
    print("Enviado : " + lineteste )
    print("Recebido: " + lineteste2)
    return line_t
##########################################################################
##########################################################################
########################################
def Rota_Sensor(ser,Sensor,Destino,Caminho,LimparTabela):
    for i in range(0,52): #Inicializa todo o pacote com ZEROS.
        pkt[i] = 0
    print("Sensor = %s, Destino = %s, Caminho = %s, Limpar Tabela: %s
\n"%(Sensor,Destino,Caminho,LimparTabela))
    pkt[14] = int(random.uniform(1, 255)) #ID único do pacote.
    pkt[8] = int(Sensor) #Sensor que deve receber a alteração de rota.Mandamos o
pacote para este sensor.
    pkt[16] = 10 #Indica que é um pacote de alteracao de rota.
    pkt[17] = int(Sensor) #Sensor que tera rota alterada.
    pkt[20] = int(Destino) #Sensor destino.
    pkt[21] = int(Caminho) #Sensor caminho.
    pkt[22] = 100 #Fim da Rota
    if(LimparTabela): #Indica se a tabela de rota deve ser apagada completamente.
        pkt[18] = 100
    else:
        pkt[18] = 0
    ser.flushOutput() #Limpa o buffer da serial.
    TXbyte = ''
    for k in range(0,52): #Transmite o pacote.
        TXbyte = TXbyte+chr(pkt[k])
    ser.write(TXbyte)
    time.sleep(1)
    line = ser.read(52)
    line_t={}
    for i in range(0,len(line)):
        line_t[i] = ord(line[i])
    return line_t
##########################################################################
##########################################################################
########################################
def Dijkstra(SCusto,destino):
#-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Listas e variáveis necessárias para o programa.
    rede = len(SCusto) # Numero de rede na rede.
    inf = numpy.inf
```

```
    Custo = numpy.zeros((rede,rede))
    CT = list() # Custo.
    S = list() # Sensores disponíveis.
    aux = list() # lista auxiliar, nela estarão todos os sensores da rede.
    caminho = list() # Contém o destino e o caminho de todos os sensores até ele.
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Inicialização da matriz principal.
    for linha in range(0,rede):
        for coluna in range(0,rede):
            Custo[linha][coluna] = SCusto[linha][coluna+1]
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Inicialização das listas.
    for i in range(0,rede):
        S.append(SCusto[i][0])
        CT.append(numpy.inf)
        aux.append(SCusto[i][0])
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Cria, agora, a matriz principal.
    C = numpy.zeros((rede,4)) #Primeira Coluna: sensor, Segunda Coluna:
Permissão, Terceira Coluna: Custo, Quarta Coluna: Caminho.
    for i in range(0,rede): # Copia a tabela inicial.
        C[i][0] =  S[i]
        C[i][1] = True
        C[i][2] = numpy.inf
        C[i][3] = destino # Inicializa-se com o caminho direto.
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Caso a base não seja o destino.
    if destino != 0: # A base só é levada em conta se ela for o destino.
        for i in range(1,rede):
            Custo[0][i] = inf
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Caso o destino seja uma sombra.
    if destino not in S:
        return 'erroooooooooooou'
#-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
# Programa Principal_RBF.
    C[aux.index(destino)][2] = 0 #Custo do destino ao destino é zero.
    CT[aux.index(destino)] = 0 #Custo do destino ao destino é zero.
    C[aux.index(destino)][1] = False #Permissão negada.
    inter = aux.index(destino)
    proximo = destino

    while len(S) > 0:
        for sensor in range(0,rede):
```

```
            if C[sensor][2] > C[inter][2] + Custo[inter][sensor]:
                C[sensor][2] =  C[inter][2] + Custo[inter][sensor]
                C[sensor][3] = proximo

        C[inter][1] = False
        apaga = S.index(proximo) # Índice referente ao sensor que será apagado.
        S.pop(apaga) # Apaga o sensor usado.
        CT.pop(apaga) # Apaga o custo referente ao sensor usado.

        for i in range(0,len(S)): # Escreve o custo referente aos sensores.
            CT[i] = C[aux.index(S[i])][2]

        if len(CT) > 0: # Define qual é o próximo sensor a ser utilizado na lista
            indice = CT.index(min(CT))
            proximo = S[indice]
            #print ("Sensor intermediário = %s"%proximo)
            inter = aux.index(proximo)
    caminho.append(destino)
    #radiuino.ImprimeMatriz(Custo)
    for i in range(0,rede):
        caminho.append(int(C[i][3]))
    #print "Matriz C"
    #EscreveMatriz(C)
    return caminho
#########################################################################
#########################################################################
########################################
def DefineRota(ser,SCusto):
    size = len(SCusto)
    caminho_sensor = []
    S = list()
    PATH = numpy.zeros((size,size))
    limpa_tabela = True
    for i in range(0,size):
        S.append(SCusto[i][0])
    for x in range(0,size): # Percorre todos os sensores.
        caminho_sensor = Dijkstra(SCusto,S[x])
        if type(caminho_sensor) != str: # Se o sensor destino está funcionando.
            aux = caminho_sensor[0]
            caminho_sensor.pop(0)
            for y in range(0,len(PATH)):
                PATH[S.index(aux)][y] = caminho_sensor[y]
                if (y != x and y != 0):
                    Rota_Sensor(ser,y,int(aux),int(caminho_sensor[y]),limpa_tabela)
        caminho_sensor = []
        limpa_tabela = False
    #print "PATH"
    #EscreveMatriz(PATH)
    #print "\n"
    return PATH
```

```
#########################################################################
#########################################################################
########################################
def Shadows(MatrizCusto,Sensores):
    inf = numpy.inf
    rede = len(MatrizCusto)
    SCusto = numpy.zeros((rede,rede+1))
    sombra = 0
    linha = 0
    coluna = 0
#-------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------
# Programa principal.
    for linha in range(0,rede):
        for coluna in range(0,rede):
            if MatrizCusto[linha][coluna] == inf:
                sombra += 1
        if sombra == rede - 1 and linha != 0:
            MatrizCusto = numpy.delete(MatrizCusto, (linha), axis=0) # Exclui linha
referente ao sensor sombreado.
            MatrizCusto = numpy.delete(MatrizCusto, (linha), axis=1) # Exclui coluna
referente ao sensor sombreado.
            Sensores.pop(linha) # Atualiza a lista de sensores funcionais da rede.
            return Shadows(MatrizCusto,Sensores)
        sombra = 0
#-------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------
# Criação da matriz SCusto.
    for j in range(0,rede):
        for k in range(0,rede):
            SCusto[j][0] = Sensores[j] # Lista de sensores funcionais da rede.
            SCusto[j][k+1] = MatrizCusto[j][k]
    #print "SCusto"
    #EscreveMatriz(SCusto)
    #print "\n"
    return SCusto
#########################################################################
#########################################################################
########################################
def PrimeiraRota(ser,n_sensores):
    SCusto = numpy.zeros((int(n_sensores),int(n_sensores)+1))
    for i in range(0,int(n_sensores)):
        SCusto[i][0] = i
    PATH = DefineRota(ser,SCusto)
    return PATH
#########################################################################
#########################################################################
########################################
def EscreveVetor(vetor):
    for i in range(0,len(vetor)):
```

```
        print vetor[i]
#########################################################################
#########################################################################
########################################
def EscreveMatriz(Matriz):
    Linha = len(Matriz)
    Coluna = len(Matriz[0])
    linha = {}
    for i in range(0,Linha):
        for j in range(0,Coluna):
            linha = str(linha) + " " + str(Matriz[i][j])
        print linha
        linha = {}
#########################################################################
#########################################################################
########################################
def ArquivaMatriz(nome,RSSI,MEDIA,DP,SALTOS,PER,SCusto):
    name = nome + ".txt"
    file = open(name, 'a')
    S = list()
    i = len(SCusto)
    j = len(SCusto)
    for l in range(0,len(SCusto)):
        S.append(SCusto[l][0])
    for a in range(0,len(S)):
        for b in range(0,len(S)):
            if RSSI[a][b] < 1e+45:
                #file.write(str(S[a]) + ";" + str(S[b]) + ";" + str(RSSI[a][b])  + ";" +
str(MEDIA[a][b]) + ";" + str(DP[a][b]) + ";" + str(PER[a][b]) + ";" +
str(int(SALTOS[a][b])) + ",")
                file.write(str(S[a]) + "->" + str(S[b]) + ": " + str(RSSI[a][b])  + " | " +
str(MEDIA[a][b]) + " | " + str(DP[a][b]) + " | " + str(PER[a][b]) + " | " +
str(int(SALTOS[a][b])) + ";")
            else:
                #file.write(str(S[a]) + "," + str(S[b]) + "," + "ERRO;")
                file.write(str(S[a]) + "->" + str(S[b]) + ": RSSI | MEDIA | DP | PER |
SALTOS;")
    file.write("\n")
    file.close()
    fil = open("PROGRAMA.txt", 'a')
    i = len(SCusto)
    j = len(SCusto)
    a = 0
    b = 0
    for a in range(0,i):
        for b in range(0,j):
            if RSSI[a][b] < 1e+45:
                #fil.write(str(S[a]) + ";" + str(S[b]) + ";" + str(RSSI[a][b])  + ";" +
str(MEDIA[a][b]) + ";" + str(DP[a][b]) + ";" + str(PER[a][b]) + ";" +
str(int(SALTOS[a][b])) + "\n")
```

```
            fil.write(str(S[a]) + "->" + str(S[b]) + ": " + str(RSSI[a][b])  + " | " + str(MEDIA[a][b]) + " | " + str(DP[a][b]) + " | " + str(PER[a][b]) + " | " + str(int(SALTOS[a][b])) + "\n")
        else:
            #fil.write(str(S[a]) + ";" + str(S[b]) + ";" + "ERRO\n")
            fil.write(str(S[a]) + "->" + str(S[b]) + ": RSSI | MEDIA | DP | PER | SALTOS\n")
    fil.close()
#########################################################################
#########################################################################
########################################
def ArquivaPATH(nome,PATH,rota,SCusto):
    S = list()
    name = nome + ".txt"
    file = open(name, 'a')
    string = time.strftime("%d/%m/%Y")
    string2 = time.strftime("%H:%M:%S")
    file.write(string +"," + string2 + ";")
    file.write(rota)
    i = len(SCusto)
    j = i
    for l in range(0,i):
        S.append(SCusto[l][0])
    for a in range(0,i):
        for b in range(0,j):
            #file.write(str(S[a]) + ";" + str(S[b]) + ";" + str(int(PATH[b][a])) + ",")
            file.write(str(S[a]) + "->" + str(S[b]) + ":" + str(int(PATH[b][a])) + ";")
    file.close()
    fil = open('Programa.txt', 'a')
    i = len(SCusto)
    j = i
    a = 0
    b = 0
    for a in range(0,i):
        for b in range(0,j):
            #fil.write(str(S[a]) + "," + str(S[b]) + "," + str(int(PATH[b][a])) + "\n")
            fil.write(str(S[a]) + "->" + str(S[b]) + ":" + str(int(PATH[b][a])) + "\n")
    fil.close()
#########################################################################
#########################################################################
########################################
def CalculaRSSI(dados):
    if dados >= 128:
      x=((dados-256)/2.0)-74
    if dados == 0:
      x = 0
    else:
      x=(dados/2.0)-74
    return x
#########################################################################
```

```
#########################################################################
########################################
def Saltos(PATH,SCusto):
    salto = 1
    S = list()
    SALTOS = numpy.zeros((len(PATH),len(PATH)))
    for i in range(0,len(SCusto)):
        S.append(SCusto[i][0])
#-------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------
# Programa principal.
    for a in range(0,len(PATH)):
        for b in range(0,len(PATH)):
            dest = S[a]
            orig = S[b]
            origem = orig
            while PATH[S.index(dest)][S.index(orig)] != orig and orig != dest:
                salto = salto + 1
                orig = PATH[S.index(dest)][S.index(orig)]
                if PATH[S.index(dest)][S.index(orig)] == orig:
                    return SALTOS
            if orig == dest:
                salto = 0
            SALTOS[S.index(dest)][S.index(origem)] = salto
            salto = 1
    return SALTOS
#########################################################################
#########################################################################
########################################
def PER_A(PATH,PER,SCusto):
    Acerto = 1
    n =0
    S = list()
    rede = int(len(PATH))
    for i in range(0,len(SCusto)):
        S.append(SCusto[i][0])
#-------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------
# Programa principal
    PERA = numpy.zeros((rede,rede))
    for a in range(0,rede):
        for b in range(0,rede):
            dest = S[a]
            orig = S[b]
            origem = orig
            if PATH[S.index(dest)][S.index(orig)] == dest:
                PERA[S.index(dest)][S.index(orig)] = PER[S.index(dest)][S.index(orig)]
            else:
                while PATH[S.index(dest)][S.index(orig)] != dest and orig != dest:
                    x = PATH[S.index(dest)][S.index(orig)]
```

```
                right = 1 - PER[S.index(x)][S.index(orig)]
                Acerto = Acerto*right
                orig = x
                n += 1
            if n > 0:
                x = PATH[S.index(dest)][S.index(orig)]
                right = 1 - PER[S.index(x)][S.index(orig)]
                Acerto = Acerto*right
                n = 0
            PERA[S.index(dest)][S.index(origem)] = 1 - Acerto
            Acerto = 1
    print "PER"
    EscreveMatriz(PERA)
    print "\n"
    return PERA
#########################################################################
#########################################################################
########################################
```

Anexo D – Implementação da Rotina de Análise de Dados

Arquivo: analise.sci

```
//*********************************************************************************//
// Análise de Resultados                                                           //
// *********************************************************************************//
// Lucas Augusto de Araujo Marques Leão
// Programa de Mestrado em Engenharia Elétrica
// PUC Campinas
//
// *********************************************************************************//

//=================================================================================
======
// Inicializações
//=================================================================================
======
clearglobal

stacksize('max')
gstacksize('max')

TAMANHO_REDE = 15
EXIBIR_GRAFICO = 0 // 1 - todos os gráficos // 2 - somente o primeiro
EXIBIR_RESULTADO = 1

//=================================================================================
======
// Leitura do Arquivo de Log
//=================================================================================
======
//---- FUZZY ----------------------------------------------------------------
//abre o arquivo de log

//Cenário 1 - Fuzzy
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\Fuzzy - 4 Sensores --
16-12-2014 (8h ~ 13h)\DADOS Fuzzy - 4 Sensores -- 16-12-2014 (8h ~ 13h).txt', 'rt')

//Cenário 2 - Fuzzy
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\Fuzzy - 6 Sensores --
16-12-2014 (13h ~ 17h)\DADOS Fuzzy - 6 Sensores -- 16-12-2014 (13h ~ 17h).txt', 'rt')

//Cenário 3 - Fuzzy
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\Fuzzy - 8 Sensores --
16-12-2014 (17h ~ 21h)\DADOS Fuzzy - 8 Sensores -- 16-12-2014 (17h ~ 21h).txt', 'rt')

//Cenário 4 - Fuzzy
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\Dia 18\Fuzzy - 10
Sensores -- 18-12-2014 (8h ~ 12h)\DADOS Fuzzy - 10 Sensores -- 18-12-2014 (8h ~ 12h).txt', 'rt')

//Cenário 5 - Fuzzy
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\Dia 18\Fuzzy - 12
Sensores -- 18-12-2014 (12h ~ 16h)\DADOS Fuzzy - 12 Sensores -- 18-12-2014 (12h ~ 16h).txt', 'rt')

//Cenário 6 - Fuzzy
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\Dia 18\Fuzzy - 14
Sensores -- 18-12-2014 (16h ~ 20h)\DADOS Fuzzy - 14 Sensores -- 18-12-2014 (16h ~ 20h).txt', 'rt')

//Cenário FIXO - Fuzzy
fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Fuzzy - 14 - fixo.txt', 'rt')

//faz a leitura de uma linhas do arquivo
str_f = mgetl(fd_r)

//fecha o arquivo
mclose(fd_r);
```

```
//---- RBF ----------------------------------------------------------------
//abre o arquivo de log

//Cenário 1 - RBF
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\RBF - 4 Sensores -- 11-12-14 (8h ~ 12h)\DADOS RBF - 4 Sensores -- 11-12-14 (8h ~ 12h).txt', 'rt')

//Cenário 2 - RBF
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\RBF - 6 Sensores -- 11-12-2014 (12h ~ 16h)\DADOS RBF - 6 Sensores -- 11-12-2014 (12h ~ 16h).txt', 'rt')

//Cenário 3 - RBF
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\RBF - 8 Sensores -- 11-12-2014 (16h ~ 20h)\DADOS RBF - 8 Sensores -- 11-12-2014 (16h ~ 20h).txt', 'rt')

//Cenário 4 - RBF
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\Dia 17\RBF - 10 Sensores -- 17-12-2014 (8h ~ 12h)\DADOS RBF - 10 Sensores -- 17-12-2014 (8h ~ 12h).txt', 'rt')

//Cenário 5 - RBF
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\Dia 17\RBF - 12 Sensores -- 17-12-2014 (13h ~ 17h)\DADOS RBF - 12 Sensores -- 17-12-2014 (13h ~ 17h).txt', 'rt')

//Cenário 6 - RBF
//fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\Resultados\Dia 17\RBF - 14 Sensores -- 17-12-2014 (17h ~ 21h)\DADOS RBF - 14 Sensores -- 17-12-2014 (17h ~ 21h).txt', 'rt')

//Cenário FIXO - RBF
fd_r = mopen('C:\Users\leao.lucas\Google Drive\PUC Mestrado\Projeto\Experimento\RBF - 14 - fixo.txt', 'rt')

//faz a leitura de uma linhas do arquivo
str_r = mgetl(fd_r)

//fecha o arquivo
mclose(fd_r);

//=========================================================================
======
// Definição de Funções
//=========================================================================
======
//*************************************************************************//
// Gráfico da RSSF
// Exibe o gráfico que representa a posição para cada sensor da rede
// *************************************************************************//

function exibeGrafico(rssf_temp, janela, titulo)

    cenBase([1],[1]) = rssf_temp(1).x
    cenBase([1],[2]) = rssf_temp(1).y

    tamRede = size(rssf)

    for (i=2:tamRede(1))
        cenSensores([i],[1]) = rssf_temp(i).x
        cenSensores([i],[2]) = rssf_temp(i).y
    end

    scf(janela);
    title(titulo);
    //Limites do gráfico
    MaxX = max(cenSensores([:],[1]))+1
    MinX = min(cenSensores([:],[1]))-1
    MaxY = max(cenSensores([:],[2]))+1
    MinY = min(cenSensores([:],[2]))-1
    plot2d(0,0,1,rect=[MinX,MinY,MaxX,MaxY], frameflag=3)
```

```
    //--Base
    xpoly(cenBase(:,[1]),cenBase(:,[2]),"marks")
    aeBase=gce();
    set(aeBase,"mark_style",1);


    //--Sensores
    for (i=2:tamRede(1))
        //--Exibe o gráfico
        xpoly(rssf(i).x,rssf(i).y,"marks")
        aeSens=gce();
        set(aeSens,"mark_style",2);
    end

endfunction

//*****************************************************************************//
// Separação dos Dados e Agrupamento da Rede
// *****************************************************************************//
function experimento=TratamentoDeDados(str)
    tamanho = size(str)
    experimento = list()
    rssf = list()
    vizinho_aux = list([],[],[],[],[],[],[],[],[],[],[],[],[],[])
    //xy=[0,3;6,6;3,6;0,6;-3,6;-6,6;6,3;3,3;-3,3;-6,3;6,0;3,0;-3,0;-6,0;0,0]
    xy=[0,0;-3,3;0,3;3,3;-3,0;3,0;0;-3,-3;0,-3;3,-3;6,6;0,6;,6,0;3,6;6,3;-6,3]

    //--Cria a variável estruturada para os Sensores
    a=1
    for (i=1:TAMANHO_REDE)
        rssf(i) = struct('id',(i-1),'tipo',a,'x',xy(i,1),'y',xy(i,2),'vizinho',vizinho_aux)
        a=0
    end

    for(ind=1:tamanho(1))

        log_info = strsplit(str(ind),";")
        i = 3
        aux = -1
        count = 0
        while(log_info(i)<>"")
            if(strindex(log_info(i),'|')==[])
                temp = strsplit(log_info(i),["->",":"])
                s_fonte = strtod(temp(1))
                s_destino = strtod(temp(2))
                s_caminho = strtod(temp(3))
                if(s_fonte<>s_destino)
                    rssf(s_fonte+1).vizinho(s_destino+1)          =          struct('destino',s_destino,'rssi',-1,'media',-1,'dp',-1,'per',-
1,'caminho',s_caminho)
                end
            else
                temp = strsplit(log_info(i),["->","-","|"])
                s_fonte = strtod(temp(1))
                s_vizinho = strtod(temp(2))
                if(s_fonte<>s_vizinho)
                    if(isnum(temp(3)))
                        rssi = strtod(temp(3))
                        media = strtod(temp(4))
                        dp = strtod(temp(5))
                        per = strtod(temp(6))
                        if(rssi<>per)
                            rssf(s_fonte+1).vizinho(s_vizinho+1).rssi = rssi
                            rssf(s_fonte+1).vizinho(s_vizinho+1).media = media
                            rssf(s_fonte+1).vizinho(s_vizinho+1).dp = dp
                            rssf(s_fonte+1).vizinho(s_vizinho+1).per = per
                        end
                    else
```

```
                rssf(s_fonte+1).vizinho(s_vizinho+1).rssi = -1000
                rssf(s_fonte+1).vizinho(s_vizinho+1).media = -1000
                rssf(s_fonte+1).vizinho(s_vizinho+1).dp = 1000
                rssf(s_fonte+1).vizinho(s_vizinho+1).per = 1
            end
        end
    end
    i = i+1
end

//--Cálculo do Sucesso acumulado e Saltos ----------------------------------
    prob = []
    saltos = []

    tam=size(rssf)
    k=tam(1)
    prob(1) = 1
    saltos(1) = 0
    void=0
    for (i=2:k)
        j=i
        if(isempty(rssf(i).vizinho))
            prob(i) = 0
            saltos(i) = 1
            void=void+1
        else
            prob(i) = 1 - rssf(j).vizinho(rssf(j).vizinho(1).caminho+1).per
            saltos(i) = 1
            j=rssf(j).vizinho(1).caminho+1
            while(1)
                if (j==1) then
                    break
                else
                    prob(i) = prob(i) * (1 - rssf(j).vizinho(rssf(j).vizinho(1).caminho+1).per)
                    j=rssf(j).vizinho(1).caminho+1
                    saltos(i) = saltos(i) + 1
                end
            end
        end
    end

//--Acumula resultados obtidos ---------------------------------------------
    experimento($+1)                                                    =                                   struct("iteracao',
ind,'data',log_info(1),'resultado',rssf,'rota',log_info(2),'taxa_entrega',prob,'saltos',saltos,'buracos',void)
    //--Limpa a variável estruturada para os Sensores
    for (i=1:TAMANHO_REDE)
        rssf(i).vizinho=vizinho_aux
    end
  end
endfunction

//*****************************************************************************//
// Representação gráfica da comunicação dos sensores
// *****************************************************************************//
function exibeResultadoGrafico(rssf, janela, texto)
  exibeGrafico(rssf,janela, texto)
  tam=size(rssf)
  k=tam(1)
  while(k>1)
    if rssf(k).id>0  then
      if(~isempty(rssf(k).vizinho))
        x=[rssf(k).x rssf(rssf(k).vizinho(1).caminho+1).x]
        y=[rssf(k).y rssf(rssf(k).vizinho(1).caminho+1).y]
      end
      xset("color",5)
      xpoly(x,y,"lines",1)
      p=get("hdl");
```

```
            p.polyline_style=4;
            if (rssf(k).id==0) then
               str = "Base Station"
            else
               str = "Sensor " + string(rssf(k).id)
            end
            xstring(rssf(k).x,rssf(k).y,str,0,0)
         end
         k=k-1
      end
endfunction

//=========================================================================
======
// Programa
//=========================================================================
======

experimento_f = TratamentoDeDados(str_f)
experimento_r = TratamentoDeDados(str_r)

if(EXIBIR_GRAFICO>0)
   count_g=1
   tam = size(experimento_f)
   for(i=1:tam(1))
      if(experimento_f(i).rota=="RA")
         rssf = experimento_f(i).resultado
         exibeResultadoGrafico(rssf,count_g,'Resultado Experimento - FLBRA')
         count_g=count_g+1
         if(EXIBIR_GRAFICO==2 & experimento_f(i).rota=="RA")
            break;
         end
      end
   end

   tam = size(experimento_r)
   for(j=1:tam(1))
      if(experimento_r(i).rota=="RA")
         rssf = experimento_r(i).resultado
         exibeResultadoGrafico(rssf,count_g,'Resultado Experimento - RBF')
         count_g=count_g+1
         if(EXIBIR_GRAFICO==2 & experimento_r(i).rota=="RA")
            break;
         end
      end
   end
end

if(EXIBIR_RESULTADO)
   dif=[]
   dif_m=[]
   printf("Resultados FUZZY\n")
   tam_f = size(experimento_f)
   for(i=1:tam_f(1))

printf("%d;%s;%s;%s;%d\n",i,experimento_f(i).rota,strcat(string(experimento_f(i).taxa_entrega'),','),strcat(string(experim
ento_f(i).saltos'),','),experimento_f(i).buracos)
   end

   printf("\n\nResultados RBF\n")
   tam_r = size(experimento_r)
   for(i=1:tam_r(1))

printf("%d;%s;%s;%s;%d\n",i,experimento_r(i).rota,strcat(string(experimento_r(i).taxa_entrega'),','),strcat(string(experim
ento_r(i).saltos'),','),experimento_r(i).buracos)
   end
```

```
    printf("\n\nResultados para o F\n")
    printf("\nF       | Média Saltos Fuzzy | Média Saltos RBF\n")
    for(i=1:min(tam_f,tam_r))
        for(j=1:TAMANHO_REDE)
            dif_te(j,i) = experimento_f(i).taxa_entrega(j)-experimento_r(i).taxa_entrega(j)
            dif_s_f(j,i) = experimento_f(i).saltos(j)
            dif_s_r(j,i) = experimento_r(i).saltos(j)
        end
        max_s_f(i) = max(experimento_f(i).saltos)
        max_s_r(i) = max(experimento_r(i).saltos)
        temp_f = sum(dif_s_f(:,i))/nnz(dif_s_f(:,i))
        temp_r = sum(dif_s_r(:,i))/nnz(dif_s_r(:,i))
        printf("% .3f | % .3f           | % .3f\n",mean(dif_te(:,i)), temp_f, temp_r)
    end

    temp_f = sum(dif_s_f)/nnz(dif_s_f)
    temp_r = sum(dif_s_r)/nnz(dif_s_r)
    printf("\n\nFator F=%f\nMédia de Saltos Fuzzy=%f\nMédia de Saltos RBF=%f\n",mean(dif_te),temp_f,temp_r)
    printf("\nSaltos do sensor mais distante:\nFuzzy: %d\nRBF: %d ",max(max_s_f),max(max_s_r))
end
```