

ROSANA LACHOWSKI

**Algoritmos Distribuídos para Construção de
Spanning Trees em Redes de Sensores sem Fio**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

CURITIBA

2014

ROSANA LACHOWSKI

**Algoritmos Distribuídos para Construção de
Spanning Trees em Redes de Sensores sem Fio**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

Área de Concentração: Ciência da Computação

Orientador: Prof. Dr. Marcelo Eduardo Pellenz
Co-orientador: Prof. Dr. Manoel Camillo

CURITIBA

2014

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

L138a
2014 Lachowski, Rosana
Algoritmos distribuídos para construção de spanning trees em redes de sensores sem fio / Rosana Lachowski ; orientador, Marcelo Eduardo Pellenz ; co-orientador, Manoel Camillo. – 2014.
xvi, 51 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2014
Bibliografia: f. 48-51

1. Informática. 2. Algoritmos. 3. Sistemas operacionais distribuídos (Computadores). 4. Redes de sensores sem fio. I. Pellenz, Marcelo Eduardo. II. Camillo, Manoel. III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada. IV. Título.

CDD 20. ed. – 004

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 01/2014

Aos 30 dias do mês de Abril de 2014 realizou-se a sessão pública de Defesa da Dissertação “**Algoritmos Distribuídos para Construção de Spaming Trees em Redes de Sensores Sem Fio**” apresentado pela aluna **Rosana Lachowski**, como requisito parcial para a obtenção do título de Mestre em Informática, perante uma Banca Examinadora composta pelos seguintes membros:

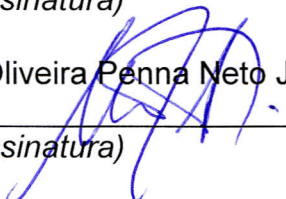
Prof. Dr. Marcelo Eduardo Pellenz
PUCPR (Orientador)



(assinatura)

APROVADO
(Aprov/Reprov)

Prof. Dr. Manoel Camillo de Oliveira Penna Neto Jr
PUCPR (co-orientador)



(assinatura)

APROVADO
(Aprov/Reprov)

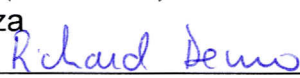
Prof. Dr. Edgard Jamhour
PUCPR



(assinatura)

APROVADO
(Aprov/Reprov)

Prof. Dr. Richard Demo Souza
UTFPR



(assinatura)

APROVADO
(Aprov/Reprov)

Conforme as normas regimentais do PPGIa e da PUCPR, o trabalho apresentado foi considerado APROVADO (aprovado/reprovado), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora registradas no Livro de Defesas do programa.


Prof. Dr. Mauro Serio Pereira Fonseca
Diretor do Programa de Pós-Graduação em Informática



A Deus, fonte de toda sabedoria.

Agradecimentos

Às minhas filhas Ana Vitória e Ana Thereza pela compreensão nos momentos de ausência.

Aos meus pais Romoaldo e Teresa e a minha querida tia Elza por sempre permaneceram ao meu lado.

Aos meus orientadores Prof. Dr. Marcelo Eduardo Pellenz e Prof. Dr. Manoel Camillo por compartilharem seus conhecimentos e pelo exemplo de dedicação.

Ao Prof. Dr. Edgard Jamhour pelas sugestões, por disponibilizar a biblioteca Sensor-Sim e por despendar seu tempo explicando o funcionamento da mesma.

Aos colegas de curso pelas dicas e pela amizade.

A todos os funcionários da PUC/PR que sempre foram muito prestativos e educados.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela concessão da bolsa de estudos.

A todas as pessoas que diretamente ou indiretamente tornaram este trabalho possível.

Sumário

Lista de Figuras	xii
Lista de Tabelas	xiii
Lista de Símbolos	xiv
Lista de Abreviações	xvi
Resumo	xvii
Abstract	xviii
1 Introdução	1
1.1 Motivação	2
1.2 Objetivos	3
1.3 Estrutura do Documento	3
2 Fundamentos Teóricos	5
2.1 Características das Redes de Sensores Sem Fio	5
2.2 Arquitetura do Nó Sensor	6
2.2.1 Estados operacionais dos nós sensores	8
2.3 Controle de Topologia	9
2.3.1 Ajuste de potência	11
2.3.2 Modo de potência	11
2.3.3 <i>Clustering</i>	12
2.4 Tempo de Vida da Rede	12
2.5 <i>Convergecast</i> e Árvores de Roteamento	13
2.6 Considerações Finais	15
3 Trabalhos Relacionados	17
3.1 Algoritmos Distribuídos para Construção de <i>Spanning Trees</i>	17

3.1.1	Algoritmo Bellman-Ford	18
3.1.2	Algoritmo Shortest Hop Multipath	20
3.1.3	Algoritmo Depth-First Search	24
3.1.4	Comparações entre Algoritmos Distribuídos para Construção de <i>Spanning Trees</i>	28
3.2	Considerações Finais	30
4	Algoritmo Proposto	31
4.1	Descrição do Algoritmo Proposto	32
4.2	Considerações Finais	34
5	Resultados	37
5.1	Modelagem da Rede e Ambiente de Simulação	37
5.1.1	Procedimentos para Geração das Topologias	38
5.1.2	Métrica de Custo do Algoritmo Proposto	39
5.1.3	Métricas para Avaliação dos Algoritmos	40
5.2	Primeira Etapa - Determinação Empírica de Alfa	41
5.3	Segunda Etapa - Avaliação do Algoritmo Proposto	44
5.4	Considerações Finais	46
6	Conclusão	49
6.1	Trabalhos Futuros	50

Lista de Figuras

2.1	<i>Mica mote</i> implantado no ambiente [Polastre et al., 2013]	7
2.2	Arquitetura Típica de uma RSSF	8
2.3	Energia despendida em diferentes modos operacionais [Anastasi et al., 2004]	9
2.4	Possíveis topologias de uma RSSF (adaptado de [Blough et al., 2006])	10
2.5	Perspectiva da eficiência energética	11
2.6	RSSF antes da construção da estrutura de roteamento	14
2.7	RSSF após a construção da estrutura de roteamento	14
3.1	(a) Grafo orientado ponderado com pesos de caminhos mais curtos desde a origem s . (b) As arestas sombreadas formam uma árvore de caminhos mais curtos com raiz na origem s . (c) Outra árvore de caminhos mais curtos com a mesma raiz (adaptado de [Cormen, 2001]).	18
3.2	(a) Mensagem <i>probe</i> (início do SHM). (b) Mensagem <i>ack</i> . (c) Mensagem <i>pulse</i> e mensagens <i>probe</i> [Yilmaz et al., 2012].	22
3.3	(a) Mensagens <i>pulseAck</i> e <i>pulseNack</i> . (b) Mensagens <i>pulseNack</i> . (c) SPT e rotas alternativas [Yilmaz et al., 2012].	22
3.4	Número Médio de Mensagens por Nó	29
3.5	Tempo de Execução	29
4.1	Funcionamento do algoritmo MBF em uma rede com 5 nós e $\alpha = 0.2$	35
5.1	Topologia de Rede em Grade sem Distúrbio Aleatório na Posição dos Nós e $N = 9$	38
5.2	Topologia de Rede em Grade com Distúrbio Aleatório na Posição dos Nós e $N = 9$	38
5.3	Número Médio de Mensagens por Nó	41
5.4	Tempo de Execução	42
5.5	Distância Média até o <i>sink</i>	42
5.6	Número Médio de Saltos até o Sink	43
5.7	Número de Pais Alternativos	43
5.8	Número Médio de Mensagens por Nó	44
5.9	Tempo de Execução	45

5.10 Distância Média até o *sink* 45

5.11 Número Médio de Saltos até o Sink 46

Lista de Tabelas

2.1	Métricas utilizadas para o roteamento em RSSF [Bechkit et al., 2012]	15
3.1	Principais dificuldades para a utilização dos algoritmos DBF, SHM e DFS	30
5.1	Valor Aproximado do Parâmetro d (metros) dados N e k	39
5.2	Parâmetros das Simulações	41

Lista de Símbolos

N_i	Conjunto de nós vizinhos de i , isto é, os nós conectados a i através de um enlace
W_i^t	Peso do nó i na etapa t , ou seja, a última estimativa do custo do caminho de i até o nó raiz computada no nó i
$cost_{i,j}$	Peso atribuído ao enlace entre os nós i e j
$parent_i$	Nó pai do nó i
$level_i$	Nível do nó i
$rcdAck_i$	Contador que contabiliza o número de mensagens ack recebidas pelo nó i
NGB_i	Conjunto de nós vizinhos de i , isto é, os nós conectados a i através de um enlace
$rcdPulseAck_i$	Contador que contabiliza o número de mensagens PulseAck recebidas pelo nó i
$rcdPulseNack_i$	Contador que contabiliza o número de mensagens PulseNack recebidas pelo nó i
AP_i	Conjunto de pais alternativos do nó i
CHD_i	Conjunto de nós filhos do nó i
$PSBL_CHD_i$	Conjunto de possíveis nós filhos de i
$probe_i$	Variável que assume o valor <i>true</i> uma vez que i tenha enviado uma mensagem probe
UNVISITED	Conjunto de nós ainda não visitados
VISITED	Conjunto de nós que já foram visitados
REJECTERS	Conjunto de nós que rejeitaram ofertas de paternidade
$hops_i$	Número de saltos entre o nó i e o nó <i>sink</i>
ROUTE	Conjunto ordenado (fila) de nós que fazem parte da rota até o <i>sink</i> . Embora o conceito de conjunto ordenado não exista na teoria de conjuntos, utilizamos o mesmo para simplificar o pseudocódigo do algoritmo

$position(ROUTE, u)$	Operação que devolve a posição do nó u no conjunto ordenado Route
$REJECTERS_i$	Conjunto de nós que fatalmente rejeitariam uma oferta de paternidade feita pelo nó i
$ROUTE_i$	Conjunto de nós que fazem parte da rota do nó i até o <i>sink</i>
$adv_{i,j}$	Vantagem da rota ofertada por j com relação ao custo da rota atual do nó i
G	Grafo que representa a RSSF
V	Conjunto de nós sensores
A	Conjunto de enlaces
$dist_{i,j}$	Distância entre os nós i e j
r	Raio de transmissão
N	Número de nós
d	Espaçamento de um determinado nó i para seu vizinho j , na horizontal e na vertical
k	Grau médio de conectividade alvo
P_r	Potência recebida em dBm
P_t	Potência de transmissão em dBm
G_t	Ganho da antena para transmissão em dB
G_r	Ganho da antena para recepção em dB
$PL_{d_{i,j}}$	Perda média do sinal ao percorrer o trajeto $d_{i,j}$ em dB
PL_{d_0}	Perda média do sinal ao percorrer o trajeto até uma distância de referência d_0 em dB
η	Expoente de perda de percurso
χ	Variável aleatória chamada de sombreamento
α	Fator que determina o quanto a oferta recebida deve ser mais vantajosa que a anterior para ser aceita

Lista de Abreviações

BFS	<i>Breadth-First Search</i>
CSMA/CA	<i>Carrier Sense Multiple Access/Collision Avoidance</i>
DBF	<i>Distributed Bellman-Ford</i>
DFS	<i>Depth-First Search</i>
MAC	<i>Medium Access Control</i>
MBF	<i>Modified Bellman-Ford</i>
QoS	<i>Quality of Service</i>
RSSI	<i>Received Strength Signal Indicator</i>
RSSF	Redes de Sensores sem Fio
SHM	<i>Shortest Hop Multipath</i>
SPT	<i>Shortest Path Tree</i>
TDMA	<i>Time Division Multiple Access</i>

Resumo

A principal atividade das Redes de Sensores sem Fio (RSSF) é o monitoramento e a coleta de dados. Os dados coletados pelos nós sensores durante as atividades de monitoramento são geralmente transmitidos através de múltiplos saltos até um nó especial chamado nó *sink*. Esta operação é denominada *convergecast*. Enquanto em uma RSSF os nós sensores precisam comunicar-se somente com o nó *sink*, em uma rede Ad Hoc típica os nós precisam comunicar-se uns com os outros. Por este motivo, protocolos de roteamento para redes Ad Hoc são inadequados para RSSF. Por outro lado, árvores são estruturas clássicas de roteamento explicitamente ou implicitamente utilizadas nas RSSF. Neste trabalho, implementamos e avaliamos o desempenho de algoritmos distribuídos para construção de árvores de roteamento em RSSF descritos na literatura. Após identificarmos as dificuldades e vantagens da utilização destes algoritmos em cenários reais, propomos um novo algoritmo para construção de *spanning trees* em RSSF. O desempenho do algoritmo proposto e a qualidade da árvore construída foram avaliados através de simulações em diferentes cenários de rede. Os dados obtidos mostram que o algoritmo proposto é uma solução mais eficiente e adequada para utilização em cenários reais. Além disso, o algoritmo provê múltiplas rotas aos nós sensores, para serem utilizadas como mecanismos de tolerância a falhas e balanceamento de carga.

Palavras-chave: Redes de Sensores Sem Fio, Roteamento, Algoritmos Distribuídos para Construção de Spanning Trees.

Abstract

Monitoring and data collection is one of the main functions in Wireless Sensor Networks (WSN). Collected data during the monitoring activities of sensors nodes are transmitted through multihop communication to a special node called sink. This operation is named convergecast. While in a WSN sensor nodes need to communicate only with the sink, in a typical Ad Hoc network nodes need to communicate with each other. For this reason, routing protocols for Ad hoc are unsuitable for WSN. On the other hand, trees are classic routing structures explicitly or implicitly used in WSN. In this work, we implemented and evaluated through simulations the performance of distributed algorithms for constructing routing trees in WSN described in the literature. After identifying the problems and advantages of using these algorithms in real scenarios, we propose a new algorithm for constructing spanning trees in WSN. The performance of the proposed algorithm and the quality of the constructed tree were evaluated through simulations in different network scenarios. The data obtained show that the proposed algorithm is a more efficient and suitable solution for use in real-world scenarios. Furthermore, the algorithm provides multiple routes to the sensor nodes to be used as mechanisms for fault tolerance and load balancing.

Keywords: Wireless Sensor Networks, Routing, Distributed Spanning Tree Algorithms.

Capítulo 1

Introdução

Avanços na indústria de micro eletrônicos e na tecnologia *wireless* (sem fio) propiciaram o desenvolvimento de nós sensores: dispositivos de pequenas dimensões, baixo custo, recursos computacionais restritos e que comunicam-se a curtas distâncias. Quando esses nós são dispostos em rede em um modo ad hoc, formam as redes de sensores.

As Redes de Sensores sem Fio (RSSF) possuem um grande e crescente número de aplicações, sendo que a principal delas é o monitoramento e a coleta de dados [Akyildiz et al., 2002, Aziz et al., 2013, Dargie, 2012, Zibakalam, 2012]. Inicialmente, as RSSF surgiram para auxiliar em operações militares: monitoramento de tropas, equipamentos, munição e vigilância. Atualmente, as atividades de monitoramento são empregadas em várias outras áreas: *Smart Grids* (redes inteligentes de transmissão e distribuição de energia), *Smart Cities*, ambientes inteligentes, áreas de risco e prevenção de desastres naturais, regiões que ofereçam perigos aos seres humanos (vulcões, regiões de instabilidade sísmica, regiões suscetíveis a ocorrência de furacões etc), vazamento de óleo e gás, danos estruturais, etc.

Os dados coletados pelos nós sensores durante as atividades de monitoramento são geralmente transmitidos através de múltiplos saltos até um nó especial chamado nó *sink*. O nó *sink* provê a interface com o mundo exterior e é responsável por entregar os dados coletados para o usuário. Esta operação é usualmente denominada denominada *convergecast* [Incel et al., 2012].

Operações de *convergecast* utilizam um modelo de comunicação muitos-para-um, enquanto redes ad hoc típicas utilizam um modelo muitos-para-muitos. Por este motivo, protocolos de roteamento para redes ad hoc são inadequados para RSSF. Por outro lado, árvores são estruturas clássicas de roteamento explicitamente ou implicitamente utilizadas nas RSSF por vários pesquisadores [Bechkit et al., 2012]. Nestes casos, a árvore é enraizada no nó *sink* e os demais nós sensores formam os galhos e folhas.

1.1 Motivação

Os dois métodos mais comumente utilizados para construção de *spanning trees* são: busca em profundidade e busca em largura. A busca em profundidade constrói árvores com poucos galhos e folhas parecendo-se mais com um caminho. Devido à profundidade da árvore, os nós consomem uma grande quantidade de tempo e energia para enviar os dados até o *sink*. Algumas aplicações de segurança e de missão crítica, como por exemplo detecção de óleo e gás, necessitam que os dados sejam entregues ao usuário dentro de um tempo limite. Estas aplicações também requerem uma rápida reconstrução da árvore caso nós falhem ou morram.

A busca em largura constrói árvores com muitos galhos e pequena profundidade. É a abordagem utilizada para construção de SPTs (*Shortest Path Trees*) [árvores de caminhos mais curtos]. SPTs são assim chamadas porque o custo do caminho de qualquer nó até a raiz é mínimo. Em se tratando de redes, as métricas mais comumente empregadas para computar o custo do caminho dos nós até a raiz são distância e número de saltos.

Os algoritmos mais utilizados para construção de SPTs são o algoritmo de *Dijkstra* [Cormen, 2001] e o algoritmo *Bellman-Ford* [Lynch, 1996]. O algoritmo *Bellman-Ford* possui uma versão assíncrona distribuída e por este motivo é mais utilizado em RSSF. A abordagem centralizada deve ser evitada devido ao inaceitável *overhead* de comunicação envolvido na descoberta da topologia da rede, pois os nós sensores possuem recursos computacionais escassos. Além disso, RSSF não possuem uma administração centralizada e podem exigir reconfiguração frequente.

O algoritmo *Bellman-Ford* é simples, não demanda muitos recursos computacionais e não exige dos nós o conhecimento da topologia da rede. No entanto, é inviável para RSSF. Especialmente em uma rede constituída por um grande número de nós, o número de mensagens necessárias para construção da árvore é excessivamente alto. Isto é um fator importante porque a maior parte da energia dos nós sensores é consumida pela comunicação de dados [Akyildiz et al., 2002, Santi, 2005, Holger and Willig., 2005, Dargie, 2012, Yilmaz et al., 2012]. Em algumas aplicações, por exemplo quando os nós são implantados em regiões inóspitas, é impossível recarregar ou trocar a bateria dos nós sensores. Nestes casos deve-se tentar maximizar o tempo de vida dos nós.

Em [Yilmaz et al., 2012] os autores propõem um algoritmo denominado *Shortest Hop Multipath* (SHM), cujo objetivo é reduzir o *overhead* de comunicação do algoritmo *Bellman-Ford* e construir múltiplos caminhos entre os nós para serem usados como mecanismos de tolerância a falhas e balanceamento de carga. No entanto, o algoritmo exige que os nós conheçam todos os outros nós localizados a um salto de distância. Além disso, o SHM não constrói a árvore de roteamento caso ocorram perdas de mensagens, um evento muito comum em redes sem fio.

1.2 Objetivos

O principal objetivo desta dissertação de mestrado é a concepção e implementação de um algoritmo para construção de árvores de roteamento apropriado para RSSF e baseado na versão assíncrona distribuída do algoritmo Bellman-Ford. O algoritmo Bellman-Ford possui qualidades desejáveis para as RSSF: simplicidade, tolerância a perda de mensagens e flexibilidade quanto à métrica utilizada para construção da árvore. A intenção é reduzir o *overhead* do algoritmo Bellman-Ford, tornando-o uma solução mais escalável, sem degradar a qualidade da árvore construída. Assim como o algoritmo SHM, a versão modificada do Bellman-Ford deve ser resiliente e construir múltiplos caminhos entre os nós sem, no entanto, exigir o conhecimento da topologia da rede.

O objetivo secundário é a implementação e avaliação de desempenho de algoritmos distribuídos para construção de árvores de roteamento em RSSF propostos na literatura a fim de identificar as vantagens e dificuldades na utilização dos referidos algoritmos.

1.3 Estrutura do Documento

O restante deste documento está estruturado da seguinte forma: O Capítulo 2 apresenta os conceitos fundamentais relacionados às RSSF. O Capítulo 3 discute os trabalhos relacionados com a pesquisa. O Capítulo 4 apresenta e descreve o funcionamento do algoritmo proposto. O Capítulo 5 descreve as simulações realizadas para convalidar o algoritmo proposto e exhibe os resultados obtidos. Finalmente, o Capítulo 6 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Fundamentos Teóricos

2.1 Características das Redes de Sensores Sem Fio

As Redes de Sensores sem Fio são uma classe especial de redes ad hoc constituídas por nós que interagem com o ambiente e trabalham de forma colaborativa para desempenhar uma tarefa específica. Os nós sensores são densamente implantados e possuem capacidade limitada de processamento e memória. São propensos a falhas e podem exaurir suas reservas energéticas. Além disso, devem possuir baixo custo de produção, serem autônomos e operar sem assistência.

As RSSF diferem de outras redes de comunicações de dados em vários aspectos. Uma rede de sensores não é simplesmente uma infraestrutura para o transporte de dados. Os nós sensores são cientes dos dados que transportam e podem processar ou agregar os dados uns dos outros. As principais características das RSSF são [Holger and Willig., 2005]:

1. Diferentes requisitos de Quality of Service (QoS) – Requisitos de QoS como atraso e perda de pacotes podem ser irrelevantes para algumas das aplicações das RSSFs. Por outro lado, a quantidade e a qualidade das informações fornecidas são parâmetros importantes, assim como a detecção confiável de eventos;
2. Tolerância a falhas – RSSFs devem ser capazes de tolerar as falhas e o esgotamento de energia de seus nós e quebra de enlaces. Geralmente, os nós sensores são implantados em número maior do que o estritamente necessário;
3. Escalabilidade – Como uma RSSF pode ser formada por uma grande quantidade de nós, as arquiteturas e protocolos empregados devem ser escaláveis;
4. Densidade – Densidade diz respeito ao número de nós por unidade de área. A densidade da RSSF irá depender de sua aplicação e pode variar ao longo do tempo e do espaço pela falha dos nós, dos enlaces, mobilidade etc. Redes não homogêneas também podem ser re-

sultado da implantação dos nós, já que os mesmos podem ser simplesmente arremessados na área a ser monitorada. A RSSF deve ser capaz de adaptar-se a essas variações;

5. Topologia dinâmica – A topologia de uma RSSF pode ser alterada rapidamente devido às mudanças no ambiente e às próprias características dos nós sensores: suscetibilidade a falhas, mobilidade e reserva limitada de energia;
6. Autoconfiguração – Uma RSSF deve ser capaz de autoconfigurar-se em uma rede totalmente conectada. Assim como deve ser capaz de tolerar falhas, deve também ser capaz de autoconfigurar-se após a implantação incremental de novos nós como consequência de uma falha ou a fim de prolongar o tempo de vida da rede;
7. Comunicação *wireless* (sem fio) através de múltiplos saltos – Os nós sensores comunicam-se a curtas distâncias. Além disso, a comunicação direta entre dois nós, especialmente a longas distâncias, pode ser proibitiva em termos de energia despendida;
8. Eficiência energética – Um dos principais desafios da RSSF é o baixo consumo energético. Portanto, todos os aspectos da RSSF devem ser energeticamente eficientes, incluindo algoritmos, protocolos de comunicação e a própria arquitetura do nó sensor;
9. Trabalho colaborativo – Em certas circunstâncias, um nó sensor isolado não é capaz de detectar um evento. Ao invés disso, é necessário que vários nós trabalhem de forma colaborativa e somente os dados conjuntos são capazes de fornecer informações suficientes;
10. Interação com o ambiente – Devido à interação com o ambiente, as características do tráfego das RSSFs são muito peculiares. Dependendo da aplicação, longos períodos de inatividade podem alternar com curtos períodos de atividade muito intensa quando algo acontece, um fenômeno conhecido como tempestade de alarmes;
11. Simplicidade e escassez de recursos computacionais – Os nós sensores são dispositivos simples, cujo suplemento de energia é escasso e os recursos computacionais restritos, logo, softwares operacionais e de rede devem ser igualmente simples;
12. Centrada em dados – A identidade do nó sensor que provê os dados de sensoriamento é irrelevante. Ao contrário, a importância é dada à própria informação fornecida que pode ser o resultado do sensoriamento de vários nós.

2.2 Arquitetura do Nó Sensor

Um nó sensor básico compreende cinco componentes principais [Akyildiz et al., 2002, Holger and Willig., 2005]:

1. Controlador ou unidade de processamento – Processa dados relevantes, capaz de executar códigos;
2. Memória ou unidade de armazenagem – Armazena programas e dados intermediários. Diferentes tipos de memória são utilizados para programas e dados;
3. Sensores ou unidade de sensoriamento – Dispositivos que podem observar ou controlar parâmetros físicos do ambiente;
4. Transceptor – Dispositivo de comunicação utilizado para a troca de dados entre os nós, geralmente através de frequências de rádio. Possui essencialmente duas tarefas: transmitir e receber dados;
5. Fonte de alimentação – Geralmente é uma bateria que prove a energia necessária ao funcionamento do nó sensor. No entanto, existem outras formas de obtenção de energia através do ambiente como, por exemplo, através da utilização de células solares.

A Figura 2.1 exhibe, como exemplo, o nó sensor *Mica mote* desenvolvido na Universidade da Califórnia em Berkeley.

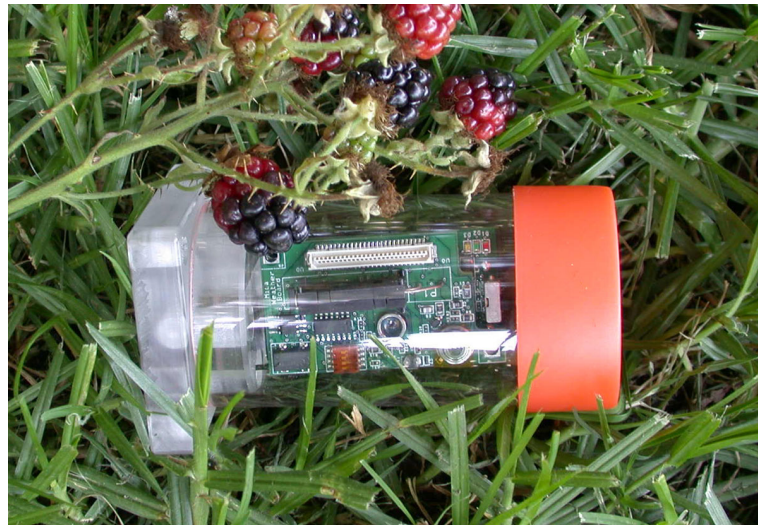


Figura 2.1: *Mica mote* implantado no ambiente [Polastre et al., 2013]

Nós especiais denominados nós *sink* são responsáveis pela coleta de dados. Os dados coletados pelos nós sensores são encaminhados para o nó *sink* que provê a interface para o mundo exterior. O nó *sink* possui rádios de longo alcance, capacidade de processamento e armazenamento superiores aos dos nós sensores comuns e não possui, em geral, restrições quanto ao consumo de energia. A Figura 2.2 exhibe a arquitetura típica de uma RSSF.

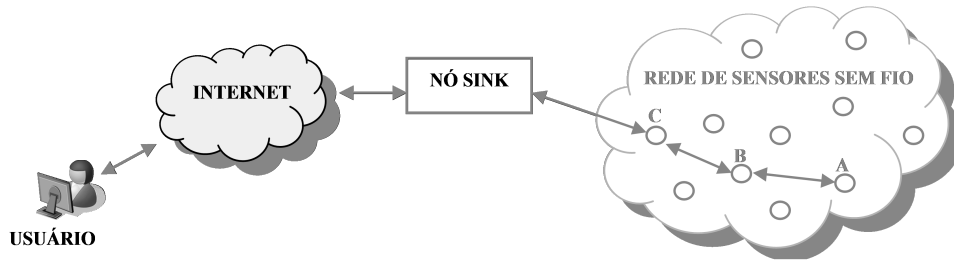


Figura 2.2: Arquitetura Típica de uma RSSF

2.2.1 Estados operacionais dos nós sensores

Os componentes de hardware que constituem um nó sensor possuem diferentes modos de operação e são capazes de transitar entre esses modos. A ideia básica é desativar os componentes quando não são necessários e reativá-los quando houver necessidade [Sinha and Chandrakasan, 2001]. Um exemplo simples, porém não relacionado às RSSFs, é a desativação de alguns dos componentes de *laptops* e *palmtops* após certo período de ociosidade.

Cada modo de operação também é chamado modo de potência. Para o controlador, modos típicos são: *active*, *idle* e *sleep*. O transceptor geralmente pode transitar pelos modos: transmitir, receber, *idle* e *sleep*. Memória e sensores podem ser ligados e desligados. O modo *idle* representa um modo intermediário, no qual o componente não está totalmente ativo, porém não foi desativado e pode retornar à completa atividade devido a um estímulo externo ou decorrido determinado espaço de tempo. Cada nó sensor possui diferentes estados de dormência que correspondem às combinações dos modos de operação de seus componentes. Entretanto, nem todas as combinações possíveis podem ser utilizadas. Por exemplo, não faz sentido a memória estar ativa e todos os outros componentes desativados [Lin et al., 2006]. Quanto mais profundo o estado de dormência do nó sensor, menos energia é consumida, entretanto maior é o atraso e a energia requerida para acordar [Lin et al., 2006]. A Figura 2.3 mostra a energia despendida em cada um dos cinco modos operacionais dos nós sensores Mica2dot e Mica2.

Dentre as tarefas desempenhadas pelo nó sensor, a comunicação de dados é a mais dispendiosa em termos de consumo energético e existem dois motivos principais para isto [Dargie, 2012]: a escuta inútil e *overhearing*. A escuta inútil ocorre quando um nó não tem conhecimento sobre quando irão chegar pacotes endereçados a ele e por esse motivo permanece ociosamente pronto para recebê-los. *Overhearing* ocorre quando o nó recebe e processa pacotes que não são destinados a ele. O transceptor também consome grande quantidade de energia quando transmite ou recebe pacotes. Para o transceptor RFM TR1000 a transmissão de um único bit requer aproximadamente $1 \mu\text{J}$ enquanto a recepção requer $0.5 \mu\text{J}$ [Hill et al., 2000]. Já um microcontrolador requer tipicamente cerca de 1 nJ para processar uma instrução [Holger and Willig., 2005]. Em [Pottie and Kaiser, 2000] outra comparação entre a energia despendida

na computação e comunicação de dados pode ser encontrada: transmitir 1 kB de dados sobre 100 m consome aproximadamente a mesma quantidade de energia necessária para computar três milhões de instruções.

Fazer com que o rádio transite para estados de dormência sempre que estiver ocioso parece uma boa estratégia para economizar energia. No entanto, desligar o rádio cegamente durante cada período de ociosidade acaba por consumir mais energia do que se o mesmo permanecesse ligado [Akyildiz et al., 2002]. A energia despendida para que o rádio retorne ao estado anterior não pode ser negligenciada. A economia energética somente é obtida se a energia consumida durante o estado de dormência somada à energia necessária para que o rádio retorne ao estado anterior seja menor ou igual à energia que o rádio consumiria caso permanecesse ligado durante o período de dormência somado ao tempo despendido durante a transição entre os estados [Dargie, 2012].

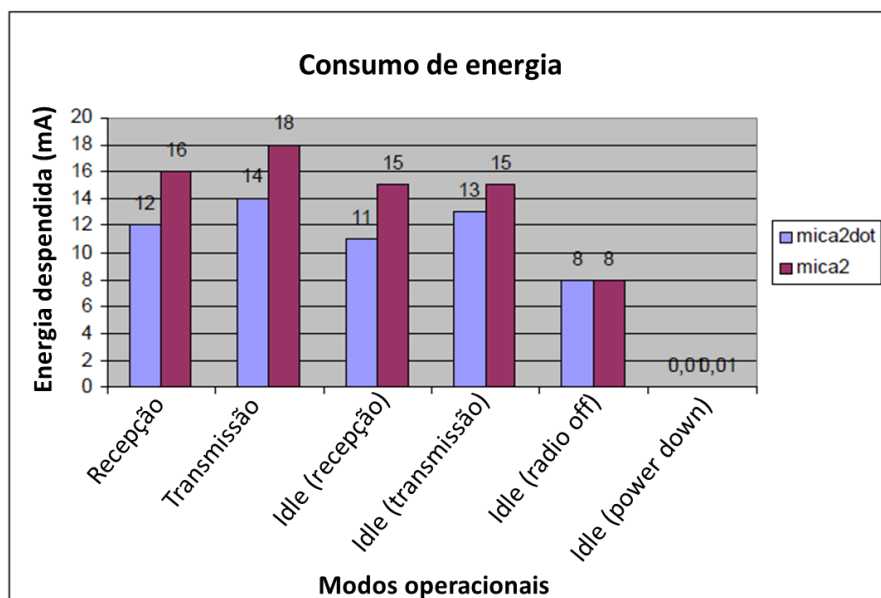


Figura 2.3: Energia despendida em diferentes modos operacionais [Anastasi et al., 2004]

2.3 Controle de Topologia

A ideia básica do controle de topologia da rede é deliberadamente restringir o conjunto de nós vizinhos de um determinado nó [Holger and Willig., 2005]. Através do controle de topologia é possível:

1. Economizar energia e conseqüentemente aumentar o tempo de vida da rede;
2. Reduzir a interferência entre os nós;

3. Reduzir a contenção na subcamada Medium Access Control (MAC);
4. Otimizar o reuso espacial e consequentemente aumentar a capacidade da rede;
5. Alcançar determinada propriedade da rede (conectividade, *throughput*, etc).

A fim de construir a topologia da rede, os nós tomam suas decisões baseando-se em informações colhidas sobre seus vizinhos e respectivos enlaces utilizando potência máxima de transmissão [Aziz et al., 2013]. No entanto, a topologia gerada pode ser muito densa e suscetível a interferências ou muito esparsa e sujeita ao particionamento como mostra a Figura 2.4. Topologias de rede adequadas podem ser geradas através das técnicas de controle de topologia que consistem basicamente na manipulação de certos parâmetros da rede pelos nós. Os nós podem: variar a potência de transmissão utilizada na comunicação de dados, transitar entre diferentes modos de operação ou criar hierarquias na rede e atribuir determinadas tarefas apenas a um conjunto de nós.

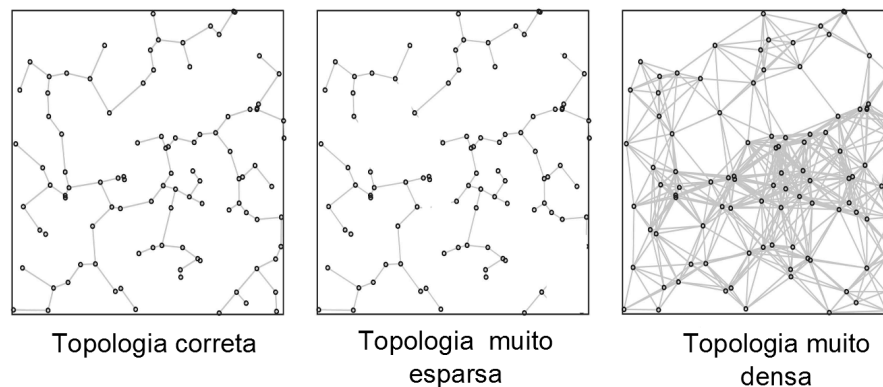


Figura 2.4: Possíveis topologias de uma RSSF (adaptado de [Blough et al., 2006])

Em [Aziz et al., 2013] algoritmos de controle de topologia distribuídos são classificados em quatro categorias de acordo com os parâmetros de rede manipulados pelos nós: ajuste de potência, modo de potência, *clustering* e híbrido. Algoritmos de ajuste de potência visam diminuir a energia despendida durante a transmissão dos dados. Para isso, ao invés de transmitir os dados com potência máxima, os nós trabalham de forma colaborativa para encontrar a potência de transmissão adequada. Algoritmos pertencentes ao grupo modo de potência visam a economia de energia colocando o maior número possível de nós em estado de dormência. Algoritmos do tipo *clustering* modificam a arquitetura da rede selecionando conjuntos de nós para formarem *clusters*. O encaminhamento e a agregação de dados são restringidos aos nós pertencentes ao *cluster*. Geralmente um dos nós é eleito *clusterhead* e torna-se responsável pelo encaminhamento dos dados até o destino. Algoritmos híbridos procuram maximizar os ganhos utilizando técnicas de *clustering* combinadas com técnicas de ajuste de potência ou modo de potência.

A seguir são detalhadas as características das quatro categorias de algoritmos de controle de topologia distribuídos.

2.3.1 Ajuste de potência

A energia despendida na transmissão dos dados deve-se a dois fatores [Holger and Willig., 2005]: a geração do sinal de radiofrequência e o funcionamento dos componentes eletrônicos do transmissor. A energia gasta na geração do sinal de radiofrequência depende principalmente da modulação escolhida e da potência de transmissão selecionada. Algoritmos de ajuste de potência objetivam a economia de energia permitindo que os nós selecionem a potência de transmissão apropriada ao invés de transmitirem usando a potência máxima. Como resultado, enlaces de longa distância são eliminados e conseqüentemente a rede torna-se menos densa e a interferência entre os nós é mitigada. No entanto, a perspectiva da eficiência energética deve ser da rede e não dos nós isoladamente como demonstra a Figura 2.5.

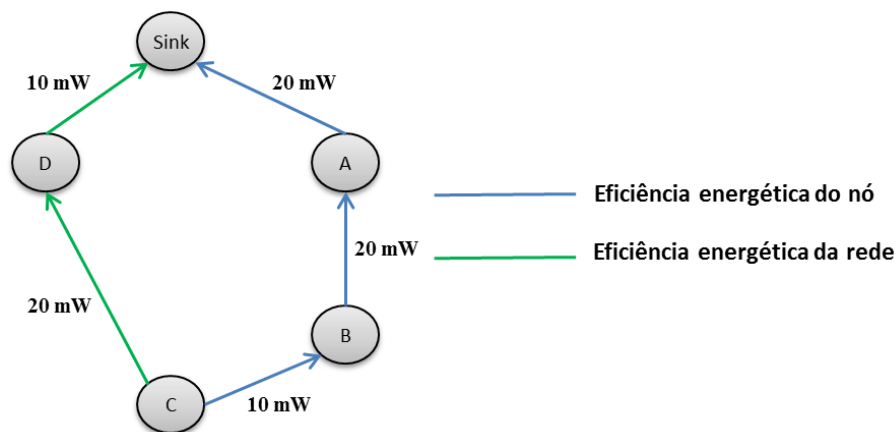


Figura 2.5: Perspectiva da eficiência energética

Caso o objetivo do nó *C* fosse sua própria eficiência energética, então a melhor rota até o sink seria através do nó *B* e a potência de transmissão selecionada corresponderia a 10 mW. Porém, a eficiência energética da rede inteira somente é obtida através do nó *D* cujo alcance exige que a potência de transmissão selecionada corresponda a 20 mW.

2.3.2 Modo de potência

Existem duas abordagens distintas utilizadas pelos algoritmos de controle de topologia para colocar os nós em estados de dormência [Wawryszczuk and Amanowicz, 2012]. A primeira abordagem consiste em manter apenas um pequeno subgrupo de nós operacionais, enquanto os outros nós transitam para estados de dormência. A segunda abordagem coloca os

nós em estado de dormência temporariamente. Além de economizar energia, ambas as abordagens diminuem o número de enlaces ativos e conseqüentemente são capazes de mitigar a interferência.

Uma estratégia muito empregada para colocar os nós temporariamente em estado de dormência, é a utilização do protocolo *Time Division Multiple Access* (TDMA) [acesso múltiplo com divisão de tempo] para acesso ao meio [Sheikh et al., 2012]. O protocolo TDMA divide o canal de comunicação em slots de tempo que são organizados em um frame. Cada nó somente transmite seus dados no *slot* de tempo que lhe fora previamente atribuído. Durante os outros *slots* de tempo o nó pode desativar seus componentes. Dessa forma, evita-se colisões e retransmissões, escuta inútil e *overhearing*.

2.3.3 Clustering

A abordagem *clustering* explora as vantagens de se criar hierarquias na rede. Os nós são divididos em pequenos grupos denominados *clusters*. Cada *cluster* possui um coordenador chamado *clusterhead* e certo número de nós membros. Os nós membros encaminham dados para o *clusterhead* diretamente ou através de múltiplos saltos formando um *backbone* virtual. *Clusterheads* podem ainda conectar-se uns aos outros através de nós denominados *gateways*. Como somente um grupo de nós torna-se responsável pelo encaminhamento de dados até o destino, o número de transmissões é diminuído e conseqüentemente a interferência e a contenção na subcamada MAC são também mitigadas. A quantidade de informações em trânsito na rede pode também ser reduzida através de técnicas de agregação de dados que, além disso, proveem informações mais precisas [Tan and Korpeoglu, 2003, Heinzelman et al., 2000]. Para evitar interferências dentro do *cluster*, uma abordagem comum é utilizar o protocolo TDMA. Geralmente, os *clusterheads* criam os *slots* de tempo e distribuem para os nós membros. Nós *clusterheads* tendem a exaurir suas reservas energéticas mais rapidamente devido à grande quantidade de tarefas atribuídas. Além disso, como qualquer outro nó, podem falhar ou mover-se desconectando os nós membros da rede. Uma solução para atenuar estes problemas é recriar os *clusters* periodicamente. Até mesmo porque, alguns parâmetros utilizados na eleição dos *clusterheads* são dinâmicos (energia residual, grau do nó, etc.).

A fim de maximizar os benefícios do controle de topologia, alguns algoritmos utilizam técnicas de *clustering* combinadas com técnicas de ajuste de potência ou modo de potência gerando abordagens híbridas.

2.4 Tempo de Vida da Rede

Em alguns cenários, substituir as baterias dos nós sensores pode ser inviável. Os nós podem ser implantados em regiões inacessíveis e inóspitas, como por exemplo no fundo dos

oceanos. Nestes casos, é importante maximizar o tempo de vida de cada um dos nós utilizando eficientemente seus recursos. Quando um dos nós morre, a rede inteira pode perder a conectividade deixando de exercer suas funções.

O termo tempo de vida da rede diz respeito ao tempo em que a rede permanece operacional. Não existe um consenso entre os pesquisadores sobre quando o tempo de vida da rede expira. As opções mais utilizadas são:

- O tempo até que o primeiro nó morre;
- O tempo até que o último nó morre;
- Número de nós vivos como uma função do tempo;
- Tempo decorrido até a rede falhar na construção do *backbone* - Aplicável à redes hierarquizadas;
- Tempo decorrido até a rede falhar na notificação de um evento.

2.5 *Convergecast* e Árvores de Roteamento

Convergecast é o termo utilizado para designar a coleta de dados pelos nós sensores e posterior encaminhamento destes dados até o nó *sink*. É uma operação fundamental em RSSFs [Incel et al., 2012]. Enquanto em uma rede ad hoc típica cada nó precisa comunicar-se com todos os outros nós da rede, em uma RSSF os nós sensores precisam comunicar-se somente com o nó *sink*. É um padrão de comunicação muitos-para-um que naturalmente conduz para uma estrutura de roteamento baseada em árvore. Exemplos de RSSF que utilizam árvores de roteamento podem ser encontrados em [Pan et al., 2013, Elhabyan and Yagoub, 2013, Bechkit et al., 2012, Incel et al., 2012, Saadat and Mirjalily, 2012]. A Figura 2.6 exhibe um grafo representando uma RSSF constituída por 100 nós antes da construção de uma estrutura de roteamento. A Figura 2.7 representa uma RSSF constituída por 100 nós após a construção de uma estrutura de roteamento baseada em árvore.

Em ambas as Figuras 2.6 e 2.7 o vértice em vermelho representa o nó *sink* e os demais vértices representam os nós sensores. Nota-se na Figura 2.7 que o nó *sink* é a raiz da árvore de roteamento construída, enquanto que os nós sensores formam os galhos e as folhas. Na Figura 2.6 as arestas entre os nós representam todos os enlaces da RSSF. Na Figura 2.7 as arestas representam somente os enlaces utilizados para o encaminhamento dos dados até o nó *sink*.

A fim de determinar quais são os melhores caminhos, dentre todos os caminhos possíveis, alguns algoritmos para construção de *spanning trees* atribuem peso às arestas. Em uma RSSF este peso representa o custo de se percorrer determinado enlace. Várias métricas distintas

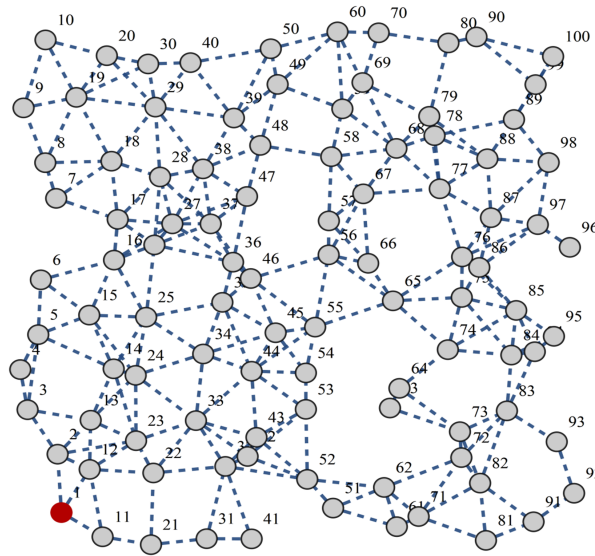


Figura 2.6: RSSF antes da construção da estrutura de roteamento

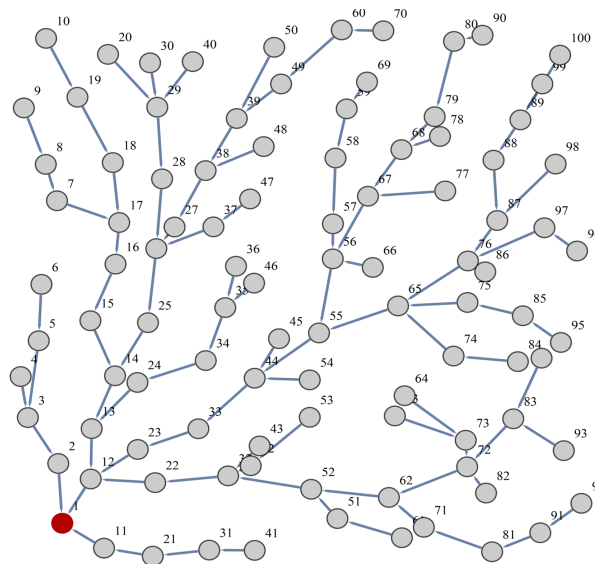


Figura 2.7: RSSF após a construção da estrutura de roteamento

para computar o custo dos enlaces foram propostas na literatura. Abordagens básicas utilizam o número de saltos e a distância até o nó *sink* como métrica. A intenção nestes dois casos é utilizar os caminhos mais curtos, pois a dissipação de energia e o atraso fim-a-fim estão correlacionados com o comprimento do percurso [Bechkit et al., 2012]. Outras métricas utilizadas e respectivos objetivos estão descritas na Tabela 2.1.

A utilização de estruturas de roteamento baseadas em árvore em RSSF apresenta várias vantagens. Os nós sensores necessitam transmitir seus dados somente para o nó pai. A

Métrica	Objetivos
Dissipação de energia	Minimizar o consumo de energia
Energia residual	Adiar o tempo em que o primeiro nó morre
Qualidade do enlace	Reduzir o número de retransmissões
Latência	Minimizar o atraso fim-a-fim

Tabela 2.1: Métricas utilizadas para o roteamento em RSSF [Bechkit et al., 2012]

inundação, abordagem na qual os nós retransmitem para todos os vizinhos os dados recebidos independentemente de os vizinhos já terem recebido os referidos dados de outros nós, é inadequada para operações de *convergecast*. A inundação possui dois problemas primários: a implosão, quando múltiplas cópias de dados são enviadas para um nó e o encaminhamento redundante, quando um nó despande energia para receber e encaminhar os mesmos dados [Chen et al., 2010]. Outra vantagem é a facilidade em eliminar enlaces não utilizados para o encaminhamento dos dados através do ajuste de potência, técnica de controle de topologia descrita na seção 2.3.1. Conhecendo o nó receptor, é possível saber que nível de potência utilizar para a transmissão dos dados ao invés de utilizar a potência máxima de transmissão. Quando enlaces desnecessários são eliminados, a rede torna-se menos densa e a interferência entre os nós é minimizada. Árvores são ainda a estrutura mais adequada para a utilização do protocolo TDMA, cujo funcionamento e vantagens estão descritos na seção 2.3.2. Quando o protocolo TDMA é utilizado, a contenção na subcamada MAC é reduzida, o reuso espacial é otimizado e consequentemente a capacidade da rede aumenta. Vários algoritmos visando minimizar o número de *slots* de tempo necessários para que a interferência nas RSSF seja completamente eliminada foram propostos na literatura. Exemplos destes algoritmos e da utilização de árvores em conjunto com o protocolo TDMA em RSSF podem ser encontrados em [Shrivastava and Pokle, 2014, Incel et al., 2012, Amdouni et al., 2012, Ergen and Varaiya, 2010, Tsai and Chen, 2008, Gandham et al., 2006].

2.6 Considerações Finais

Este Capítulo discorreu sobre as características das RSSF e mostrou porque as mesmas são diferentes de outras redes de comunicação de dados. A arquitetura do nó sensor foi apresentada e mostrou-se que o dispositivo possui diferentes modos operacionais e é capaz de transitar entre estes modos desativando/ativando seus componentes. Foram ainda discutidas as quatro diferentes técnicas de Controle de Topologia, que consistem na manipulação de certos parâmetros da rede pelos nós a fim de alcançar determinado objetivo. Explicou-se o significado do termo tempo de vida da rede e quais são as opções mais utilizadas para determinar quando a rede deixa de exercer suas funções. Por fim, o termo *covergecast* foi introduzido e esclareceu-

se porque protocolos de roteamento para redes ad hoc são inadequados para RSSF e porque estruturas de roteamento baseadas em árvore são apropriadas e vantajosas.

Capítulo 3

Trabalhos Relacionados

3.1 Algoritmos Distribuídos para Construção de *Spanning Trees*

Os métodos mais comuns para construção de *spanning trees* são busca em largura (*Breadth-First Search* - BFS) e busca em profundidade (*Depth-First Search* - DFS). A estratégia da busca em profundidade é percorrer os nós filhos antes de percorrer os nós irmãos, construindo árvores com poucos galhos e folhas e grande profundidade [Saadat and Mirjalily, 2012]. A profundidade da estrutura construída introduz atrasos e aumenta a latência na rede. Além disso, uma grande quantidade de tempo é necessária para construir a árvore DFS, pois a busca em profundidade é inerentemente sequencial [Reif, 1985]. Certas aplicações requerem uma rápida reconstrução da rede em caso de falhas. Por exemplo, em aplicações de missão crítica e salvamento os dados necessitam ser entregues dentro de um tempo limite para evitar eventos imprevisíveis e catastróficos [Incel et al., 2012].

A estratégia da busca em largura é percorrer os nós irmãos antes de percorrer os nós filhos, construindo árvores com muitos galhos e pouca profundidade [Saadat and Mirjalily, 2012]. A busca em largura é a abordagem utilizada para a construção de SPTs (*Shortest Path Trees*) [árvores de caminhos mais curtos], estruturas de roteamento muito utilizadas em RSSF pela eficiência em termos de tempo e energia [Yilmaz et al., 2012]. A SPT é assim denominada porque o caminho de qualquer um dos nós até a raiz é sempre, dentre todos os caminhos possíveis, o caminho mais curto. A Figura 3.1 exhibe exemplos de SPTs.

Existem vários algoritmos para construir SPTs, incluindo os algoritmos de Dijkstra e Bellman-Ford. O algoritmo Bellman-Ford possui uma versão assíncrona distribuída proposta por Bertsekas e Gallager em [Bertsekas and Gallager, 1992] para sistemas distribuídos e portanto apropriada para redes ad hoc. Exemplos da utilização do algoritmo Bellman-Ford em RSSF podem ser encontrados em [Bechkit et al., 2012, Chang and Tassiulas, 2004, Coleri and

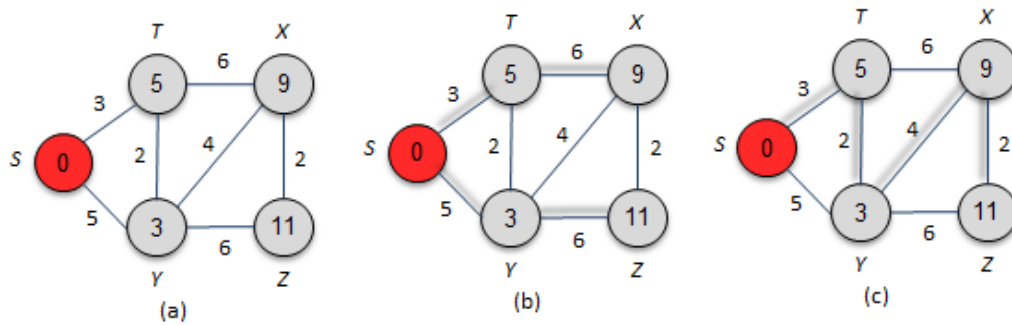


Figura 3.1: (a) Grafo orientado ponderado com pesos de caminhos mais curtos desde a origem s . (b) As arestas sombreadas formam uma árvore de caminhos mais curtos com raiz na origem s . (c) Outra árvore de caminhos mais curtos com a mesma raiz (adaptado de [Cormen, 2001]).

Varaiya, 2004]. No entanto, o alto *overhead* de comunicação do Bellman-Ford o torna inviável para RSSF. O grande número de mensagens necessárias para a construção da árvore antecipa a exaustão dos recursos energéticos dos nós sensores e conseqüentemente reduz o tempo de vida da rede. O objetivo do algoritmo *Shortest Hop Multipath* proposto em [Yilmaz et al., 2012] é reduzir o *overhead* de comunicação do Bellman-Ford e fornecer uma solução adequada para construção de árvores de roteamento para RSSF. O SHM ainda constrói múltiplos caminhos entre os nós para serem utilizados como mecanismos de tolerância a falhas e balanceamento de carga.

As próximas subseções descrevem o funcionamento de um algoritmo distribuído de busca em profundidade e dos algoritmos Bellman-Ford e SHM. Os pseudocódigos empregam a notação de teoria dos conjuntos.

3.1.1 Algoritmo Bellman-Ford

Esta subseção descreve o funcionamento da versão assíncrona distribuída do algoritmo Bellman-Ford (*Distributed Bellman-Ford* - DBF). O DBF constrói a SPT através da seguinte iteração [Bertsekas and Gallager, 1992]:

$$W_i^{t+1} = \min_{j \in N_i} \text{cost}_{i,j} + W_j^t, \quad i \neq 1 \quad (3.1)$$

Partindo das seguintes condições:

$$W_i^0 = \infty, \quad i \neq 1 \quad (3.2)$$

$$W_1^0 = 0 \quad (3.3)$$

Sendo que:

N_i = conjunto de nós vizinhos de i , isto é, os nós conectados a i através de um enlace.

W_i^t = peso do nó i na etapa t , ou seja, a última estimativa do custo do caminho de i até o nó raiz computada no nó i .

$cost_{i,j}$ = peso atribuído ao enlace entre os nós i e j .

O pseudocódigo do DBF adaptado para as RSSF é exibido nos Algoritmos 1 e 2. A variável $parent_i$ representa o nó pai do nó i na RSSF. Inicialmente, o nó pai de i é nulo e o pai do nó $sink$ é sempre ele mesmo.

Algorithm 1 Distributed Bellman-Ford - Initiator Node

- 1: $parent_{sink} = sink; W_{sink} = 0$
 - 2: broadcast $\langle sink, W_{sink} \rangle$
-

Algorithm 2 Distributed Bellman-Ford - Non Initiator Node

- 1: $parent_i = null; W_i = \infty$
 - 2: **while** node i receives $\langle j, W_j \rangle$ message from node j
 - 3: **if** $W_i > W_j + cost_{i,j}$ **then**
 - 4: $parent_i = j$
 - 5: $W_i = W_j + cost_{i,j}$
 - 6: broadcast $\langle i, W_i \rangle$
 - 7: **end if**
 - 8: **end while**
-

O nó $sink$ inicia o processo de construção da SPT enviando uma mensagem em *broadcast* informando seu peso. Esta etapa é representada pelo passo 2 do Algoritmo 1. Quando um determinado nó i recebe a mensagem de configuração contendo o peso do nó emissor j , computa o custo do caminho até o $sink$ caso escolhesse j como nó pai. Para isso, o custo do enlace entre os nós i e j é somado ao peso fornecido. Caso o custo da rota atual seja superior ao custo da rota que está sendo ofertada (passo 3 do Algoritmo 2), o nó i executa os seguintes procedimentos, representados no Algoritmo 2:

1. Seleciona j como pai temporário (passo 4);
2. Atualiza seu peso (passo 5);
3. Envia uma mensagem em *broadcast* informando seu novo peso (passo 6).

O algoritmo termina quando não existem mais mensagens em trânsito na rede. Bertsekas e Gallager [Bertsekas and Gallager, 1992] provaram que a SPT é construída mesmo que alguns nós demorem mais que outros para propagar ou calcular seu peso.

3.1.2 Algoritmo Shortest Hop Multipath

O algoritmo SHM foi concebido com o objetivo de oferecer uma solução adequada para a construção de SPTs em RSSF. A métrica utilizada é o número de saltos. Os autores argumentam que o algoritmo DBF é inadequado para esta tarefa devido ao alto custo energético e também porque o nó *sink* não possui meios de saber quando a execução do algoritmo termina. Para que o nó *sink* tenha acesso à esta informação, a execução do algoritmo SHM é sincronizada através de um mecanismo denominado *sincronizadores* β . O nó *sink* é chamado nó sincronizador e é responsável por iniciar e finalizar a construção da *spanning tree* que é feita camada a camada. A cada turno, um dos níveis da *spanning tree* é construído. O pseudocódigo do SHM é exibido nos Algoritmos 3 e 4. As Figuras 3.2 e 3.3 exibem um exemplo da execução do algoritmo SHM. Na explicação detalhada dos pseudocódigos que será feita a seguir, os procedimentos realizados pelo nó *sink* dizem respeito ao Algoritmo 3 e os procedimentos realizados pelos demais nós dizem respeito ao Algoritmo 4.

Algorithm 3 Shortest Hop Multipath Algorithm - Initiator Node

```

1:  $level = 1$ ;  $rcdAck_{sink} = 0$ ;  $NGB_{sink} = \{\text{nodes one hop away from sink}\}$ 
2: broadcast probe  $\langle sink, level \rangle$  message
3: while sink receives ack  $\langle j, parent_j \rangle$  message from node  $j$ 
4:  $rcdAck_{sink} ++$ 
5: if  $rcdAck_{sink} == |NGB_{sink}|$  then
6:    $rcdPulseAck_{sink} = rcdPulseNack_{sink} = 0$ 
7:   broadcast pulse  $\langle sink, level \rangle$  message
8: end if
9: end while
10: while sink receives pulseAck message
11:  $rcdPulseAck_{sink} ++$ 
12: if  $rcdPulseAck_{sink} + rcdPulseNack_{sink} == |NGB_{sink}|$  then
13:    $level ++$ 
14:    $rcdPulseAck_{sink} = rcdPulseNack_{sink} = 0$ 
15:   broadcast pulse  $\langle sink, level \rangle$  message
16: end if
17: end while
18: while sink receives pulseNack message
19:  $rcdPulseNack_{sink} ++$ 
20: if  $rcdPulseAck_{sink} + rcdPulseNack_{sink} == |NGB_{sink}|$  then
21:   if  $rcdPulseAck_{sink} == 0$  then
22:      $terminate = true$ 
23:   end if
24: else
25:    $level ++$ 
26:    $rcdPulseAck_{sink} = rcdPulseNack_{sink} = 0$ 
27:   broadcast pulse  $\langle sink, level \rangle$  message
28: end if
29: end while

```

Algorithm 4 Shortest Hop Multipath Algorithm - Non-initiator Node

```

1:  $parent_i = null$ ;  $AP_i = \emptyset$ ;  $CHD_i = \emptyset$ ;  $rcdAck_i = 0$ ;  $NGB_i = \{\text{nodes one hop away from } i\}$ ;
    $PSBL\_CHD_i = NGB_i$ 
2: while node  $i$  receives probe  $\langle j, level \rangle$  message from node  $j$ 
3: if  $parent_i == null$  then
4:    $parent_i = j$ 
5:    $level_i = level$ 
6:   broadcast ack  $\langle i, parent_i \rangle$  message
7: else if  $level_i == level$  then
8:    $AP_i = AP_i \cup \{j\}$ 
9: end if
10: end while
11: while node  $i$  receives pulse  $\langle j, level \rangle$  message from node  $j$ 
12: if  $parent_i == j$  then
13:   if  $PSBL\_CHD_i == \emptyset$  then
14:     unicast pulseNack message to  $parent_i$ 
15:   else if  $level_i == level$  then
16:      $probe_i = true$ 
17:     broadcast probe  $\langle i, level_i + 1 \rangle$  message
18:   else if  $level_i < level$  then
19:      $rcdPulseAck_i = rcdPulseNack_i = 0$ 
20:     broadcast pulse  $\langle i, level \rangle$  message
21:   end if
22: end if
23: end while
24: while node  $i$  receives ack  $\langle j, parent_j \rangle$  message from node  $j$ 
25:  $rcdAck_i ++$ 
26: if  $parent_j \neq i$  then
27:    $PSBL\_CHD_i = PSBL\_CHD_i \setminus \{j\}$ 
28: else
29:    $CHD_i = CHD_i \cup \{j\}$ 
30: end if
31: if  $probe_i == true$  and  $rcdAck_i == |NGB_i|$  then
32:   unicast pulseAck message to  $parent_i$ 
33: end if
34: end while
35: while node  $i$  receives pulseAck message
36:  $rcdPulseAck_i ++$ 
37: if  $rcdPulseAck_i + rcdPulseNack_i == |CHD_i|$  then
38:   unicast pulseAck message to  $parent_i$ 
39: end if
40: end while
41: while node  $i$  receives pulseNack message
42:  $rcdPulseNack_i ++$ 
43: if  $rcdPulseAck_i + rcdPulseNack_i == |CHD_i|$  then
44:   if  $rcdPulseAck_i == 0$  then
45:     unicast pulseNack message to  $parent_i$ 
46:   else
47:     unicast pulseAck message to  $parent_i$ 
48:   end if
49: end if
50: end while

```

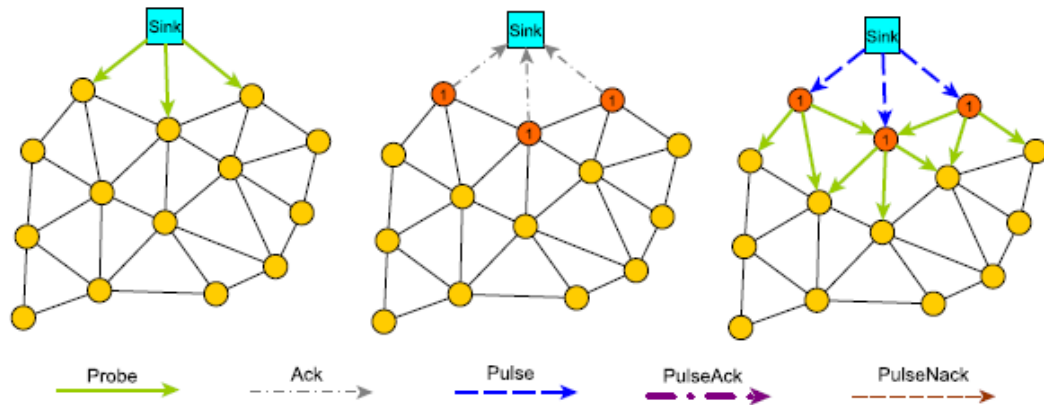


Figura 3.2: (a) Mensagem *probe* (início do SHM). (b) Mensagem *ack*. (c) Mensagem *pulse* e mensagens *probe* [Yilmaz et al., 2012].

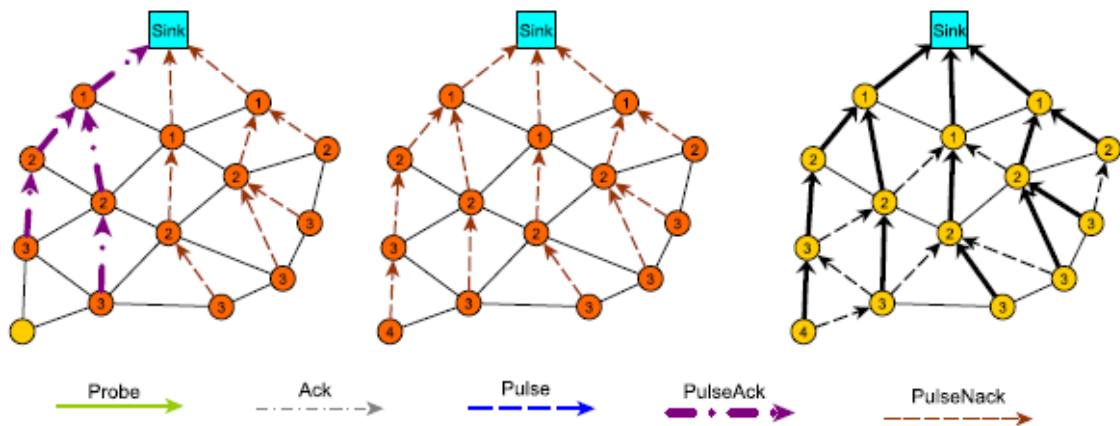


Figura 3.3: (a) Mensagens *pulseAck* e *pulseNack*. (b) Mensagens *pulseNack*. (c) SPT e rotas alternativas [Yilmaz et al., 2012].

O nó *sink* inicia o processo de construção da SPT enviando uma mensagem **probe** em *broadcast* (passo 2). O objetivo de uma mensagem **probe** é informar aos nós receptores seu nível na árvore que está sendo construída e fazer uma oferta de paternidade. Quando um determinado nó i que ainda não definiu a identidade do nó pai recebe uma mensagem **probe** de um nó emissor j , executa os seguintes procedimentos:

1. Seleciona j como pai (passo 4);
2. Configura seu nível para o nível informado na mensagem (passo 5);
3. Envia uma mensagem **ack** em *broadcast* informando a identidade de seu pai (passo 6).

Caso o nó i já tenha definido seu pai ao receber uma mensagem **probe**, verifica se o nível informado na mensagem é igual ao seu nível na árvore que está sendo construída (passo

7). Caso seja, então i adiciona j ao conjunto pais alternativos (passo 8). Esta verificação é necessária porque i não ouve somente as mensagens dos nós que estão no nível superior da árvore, mas também dos nós que estão no nível inferior e que não são possíveis pais.

O nó *sink* ao receber uma mensagem **ack**, incrementa o contador *rcdAck* (passo 4) e compara o valor calculado com o número de nós vizinhos (passo 5). Esta comparação faz parte do mecanismo de sincronização do algoritmo. Somente quando todos os nós vizinhos respondem à mensagem enviada, é que o nó *sink* começa a próxima etapa para a construção da árvore iniciando os contadores *rcdPulseAck* e *rcdPulseNack* (passo 6) e enviando uma mensagem **pulse** em *broadcast* (passo 7). A mensagem **pulse** tem como objetivo informar aos nós qual nível da árvore deve enviar mensagens **probe**. Um nó i qualquer \neq *sink*, ao receber uma mensagem **ack** contendo a identificação do pai do nó emissor j , executa os seguintes procedimentos:

1. Incrementa o contador *rcdAck* (passo 25);
2. Verifica se é o pai de j (passo 26). Caso seja, então adiciona j ao conjunto de nós filhos (passo 29). Caso contrário, retira j do conjunto de possíveis filhos (passo 27);
3. Compara o valor de *rcdAck* com o número de nós vizinhos (passo 31). Caso os números sejam iguais e i tenha ofertado sua paternidade anteriormente, então envia uma mensagem **pulseAck** para seu pai (passo 32).

Um nó i ao receber uma mensagem **pulse** de seu pai j (passo 12), verifica se seu conjunto de possíveis filhos está vazio (passo 13). Caso esteja, significa que i não tem motivos para ofertar sua paternidade e então envia uma mensagem **pulseNack** para j como resposta. Caso seu conjunto de possíveis nós filhos não esteja vazio, i compara seu nível na árvore com o nível informado na mensagem (passo 15). Se forem iguais, i envia uma mensagem **probe** em *broadcast* (passo 17), pois significa que i está na vez de ofertar sua paternidade. Caso os níveis sejam diferentes, i apenas retransmite a mensagem **pulse** recebida (passo 20).

O nó *sink* ao receber mensagens **pulseAck** e **pulseNack** incrementa seus contadores *rcdPulseAck* (passo 11) e *rcdPulseNack* (passo 19) respectivamente. As mensagens **pulseAck** e **pulseNack** também fazem parte do mecanismo de sincronização do algoritmo. Caso a soma dos contadores *rcdPulseAck* e *rcdPulseNack* seja igual ao número de nós vizinhos (passos 12 e 20) o nó *sink* inicia a próxima etapa da construção da árvore através de novas mensagens **pulse** (passos 15 e 27). Caso receba somente mensagens **pulseNack** como resposta à uma mensagem **pulse**, o nó *sink* sabe que a construção da SPT está finalizada (passos 21 e 22). Um nó $i \neq$ *sink*, ao receber mensagens **pulseAck** e **pulseNack** realiza procedimentos semelhantes: incrementa seus contadores *rcdPulseAck* (passo 36) e *rcdPulseNack* (passo 42) e verifica se a soma dos mesmos é igual ao número de elementos de seu conjunto de filhos (passos 37 e 43). Caso os

números sejam iguais, i envia uma mensagem **pulseAck** para o nó pai (passos 38 e 45). Se no entanto, i receber somente mensagens **pulseNack**, envia também uma mensagem **pulseNack** para seu pai (passo 45) pois sabe que seus descendentes não possuem nós vizinhos para quem possam ofertar a paternidade. Isto significa que a subárvore enraizada em i já está construída.

3.1.3 Algoritmo Depth-First Search

O algoritmo descrito a seguir está baseado no algoritmo distribuído de busca em profundidade proposto em [Makki and Havas, 1994]. As modificações realizadas na versão original objetivaram a utilização do número de saltos como métrica para a construção da *spanning tree*. O pseudocódigo do algoritmo é exibido nos Algoritmos 5 e 6. Na explicação detalhada dos pseudocódigos que será feita a seguir, os procedimentos realizados pelo nó *sink* dizem respeito ao Algoritmo 5 e os procedimentos realizados pelos demais nós dizem respeito ao Algoritmo 6.

O nó *sink* inicia a construção da *spanning tree* enviando uma mensagem **forward** para um de seus vizinhos (passo 3). Uma mensagem **forward** é uma oferta de paternidade que contém informações que auxiliam o nó receptor a aceitá-la ou rejeitá-la: a rota conhecida até o nó *sink* (ROUTE), os nós que rejeitaram rotas ofertadas por conhecerem rotas mais curtas (REJECTERS) e os nós que já foram percorridos (VISITED). Um determinado nó i ao receber uma mensagem **forward** de um nó j , compara sua rota com a rota que esta sendo ofertada (passos 3 e 4) e com as rotas dos nós que rejeitaram ofertas anteriores (passos 8 e 9). Caso possua algum nó vizinho que faça parte da rota ofertada ou que faça parte do conjunto de nós que rejeitaram ofertas anteriores e cujo número de saltos até o $sink + 1$ seja menor que o número de saltos de sua rota atual, o nó i atualiza sua rota (passos 5 e 11). Caso o número de saltos da rota atual de i seja menor que o número de saltos da rota ofertada + 1, i envia uma mensagem **reject** para j rejeitando a paternidade ofertada (passos 14 e 15). Caso contrário, o nó i realiza os seguintes procedimentos:

1. Retira-se do conjunto de nós que rejeitaram ofertas anteriores, caso faça parte do mesmo (passos 17 e 18);
2. Adiciona j à rota ofertada (passo 20);
3. Adiciona-se ao conjunto de nós já percorridos (passo 21);
4. O número de saltos para alcançar o *sink* é igual ao número de nós que fazem parte da rota ofertada (passo 22);
5. A rota ofertada passa a ser a rota atual de i (passo 23);
6. Seleciona j como nó pai (passo 24);

7. Examina o conjunto de nós que rejeitaram ofertas anteriores e adiciona ao seu conjunto **REJECTERS** aqueles que fatalmente iriam rejeitar sua oferta de paternidade por conhecerem rotas mais curtas (passos 26-30);
8. Constrói o conjunto de nós a serem percorridos. O conjunto é formado pelos nós vizinhos de i que ainda não foram visitados e que não fazem parte do conjunto de nós que iriam fatalmente rejeitar a oferta de paternidade (passo 31);
9. Caso o conjunto não seja vazio, uma mensagem **forward** é enviada para um de seus elementos (passos 32-34);
10. Caso o conjunto não possua elementos, uma mensagem **return** é enviada para o nó pai (passo 36).

O nó *sink* ao receber uma mensagem **return** verifica se possui nós vizinhos ainda não visitados (passos 5 e 6). Caso não possua, o nó *sink* finaliza a execução do algoritmo. Caso contrário, uma mensagem **forward** é enviada para um dos nós vizinhos ainda não visitados (passos 9 e 10). Um nó $i \neq sink$ ao receber uma mensagem **return**, realiza os seguintes procedimentos:

Algorithm 5 Depth First Search - Initiator Node

```

1:  $NGB_{sink} = \{\text{nodes one hop away from sink}\}$ ;  $UNVISITED = NGB_{sink}$ 
2: choose  $i$  such that  $\{i\} \in UNVISITED$ 
3: sink unicasts forward  $\langle sink, \emptyset, \emptyset, \{sink\} \rangle$  message to  $i$ 
4: while sink receives return  $\langle j, REJECTERS, VISITED \rangle$  message from node  $j$ 
5:  $UNVISITED = NGB_{sink} \setminus VISITED$ 
6: if  $|UNVISITED| == 0$  then
7:    $terminate = true$ 
8: else
9:   choose  $i$  such that  $\{i\} \in UNVISITED$ 
10:  sink unicasts forward  $\langle \emptyset, \emptyset, \{sink\} \rangle$  message to  $i$ 
11: end if
12: end while

```

Algorithm 6 Depth First Search - Non-initiator Node

```

1:  $hops_i = 0$ ;  $NGB_i = \{\text{nodes one hop away from } i\}$ ;  $CHD_i = \emptyset$ 
2: while node  $i$  receives forward  $\langle j, \text{ROUTE}, \text{REJECTERS}, \text{VISITED} \rangle$  message from node  $j$ 
3: for all  $\{u\} \in (\text{REJECTERS} \cap NGB_i)$  do
4:   if  $hops_i == 0$  or  $hops_i > hops_u + 1$  then
5:      $hops_i == hops_u + 1$ 
6:   end if
7: end for
8: for all  $\{u\} \in (\text{ROUTE} \cap NGB_i)$  do
9:    $hops_u = \text{position}(\text{ROUTE}, u) - 1$ 
10:  if  $hops_i == 0$  or  $hops_i > hops_u + 1$  then
11:     $hops_i == hops_u + 1$ 
12:  end if
13: end for
14: if  $hops_i > 0$  and  $hops_i < |\text{ROUTE}| + 1$  then
15:  unicast reject  $\langle i, \text{VISITED}, \text{REJECTERS}, hops_i \rangle$  message to  $j$ 
16: else
17:  if  $\{i\} \in \text{REJECTER}$  then
18:     $\text{REJECTERS} = \text{REJECTERS} \setminus \{i\}$ 
19:  end if
20:   $\text{ROUTE} = \text{ROUTE} \cup \{j\}$ 
21:   $\text{VISITED} = \text{VISITED} \cup \{i\}$ 
22:   $hops_i = |\text{ROUTE}|$ 
23:   $\text{ROUTE}_i = \text{ROUTE}$ 
24:   $\text{parent}_i = j$ 
25:   $\text{REJECTERS}_i = \emptyset$ 
26:  for all  $\{u\} \in \text{REJECTERS}$  do
27:    if  $hops_u < hops_i + 1$  then
28:       $\text{REJECTERS}_i = \text{REJECTERS}_i \cup \{u\}$ 
29:    end if
30:  end for
31:   $\text{UNVISITED} = NGB_i \setminus \text{VISITED} \setminus \text{REJECTERS}_i$ 
32:  if  $|\text{UNVISITED}| > 0$  then
33:    choose  $u$  such that  $\{u\} \in \text{UNVISITED}$ 
34:    unicast forward  $\langle i, \text{ROUTE}_i, \text{VISITED}, \text{REJECTERS} \rangle$  message to  $u$ 
35:  else
36:    unicast return  $\langle i, \text{VISITED}, \text{REJECTERS} \rangle$  message to  $\text{parent}_i$ 
37:  end if
38: end if
39: end while

```

Algorithm 6 Depth First Search - Non-initiator Node

```

40: while node  $i$  receives return  $\langle j, \text{REJECTERS}, \text{VISITED} \rangle$  message from node  $j$ 
41:  $\text{CHD}_i = \text{CHD}_i \cup \{j\}$ 
42:  $\text{REJECTERS}_i = \emptyset$ 
43: for all  $\{u\} \in \text{REJECTERS}$  such that  $\text{hops}_u < \text{hops}_i + 1$  do
44:    $\text{REJECTERS}_i = \text{REJECTERS}_i \cup \{u\}$ 
45: end for
46:  $\text{UNVISITED} = \text{NGB}_i \setminus \text{VISITED} \setminus \text{REJECTERS}_i$ 
47: if  $|\text{UNVISITED}| > 0$  then
48:   choose  $u$  such that  $\{u\} \in \text{UNVISITED}$ 
49:   unicast forward  $\langle i, \text{ROUTE}_i, \text{VISITED}, \text{REJECTERS} \rangle$  message to  $u$ 
50: else
51:   unicast return  $\langle i, \text{VISITED}, \text{REJECTERS} \rangle$  message to parent $i$ 
52: end if
53: end while
54: while node  $i$  receives reject  $\langle j, \text{VISITED}, \text{REJECTERS}, \text{hops}_j \rangle$  message from node  $j$ 
55:  $\text{REJECTERS} = \text{REJECTERS} \cup \{(j, \text{hops}_j)\}$ 
56:  $\text{REJECTERS}_i = \emptyset$ 
57: if  $\text{hops}_j + 1 < \text{hops}_i$  then
58:    $\text{hops}_i = \text{hops}_j + 1$ 
59:    $\text{VISITED} = \text{VISITED} \setminus \{i\}$ 
60:   unicast reject  $\langle i, \text{VISITED}, \text{REJECTERS}, \text{hops}_i \rangle$  message to parent $i$ 
61: else
62:   for all  $\{u\} \in \text{REJECTERS}$  do
63:     if  $\text{hops}_u < \text{hops}_i + 1$  then
64:        $\text{REJECTERS}_i = \text{REJECTERS}_i \cup \{u\}$ 
65:     end if
66:   end for
67:    $\text{UNVISITED} = \text{NGB}_i \setminus \text{VISITED} \setminus \text{REJECTERS}_i$ 
68:   if  $|\text{UNVISITED}| > 0$  then
69:     choose  $u$  such that  $\{u\} \in \text{UNVISITED}$ 
70:     unicast forward  $\langle i, \text{ROUTE}_i, \text{VISITED}, \text{REJECTERS} \rangle$  message to  $u$ 
71:   else
72:     unicast return  $\langle i, \text{VISITED}, \text{REJECTERS} \rangle$  message to parent $i$ 
73:   end if
74: end if
75: end while

```

1. Adiciona j ao seu conjunto de nós filhos (passo 41);
2. Examina o conjunto de nós que rejeitaram ofertas anteriores e adiciona ao seu conjunto REJECTERS aqueles que fatalmente iriam rejeitar sua oferta de paternidade por conhecerem rotas mais curtas (passos 42-45).
3. Constrói o conjunto de nós a serem percorridos. O conjunto é formado pelos nós vizinhos de i que ainda não foram visitados e que não fazem parte do conjunto de nós que iriam fatalmente rejeitar a oferta de paternidade (passo 46);

4. Caso o conjunto não seja vazio, uma mensagem **forward** é enviada para um de seus elementos (passos 47-49);
5. Caso o conjunto não possua elementos, uma mensagem **return** é enviada para o nó pai (passo 51).

Um nó i ao receber uma mensagem **reject** de um nó j , adiciona j ao conjunto de nós que rejeitaram ofertas de paternidade (REJECTERS) (passo 55). Caso o número de saltos de j até o nó $sink + 1$ for menor que o número de saltos de i até o $sink$, o nó i atualiza seu número de saltos, retira-se do conjunto de nós que já foram percorridos e envia uma mensagem **reject** para o nó pai (passos 57-60). Caso contrário, o nó i realiza os seguintes procedimentos:

1. Examina o conjunto de nós que rejeitaram ofertas anteriores e adiciona ao seu conjunto REJECTERS aqueles que fatalmente iriam rejeitar sua oferta de paternidade por conhecerem rotas mais curtas (passos 62-66);
2. Constrói o conjunto de nós a serem percorridos. O conjunto é formado pelos nós vizinhos de i que ainda não foram visitados e que não fazem parte do conjunto de nós que iriam fatalmente rejeitar a oferta de paternidade (passo 67);
3. Caso o conjunto não seja vazio, uma mensagem **forward** é enviada para um de seus elementos (passos 68-70);
4. Caso o conjunto não possua elementos, uma mensagem **return** é enviada para o nó pai (passo 72).

3.1.4 Comparações entre Algoritmos Distribuídos para Construção de *Spanning Trees*

Para comparar o desempenho dos algoritmos descritos nas subseções anteriores (DBF, SHM e DFS), foram realizadas simulações utilizando o *software* Mathematica [Wolfram Research, 2012]. A modelagem da rede e o ambiente de simulação estão descritos na Seção 5.1. Os cenários avaliados envolveram topologias de rede constituídas por 50 e 100 nós. As métricas utilizadas foram o número de mensagens necessárias para a construção da árvore e o tempo de execução dos algoritmos. As Figuras 3.4 e 3.5 apresentam os resultados obtidos.

O algoritmo Bellman-Ford apresentou, dentre os algoritmos avaliados, o menor tempo de execução. No entanto, o desempenho com relação ao número de mensagens degrada consideravelmente com o aumento do número de nós. Em uma rede constituída por 50 nós, o DBF necessita de aproximadamente 25 mensagens por nó para a construção da *spanning tree*. Em

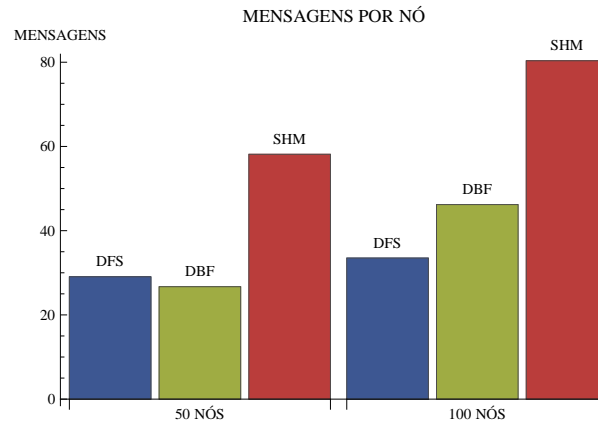


Figura 3.4: Número Médio de Mensagens por Nó

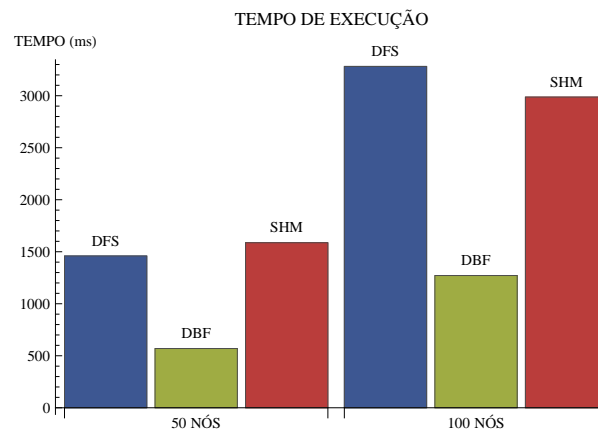


Figura 3.5: Tempo de Execução

uma rede com 100 nós, o número de mensagens necessárias é de aproximadamente 45 mensagens por nó. Portanto, um aumento de 80%. AS RSSF são geralmente formadas por um grande número de nós, o que inviabiliza a utilização do DBF. Além disso, o nó *sink* não possui meios de saber quando a execução do algoritmo termina e caso necessite disseminar dados na rede, precisa esperar por um tempo indeterminado. Para que o nó *sink* tenha acesso à esta informação, o algoritmo SHM paga um alto preço tanto com relação ao número de mensagens quanto com relação ao tempo de execução do algoritmo. Comparativamente ao algoritmo DBF, o aumento no tempo de execução do SHM é de aproximadamente 130%. Isto torna o SHM inadequado para aplicações que necessitam de uma rápida reconfiguração da estrutura de roteamento em caso de falhas. Além disso, os nós sensores necessitam conhecer todos os nós localizados a um salto de distância e os autores não explicam como esta informação é obtida. A métrica utilizada para a construção da árvore não é flexível, admitindo-se somente o número de saltos até o *sink*. No entanto, a maior dificuldade na utilização do SHM em um cenário real é o fato de que o

algoritmo não está preparado para a perda de mensagens, um evento bastante comum em redes *wireless*. Isto se deve ao próprio funcionamento do algoritmo: sincronização e mensagens enviadas em *unicast*. Um nó pai para dar prosseguimento à execução do algoritmo, aguarda o recebimento de uma mensagem (**pulseAck** ou **pulseNack**) enviada por cada um de seus filhos. Caso uma destas mensagens não seja entregue, o nó pai permanece esperando o recebimento da mesma por um tempo indeterminado. Um modo de resolver esta dificuldade seria utilizar um mecanismo de confirmação de mensagens recebidas. No entanto, este mecanismo aumentaria ainda mais o número de mensagens necessárias para a construção da árvore e o tempo de execução do algoritmo. A vantagem do algoritmo SHM é a resiliência, pois os nós possuem pais alternativos. Caso o nó pai falhe ou morra, os nós filhos possuem rotas alternativas para enviar os dados coletados até o *sink*. As rotas alternativas podem também ser utilizadas para o balanceamento de carga na rede.

O algoritmo DFS, embora tenha apresentado o melhor desempenho com relação ao número de mensagens, apresentou o pior desempenho com relação ao tempo de execução. O motivo para o baixo desempenho é o fato de que o DFS é um algoritmo sequencial. Por este mesmo motivo, o DFS assim como o SHM, não está preparado para perdas de mensagens e exige que os nós conheçam todos os outros nós localizados a um salto de distância. A Tabela 3.1 resume as principais dificuldades para a utilização dos algoritmos descritos.

Dificuldade	DBF	SHM	DFS
O <i>sink</i> não sabe quando a execução do algoritmo termina	•		
Métrica fixa		•	•
Os nós precisam conhecer seus vizinhos		•	•
Necessidade de confirmação das mensagens recebidas		•	•
Alto <i>overhead</i> de comunicação	•	•	
Alto <i>overhead</i> de tempo		•	•

Tabela 3.1: Principais dificuldades para a utilização dos algoritmos DBF, SHM e DFS

3.2 Considerações Finais

Este Capítulo explicou quais são os principais métodos utilizados para construir *spanning trees* e apresentou exemplos de algoritmos distribuídos que utilizam estes métodos para a construção de estruturas de roteamento em RSSF. O funcionamento destes algoritmos foi detalhado e o desempenho dos mesmos foi avaliado através de testes e simulações. Mostrou-se quais as vantagens e dificuldades em empregar os referidos algoritmos para construção de árvores de roteamento em um cenário real. As dificuldades apontadas para a utilização dos algoritmos avaliados evidenciaram a necessidade de um algoritmo adequado para RSSF que possa ser empregado em cenários reais.

Capítulo 4

Algoritmo Proposto

Este capítulo apresenta o algoritmo proposto, denominado Modified Bellman-Ford (MBF), baseado na versão assíncrona distribuída do algoritmo Bellman-Ford e cujo objetivo é fornecer uma solução viável para construção de árvores de roteamento em RSSF. A concepção do algoritmo foi orientada pelos seguintes objetivos:

1. Baixo *overhead* de comunicação – A troca extensiva de mensagens abrevia o tempo de vida da rede;
2. Baixo *overhead* de tempo – Algumas aplicações das RSSF requerem que os dados cheguem até o destino dentro de um determinado espaço de tempo e que a rede seja rapidamente reconfigurada caso os nós falhem ou morram;
3. Simplicidade – O algoritmo deve demandar poucos recursos computacionais e não exigir o conhecimento da topologia da rede;
4. Escalabilidade – A solução deve ser também eficiente para redes formadas por um grande número de nós;
5. Utilizar abordagem distribuída - A abordagem centralizada não é adequada para RSSF como anteriormente explanado na Seção 1.1;
6. Adequado para redes sem fio – O algoritmo deve ser capaz de construir a estrutura de roteamento mesmo que ocorram perdas de mensagens;
7. Resiliência – Assim como o algoritmo SHM, a solução deve prover aos nós múltiplos caminhos até o sink para serem utilizados como mecanismos de tolerância a falhas e balanceamento de carga;
8. Qualidade da árvore – a qualidade da árvore construída deve ser semelhante à qualidade da estrutura construída pelo algoritmo DBF.

4.1 Descrição do Algoritmo Proposto

Por estar baseado na versão assíncrona distribuída do algoritmo Bellman-Ford o algoritmo proposto MBF herda algumas de suas características como a simplicidade e a capacidade de construir a estrutura de roteamento, mesmo que ocorram perdas de mensagens durante a execução do algoritmo. Esta tolerância à perda de mensagens ocorre porque as mensagens são sempre enviadas em *broadcast* e não existe necessidade de uma estratégia de entrega confiável das mensagens. Portanto, não existe também a possibilidade de que um nó fique aguardando indefinidamente uma mensagem para dar prosseguimento à execução do algoritmo. Além disso, basta uma única mensagem para que o nó selecione seu pai na estrutura de roteamento e a rede mantenha-se conectada. Logicamente, quanto maior o número de mensagens perdidas, menor a qualidade da árvore gerada. O algoritmo também não requer que hajam confirmações das mensagens recebidas. Assim como o DBF, o MBF demanda poucos recursos computacionais e os nós não necessitam conhecer seus vizinhos.

O excessivo *overhead* na troca de mensagens do algoritmo DBF é a principal dificuldade para sua utilização nas RSSF. Quanto mais conectada a rede e quanto maior o número de nós, pior é o desempenho do DBF. O motivo para o baixo desempenho é a utilização do mecanismo de inundação no processo de construção da árvore. Os nós aceitam e retransmitem para todos os seus vizinhos qualquer oferta de paternidade para que os dados cheguem até o destino, bastando que a mesma seja mais vantajosa que a oferta anteriormente recebida em termos da métrica de construção da árvore.

A principal modificação proposta é uma estratégia para diminuir o número de mensagens necessárias para a construção da árvore e o tempo de execução do algoritmo DBF, tornando-o apropriado para as RSSF. Diferentemente do algoritmo original, somente são aceitas as ofertas cuja vantagem com relação à oferta anterior seja maior ou igual a determinado fator denominado *alfa*. O fator *alfa* determina o quanto a oferta recebida deve ser mais vantajosa que a anterior para ser aceita, de modo que sua rejeição não cause impacto na qualidade da árvore.

Com relação aos múltiplos caminhos, a estratégia consiste em fazer com que os nós armazenem todas as ofertas recebidas e deste modo possuam um conjunto de pais alternativos, para quem os dados possam ser enviados. Caso um mesmo candidato a pai envie várias ofertas, somente a última oferta é armazenada.

De forma semelhante ao algoritmo Bellman-Ford assíncrono distribuído e assumindo-se que o nó i representa o nó *sink*, a construção da *spanning tree* pode ser dada pela iteração:

$$W_i^{t+1} = \min_{j \in N_i} \text{cost}_{i,j} + W_j^t, \quad i \neq 1 \quad (4.1)$$

Sujeito à restrição:

$$adv_{i,j} \geq \alpha \quad (4.2)$$

Onde:

$$adv_{i,j} = (W_i^t - (cost_{i,j} + W_j^t)) / W_i^t \quad (4.3)$$

Partindo das seguintes condições:

$$W_i^0 = \infty, \quad i \neq 1 \quad (4.4)$$

$$W_1^0 = 0 \quad (4.5)$$

Sendo que:

N_i = conjunto de nós vizinhos de i , isto é, os nós localizados a um salto de distância.

W_i^t = peso do nó i na etapa t , ou seja, a última estimativa do custo do caminho de i até o *sink* computada no nó i .

$cost_{i,j}$ = estimativa do custo do enlace entre os nós i e j .

$adv_{i,j}$ = vantagem da rota ofertada por j com relação ao custo da rota atual do nó i .

O pseudocódigo do Algoritmo MBF é apresentado nos algoritmos 7 e 8. A Figura 4.1 apresenta um exemplo do funcionamento do algoritmo em uma rede com cinco nós e parâmetro $\alpha = 0.2$. O *sink* inicia o processo de construção enviando uma mensagem em *broadcast* informando seu peso. Esta etapa é representada pelo passo 2 do Algoritmo 7. Quando um determinado nó i recebe a mensagem de configuração contendo o peso do nó emissor j , computa o custo do caminho até o *sink* caso escolhesse j como nó pai. Para isso, o custo do enlace (i,j) é somado ao peso fornecido. Caso a nova rota ofereça vantagem com relação à rota atual no mínimo igual ao fator α (passo 11 do Algoritmo 8), o nó i executa os seguintes procedimentos, representados no Algoritmo 8:

1. Confere se j é seu pai. Caso não seja, então o nó pai é adicionado ao conjunto de pais alternativos (passos 12 e 13). Esta verificação é necessária porque mesmo o nó pai pode encontrar rotas mais vantajosas;
2. Seleciona j como pai temporário (passo 14);
3. Atualiza seu peso (passo 16);
4. Envia uma mensagem em *broadcast* informando seu novo peso (passo 17).

Se, entretanto, a nova rota não for considerada suficientemente vantajosa, o nó i simplesmente adiciona j juntamente com seu respectivo peso somado ao custo do enlace (i,j) ao conjunto de pais alternativos (passos 19 e 20). Inevitavelmente j torna-se o nó pai ou passa a

fazer parte do conjunto de pais alternativos. Por este motivo, o nó receptor retira j do conjunto de pais alternativos, caso j já faça parte do mesmo, assim que recebe a mensagem de configuração (passos 8 e 9). As medidas de complexidade não podem ser fornecidas, uma vez que o algoritmo MBF é assíncrono [Yilmaz et al., 2012].

Algorithm 7 Modified Distributed Bellman-Ford - Initiator Node

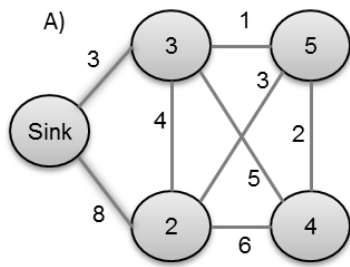
- 1: $parent_{sink} = sink; W_{sink} = 0$
 - 2: broadcast $\langle sink, W_{sink} \rangle$ message
-

Algorithm 8 Modified Distributed Bellman-Ford - Non Initiator Node

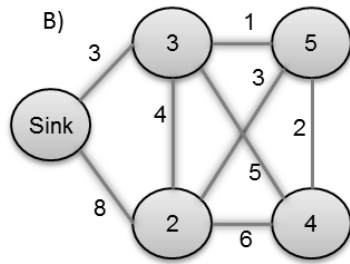
- 1: $parent_i = null; W_i = \infty; AP_i = \emptyset$
 - 2: **while** node i receives $\langle j, W_j \rangle$ message from node j
 - 3: **if** $parent_i = null$ **then**
 - 4: $parent_i = j$
 - 5: $W_i = W_j + cost_{i,j}$
 - 6: broadcast $\langle i, W_i \rangle$ message
 - 7: **else**
 - 8: **if** $(\exists(j, x) \mid x \in \mathfrak{R} \wedge (j, x) \in AP_i)$ **then**
 - 9: $AP_i = AP_i \setminus \{(j, x)\}$
 - 10: **end if**
 - 11: **if** $(W_i - (W_j + cost_{i,j})) / W_j \geq \alpha$ **then**
 - 12: **if** $parent_i \neq j$ **then**
 - 13: $AP_i = AP_i \cup \{(parent_i, W_i)\}$
 - 14: $parent_i = j$
 - 15: **end if**
 - 16: $W_i = W_j + cost_{i,j}$
 - 17: broadcast $\langle i, W_i \rangle$ message
 - 18: **end if**
 - 19: **if** $parent_i \neq j$ **then**
 - 20: $AP_i = AP_i \cup \{(j, W_j + cost_{i,j})\}$
 - 21: **end if**
 - 22: **end if**
 - 23: **end while**
-

4.2 Considerações Finais

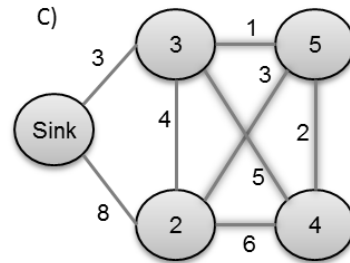
Este capítulo apresentou o algoritmo Modified Bellman-Ford e mostrou de que maneira o algoritmo se propõe a cumprir seus objetivos a fim de fornecer uma solução apropriada para construção de árvores de roteamento em RSSF. Foi ainda detalhado e demonstrado o funcionamento do MBF e as estratégias utilizadas para diminuir o número de mensagens e o tempo necessário para construção da árvore. Além disso, o algoritmo proposto apresenta a propriedade de ser resiliente em relação a perda de mensagens e possibilita o uso de pais alternativos para o balanceamento de carga durante a operação da RSSF.



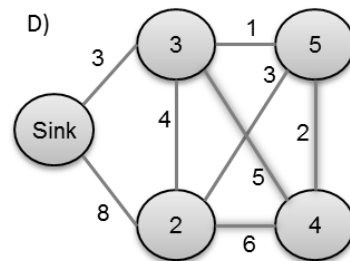
Nó	Pai	Peso	Oferta	Vantagem	Broadcast
2	null	∞	8	1ª oferta	sim
3	null	∞	3	1ª oferta	sim
4	null	∞	-	-	-
5	null	∞	-	-	-



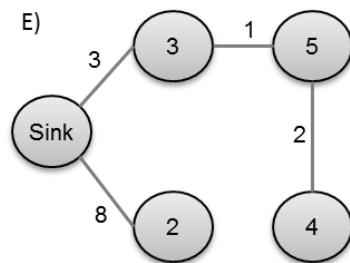
Nó	Pai	Peso	Oferta	Vantagem	Broadcast
2	sink	8	7	0,125	não
3	sink	3	12	-3	não
4	null	∞	8-14	1ª oferta	sim
5	null	∞	4-11	1ª oferta	sim



Nó	Pai	Peso	Oferta	Vantagem	Broadcast
2	sink	8	7 14	0,125 -0,75	não
3	sink	3	13 5	-3,33 -0,66	não
4	3	8	6	0,25	sim
5	3	4	10	-1,5	não



Nó	Pai	Peso	Oferta	Vantagem	Broadcast
2	sink	8	12	-0,5	não
3	sink	3	11	-2,66	não
4	5	6	-	-	-
5	3	4	8	-1	não



Nó	Pai	Peso	Oferta	Vantagem	Broadcast
2	sink	8	-	-	-
3	sink	3	-	-	-
4	5	6	-	-	-
5	3	4	-	-	-

Figura 4.1: Funcionamento do algoritmo MBF em uma rede com 5 nós e $\alpha = 0.2$

Capítulo 5

Resultados

Este capítulo descreve as simulações realizadas para a validação e avaliação de desempenho do algoritmo proposto *Modified Bellman-Ford*. São apresentados o ambiente de simulação, os cenários de rede avaliados e os resultados de desempenho do algoritmo. As simulações estão divididas em duas etapas. A primeira etapa tem por objetivo determinar empiricamente o fator α ideal, capaz de minimizar o número de mensagens e tempo de execução para construção da árvore de roteamento, mantendo a qualidade da estrutura. A segunda etapa objetiva comparar o desempenho dos algoritmos DBF e MBF e a qualidade das árvores construídas.

5.1 Modelagem da Rede e Ambiente de Simulação

Para realizar as simulações, utilizamos o *software* Mathematica [Wolfram Research, 2012] juntamente com a biblioteca SensorSim desenvolvida por Edgard Jamhour [Jamhour, 2011]. A biblioteca SensorSim utiliza os parâmetros de transmissão do protocolo Zigbee padrão IEEE 802.15.4. Do ponto de vista da arquitetura da RSSF, o simulador e os algoritmos implementados consideram as seguintes premissas:

1. Cada nó possui uma identificação distinta;
2. Os enlaces entre os nós são simétricos. Deste modo, se existe um enlace de i para j , existe um enlace reverso de j para i ;
3. Os nós não conhecem sua posição geográfica;
4. Todos os nós são iguais em termos de capacidade de processamento, rádio, bateria e memória, com exceção do nó *sink*;
5. Assume-se que o nó *sink* possui capacidade ilimitada de processamento e memória, além de não possuir restrições quanto ao consumo de energia;

6. O protocolo CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*) ideal [Duarte-Melo and Liu, 2003] é utilizado para acesso ao meio. Existe a competição pelo acesso ao meio, no entanto não existem colisões.

5.1.1 Procedimentos para Geração das Topologias

Com base nas premissas consideradas pelo simulador e algoritmos implementados, a rede é modelada como um grafo $G = (V, A)$ conectado, sendo que V e A representam respectivamente os nós sensores e o conjunto de enlaces. Existe um enlace (i, j) se a distância entre os nós i e j é menor ou igual ao raio de transmissão de ambos os nós i e j , ou seja:

$$(i, j) \Leftrightarrow dist_{i,j} \leq r \quad (5.1)$$

Sendo que $dist_{i,j}$ = distância entre os nós i e j e r = raio de transmissão dos nós i e j .

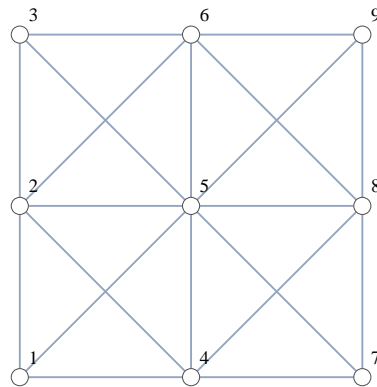


Figura 5.1: Topologia de Rede em Grade sem Distúrbio Aleatório na Posição dos Nós e $N = 9$

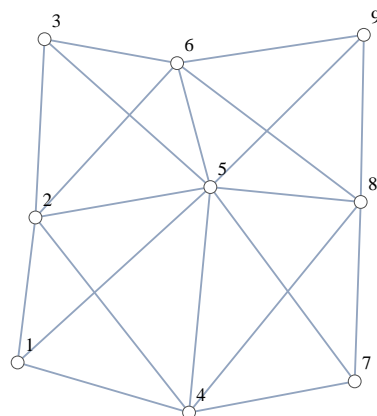


Figura 5.2: Topologia de Rede em Grade com Distúrbio Aleatório na Posição dos Nós e $N = 9$

Para os cenários de simulação, utilizamos topologias de rede em grade (*grid*), com distúrbio aleatório na posição dos nós. Assume-se que o nó *sink* é sempre o nó 1. As topologias são constituídas por N nós e o espaçamento de cada nó i para seu vizinho j , na horizontal e na vertical é definido pelo parâmetro d . O valor do parâmetro d é dependente de N e do parâmetro k , que representa o grau médio de conectividade alvo. A Tabela 5.1 exibe os valores assumidos por d , dados N e k . As Figuras 5.1 e 5.2 exibem respectivamente topologias de rede em grade sem e com distúrbio na posição dos nós.

N	k						
	4	5	6	7	8	9	10
50	220	190	180	170	155	145	138
100	220	200	185	169	162	151	145
150	225	200	190	174	165	155	145
200	225	210	190	175	165	155	148
250	225	210	190	175	165	157	148
300	225	205	190	177	167	157	148

Tabela 5.1: Valor Aproximado do Parâmetro d (metros) dados N e k

5.1.2 Métrica de Custo do Algoritmo Proposto

A distância entre os nós foi a métrica escolhida para computar o custo dos enlaces por ser um dos métodos mais comumente utilizados para retransmitir mensagens em uma ampla variedade de redes [Yilmaz et al., 2012]. No entanto, qualquer outra métrica pode ser utilizada. As principais métricas descritas na literatura são comentadas na seção 3.2.

Para estimar a distância até o nó emissor, o nó receptor pode utilizar o RSSI (*Received Strength Signal Indicator*) [Indicador de Potência do Sinal Recebido] [Trevisan et al., 2013]. O RSSI representa a intensidade do sinal recebido. Os nós sensores possuem circuitos integrados com capacidade de coletar esta informação. A Potência recebida (em dBm) a uma distância $d_{i,j}$ é dada pela seguinte Fórmula:

$$P_r = P_t + G_t + G_r - PL_{d_{i,j}} \quad (5.2)$$

Sendo que:

P_r = Potência recebida em dBm.

P_t = Potência de transmissão em dBm.

G_t = Ganho da antena para transmissão em dB.

G_r = Ganho da antena para recepção em dB.

$PL_{d_{i,j}}$ = Perda média do sinal ao percorrer o trajeto $d_{i,j}$ em dB.

Através do modelo de propagação Log-Distância também é possível estimar $PL_{d_{i,j}}$ [Goldsmith, 2005]:

$$PL_{d_{i,j}} = PL_{d_0} + 10 \cdot \eta \cdot \log \frac{d_{i,j}}{d_0} + \chi \quad (5.3)$$

Sendo que:

PL_{d_0} = Perda média do sinal ao percorrer o trajeto até uma distância de referência d_0 em dB.

η = Expoente de perda de percurso;

χ = Variável aleatória chamada de sombreamento. Tem distribuição normal em dB com média zero e variância σ^2 constante com a distância.

Como o valor de $PL_{d_{i,j}}$ pode ser estimado através da Equação 5.2, é possível isolar $d_{i,j}$ na Equação 5.3 e estimar a distância entre i e j :

$$d_{i,j} = 10^{\frac{(PL_{d_{i,j}} - PL_{d_0} - \chi)}{10 \cdot \eta}} * d_0 \quad (5.4)$$

5.1.3 Métricas para Avaliação dos Algoritmos

Durante a primeira etapa das simulações, cujo objetivo é determinar o fator α ideal, os parâmetros α , N e k assumiram respectivamente os seguintes valores: $\alpha \in \{0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.2, 0.4\}$, $N = 100$ e $k \in \{4, 6, 8, 10\}$. Durante a segunda etapa das simulações, cujo objetivo é avaliar os algoritmos DBF e MBF e a qualidade das árvores construídas, os parâmetros assumiram os seguintes valores: $\alpha = 0.1$, $N \in \{50, 100, 150, 200, 250, 300\}$ e $k = 8$.

Tanto durante a 1ª etapa quanto durante a 2ª, as seguintes métricas foram utilizadas:

1. Número de mensagens necessárias para construção da árvore;
2. Tempo de execução do algoritmo;
3. Distância média até o sink;
4. Número médio de saltos até o sink;

As duas primeiras métricas (número de mensagens e tempo de execução) visam avaliar o desempenho dos algoritmos. As duas últimas (distância e número de saltos) tem por objetivo avaliar a qualidade da árvore. Quanto menor a distância e número de saltos dos nós sensores até o *sink*, maior a qualidade da árvore porque distância e número de saltos até o *sink* impactam no consumo energético e atraso fim-a-fim durante a operação da rede.

Também investigamos o número de pais alternativos em função do parâmetro α e do grau de conectividade da rede, $k \in \{4, 6, 8, 10\}$.

A Tabela 5.2 exibe um resumo dos parâmetros utilizados durante as simulações.

Parâmetro	1ª Etapa	2ª Etapa
N	100	(50-300)
k	(4, 6, 8, 10)	8
MAC	CSMA\CA	CSMA\CA
Faixa de transmissão	295 m	295 m
Faixa de interferência	887 m	887 m

Tabela 5.2: Parâmetros das Simulações

5.2 Primeira Etapa - Determinação Empírica de Alfa

A primeira etapa das simulações realizadas objetivou definir o valor do parâmetro α a ser utilizado pelo algoritmo MBF durante as comparações com o algoritmo DBF. Quando $\alpha = 0.0$ o algoritmo MBF equivale ao DBF.

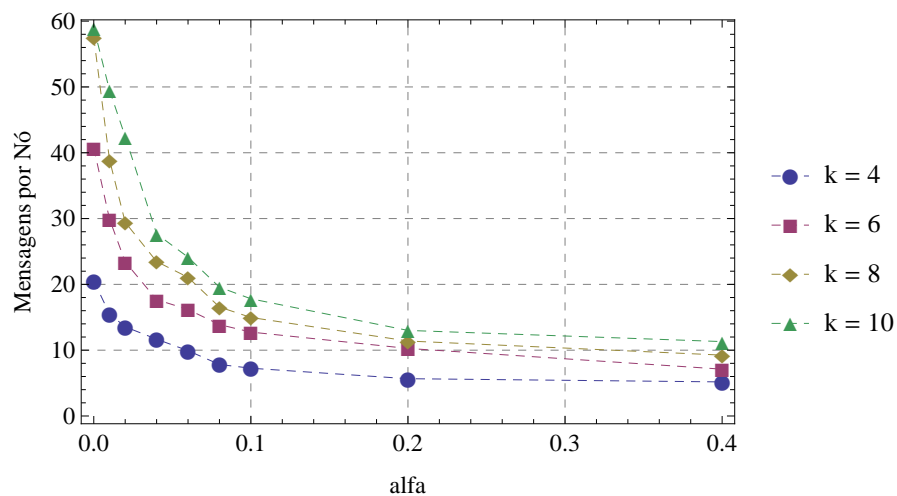


Figura 5.3: Número Médio de Mensagens por Nó

A Figura 5.3 exibe o número médio de mensagens (enviadas e recebidas) por nó durante a execução do algoritmo MBF. A redução no número de mensagens é bastante significativa até $\alpha = 0.1$. A partir deste ponto, o número médio de mensagens tende a estabilizar-se. Nota-se também que quanto mais alto o grau médio de conectividade dos nós, maior é a redução do número de mensagens. Isto ocorre porque o desempenho do algoritmo DBF piora a medida que a rede torna-se mais densa.

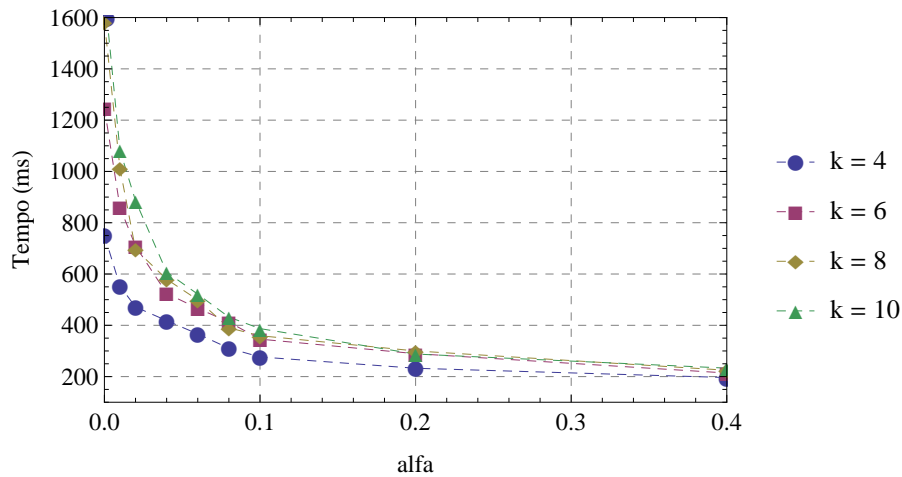


Figura 5.4: Tempo de Execução

A Figura 5.4 exibe o tempo gasto pelo algoritmo para a construção da árvore. Novamente existe uma expressiva redução no tempo de execução até $\alpha = 0.1$. Embora esta redução seja maior a medida que o grau médio de conectividade aumenta, a partir do ponto 0.1 o tempo de execução é praticamente o mesmo para todos os graus de conectividade avaliados.

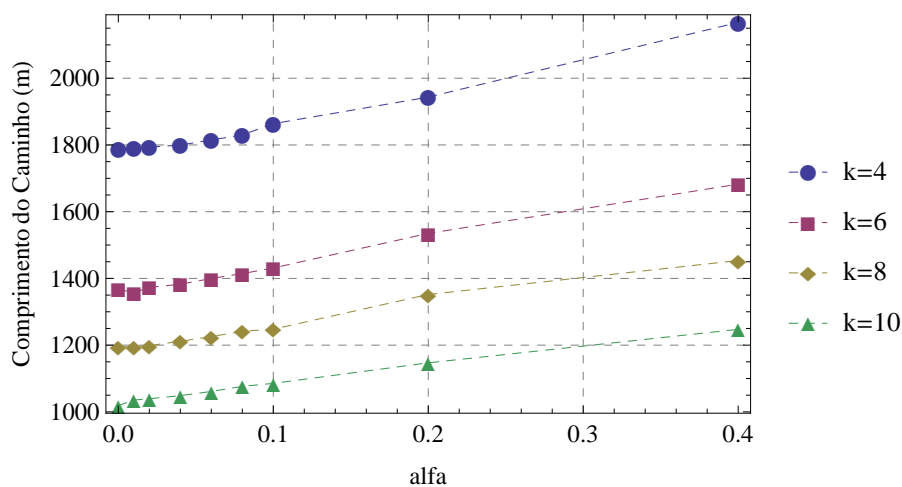


Figura 5.5: Distância Média até o *sink*

A Figura 5.5 apresenta a métrica de distância média (em metros) dos nós até o *sink*. Para valores de *alfa* até 0.1, podemos observar um pequeno aumento nesta métrica. Contudo, a partir de $\alpha = 0.1$ este aumento torna-se mais expressivo. O gráfico indica que o crescimento do tamanho do caminho em função de *alfa* é aproximadamente linear, o que sugere que devemos usar o menor valor possível de *alfa*. Podemos observar também que quanto menor o grau médio de conectividade da rede, maior a distância média até o *sink*. Isto ocorre porque quanto menor

o grau de conectividade, menor também é o número de caminhos possíveis. Por este mesmo motivo, quanto menor o grau de conectividade mais acentuado o aumento na distância a medida que o valor atribuído a *alfa* aumenta.

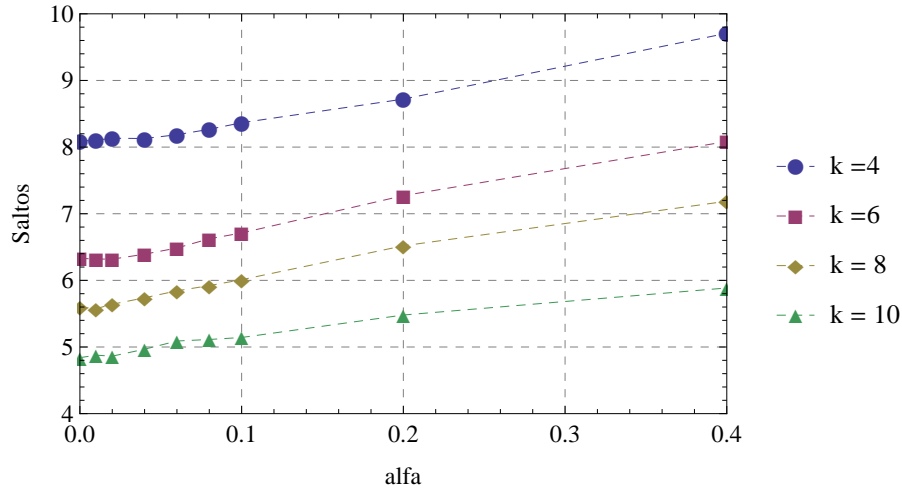


Figura 5.6: Número Médio de Saltos até o Sink

A Figura 5.6 demonstra que o fator *alfa* causa pouco impacto no número médio de saltos dos nós sensores até o sink. Até $\alpha = 0.1$ a diferença não ultrapassa um salto de distância, não importando o grau de conectividade da rede, e chega no máximo a dois saltos quando $\alpha = 0.4$. O gráfico indica que o aumento no número de saltos é aproximadamente linear, o que sugere que devemos usar o menor valor possível de *alfa*.

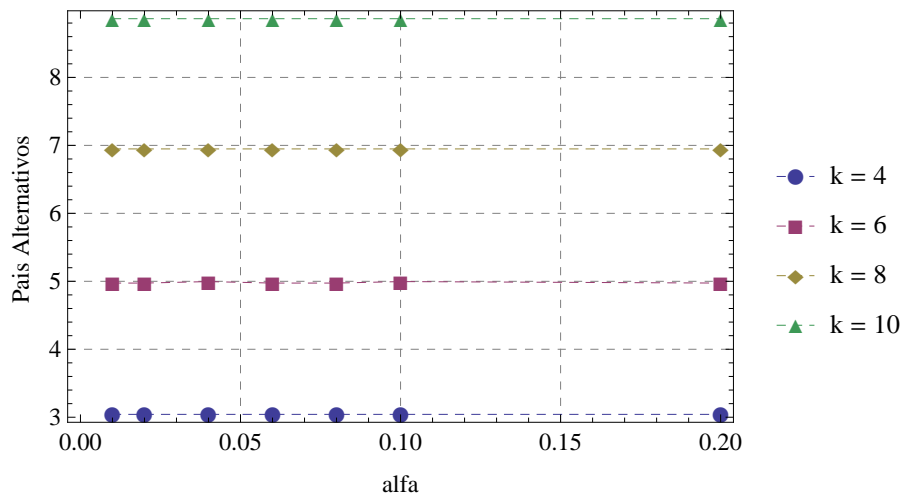


Figura 5.7: Número de Pais Alternativos

A Figura 5.7 mostra que o fator *alfa* não exerce influência sobre o número de pais alternativos. Entretanto, a métrica avaliada é influenciada pelo grau de conectividade da rede. Isto

deve-se ao próprio funcionamento do algoritmo. O nó irá inevitavelmente receber a mensagem de configuração de todos os seus vizinhos. Portanto, quanto mais densa a rede, maior o número de pais alternativos. O fator *alfa* influencia somente no número de mensagens subsequentes, e portanto na distância dos pais alternativos até o sink.

Considerando-se os resultados obtidos durante as simulações, o valor escolhido para α foi 0.1. O parâmetro α deve ser capaz de reduzir os *overheads* de comunicação e tempo do algoritmo DBF sem impactar na qualidade da árvore e a escolha do valor 0.1 é justificada pelas seguintes observações:

1. A maior redução no número de mensagens ocorre até $\alpha = 0.1$, sendo que a partir deste ponto tende a estabilizar-se;
2. A redução mais expressiva no tempo de execução do algoritmo MBF também ocorre até $\alpha = 0.1$;
3. O aumento na distância e no número de saltos dos nós até o sink acentua-se a partir de $\alpha = 0.1$.

5.3 Segunda Etapa - Avaliação do Algoritmo Proposto

A segunda etapa das simulações visou comparar o desempenho do algoritmo MBF com relação ao desempenho e a qualidade da árvore construída pelo algoritmo DBF.

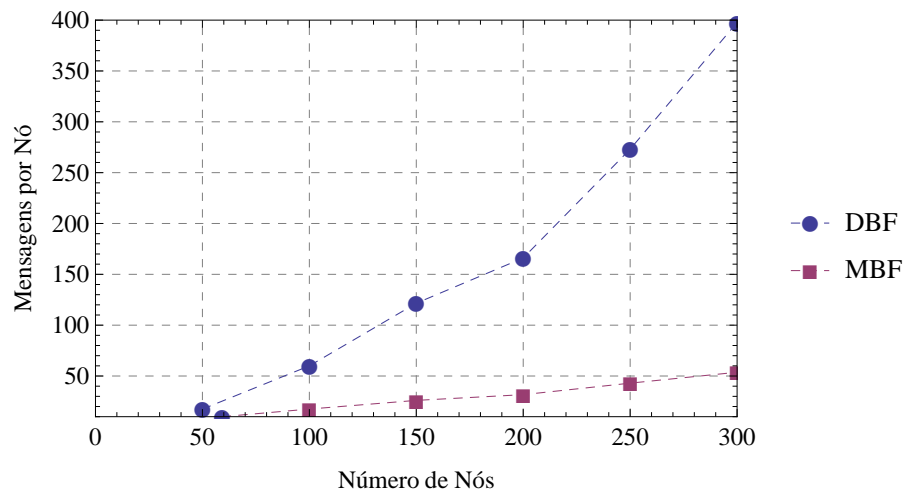


Figura 5.8: Número Médio de Mensagens por Nó

A Figura 5.8 demonstra que o algoritmo MBF cumpre o objetivo de reduzir o alto *overhead* de comunicação do algoritmo DBF. Quanto maior o número de nós, mais significativa é a diferença entre o número de mensagens requeridas pelos dois algoritmos. Em um

cenário com 50 nós, o MBF requer em média 10 mensagens por nó, enquanto o DBF requer aproximadamente 20 mensagens por nó o que representa uma redução de 50% usando a estratégia proposta. Em um cenário com 300 nós, o algoritmo DBF exige em média 400 mensagens por nó, enquanto o algoritmo MBF exige em média 50. Portanto, uma redução de 87,5%. Isto significa que quanto maior o número de nós, maior é a proporção de rotas ofertadas cuja vantagem com relação ao custo da rota atual é menor que *alfa*. Logo, quanto maior o número de nós, melhor é o desempenho do MBF com relação ao DBF. Comparando-se ainda os cenários com 50 e 300 nós, o número de mensagens aumenta 5 vezes para o MBF e 20 vezes para o DBF. Além de reduzir o *overhead* de comunicação, o MBF demonstra também ser uma solução mais escalável.

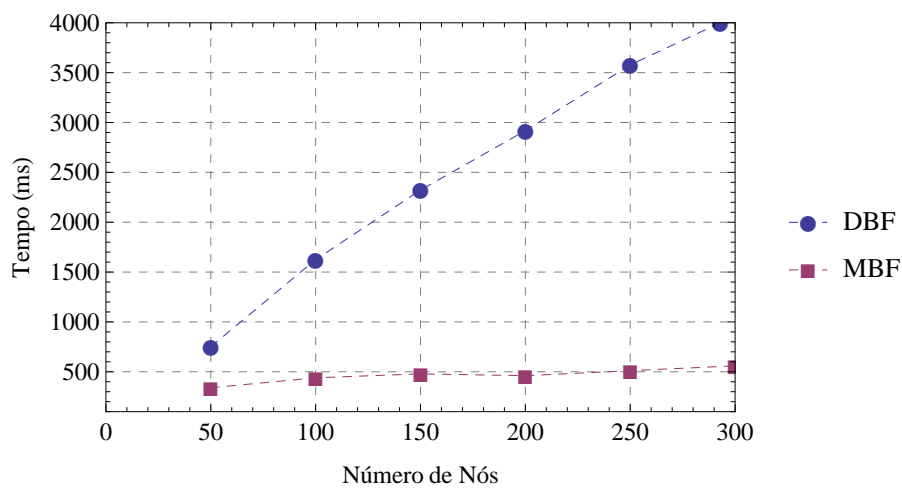


Figura 5.9: Tempo de Execução

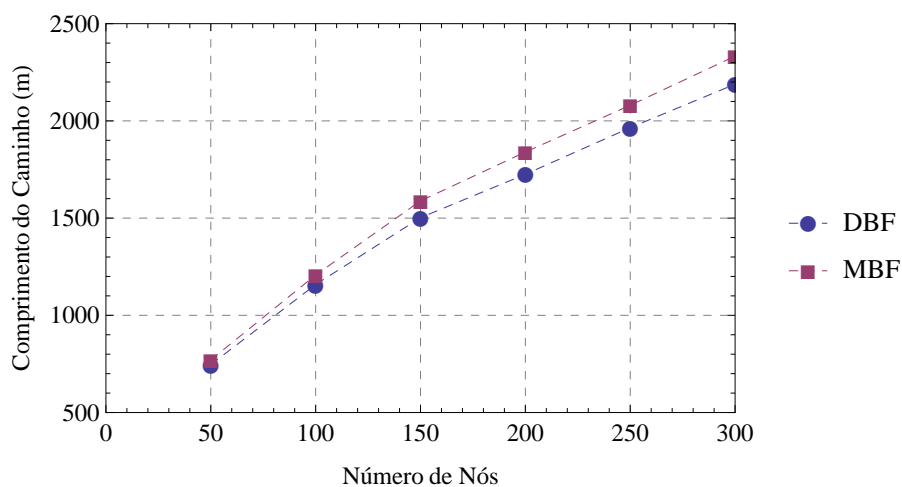


Figura 5.10: Distância Média até o *sink*

A Figura 5.9 mostra que o tempo de execução do algoritmo MBF é significativamente menor que o tempo de execução do DBF. Em um cenário com 300 nós, esta diferença é de 87,5%. Comparando-se os cenários com 50 e 300 nós, o aumento no tempo de execução do MBF é de aproximadamente 37,5% enquanto para o DBF este aumento é de aproximadamente 434%. Comprova-se portanto que o MBF cumpre outro de seus objetivos: baixo *overhead* de tempo. A Figura 5.10 demonstra que as significativas reduções no número de mensagens e tempo de execução causam pouco impacto na qualidade da árvore construída pelo algoritmo MBF. A distância média dos nós até o sink é aproximadamente a mesma para os dois algoritmos avaliados. Nas topologias envolvendo 300 nós esta diferença não ultrapassa 7%.

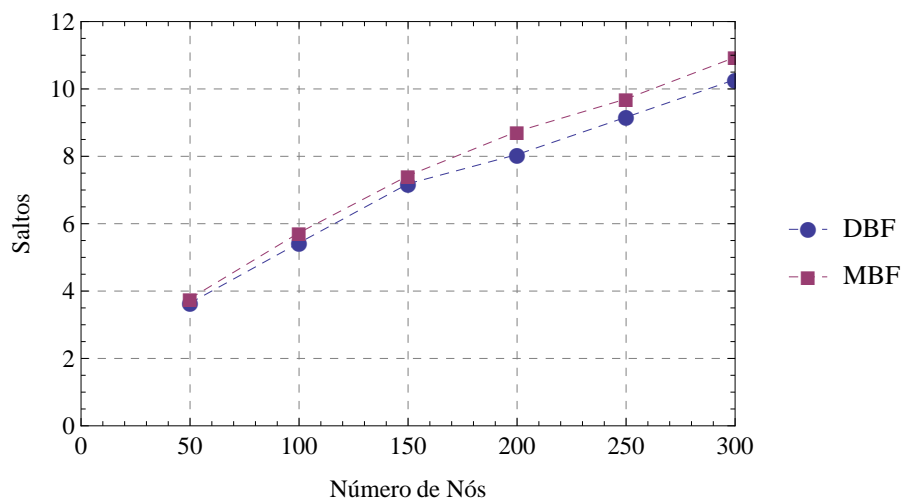


Figura 5.11: Número Médio de Saltos até o Sink

A Figura 5.11 confirma que o algoritmo MBF é capaz de garantir a qualidade da árvore construída. Assim como a distância média dos nós até o sink, o número médio de saltos até o sink é aproximadamente o mesmo. Em nenhum dos cenários avaliados, a diferença entre os caminhos construídos pelo MBF e pelo DBF chega a um salto de distância.

5.4 Considerações Finais

Este capítulo apresentou as simulações, cenários, métricas utilizadas e por fim os resultados que convalidam o algoritmo MBF. Primeiramente, justificou-se o valor escolhido para *alfa*. Em seguida, foram apresentados e discutidos os testes comparativos entre os algoritmos DBF e MBF. Os resultados demonstraram claramente que o MBF cumpre seus objetivos, possui desempenho superior ao algoritmo DBF e constrói árvores cuja qualidade são semelhantes à qualidade das árvores construídas pelo DBF. Duas métricas foram escolhidas para avaliar a qualidade das árvores: distância média dos nós até o *sink* e número médio de saltos dos nós

até o *sink*. A diferença no número de saltos não alcançou 1 salto de distância em nenhum dos cenários avaliados. Além disso, ficou demonstrado que a distância dos nós até o *sink* é aproximadamente a mesma para os dois algoritmos. Como o DBF garante a SPT, o comprimento dos caminhos dos nós até o *sink* é ótimo. Portanto, pode-se afirmar que o comprimento dos caminhos construídos pelo MBF é subótimo. A diferença no comprimento dos caminhos construídos pelos dois algoritmos avaliados não excedeu 7%. No entanto, a diferença no número de mensagens necessárias para a construção das árvores chegou a 87,5%, mesma diferença computada para o tempo de execução dos algoritmos. O MBF provou também ser uma solução mais escalável, uma vez que quanto maior o número de nós melhor é seu desempenho com relação ao DBF. As simulações e resultados comprovam que de fato, o algoritmo proposto é viável e adequado para construção de árvores de roteamento em RSSF.

Capítulo 6

Conclusão

O objetivo deste trabalho, conforme apresentado na Seção 1.2, foi alcançado com a proposição do algoritmo Modified Bellman-Ford. As simulações realizadas e os resultados obtidos e apresentados no Capítulo 5 confirmam que o algoritmo proposto é uma solução viável e eficiente para construção de árvores de roteamento em RSSF e que além disso é capaz de garantir a qualidade da estrutura construída. O objetivo específico de implementar e avaliar o desempenho de algoritmos distribuídos para construção de *spanning trees* em RSSF propostos na literatura também foi alcançado e os resultados obtidos foram apresentados na Seção 3.1.4. A implementação e avaliação de desempenho destes algoritmos auxiliou na identificação das vantagens e dificuldades na utilização dos mesmos em cenários reais e evidenciou a necessidade de uma solução para construção de árvores de roteamento apropriada para RSSF. A principal dificuldade identificada foi o fato de que alguns algoritmos não são capazes de construir a estrutura de roteamento caso ocorram perdas de mensagens, um evento comum em redes *wireless*. Outras dificuldades encontradas foram o *alto overhead* de comunicação e a exigência de que os nós conheçam a topologia da rede. Os nós sensores possuem recursos computacionais escassos e esta limitação não deve ser desconsiderada. O *alto overhead* de tempo também foi uma das dificuldades identificadas, pois algumas aplicações de monitoramento exigem uma rápida reconstrução da estrutura de roteamento em caso de falhas. Dentre as principais vantagens identificadas, estão a simplicidade e a capacidade de construir a árvore de roteamento mesmo que ocorram perdas de mensagens do algoritmo Bellman-Ford e a resiliência do algoritmo SHM.

O algoritmo MBF é uma solução eficiente e viável para construção de árvores de roteamento em RSSF porque:

1. É adequado para redes *wireless*, pois é capaz de construir a árvore de roteamento mesmo que ocorram perdas de mensagens;
2. É uma solução simples que não demanda uma grande quantidade de recursos computacionais e não requer que os nós sensores conheçam a topologia da rede;

3. Possui baixo *overhead* de tempo e de comunicação, como ficou demonstrado através das simulações realizadas;
4. É uma solução escalável, como também ficou demonstrado através das simulações e resultados obtidos. Esta é uma propriedade importante porque as RSSF podem ser formadas por um grande número de nós;
5. É resiliente, pois assim como o algoritmo SHM, o algoritmo MBF provê múltiplas rotas aos nós para serem utilizadas como mecanismos de tolerância a falhas e balanceamento de carga;
6. É flexível com relação à métrica utilizada para a construção da árvore de roteamento;
7. As simulações realizadas demonstraram que as vantagens apresentadas pelo MBF não causam impacto na qualidade da árvore construída.

6.1 Trabalhos Futuros

As seguintes questões foram identificadas para a continuação deste trabalho:

1. Inclusão de um modelo de consumo de energia;
2. Avaliação do tempo de vida da rede;
3. Utilização de um modelo de canal baseado em probabilidade de outage;
4. Proposição de um protocolo para reorganização da árvore;
5. Proposição de novas métricas para a construção da árvore;

Referências Bibliográficas

- [Akyildiz et al., 2002] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: A survey. *Computer Networks*, 38(4):pg 393–422.
- [Amdouni et al., 2012] Amdouni, I., Soua, R., Livolant, E., and Minet, P. (2012). Delay optimized time slot assignment for data gathering applications in wireless sensor networks. In *Wireless Communications in Unusual and Confined Areas (ICWCUCA), 2012 International Conference on*, pages 1–6.
- [Anastasi et al., 2004] Anastasi, G., Falchi, A., Passarella, A., Conti, M., and Gregori, E. (2004). Performance measurements of motes sensor networks. In Balsamo, S., Chiasserini, C.-F., and Donatiello, L., editors, *MSWiM*, pages 174–181. ACM.
- [Aziz et al., 2013] Aziz, A., Sekercioglu, Y., Fitzpatrick, P., and Ivanovich, M. (2013). A survey on distributed topology control techniques for extending the lifetime of battery powered wireless sensor networks. *Communications Surveys Tutorials, IEEE*, 15(1):121–144.
- [Bechkit et al., 2012] Bechkit, W., Koudil, M., Challal, Y., Bouabdallah, A., Souici, B., and Benatchba, K. (2012). A new weighted shortest path tree for convergecast traffic routing in wsn. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000187–000192.
- [Bertsekas and Gallager, 1992] Bertsekas, D. and Gallager, R. (1992). *Data Networks (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- [Blough et al., 2006] Blough, D., Leoncini, M., Resta, G., and Santi, P. (2006). The k-neighbors approach to interference bounded and symmetric topology control in ad hoc networks. *Mobile Computing, IEEE Transactions on*, 5(9):1267–1282.
- [Chang and Tassiulas, 2004] Chang, J.-H. and Tassiulas, L. (2004). Maximum lifetime routing in wireless sensor networks. *Networking, IEEE/ACM Transactions on*, 12(4):609–619.
- [Chen et al., 2010] Chen, T.-S., Tsai, H.-W., and Chu, C.-P. (2010). Adjustable convergecast tree protocol for wireless sensor networks. *Computer Communications*, 33(5):559–570.

- [Coleri and Varaiya, 2004] Coleri, S. and Varaiya, P. (2004). Fault tolerant and energy efficient routing for sensor networks. In *Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE*, pages 10–15.
- [Cormen, 2001] Cormen, T. (2001). *Introduction to algorithms*. The MIT press.
- [Dargie, 2012] Dargie, W. (2012). Dynamic power management in wireless sensor networks: State-of-the-art. *Sensors Journal, IEEE*, 12(5):1518–1528.
- [Duarte-Melo and Liu, 2003] Duarte-Melo, E. J. and Liu, M. (2003). Data-gathering wireless sensor networks: Organization and capacity. *Computer Networks*, 43:519–537.
- [Elhabyan and Yagoub, 2013] Elhabyan, R. and Yagoub, M. (2013). Weighted tree based routing and clustering protocol for wsn. In *Electrical and Computer Engineering (CCECE), 2013 26th Annual IEEE Canadian Conference on*, pages 1–6.
- [Ergen and Varaiya, 2010] Ergen, S. C. and Varaiya, P. (2010). Tdma scheduling algorithms for wireless sensor networks. *Wireless Networks*, 16(4):985–997.
- [Gandham et al., 2006] Gandham, S., Zhang, Y., and Huang, Q. (2006). Distributed minimal time convergecast scheduling in wireless sensor networks. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 50–50.
- [Goldsmith, 2005] Goldsmith, A. (2005). *Wireless Communications*. Cambridge University Press, New York, NY, USA.
- [Heinzelman et al., 2000] Heinzelman, W., Chandrakasan, A., and Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, pages 10 pp. vol.2–.
- [Hill et al., 2000] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. E., and Pister, K. S. J. (2000). System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104.
- [Holger and Willig., 2005] Holger, K. and Willig., A. (2005). *Protocols and architectures for wireless sensors networks*. John Wiley & Sons.
- [Incel et al., 2012] Incel, O., Ghosh, A., Krishnamachari, B., and Chintalapudi, K. (2012). Fast data collection in tree-based wireless sensor networks. *Mobile Computing, IEEE Transactions on*, 11(1):86–99.

- [Jamhour, 2011] Jamhour, E. (2011). A symbolic model to traffic engineering in wireless mesh networks. In *Proceedings of the 44th Annual Simulation Symposium, ANSS '11*, pages 32–38, San Diego, CA, USA. Society for Computer Simulation International.
- [Lin et al., 2006] Lin, C., He, Y.-X., and Xiong, N. (2006). An energy-efficient dynamic power management in wireless sensor networks. In *Parallel and Distributed Computing, 2006. ISPPDC '06. The Fifth International Symposium on*, pages 148–154.
- [Lynch, 1996] Lynch, N. A. (1996). *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [Makki and Havas, 1994] Makki, S. A. M. and Havas, G. (1994). Distributed algorithms for constructing a depth-first-search tree. In *ICPP (3)*, pages 270–273.
- [Pan et al., 2013] Pan, M.-S., Liu, P.-L., and Cheng, C.-F. (2013). Convergecast in zigbee tree-based wireless sensor networks. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 1470–1475.
- [Polastre et al., 2013] Polastre, J., Levis, P., Szewczyk, R., and Culler, D. (2013). Tinyos hardware.
- [Pottie and Kaiser, 2000] Pottie, G. J. and Kaiser, W. J. (2000). Embedding the Internet: wireless integrated network sensors. *Communications of the ACM*, 43(5):51–51.
- [Reif, 1985] Reif, J. H. (1985). Depth-first search is inherently sequential. *Inf. Process. Lett.*, 20(5):229–234.
- [Saadat and Mirjalily, 2012] Saadat, M. and Mirjalily, G. (2012). Efficient convergecast tree for data collection in wireless sensor networks. In *Electrical Engineering (ICEE), 2012 20th Iranian Conference on*, pages 1534–1539.
- [Santi, 2005] Santi, P. (2005). Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2):164–194.
- [Sheikh et al., 2012] Sheikh, M., Driberg, M., and Ali, N. (2012). An improved distributed scheduling algorithm for wireless sensor networks. In *Intelligent and Advanced Systems (ICIAS), 2012 4th International Conference on*, volume 1, pages 274–279.
- [Shrivastava and Pokle, 2014] Shrivastava, P. and Pokle, S. (2014). Energy efficient scheduling strategy for data collection in wireless sensor networks. In *Electronic Systems, Signal Processing and Computing Technologies (ICESC), 2014 International Conference on*, pages 170–173.

- [Sinha and Chandrakasan, 2001] Sinha, A. and Chandrakasan, A. (2001). Dynamic power management in wireless sensor networks. *Design Test of Computers, IEEE*, 18(2):62–74.
- [Tan and Korpeoglu, 2003] Tan, H. z. and Korpeoglu, I. (2003). Power efficient data gathering and aggregation in wireless sensor networks. *SIGMOD Record*, 32(4):66–71.
- [Trevisan et al., 2013] Trevisan, L., Pellenz, M., Penna, M., Souza, R., and Fonseca, M. (2013). A simple iterative positioning algorithm for client node localization in wlans. *EURASIP Journal on Wireless Communications and Networking*, 2013(1):1–11.
- [Tsai and Chen, 2008] Tsai, H.-W. and Chen, T.-S. (2008). Minimal time and conflict-free schedule for convergecast in wireless sensor networks. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 2808–2812.
- [Wawryszczuk and Amanowicz, 2012] Wawryszczuk, M. and Amanowicz, M. (2012). An energy effective method for topology control in wireless sensor networks. In *Microwave Radar and Wireless Communications (MIKON), 2012 19th International Conference on*, volume 2, pages 647–651.
- [Wolfram Research, 2012] Wolfram Research, I. (2012). *Mathematica*. Wolfram Research, Inc.
- [Yilmaz et al., 2012] Yilmaz, O., Demirci, S., Kaymak, Y., Ergun, S., and Yildirim, A. (2012). Shortest hop multipath algorithm for wireless sensor networks. *Computers & Mathematics with Applications*, 63(1):48 – 59.
- [Zibakalam, 2012] Zibakalam, V. (2012). A new tdma scheduling algorithm for data collection over tree-based routing in wireless sensor networks. *ISRN Sensor Networks*, 2012.