DANIEL FERNANDES MACEDO

UM PROTOCOLO DE ROTEAMENTO PARA REDES DE SENSORES SEM FIO ADAPTÁVEL POR REGRAS DE APLICAÇÃO

Belo Horizonte
14 de março de 2006

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

UM PROTOCOLO DE ROTEAMENTO PARA REDES DE SENSORES SEM FIO ADAPTÁVEL POR REGRAS DE APLICAÇÃO

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

DANIEL FERNANDES MACEDO

Belo Horizonte
14 de março de 2006

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

Um Protocolo de Roteamento para Redes de Sensores Sem Fio Adaptável por Regras de Aplicação

DANIEL FERNANDES MACEDO

Dissertação defendida e aprovada pela banca examinadora constituída por:

Prof. José Marcos Silva Nogueira – Orientador
Universidade Federal de Minas Gerais

Prof. Antonio Alfredo Ferreira Loureiro – Co-orientador
Universidade Federal de Minas Gerais

Profa. Linnyer Beatrys Ruiz
Universidade Federal de Minas Gerais

Prof. Carlos Frederico M. C. Cavalcanti
Universidade Federal de Ouro Preto

Belo Horizonte, 14 de março de 2006

Resumo

A troca de informação entre camadas da pilha de comunicações para otimizar o desempenho da rede tem se mostrado uma necessidade em redes de sensores sem fio (RSSF). Tal característica se faz necessária para que a rede possua um serviço confiável e eficiente mesmo diante das restrições severas de recursos encontrados nos elementos de rede que compõem uma RSSF. A otimização impõe um grande fardo ao desenvolvedor, que deve conhecer a fundo o funcionamento de cada protocolo, uma vez que uma otimização mal feita pode piorar o desempenho da rede ou mesmo impedir a sua operação.

Esta dissertação apresenta um protocolo de roteamento pró-ativo, denominado PROC (*Proactive ROuting with Coordination*). O PROC é otimizado para RSSF de disseminação contínua de dados, que são redes onde os nós sensores enviam dados para o ponto de acesso em intervalos regulares de tempo. Pelo nosso conhecimento, o PROC é o primeiro protocolo de roteamento para redes de disseminação contínua de dados que provê mecanismos simples de interação com a aplicação tendo em vista a otimização do roteamento. Esta interação é feita pelas *regras de aplicação*, que permitem que o PROC se adapte em tempo de execução a mudanças no ambiente.

As rotas são criadas em um processo de duas fases. O PROC determina na primeira fase quais nós irão repassar dados a partir das informações fornecidas pela aplicação via regras de aplicação. A segunda fase garante que todos os nós da rede possuem rotas. Com isso, o PROC permite que programadores que não conhecem o funcionamento do protocolo possam otimizar o processo de criação de rotas. O protocolo ainda implementa mecanismos simples de tolerância a falhas que aumentam a sua resiliência frente a falhas silenciosas.

Verificamos utilizando simulações que o PROC aumenta em até 12% o tempo de vida da rede em comparação aos protocolos de roteamento EAD e ao TOSB (uma versão simplificada do protocolo TinyOS Beaconing). O PROC também se recupera mais rapidamente de falhas de nós devido ao uso de algoritmos de tolerância a falhas. O PROC é escalável, pois a quantidade de mensagens enviadas por nó requerida para a construção de rotas independe de densidade ou tamanho da rede. Além disso, o protocolo proposto requer pouca memória, pois armazena uma quantidade de informação proporcional ao número de nós que são alcançados diretamente. Por fim, a implementação do protocolo na plataforma Mica2, que é a plataforma mais utilizada na pesquisa em RSSF, encontra-se disponível para *download*.

Abstract

The exchange of information among the layers of the communication stack has been frequently used in Wireless Sensor Networks (WSNs) to increase the performance of the network. Such interaction among layers enhances the reliability and efficiency of the communication, despite the severely restricted resources found on sensor nodes. However, such optimizations impose a huge burden on the developer, as an unsound decision might decrease the overall performance or even hinder the operation of the network.

This dissertation presents a proactive routing protocol, called PROC (*Proactive ROuting with Coordination*). The protocol is optimized to static continuous data dissemination WSNs, that is, every node periodically sends data towards the access point. To our knowledge, PROC is the first protocol devised specifically to continuous dissemination networks which provides simple optimization mechanisms, allowing the applications to effortlessly optimize routing operation to their needs. This interaction takes place using the *application rules*, which allow PROC to adapt in runtime to any change in the environment.

Routes are established in a two-phase process. In the first phase, PROC determines which nodes will forward data, based on the information provided by the application rules. The second phase ensures that all nodes have valid routes. Unexperienced programmers might optimize the routes created by PROC, as no knowledge concerning the operation of the protocol is required. The protocol also includes fault-tolerance mechanisms that increase its resiliency against silent failures.

Simulation results showed that PROC increases the average network lifetime by up to 12% when compared to EAD and TOSB (a simplified version of the TinyOS Beaconing protocol). PROC also quickly recovers from failed nodes due to its fault-tolerance algorithms. PROC is highly scalable, since the amount of route maintenance messages sent by each node is independent of node density or network size. Furthermore, the protocol presents a low memory footprint as nodes only store information concerning one-hop neighbors. Finally, the implementation of PROC in the Mica2 platform, which is the most frequently used platform by researches in the field, is available for download.

Agradecimentos

À minha família, que sempre apoiou este "estudante profissional", mesmo quando eu virei noites e finais de semana trabalhando, algumas vezes lhes ignorando quase que completamente. Da mesma forma, agradeço aos meus amigos pela paciência com esse Damacedo, que parece ter somente olhos para a pesquisa.

Um agradecimento especial ao Aldri e Lula, pelos nossos projetos conjuntos, as revisões nos fins de semana e pelas brincadeiras em cima dos *deadlines*. Mas, como *"Isso não resolve o nosso problema"*, voltemos ao texto. Vocês foram os companheiros que me seguiram em todas as minhas idéias mirabolantes e, como bons amigos, continuaram no barco mesmo quando a água entrava por todos os lados. Não é a toa que também vimos vários belos horizontes à bordo desse barco...

Também gostaria de agradecer aos meus orientadores, José Marcos e Loureiro, que desde a graduação estiveram presentes em todas as fases da minha carreira acadêmica. Leo e Isa, por me ensinarem a fazer pesquisa, e principalmente por me aguentarem por todos esses anos.

Ao Flip, Fabrício, Vitorino, Radicchi e Flop (viram, vocês ficaram entre "parênteses") e também aos aderentes recentes do grupo de estudo de algoritmos: Krusty, Ítalo, Otaviano, Fred e Júnior. Nunca me esquecerei das nossas competições de "quem fazia mais programas em menos tempo". Só reitero que ainda está valendo a idéia de abrir o Posto de Gasolina ACM...

Aos amigos do SIS, ATM e SensorNet, à GRAD001, e a todos os amigos do DCC. É graças a todos vocês que o nosso ambiente de trabalho é tão descontraído e produtivo.

À CAPES e ao CNPq, por me permitirem dedicar exclusivamente ao mestrado.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ACK	Acknowledgement
AM	Active Message
AODV	Ad hoc On demand Distance Vector
BER	Bit Error Rate
B-MAC	Berkeley MAC
CBS	Community-Based Security
CDS	Connected Dominatig Set
CPT	Controle de Potência de Transmissão
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear To Send
DIFS	Distributed Inter Frame Space
DSDV	Destination-Sequenced Distance Vector
DSR	Dynamic Source Routing
EAD	Energy-Aware Distributed routing
EF-Tree	Earlier-First Tree
ExOR	Extremely Opportunistic Routing
GEAR	Geographical and Energy Aware Routing
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing
GRAB	GRAdient Broadcast
IP	Internet Protocol
IDS	Intrusion Detection System

ISO	International Standard Organization
ISM	Industrial, Scientifical and Medical
LEACH	Low Energy Adaptive Clustering Hierarchy
LED	Light Emitting Diode
LGPL	Lesser General Public License
MAC	Medium Access Control
MCDS	Minimum Connected Dominant Subset
MIG	Message Interface Generator
NS-2	Network Simulator Version 2.0
NSF	National Science Foundation
OSI	Open Systems Interconnect
PA	Ponto de Acesso
PDA	Personal Digital Assistent
PEGASIS	Power-Efficient Gathering in Sensor Information Systems
PER	Packet Error Rate
PROC	Proactive ROuting with Coordination
RAM	Random Access Memory
ROM	Read-Only Memory
RSSF	Redes de Sensores Sem Fio
RTS	Request To Send
SID	Source-Initiated Dissemination
SIFS	Short Inter Frame Space
SPIN	Sensor Protocol for Information via Negotiation
TCP	Transmission Control Protocol
TOSB	TinyOS Beaconing
WRP	Wireless Routing Protocol

Capítulo 1

Introdução

Redes de Sensores Sem Fio (RSSF) são uma subclasse das redes ad hoc sem fio. As RSSF são formadas por elementos de rede chamados de nós sensores, que são dispositivos compactos compostos de sensores, processador, rádio, memória e bateria. Esses nós enviam os dados coletados do ambiente para um Ponto de Acesso (PA), que repassa os dados ao usuário final [35]. Os nós sensores têm como objetivo monitorar eventos relevantes que ocorrem no ambiente. Os dados produzidos são enviados ao PA, por meio de comunicação de múltiplos saltos entre enlaces sem fio. As RSSF ainda podem utilizar atuadores, que permitem que os nós interfiram no ambiente [2]. O *hardware* é controlado por um conjunto de *softwares* que em geral incluem um sistema operacional, protocolos de comunicação e o *software* de aplicação. Este último coleta e processa os dados brutos obtidos dos sensores, gerando dados relevantes à aplicação que são enviados ao PA.

As RSSF são utilizadas em aplicações diversas, como monitoração ambiental [3, 71, 80], ambientes inteligentes, monitoração remota de doentes [55], combate a queimadas [9], monitoração de fadiga em edifícios e pontes [47], detecção de intrusão [17], controle de chão de fábrica, agricultura de precisão [38], auxílio a resgate em áreas de catástrofes naturais ou incêndios [49], entre outros. Devido à importância dessas redes e seus inúmeros desafios de pesquisa, as RSSF foram identificadas pelo NSF (*National Science Foundation*, órgão americano de fomento de pesquisa) como um dos tópicos mais importantes de pesquisa da área de redes [60].

O desenvolvimento de redes de sensores sem fio está calcado em três requisitos básicos, que são *desempenho aceitável*, *economia de recursos* e *tolerância a falhas*. Por serem compostas de milhares de nós, o custo unitário dos sensores deve ser baixo para que a rede seja viável. Esta restrição se reflete no *hardware*, que possui capacidade limitada de processamento, memória e banda. Além disso, RSSF são empregadas em ambientes inóspitos, onde a falha de nós é frequente. Apesar destas restrições, o serviço provido pela rede deve atender a requisitos mínimos de qualidade, como segurança, latência baixa e previsível (para sistemas de tempo real), garantia de entrega de mensagens e precisão dos dados coletados.

Diferente das redes ad hoc tradicionais, em RSSF em geral não é possível trocar ou recarregar a fonte de energia dos nós sensores devido à grande quantidade de nós ou às dificuldades impostas pelo ambiente. Por conseguinte, o consumo de energia é um fator crítico no tempo de vida das RSSF. Além disto, as restrições de memória e processamento requerem o uso de soluções eficientes e compactas. Assim, os protocolos e algoritmos utilizados em RSSF procuram minimizar o consumo de recursos.

Devido ao ambiente inóspito, falhas em RSSF são frequentes. Os nós podem ser destruídos a qualquer momento, seja por deslizamento, queda de árvores ou prédios, enchentes ou outros agentes naturais. Falhas também ocorrem na comunicação, devido a interferências causadas por modificações no clima ou na movimentação de objetos no espaço sensoriado, ou mesmo a agentes maliciosos. Desta forma, protocolos e aplicações em RSSF devem prover mecanismos de tolerância a falhas.

Para garantir um serviço confiável mesmo diante das restrições severas de recursos, foram propostas soluções otimizadas para uma classe restrita de aplicações [33, 36, 87]. Assim, os protocolos atendem somente os requisitos de cada cenário, diminuindo o número de funcionalidades implementadas. Com isso, os protocolos são mais simples e eficientes. Tal alternativa é viável em RSSF pois estas redes são empregadas para um propósito específico.

Uma otimização normalmente utilizada em RSSF é o desenvolvimento de protocolos que suportam fluxos particulares de dados. Podemos classificar o fluxo de dados em RSSF em dois grandes grupos [73]. Nas *redes dirigidas a eventos*, o envio de dados é ocasional, ocorrendo somente quando um determinado evento é detectado. Redes dirigidas a evento são utilizadas na localização de animais silvestres, detecção de intrusão, monitoramento de queimadas, entre outros. Nas *redes de disseminação contínua*, os nós enviam mensagens em intervalos constantes para um PA, relatando as leituras atuais dos seus sensores. Temos como exemplo as aplicações de estudos ambientais, sistemas de tráfego inteligente, monitoração de plantas industriais, entre outros. Por possuírem padrões de comunicação distintos, é possível implementar protocolos de roteamento otimizados para cada fluxo, que diminuem ainda mais a complexidade e o consumo de energia dos protocolos.

Outra tendência em redes sem fio, em particular em RSSF, é a forte interação entre protocolos de comunicação [51, 69], pois esta interação permite ganhos consideráveis de desempenho. Como uma rede de sensores sem fio em geral executa somente uma aplicação, os protocolos podem ser otimizados ainda mais, se aproveitando de características intrínsecas de cada cenário ou até mesmo de uma instância particular da rede. A otimização impõe um grande fardo ao desenvolvedor, que deve conhecer a fundo o funcionamento de cada protocolo, uma vez que a otimização excessiva e sem critérios pode piorar o desempenho da rede ou mesmo impedir a sua operação [46].

Uma maneira de minimizar os custos e riscos da otimização dos protocolos é o uso de protocolos que expõem mecanismos de configuração simples, quer permitem modificar o comportamento

dos protocolos e ao mesmo tempo ocultam aspectos internos do algoritmo. Desta forma, o desenvolvedor pode aumentar o desempenho da rede sem conhecer em detalhes cada componente utilizado. O protocolo *Directed Diffusion* é um exemplo de protocolo de roteamento para redes dirigidas e eventos que possui mecanismos simples de configuração. No *Directed Diffusion*, as rotas são construídas utilizando os *interesses*, que descrevem as características da informação procurada [36]. O interesse não requer nenhum conhecimento da operação das rotas, e em geral é simples e intuitivo como, por exemplo, "retorne todas as leituras de temperatura maiores que 60 graus na região interior ao polígono X".

O *Directed Diffusion*, entretanto, é aplicável somente em redes dirigidas a eventos [28]. Pelo nosso conhecimento, não existe nenhum protocolo além do *Directed Diffusion* que possui mecanismos simples de configuração. Assim, faz-se necessário o desenvolvimento de um protocolo com tal característica que opere em redes de disseminação contínua de dados. O protocolo ainda deve ser tolerante a falhas, possuir um consumo exíguo de recursos e apresentar um bom desempenho.

1.1 Objetivos

Neste trabalho é proposto um protocolo de roteamento configurável que procura minimizar o consumo de recursos da rede, ao mesmo tempo provendo tolerância a falhas e alto desempenho. O protocolo é específico para redes de disseminação contínua de dados, e possui um mecanismo de configuração simples e intuitivo, que permite que mesmo desenvolvedores sem conhecimentos de protocolos de comunicação consigam ajustar o protocolo proposto às suas necessidades.

O protocolo proposto, denominado PROC (*Proactive ROuting with Coordination*), interage com a aplicação ou outros protocolos utilizando um mecanismo de pontuação, que chamamos de *regras da aplicação*. Quanto maior o número de pontos acumulados em um nó, maior a chance deste ser escolhido como roteador de dados. Apesar de simples, as regras de aplicação provêm adaptabilidade ao protocolo, pois o *software* de aplicação pode modificar o comportamento do protocolo em tempo de execução, tanto por um comando externo vindo do operador da rede quanto por algoritmos de auto-organização que ajustam o funcionamento da rede de acordo com as alterações ocorridas no ambiente.

O PROC opera em duas fases. Na primeira fase, o protocolo seleciona nós que irão participar do repasse de dados de acordo com as regras de aplicação. Na segunda fase, o protocolo verifica se os nós selecionados pela aplicação são suficientes, e adiciona nós caso necessário. A segunda fase do PROC garante que as rotas serão formadas corretamente, mesmo que as regras de aplicação sejam implementadas pelo *software* de aplicação de forma incorreta.

Para permitir uma maior resiliência a falhas, o PROC implementa mecanismos que detectam nós falhos e enlaces de baixa qualidade, diminuindo a probabilidade que os dados enviados sejam perdidos ou requeiram retransmissões.

Avaliamos o protocolo proposto utilizando simulações, que medem a sua eficiência em termos de consumo de energia, taxa de entrega e latência da comunicação. O desempenho do protocolo também foi mensurado em cenários onde ocorrem falhas de nós, que foram variadas em intensidade e tempo de falha. Demonstramos que o PROC é compatível com os recursos escassos das RSSF ao implementá-lo e testá-lo em nós sensores da plataforma Mica2, uma plataforma frequentemente utilizada em RSSF em operação e na pesquisa em RSSF.

1.2 Contribuições

O desenvolvimento e a avaliação de um protocolo de roteamento configurável trazem as seguintes contribuições:

- Especificação de um protocolo de roteamento tolerante a falhas para redes de sensores sem fio de disseminação contínua de dados (chamado PROC) que provê mecanismos de otimização de rotas pela interação com a aplicação.
- Especificação de um modelo de falhas para redes de sensores que simplifica a avaliação de aspectos de tolerância a falhas em redes de sensores. Este modelo engloba as falhas mais frequentes em redes de sensores sem fio e alguns ataques de segurança.
- Avaliação de desempenho de protocolos de roteamento para RSSF, tendo em vista parâmetros como taxa de entrega, latência e consumo de energia, em redes de tamanhos diversos. Os protocolos ainda foram avaliados em cenários onde ocorrem falhas de nós, que variam em intensidade, extensão, tipo de falha (permanente ou transiente) e duração da falha.
- Implementação de extensões ao simulador NS-2 (aplicações, protocolos de roteamento, protocolos de controle e acesso ao meio) que permitem simular redes de sensores sem fio com maior fidelidade. Estas extensões estão sendo utilizadas pelos pesquisadores do DCC/UFMG, e do DCC/UFLA, em trabalhos de dissertação e conclusão de curso. O código encontra-se disponível no endereço `http://www.dcc.ufmg.br/~damacedo/proc.html`.
- Implementação do PROC em nós sensores da plataforma Mica Motes2, amplamente utilizada em aplicações comerciais e em projetos de pesquisa. A implementação pode ser encontrada no endereço `http://www.dcc.ufmg.br/~damacedo/proc.html`.
- Co-orientação de um bolsista de iniciação científica no tema "Integração de protocolos de roteamento e protocolos de controle e acesso ao meio em redes de sensores sem fio".

Durante o desenvolvimento da dissertação, resultados parciais foram publicados em conferências e periódicos nacionais e internacionais. Obtivemos uma publicação em periódico internacional, duas publicações em conferências internacionais e dois artigos completos em conferências

nacionais, sendo que um dos artigos obteve o prêmio *Best Paper Award on Simulation*, oferecido pelo *IEEE Communications Society Technical Committee on Simulation*. Ainda obtivemos publicações nacionais e internacionais em temas desenvolvidos em paralelo ao mestrado. Vide Apêndice A para a lista completa de publicações.

1.3 Organização do Texto

Este texto está organizado em sete capítulos. O capítulo 2 apresenta uma visão geral das questões pertinentes ao roteamento em redes ad hoc e RSSF. Identificamos conceitos básicos de roteamento em redes ad hoc e, em seguida, apresentamos as peculiaridades do roteamento em RSSF que devem ser levadas em consideração no desenvolvimento de protocolos específicos para essas redes. Em seguida, apresentamos os protocolos de roteamento mais importantes da literatura, mostrando o seu funcionamento e explicitando as suas aplicações-alvo. Concluímos o capítulo descrevendo os mecanismos de tolerância a falhas frequentemente encontrados em protocolos de roteamento para RSSF.

No capítulo 3 apresentamos o protocolo PROC e descrevemos o funcionamento de cada algoritmo que compõe o protocolo. Discutimos as regras de aplicação, que possibilitam que aplicações ajustem o PROC às suas características, otimizando assim o roteamento. Mostramos possíveis usos das regras de aplicação e propomos três regras simples que podem ser utilizadas em qualquer RSSF. Em seguida, descrevemos os mecanismos de tolerância a falhas presentes no PROC, bem como o modelo de falhas que consideramos na avaliação do protocolo. Concluímos o capítulo apresentando um modelo analítico do protocolo proposto e analisando o custo do protocolo em termos de consumo de memória, número de operações do processador e quantidade de mensagens enviadas, utilizando a notação "big-Oh".

No capítulo 4 avaliamos o funcionamento do protocolo em um ambiente de simulação. A avaliação consiste de três cenários. No primeiro, avaliamos a escalabilidade do PROC variando o número de nós na rede. O segundo cenário avalia o tempo de vida da rede ao utilizarmos diversos protocolos de roteamento. Por fim, utilizamos o modelo de falhas proposto no capítulo 3 para avaliar o comportamento do PROC diante de falhas.

Descrevemos a implementação do protocolo PROC na arquitetura de nós sensores Mica2 no capítulo 5. Apresentamos o formato e função dos pacotes, as *interfaces* criadas para a interação entre o protocolo e o *software* da aplicação. Em seguida, demonstramos o funcionamento do PROC em uma rede real.

O capítulo 6 apresenta as conclusões obtidas das simulações e da implementação do PROC em um ambiente real. Também apresentamos possíveis extensões do PROC e trabalhos futuros relacionados ao roteamento em RSSF.

Capítulo 2

Roteamento em Redes Ad Hoc e Redes de Sensores Sem Fio

Este capítulo apresenta e descreve os principais protocolos e técnicas utilizadas no roteamento de dados em redes ad hoc e RSSF. Na Seção 2.1 apresentamos uma visão geral dos problemas existentes na comunicação em RSSF, e identificamos porque soluções desenvolvidas para redes ad hoc devem ser revistas para serem aplicadas em RSSF. Na Seção 2.2 descrevemos os conceitos relativos às redes ad hoc e que são utilizados em RSSF. A Seção 2.3 discute as particularidades no fluxo de dados em RSSF, que são frequentemente utilizadas na especificação de protocolos de roteamento otimizados. Em seguida, apresentamos os principais protocolos de roteamento em RSSF, classificando-os de acordo com o seu emprego. A Seção 2.4 apresenta aspectos de tolerância a falhas no roteamento em RSSF e como os protocolos de comunicação existentes abordam o tema.

2.1 Comunicação em RSSF

Devido às limitações impostas pelo tamanho e custo dos nós sensores, as baterias são de baixa capacidade (armazenando poucas centenas de mAh), assim os componentes de *hardware* e *software* procuram minimizar a quantidade de energia consumida. Dentre os componentes empregados nos nós sensores atuais, a memória *flash* e o rádio são os maiores consumidores de energia [16]. Como o acesso a dados na memória *flash* é eventual, o rádio pode ser considerado o grande consumidor de energia nos nós sensores. A Tabela 2.1 exemplifica o consumo típico dos componentes de um nó sensor nos seus principais estados de operação.

As restrições severas de energia impostas pelo *hardware* empregado em nós sensores determinam que os protocolos de comunicação adotem mecanismos para diminuir o consumo de energia. Protocolos de controle e acesso ao meio (ou MAC, do inglês *Medium Access Control*) minimizam o consumo de energia ao ajustar a potência de transmissão de dados e ao desligar o rádio

Dispositivo	Estado de operação	Corrente
Processador	Ativo	8 mA
	Repouso	8 μA
Rádio	Recepção	8 mA
	Transmissão (0 dBm)	12 mA
	Repouso	2 μA
Memória Flash	Escrita	15 mA
	Leitura	4 mA
	Inativa	2 μA
Sensor (médio)	Ativo	5 mA
	Repouso	5 μA

Tabela 2.1: Consumo do *hardware* empregado na plataforma Mica Motes2 [16].

sempre que este não estiver sendo utilizado [14, 15]. Protocolos de roteamento e aplicações contribuem para a redução do consumo de energia ao diminuir a quantidade de dados transmitidos, ou procurando minimizar a quantidade de mensagens de controle enviadas.

Existe uma vasta literatura sobre protocolos de comunicação eficientes em energia para redes ad hoc. Entretanto, apesar de serem uma subclasse de redes ad hoc, as RSSF possuem peculiaridades que impossibilitam o uso de protocolos de comunicação desenvolvidos para redes ad hoc [35]. Estas particularidades são apresentadas a seguir.

Grande número de nós: Redes de sensores sem fio são projetadas para suportar centenas de milhares de nós, ao contrário das redes ad hoc, que em geral possuem dezenas ou centenas de nós. Assim, os protocolos utilizados em redes de sensores sem fio devem ser mais simples e escaláveis que os protocolos utilizados em redes ad hoc tradicionais.

Inexistência de identificadores únicos: Em geral, em RSSF a identificação do "produtor" da informação não é relevante, mas sim a informação. Os nós sensores podem não possuir identificadores únicos, podendo ser endereçados por sua localização geográfica, por exemplo [36]. Em redes ad hoc, por outro lado, a existência de um identificador único é uma premissa fundamental da rede, sendo utilizada por diversos protocolos, que desta forma não poderiam operar em RSSF.

Limitações severas de recursos: Devido ao processador com pouca capacidade de processamento e à memória limitada, os protocolos devem ser simples e eficientes. Um algoritmo que necessita de $O(n^2)$ bytes de memória, por exemplo, pode consumir uma quantidade significativa dos recursos disponíveis (veja Seção 3.5.2). Desta forma, os protocolos de roteamento em RSSF procuram minimizar a quantidade de memória e o processamento requeridos.

Topologia altamente dinâmica: Por serem utilizadas em regiões inóspitas, as RSSF são altamente suscetíveis a falhas. Além disso, nós sensores podem ser adicionados à rede durante a sua operação, ou desligados temporariamente para possibilitar a economia de energia. Assim, os protocolos desenvolvidos para RSSF devem suportar mudanças frequentes na topologia,

bem como falhas de nós. Nas redes ad hoc, por outro lado, mudanças na topologia são menos frequentes, sendo causadas principalmente pela movimentação dos nós.

Otimizadas para uma única aplicação: Devido às severas limitações de recursos, RSSF tendem a empregar uma grande interação entre camadas de comunicação [51] para aumentar o desempenho da rede. Além disso, os protocolos tendem a ser específicos para uma determinada classe de aplicações, uma vez que a especialização permite simplificar os protocolos e aumentar o desempenho da rede [28]. Redes ad hoc, por outro lado, são planejadas tendo em vista a generalidade, pois não se sabe *a priori* qual será o uso da rede (tráfego de voz, tráfego de dados insensíveis a latência, comunicação com restrições de tempo real, requisitos de qualidade de serviço, etc.).

Apesar de possuírem inúmeras diferenças, vários conceitos desenvolvidos para redes ad hoc são aplicados em protocolos e serviços para RSSF. A próxima seção apresenta estes conceitos.

2.2 Roteamento em Redes Ad Hoc

As redes ad hoc se baseiam na tecnologia IP (*Internet Protocol*), o protocolo utilizado na Internet, e formam uma rede multi-saltos que provê comunicação em ambientes onde não existe infra-estrutura prévia (e.g. redes cabeadas, pontos de acesso sem fio). Ao contrário das redes de sensores, que são utilizadas em uma aplicação específica, redes ad hoc são "genéricas", suportando qualquer tipo de aplicação. Por fim, redes ad hoc permitem a mobilidade do usuário, pois assume-se que este utilize tecnologias móveis (PDAs, *notebooks* ou telefones celulares) no acesso à rede. RSSF, por outro lado, em geral são estáticas.

Os protocolos de roteamento ad hoc podem ser classificados em estado do enlace (ou *link state*) e vetor de distância (ou *distance vector*), como ocorre nas redes fixas [27, 72]. Os protocolos baseados em estado de enlace requerem o conhecimento de toda a topologia para construir as rotas. Já os protocolos baseados em vetor de distância requerem apenas informações dos nós ao seu alcance. Por exemplo, o protocolo DSR (*Dynamic Source Routing*) [40] é baseado em estado de enlace, enquanto o protocolo DSDV (*Destination-Sequenced Distance Vector*) [63] é baseado em vetor de distância. Os protocolos ad hoc podem ainda ser classificados como *reativos* e *pró-ativos*.

Os protocolos reativos são aqueles em que as rotas de um nó a outro são construídas somente quando necessárias. Antes de iniciar uma conexão, o nó origem irá verificar se possui uma rota para o nó destino. Se esta rota existe, a comunicação é iniciada. Caso contrário, o nó origem utilizará o protocolo de roteamento para identificar uma rota até o nó de destino. O AODV (*Ad hoc On demand Distance Vector*) e o DSR são exemplos de protocolos reativos [40, 64].

Nos protocolos pró-ativos, um nó possui rotas para qualquer outro nó da rede antes da requisição de comunicação, pois o protocolo de roteamento periodicamente determina a melhor

rota para se alcançar cada nó da rede. Os protocolos DSDV e WRP (*Wireless Routing Protocol*) são exemplos de protocolos pró-ativos [58, 63].

A escolha entre protocolos reativos e pró-ativos dependerá das características do fluxo de dados da rede. Protocolos reativos diminuem o tráfego necessário para a formação de rotas quando a frequência de transmissões de dados na rede é baixa. A primeira comunicação de um dado nó origem com um dado nó destino, entretanto, apresenta uma alta latência inicial, pois o protocolo de roteamento deve encontrar uma rota até o destino. Os protocolos pró-ativos, por outro lado, não adicionam um atraso na primeira comunicação, entretanto devem ser evitados em redes com comunicações infrequentes, pois o custo de manutenção das rotas sobrepuja o custo de transmissão de dados. Para redes onde muitos nós comunicam entre si, os protocolos pró-ativos são os mais eficientes, pois o custo da reconstrução periódica de rotas é amortizado pela grande frequência das comunicações.

2.3 Roteamento em RSSF

Como nas redes ad hoc, protocolos de roteamento em redes de sensores podem ser classificados em pró-ativos e reativos. Devido às severas limitações de recursos e à grande quantidade de nós empregados, RSSF evita-se utilizar protocolos de roteamento baseados em estado de enlace, uma vez que o custo e tempo necessário para propagar o estado da rede a todos os outros nós é proibitivo. Outra diferença entre protocolos de roteamento pró-ativos desenvolvidos para redes ad hoc e os protocolos para RSSF é que, em RSSF, os protocolos pró-ativos não possuem rotas para todos os nós da rede pois tal abordagem permite um menor consumo de memória.

Para economizar recursos e aumentar o desempenho da rede, os protocolos de roteamento para RSSF, em geral, otimizam o seu funcionamento de acordo com as características de tráfego de cada aplicação. Tilak et al. [73] propuseram uma taxonomia para RSSF que engloba aspectos da dinâmica da rede e as características do tráfego de dados. A seguir apresentamos os modelos de entrega de dados levantados por Tilak et al., e em seguida apresentamos e estendemos os modelos de fluxo de dados levantados pelo mesmo.

2.3.1 Modelos de Entrega de Dados em RSSF

A primeira característica analisada por Tilak et al. é a forma como a comunicação entre os nós sensores e um ponto de acesso (PA) é realizada. Os autores denominam estes padrões de comunicação como o *modelo de entrega de dados*. Os seguintes modelos de entrega de dados foram levantados:

Contínua: Nestas redes, também chamadas de *redes de disseminação contínua*, os nós enviam mensagens em intervalos constantes para o PA, relatando as leituras atuais dos seus sensores. Com o envio periódico de dados, é possível construir um modelo no espaço e no tempo do

comportamento da rede, bem como acompanhar a evolução do fenômeno observado em tempo real. Aplicações dessas redes incluem estudos ambientais [71], sistemas de tráfego inteligente, monitoração de plantas industriais, construção do mapa de temperatura em incêndios [49], entre outros. Nestas redes, os protocolos de roteamento tendem a utilizar abordagens pró-ativas e baseadas no vetor de distância, pois o fluxo de dados contínuo é beneficiado pela atualização constante das rotas.

Dirigida a eventos: Nestas redes, o envio de dados ao PA é ocasional, ocorrendo somente quando um determinado evento é detectado, ou quando ocorre uma alteração importante no comportamento do fenômeno monitorado. Redes dirigidas a evento são utilizadas na localização de animais silvestres, detecção de intrusão, monitoramento de queimadas [9], entre outros. Devido ao fluxo escasso de dados, estas redes tendem a empregar protocolos de roteamento reativos.

Iniciada pelo observador: São redes onde os dados são enviados ao PA somente quando explicitamente requisitados. Um exemplo é a rede descrita por Burrel et al. em [11], onde nós sensores acoplados em enxadas e tesouras de poda coletam informações sobre a temperatura e humidade de uma vinícola durante todo o dia. Ao final do dia, os sensores são "guardados", e o PA solicita aos mesmos um relatório da atividade diária. Esse modelo de entrega de dados também é empregado em redes onde o PA é móvel, estando acoplado em um dirigível, por exemplo. Os sensores armazenam os dados coletados, que são requisitados pelo PA quando este detecta a sua proximidade aos nós [85].

Híbrida: É a combinação de dois ou mais modelos apresentados anteriormente. Redes híbridas são utilizadas para monitoramento de catástrofe, por exemplo. Nestas situações, nós sensores possuem poucos dados a reportar em situações corriqueiras, operando assim em um modelo dirigido a eventos. Entretanto, na ocorrência de uma catástrofe, o operador pode requisitar um fluxo contínuo de informações, como em redes de disseminação contínua, para obter o máximo possível de dados sobre o evento. Como o tráfego é variável, essas redes podem empregar tanto protocolos pró-ativos quanto protocolos reativos. Outra solução proposta na literatura é o uso de protocolos adaptativos, que alternam entre uma solução pró-ativa e uma solução reativa quando ocorre uma mudança significativa no tráfego da rede [28].

Os modelos de entrega são compostos por diversos tipos de fluxos, que também podem ser divididos em categorias. A seguir detalhamos os fluxos de dados encontrados em RSSF.

2.3.2 Fluxos de Dados em RSSF

Enquanto o modelo de entrega de dados se relaciona ao modo como os nós sensores se comunicam com o PA, o fluxo de dados se relaciona à comunicação entre todo tipo de nó. Além disto, um modelo de entrega de dados pode se utilizar de fluxos de dados de diversos tipos para implementar a comunicação com o PA. A seguir mostramos os fluxos de dados definidos por Tilak et al. [73], adicionando o fluxo *convergecast*, que é frequentemente visto em redes de disseminação

contínua de dados.

Inundação (flooding): Este fluxo de dados se caracteriza pela comunicação de um nó com seus vizinhos a um determinado número de saltos de distância. O emissor da mensagem a envia para todos os seus vizinhos, e os vizinhos do nó as repassam para seus vizinhos e assim em diante. Esta comunicação é facilitada em redes sem fio devido às características do meio de comunicação, pois todos os nós ao alcance de rádio do emissor escutam a transmissão. O PA frequentemente se utiliza de inundação de pacotes para, por exemplo, alertar os nós para que monitorem um novo tipo de evento [36].

Comunicação ponto-a-ponto (unicast): É utilizado na comunicação direta entre nós, por exemplo quando sensores alertam o PA de um novo evento de interesse na área monitorada [36].

Multicast: Como na Internet [72], é utilizado para propagar dados a um conjunto limitado de nós em situações onde é custoso enviar os dados a cada um dos destinatários utilizando comunicação *unicast*. Este tipo de comunicação ocorre, por exemplo, quando o PA deseja alertar nós em uma dada região sobre uma nova configuração (para, por exemplo, aumentar a taxa de envio de dados). A área pode tanto ser definida em saltos quanto em coordenadas geográficas [87].

Convergecast (many-to-one): Este fluxo de dados se caracteriza por várias fontes de dados que possuem como destino um único nó. Exemplos de tais comunicações são redes hierárquicas, onde nós são divididos em grupos, e os nós do grupo enviam os seus dados ao líder de grupo [79]. Outro exemplo de tráfego *convergecast* são as redes de disseminação contínua, onde os nós sensores enviam os dados produzidos diretamente ao PA utilizando caminhos de múltiplos saltos [10, 42, 84].

Como as aplicações possuem modelos de entrega de dados e fluxos de dados bem definidos, é possível construir protocolos de roteamento que se aproveitam destas características para economizar energia e diminuir o custo do estabelecimento de rotas. Protocolos de roteamento podem, por exemplo, suportar somente um modelo de entrega de dados. Outra otimização frequentemente adotada é a criação de rotas somente entre os nós sensores e o PA, não existindo rotas de um sensor a outro sensor qualquer [10, 24, 28, 42, 53, 70, 84].

A seção seguinte apresenta os principais protocolos de roteamento para redes de sensores sem fio, analisando os modelos de entrega de dados e fluxos de dados que estes suportam. Avaliamos os protocolos quanto à sua capacidade de adaptação ao ambiente e às peculiaridades de cada aplicação, e também mencionamos as características e os procedimentos encontrados no PROC. Uma avaliação exaustiva de protocolos de roteamento em RSSF pode ser encontrada em [1].

2.3.3 Protocolos de Roteamento em RSSF

O LEACH (*Low Energy Adaptive Clustering Hierarchy*) [79] é um protocolo de roteamento para RSSF de disseminação contínua de dados. O LEACH suporta tráfego *convergecast* dos nós sensores até o PA, enquanto a comunicação do PA com os nós da rede é feito via *broadcast* ou

unicast, utilizando rotas de apenas um salto. O protocolo divide a periodicamente a rede em grupos, utilizando um cálculo probabilístico que leva em conta a localização e energia residual de cada nó do grupo. Os grupos possuem um líder, que repassa os dados gerados pelos seus companheiros de grupo para o PA. Este repasse é feito via comunicação direta, o que limita o uso do protocolo a redes onde os nós conseguem alcançar o PA diretamente. O protocolo PEGASIS (*Power-Efficient Gathering in Sensor Information Systems*) [70] é uma variação do LEACH que utiliza a passagem de *tokens* entre líderes de grupo para agrupar os dados coletados e evitar colisões ao transmitir para o PA. Existem outros protocolos baseados no LEACH [24, 53] que estendem a sua aplicabilidade, mas todos necessitam de informação precisa de localização, dado não disponível facilmente. O protocolo proposto nesta dissertação não necessita de dados de localização, mas pode utilizá-los no roteamento caso a aplicação ofereça este suporte.

O TinyOS Beaconing é o protocolo de roteamento utilizado como padrão na plataforma de nós sensores Mica Motes [52]. Este protocolo recria periodicamente uma árvore de roteamento de menor caminho. O TinyOS Beaconing somente cria rotas dos nós sensores ao PA. Assim, a comunicação dos sensores ao PA deve ser feita por inundação.

A fim de criar as rotas, o PA periodicamente propaga por inundação uma mensagem *beacon* para a rede, utilizada pelos nós para determinar a sua distância em saltos até o PA. Os nós sensores que recebem a mensagem de *beacon* diretamente do PA identificam que são seus vizinhos, ou seja, possuem distância de um salto. Estes nós repassam a mensagem, adicionando seu identificador e a sua distância ao PA (um salto). Os nós que estão a dois saltos do PA receberão o *beacon*, e selecionarão como próximo salto o nó que irá atingir o PA com menor distância. Este processo é repetido até que a mensagem seja enviada para todos os nós da rede.

O algoritmo de seleção das rotas emprega uma medida de confiabilidade do enlace, calculada pelo número de pacotes corretamente enviados pelos nós vizinhos. Os nós sensores escutam a rede em modo promíscuo, e monitoram o envio e recepção de dados de todos os seus vizinhos. A confiabilidade do enlace é dada pela proporção entre número de pacotes enviados e o número de confirmações recebidas para cada vizinho. Suponha, por exemplo, que um dado nó A detectou que um dos seus vizinhos, o nó B, enviou 5 pacotes para C, e C não confirmou a recepção de três. O nó A irá registrar que a comunicação de B possui confiabilidade de 40%. O uso de estimadores de confiabilidade permite o estabelecimento de rotas mais eficientes, pois os dados sofrerão erros de transmissão com menor frequência. Este mecanismo, entretanto, impossibilita o desligamento periódico do rádio, que é uma técnica frequentemente empregada por protocolos MAC para diminuir o consumo de energia dos nós sensores [14].

O protocolo de roteamento EAD (*Energy-Aware Distributed routing*), proposto por Boukerche et al., cria uma árvore de roteamento que maximiza o número de nós folha [10]. Os nós folha não roteiam dados, e portanto podem manter o seu rádio desligado por períodos prolongados de tempo. O protocolo foi desenvolvido para aplicações que possuem padrão de tráfego *convergecast* no repasse de dados ao PA. Inicialmente desenvolvido tanto para redes de disseminação contínua

quanto para redes dirigidas a eventos, o uso de uma estratégia pró-ativa no EAD beneficia as redes de disseminação contínua, pois todas as rotas são reconstruídas periodicamente.

Como no TinyOS Beaconing, o EAD utiliza a propagação de uma mensagem de *beacon* para construir as suas rotas. Esta mensagem carrega o identificador do emissor da mensagem, sua energia residual, seu pai na árvore de roteamento e seu estado atual, que pode ser "nó folha" ou "nó não-folha". Um nó sensor é marcado como não-folha somente quando um dos seus nós vizinhos o marcou como próximo salto até o PA. Neste caso, como o nó deverá repassar os dados produzidos pelo vizinho, ele será mantido ligado durante todo o período em que permanecer um nó não-folha. O critério de seleção do nó pai dá preferência aos nós com menor distância ao PA, seguido pelos nós não-folha. Assim, caso a distância mínima ao PA seja atingida por dois nós, um folha e outro não-folha, o nó irá escolher o seu vizinho não-folha como pai. Entretanto, caso o nó folha esteja mais próximo do PA que o nó não-folha, o nó folha será escolhido como pai.

Para evitar que muitos nós sejam marcados como não-folha, o envio da mensagem de *beacon* é atrasada por um tempo aleatório. Para os nós não-folha, este tempo será inferior ao tempo de envio dos nós folha. Assim, os nós folha irão receber, com uma grande probabilidade, pelo menos um pacote de *beacon* de um dos seus nós vizinhos não-folha antes de repassarem a mensagem de *beacon*. Desta forma, os nós folha podem escolher um nó pai não-folha, evitando assim que um nó folha seja convertido a nó não-folha. O tempo de atraso ainda é baseado na energia residual, que diminui significativamente a quantidade de colisões e homogeneiza o consumo de energia da rede. Quanto maior a energia residual, menor será o atraso para envio dos dados. Logo, a mensagem de *beacon* será enviada primeiro pelos nós com maior energia residual, desta forma os nós folha irão frequentemente escolher como pai o nó com maior energia residual.

Os protocolos Minimum Cost Forwarding [84] e GRAB [86] constroem uma árvore de roteamento baseada em uma função custo, que é definida no momento da programação do nó. Ao enviar um dado ao PA, o nó escolhe como próximo salto da comunicação o vizinho que apresenta o menor custo de repasse da mensagem ao PA. O custo pode ser definido como menor latência, menor distância em saltos até o PA ou menor consumo de energia, por exemplo. Para que não ocorram ciclos no roteamento, a função custo deve ser monotonicamente crescente à medida que os nós se distanciam do PA. No Minimum Cost Forwarding, o pacote é repassado por somente uma rota, o que pode ocasionar a perda de dados devido a enlaces de baixa qualidade. O GRAB procura contornar este problema enviando os dados por várias rotas. O número de rotas utilizadas é determinado pelo custo máximo a ser empregado no repasse dos pacotes. Enquanto no Minimum Cost Forwarding o custo alocado para cada mensagem é o custo mínimo, no GRAB os nós alocam um custo superior ao mínimo. Esta "folga" é utilizada no envio de dados sobre múltiplas rotas, e determina quantas rotas alternativas serão formadas. Como no PROC e no Directed Diffusion, o Minimum Cost Forwarding e o GRAB provêm mecanismos para que a aplicação modifique as rotas conforme sua necessidade. Entretanto, ambos os protocolos não possuem mecanismos para a redefinição em tempo de execução da função custo. Além disso, o

mecanismo de tolerância a falhas do GRAB é custoso, pois requerer um maior número de repasses de mensagens.

Dentre os protocolos existentes em RSSF, em geral somente os protocolos geográficos permitem a comunicação entre qualquer nó da rede. Os protocolos geográficos utilizam informações de posicionamento dos nós no roteamento. São exemplos de protocolos geográficos os protocolos GEAR (*Geographical and Energy Aware Routing*) e GPSR (*Greedy Perimeter Stateless Routing*) [45, 87]. Estes protocolos utilizam uma estratégia gulosa, onde o próximo salto da comunicação será o nó mais próximo do destino da comunicação. Os protocolos diferem no tratamento dos casos limítrofes, por exemplo quando não existe um nó mais próximo, situação em que uma rota alternativa deve ser encontrada. Por permitirem a comunicação entre qualquer nó da rede, os protocolos geográficos em geral suportam todos os modelos de entrega de dados utilizados em RSSF. Em geral, estes protocolos permitem fluxos ponto-a-ponto e inundação. Alguns protocolos, como o GEAR, permitem ainda envio de dados para uma região delimitada pelas coordenadas de um polígono, como em um fluxo *multicast*.

O uso de protocolos geográficos é limitado, devido ao alto custo de mecanismos de posicionamento, como o GPS (*Global Positioning System* – Sistema de Posicionamento Global) e protocolos de determinação de coordenadas. Entretanto, como as coordenadas geográficas são frequentemente utilizada em várias aplicações e protocolos, o custo da determinação da localização pode ser amortizado. Os protocolos geográficos ainda podem utilizar coordenadas "virtuais", que são construídas em função da distância em saltos dos nós sensores em relação a nós de referência [67]. Neste método, não é necessário que os nós possuam mecanismos de localização, entretanto a precisão das coordenadas virtuais (medida pela diferença entre a posição atual do nó e a sua posição virtual) aumenta com o uso de tais dispositivos.

O uso de informações da aplicação no roteamento foi introduzido no protocolo Directed Diffusion [36]. Este protocolo, desenvolvido para redes dirigidas a eventos, permite que os dados requisitados pela aplicação determinem as rotas estabelecidas para atender cada requisição. A aplicação decide se um pacote será repassado, descartado ou fundido com outros pacotes recebidos, reduzindo o consumo de energia. O protocolo utiliza a comunicação ponto-a-ponto nos dados enviados dos nós sensores para o PA, e adota a estratégia de inundação para a propagação de dados de um PA para a rede. Estudos posteriores mostraram que o Directed Diffusion é ineficiente e de difícil adaptação quando aplicado em redes de disseminação contínua de dados [28, 59].

Liu et al. apresentam um protocolo de roteamento específico para redes dirigidas em eventos, onde os sensores realizam agregação de dados durante o repasse [54]. O protocolo proposto procura maximizar a informação obtida por uma consulta à rede e minimizar o consumo de energia. Isto é feito utilizando técnicas de processamento de sinais, que identificam qual caminho maximiza a quantidade de informação adquirida em relação ao evento procurado. O protocolo também define a rota de acordo com um consumo máximo de energia definido pelo operador. Para reduzir o tráfego de dados na rede, os nós realizam a agregação dos dados a cada passo da

comunicação, que aumenta a precisão dos dados a cada repasse. Como o Directed Diffusion, o protocolo proposto por Liu et al. utiliza as características dos dados requisitados para otimizar o seu funcionamento. Entretanto, o protocolo requer que o programador modele a requisição como um sinal contínuo, o que pode ser difícil e não intuitivo para certas aplicações.

Figueiredo et al. [28] desenvolveram um protocolo de roteamento, chamado Multi, que modifica o seu algoritmo de criação de rotas de acordo com a frequência da ocorrência de eventos. Este protocolo é composto por dois outros protocolos, o SID e o EF-Tree. O SID (*Source-Initiated Dissemination*) é um protocolo que constrói as rotas a partir do nó que detectou um evento de interesse do ponto de acesso, enquanto o EF-Tree (*Earlier-First Tree*) constrói uma árvore de roteamento de menor distância com raiz no PA. O SID é apropriado para tráfego escasso de dados, enquanto o EF-Tree é indicado para situações onde o tráfego é mais intenso. O Multi implementa os dois protocolos, e escolhe qual deles deve ser utilizado de acordo com o tráfego percebido no PA. O Multi, como o PROC, adapta-se automaticamente a uma característica da aplicação, que é o volume de pacotes de dados produzidos na rede. O PROC, por outro lado, possui um mecanismo genérico de adaptação, comandados não pelo roteamento, mas pela aplicação, por um operador ou por outros protocolos, que permite um maior grau de adaptação que o oferecido pelo Multi.

A partir dos resultados obtidos no protocolo Multi, Figueiredo et al. propuseram a criação de novos protocolos de roteamento baseados em políticas [29]. Este trabalho se assemelha à proposta do PROC, ao sugerir que políticas de adaptação, assim como as regras de aplicação do PROC, sejam utilizadas para que a aplicação ou outros protocolos modifiquem o comportamento do protocolo de roteamento. Este trabalho, que se encontra em fase inicial, propõe uma arquitetura de adaptação onde as regras são codificadas como *scripts*. Como o projeto ainda se encontra em fase de especificação, não é possível realizar uma comparação detalhada com o PROC.

2.4 Tolerância a Falhas em RSSF

Tolerância a falhas é um assunto que permeia qualquer sistema de computação. Assim como as falhas, os ataques de segurança interrompem o funcionamento correto de um sistema. Por este motivo, Avizienis et al. propõem uma taxonomia onde ataques de segurança são classificados como um tipo particular de falha, chamado de *falha maliciosa* [6].

Tolerância a falhas é um requisito essencial para o projeto de protocolos e aplicações em RSSF, pois falhas de hardware e comunicação são frequentes. Falhas podem ocorrer devido às condições severas do ambiente onde os nós se encontram [35], ou devido à má qualidade dos enlaces sem fio [30, 68]. As falhas podem ser caracterizadas em dois grandes grupos, que são as *falhas silenciosas* e as *falhas bizantinas* [6]. As falhas silenciosas se caracterizam pela ausência de dados de saída (impressão de relatórios, envio de mensagens, etc) enquanto o componente ou sistema está falho. Nas falhas bizantinas, por outro lado, o sistema continua a gerar dados de

saída, que ora são corretos e ora são incorretos. Falhas bizantinas podem ocorrer pela existência de erros nos algoritmos ou pela ação de agentes maliciosos, que possuem o objetivo de prejudicar o funcionamento da rede. Realizamos um levantamento das principais falhas silenciosas que ocorrem em RSSF, mostrado a seguir.

2.4.1 Falhas Silenciosas em RSSF

A partir da literatura corrente em comunicação em redes ad hoc sem fio e em RSSF, investigamos as falhas de comunicação silenciosas que ocorrem em RSSF. Estas falhas são causadas por diversos agentes e eventos internos e externos à rede, como mostramos a seguir.

Fenômenos atmosféricos: Mudanças nas condições atmosféricas alteram a propagação do sinal, causando um aumento na taxa de erros da comunicação. Condições do ambiente como humidade, temperatura, entre outros, modificam a qualidade dos enlaces. Como as características do ambiente são dinâmicas, a qualidade da comunicação varia com o tempo.

Fontes móveis de interferência: Aparelhos que operam em faixas de frequência próximas às utilizadas em RSSF, veículos, animais e pessoas, podem gerar interferência na comunicação. Por operarem em frequências na faixa ISM (*Industrial, Scientifical and Medical*), que não requer licença para operação, RSSF estão expostas a interferência de outros aparelhos que também operam nesta faixa de frequência. Para baratear o custo dos nós sensores, o rádio geralmente emprega um único canal e possui modulação fixa. Estas limitações impedem o uso de mecanismos como seleção de canais com menor interferência, saltos de frequência ou troca dinâmica da técnica de modulação empregada [76].

Desastres naturais: Nós sensores podem ser depositados ao ar livre ou em regiões de desastre, estando assim expostos a deslizamentos, terremotos e enchentes. Desastres naturais podem ocasionar a destruição dos nós ou a inutilização de componentes do hardware dos nós sensores. Ao contrário das falhas decorrentes das condições atmosféricas, nós falhos devido a desastres naturais permanecem inoperantes.

Quebra acidental: Nós sensores podem ser destruídos acidentalmente, por exemplo devido ao pisoteamento por animais ou à queda de árvores sobre os mesmos. Em geral, os nós sensores estarão espalhados a alguns metros de distância uns dos outros, assim apenas um nó falha por vez.

Bloqueio do processador: Sistemas embutidos em geral utilizam escalonadores de multitarefa cooperativa, também chamados de *run to completion* [52]. Na multitarefa cooperativa, cada tarefa sinaliza quando pode ser substituída por outra tarefa esperando para ser executada. Assim, um software defeituoso ou mal programado pode bloquear o processador por tempo indeterminado, pois este nunca permite que o sistema operacional troque a tarefa em execução. Para evitar tais situações, são utilizados temporizadores chamados de *watchdogs*, que devem ser reiniciados periodicamente pela aplicação. Caso o *watchdog* não seja reiniciado, o processador

reinicia o sistema. Desta forma, o processador fica bloqueado por um tempo finito, e em seguida retorna à operação normal.

Falhas maliciosas: RSSF estão expostas a falhas maliciosas, como decorrentes de ataques de segurança. Nestas falhas, um nó malicioso ou uma entidade externa provocam erros na rede. Este trabalho não aborda técnicas de prevenção a ataques. Entretanto, é possível utilizar mecanismos simples de tolerância a falhas para identificar regiões ou nós sobre ataque e evitá-las [83]. Neste trabalho abordamos apenas mecanismos para contornar alguns ataques de negação de serviço (DoS – *Denial of Service*), que são: *ataques de interferência*, de *colisão* e de *sinkhole*). Estes ataques se comportam como falhas silenciosas.

Ataques de interferência, também chamados de ataques de *jamming*, ocorrem quando um agente malicioso envia um sinal de rádio em alta potência, que confunde os rádios dos nós sensores e impossibilita a recepção correta de dados. Ataques de colisão, por sua vez, são ataques onde o agente malicioso somente transmite sinais no momento em que um nó está transmitindo os seus dados, tendo por objetivo causar uma colisão no receptor. Já os ataques de *sinkhole* ocorrem quando um nó malicioso se passa pelo PA, redirecionando todo o tráfego da rede para si e o descartando em seguida. Estes ataques criam uma região onde os nós se comportam como nós falhos, pois os nós sobre ataque não respondem às requisições ou enviam dados.

Esgotamento da bateria: O esgotamento da bateria dos nós sensores pode gerar uma falha de comunicação. O emprego de nós em áreas de difícil acesso ou a grande quantidade de nós empregados pode tornar inviável o recarregamento de baterias. Por possuírem hardware limitado, os nós sensores atuais não permitem a aferição confiável do nível atual de energia, dessa forma os nós sensores não possuem meios para identificar quando a sua energia está próxima de acabar.

Devido ao grande número de falhas silenciosas existentes em RSSF, criamos um modelo de falhas que identifica as características mais importantes das falhas silenciosas no ponto de vista do roteamento. Este modelo simplifica a análise das falhas em RSSF ao abstrair seus detalhes menos relevantes. Esta abstração permite que classifiquemos as falhas em quatro categorias, de acordo com a sua persistência e extensão.

2.4.2 Modelo de Falhas

Esta seção apresenta um agrupamento das falhas descritas na Seção 2.4.1 de acordo com as suas características. Esse agrupamento tem como objetivo facilitar o estudo de falhas em RSSF, sendo resumido na Tabela 2.2. As falhas são caracterizadas quanto à sua persistência e extensão:

Persistência: Indica se o nó retornará a operar corretamente após um período de tempo falho (**falhas transientes**), ou se a falha é **permanente** [6]. Do ponto de vista do roteamento, falhas transientes são aquelas em que a falha pode ser contabilizada em minutos, enquanto falhas permanentes podem ser contabilizadas em horas. Protocolos de roteamento tendem a armazenar informações por períodos curtos de tempo, pois a topologia da rede tende a mudar frequente-

mente. Assim, o comportamento da rede frente a uma falha que dura horas é semelhante ao comportamento frente a uma falha permanente, pois em ambas as situações os nós falhos ficarão inativos durante um número elevado de iterações do algoritmo de roteamento. Assim, consideramos que situações de falha devido às mudanças no clima, por exemplo, serão caracterizadas como permanentes.

Extensão: Indica o número de nós afetados. As falhas podem ser **isoladas**, no caso da falha de um único nó, ou **agrupadas**, onde um conjunto de nós falha. A severidade da falha depende do número de nós faltosos, uma vez que grande parte da vizinhança de um nó se tornará falha, diminuindo o número de vizinhos que poderão rotear dados, como mostra a Figura 2.1. Na figura, setas representam as rotas dos nós e as "caveiras" representam nós falhos.

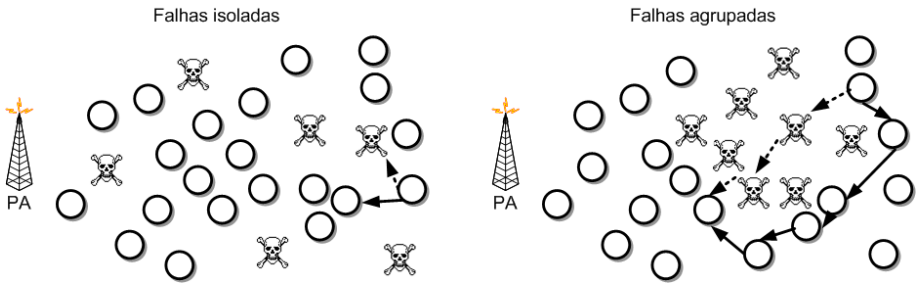

Figura 2.1: Exemplo de falhas de nós, classificadas quanto a extensão.

As falhas maliciosas decorrentes de ataques de colisão e de *sinkhole* variam de duração de acordo com a intenção do atacante, podendo ser breves para evitar detecção, ou prolongadas, aumentando o estrago causado. Assim, classificamos essas falhas tanto como transientes quanto como permanentes. Ataques de interferência, por outro lado, serão permanentes, pois estes são mais efetivos se empregados por um período prolongado, mesmo ocorrendo a detecção do ataque [83].

Para amenizar o efeito das falhas no roteamento, o PROC implementa as técnicas de tolerância a falhas apresentadas a seguir. O PROC ainda procura retardar as falhas por esgotamento da bateria dos nós ao priorizar os nós com maior energia residual na seleção do próximo salto na comunicação (veja Seção 3.2.3). Por fim, regras de aplicação podem ser utilizadas para melhorar a tolerância a falhas da rede, como por exemplo pelo uso das regras descritas na Seção 3.3, que distribuem a carga da rede entre vários nós, evitando assim que sua energia acabe muito antes que a dos seus vizinhos.

Ao analisar a Tabela 2.2, percebemos que nenhuma falha silenciosa encontrada em RSSF se enquadra na categoria transiente e agrupada. Assim, apesar do modelo de falhas definir quatro categorias de falha, somente três delas são verificadas em RSSF. Em seguida, realizamos um estudo dos mecanismos de tolerância a falhas empregado nos protocolos de roteamento existentes em RSSF.

Causa da falha	Persistência	Extensão
Fenômenos atmosféricos	permanente	agrupadas
Fontes móveis de interferência	transiente	isolada
Desastres naturais	permanente	agrupadas
Quebra acidental	permanente	isolada
Bloqueio do processador	transiente	isolada
Ataques de interferência	permanente	agrupadas
Ataques de colisão	ambos	isolada
Ataques de *sinkhole*	ambos	isolada

Tabela 2.2: Caracterização das falhas de acordo com a sua causa.

2.4.3 Mecanismos de Tolerância a Falhas no Roteamento

Devido às severas restrições de recursos encontradas nos nós sensores atuais, as técnicas de tolerância a falhas utilizadas devem ser simples, utilizando o mínimo possível de ciclos e memória. Desta forma, limitamos o escopo das falhas tratadas pelo PROC às falhas silenciosas. Como mostraremos a seguir, estas falhas podem ser contornadas com o uso de mecanismos de baixo custo computacional, permitindo assim que o PROC seja empregado em ambientes com severas restrições de recursos. Falhas bizantinas, por outro lado, requerem o uso de protocolos que provêm segurança na comunicação de dados e técnicas de diagnóstico da corretude das mensagens, que requerem uma maior quantidade de recursos da rede [43].

Os primeiros protocolos de roteamento propostos para RSSF [41, 79] consideram apenas falhas devido ao esgotamento da bateria. Assim, os protocolos propõem mecanismos para aumentar a vida do nó e distribuir o consumo de energia. Outros protocolos se preocupam apenas com falhas na comunicação ocasionadas por quebra de nós, amenizando este problema pelo uso de múltiplas rotas na transmissão de dados [22, 37, 42, 86]. Visto que cópias dos dados são enviados por múltiplos caminhos, os dados possuem uma maior probabilidade de serem recebidos pelo PA, ao custo de um maior consumo de energia.

Ganesan et al. estudaram a probabilidade de recepção de um pacote que trafega em múltiplas rotas, variando o seu grau de similaridade das rotas [23]. O estudo demonstrou que rotas parcialmente disjuntas são tão eficazes quanto rotas totalmente disjuntas, e possuem um menor custo para serem estabelecidas. De et al. mostraram analiticamente que a perda de dados em rotas parcialmente disjuntas é inferior à perda de dados em rotas totalmente disjuntas [20]. Além disso, ao considerar o uso de técnicas de correção de erro, os autores mostraram que o consumo de energia em rotas parcialmente disjuntas é menor, pois estas requerem menos blocos de correção de erro por pacote.

O numero de rotas utilizadas é, em geral, determinado pelo do consumo de energia requerido

para o repasse do pacote [42, 86] e a qualidade dos enlaces da rede [22]. Quando o número de rotas é determinado pelo consumo de energia, os pacotes possuem uma quantidade máxima de energia que pode ser utilizada no seu repasse. Esta determina quando um nó irá repassar a mensagem para dois nós ou apenas um nó, desta forma criando novas rotas ou mantendo o número atual de rotas. Nos métodos que utilizam a qualidade do enlace, o número de repasses de mensagens é determinado pela qualidade dos enlaces nas rotas até o PA. Quando existe uma rota com boa confiabilidade, o dado é repassado somente por aquela rota. Entretanto, caso nenhuma rota atenda aos requisitos mínimos de qualidade, são utilizadas várias rotas. Quanto pior os enlaces, maior é o número de rotas adicionais.

O envio de múltiplas cópias de dados possui um custo elevado, assim é desejável manter apenas uma rota composta por enlaces de boa qualidade. De Couto et al. sugerem uma modificação no protocolo de roteamento ad hoc DSR para que este considere a qualidade dos enlaces que compõem as rotas [21], amenizando falhas de comunicação. O protocolo DSR, ao propagar uma requisição de estabelecimento de rota, também propaga um valor acumulado que indica a qualidade do sinal na rota. O nó utiliza sempre a rota com a maior qualidade, evitando enlaces com comunicação intermitente. Alec Woo et al. [82] propuseram um mecanismo para protocolos de roteamento em RSSF similar ao desenvolvido por De Couto et al., que considera apenas a qualidade do enlace no próximo salto da comunicação. A cada passo da comunicação, o nó responsável por repassar os dados escolhe o vizinho com a melhor qualidade de enlace.

Biswas e Morris propõem um protocolo de roteamento que envia dados em uma única rota, entretanto os protocolo de roteamento pode utilizar "nós alternativos" quando o nó preferencial falha [8]. No ExOR (*Extremely Opportunistic Routing*), cada pacote de dados possui uma lista de nós (L), ordenados por prioridade, que irão repassar o pacote. A prioridade de repasse de dados não é tratada pelo protocolo, que delega tal tarefa a um algoritmo auxiliar qualquer. Um nó, ao escutarem a transmissão de um pacote que o inclui em L, envia um ACK com o seu endereço. Os nós ativos que compõem L escutam o meio para determinar quais outros nós da lista receberam o pacote. A partir dos ACKs recebidos, os nós verificam se devem repassar a mensagem, para tanto verificando se um nó de maior prioridade na lista enviou um ACK. Desta forma, as mensagens são repassadas por apenas um nó.

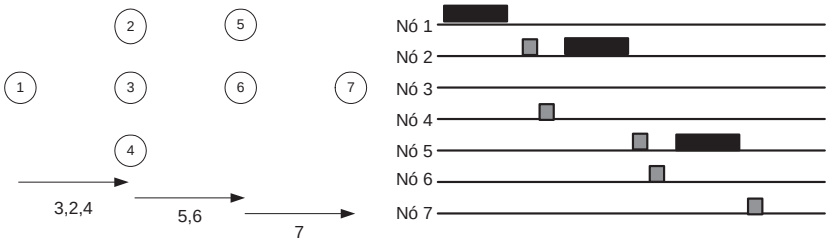

Figura 2.2: Exemplo da operação do protocolo ExOR.

A Figura 2.2 apresenta um exemplo do funcionamento do ExOR. Na figura, o nó 1 deseja enviar dados para o nó 8. Para tanto, o nó 1 pode enviar os dados para os nós 2, 3 ou 4. O nó 1 envia os dados (caixa preta na linha do tempo), listando o nó 3 como preferencial e os nós 2 e 4 como segunda e terceira opção, respectivamente, como mostram as setas da figura. O nó 3 não envia um ACK (caixa cinza na linha de tempo), enquanto os nós 2 e 4 enviam um ACK confirmando a recepção. Como o nó 4 recebeu o ACK de 2, ele sabe que o nó 2 irá enviar os dados, pois o nó 2 o precede na lista de prioridades para repasse do pacote. O nó 2 identifica que deve repassar os dados pois nenhum nó com maior prioridade (neste caso, o nó 3) confirmou a recepção do pacote. O nó 2 então decide repassar o pacote, e determina utilizando um algoritmo qualquer que os dados podem ser enviados para os nós 5 ou 6, nesta ordem de prioridade. Como ambos os nós confirmam o recebimento, o nó 5 repassa a mensagem, pois este possui maior prioridade. Finalmente, o nó 7 recebe a mensagem e confirma a sua recepção.

Outro mecanismo similar, chamado CBS (*Community-Based Security*), utiliza o conceito de "comunidades", dispensando a lista de prioridades enviada pelo ExOR [48]. O estabelecimento das rotas é feito como em um protocolo tradicional, definindo um nó para cada salto da comunicação. Digamos que uma rota é definida por quatro nós, A, B, C e D, como mostra a Figura 2.3. Os vizinhos de B e C irão criar comunidades, representadas na figura pelas regiões em cinza, sendo B e C os líderes das respectivas comunidades. Os nós da comunidade, digamos a comunidade onde B é o líder, são definidos como os nós que conseguem captar as transmissões do salto anterior e posterior (na figura, o alcance da transmissão dos nós que compõem a rota é representada pelos círculos maiores), neste caso os nós A e C, e do líder, o nó B. Os membros da comunidade detectam a falha do nó líder ao verificarem que uma transmissão de A para B não foi seguida por uma transmissão de B para C. A solução evita ataques maliciosos, pois os nós da comunidade captam os quadros de confirmação do próximo salto. Isto evita que os nós da comunidade sejam enganados por um nó líder que envia informações falsas (criptografada com a chave incorreta, por exemplo), pois o receptor da mensagem não irá confirmar a sua recepção. Quando uma falha é identificada, um dos nós da comunidade toma para si a liderança, passando a repassar os dados no lugar do antigo líder.

O estabelecimento das comunidades no CBS não requer o envio de mensagens adicionais ou a adição de cabeçalhos nas mensagens, como acontece no ExOR. Segundo os autores, o CBS pode ser empregado tanto em protocolos reativos como proativos, e armazena estados nos nós somente para os fluxos em andamento. Os protocolos ExOR e CBS, entretanto, requerem que os nós escutem o meio de forma promíscua. Assim, os nós terão menos oportunidades para desligar o seu rádio, aumentando o consumo de energia da rede. Além disso, os nós da comunidade ou da lista de prioridade devem empregar *buffers* de pacotes maiores, pois devem armazenar uma cópia dos seus pacotes e dos pacotes endereçados a outros nós da comunidade ou da lista de prioridades. Isto pode ser muito custoso em RSSF, dado a memória limitada dos nós sensores.

Dada a ocorrência de uma rota falha, é necessário identificar uma rota alternativa. Vieira

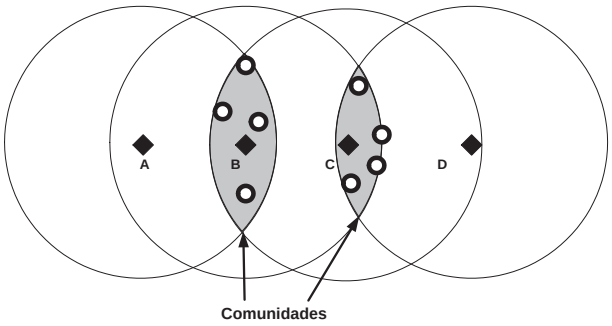

Figura 2.3: Comunidades criadas pelo protocolo CBS para aumentar a tolerância a falhas das rotas.

et al. identificaram duas soluções para a falha de nós devido ao esgotamento de energia [75]. Na primeira, o PA roda um *software* chamado de *Smart-Sink*, que reconstrói uma rota sempre que detecta que um dos nós que a compõem possui baixa energia residual. Na segunda solução, chamada de *lista de padrastos*, um nó possui uma lista de rotas sobressalentes ao PA. No caso de falha na rota padrão, uma das rotas sobressalentes é utilizada. Ambas as abordagens funcionam apenas para falhas locais, e necessitam de um mecanismo de detecção de falhas, que não é tratado no artigo.

Como as falhas de nós são frequentes, RSSF costumam empregar mais nós que o necessário para cobrir toda a área de comunicação [56]. Assim, caso um nó falhe, existirão outros nós que poderão assumir a função do nó falho. Este excesso de nós, entretanto, pode deteriorar o funcionamento da rede devido a um maior número de transmissões de dados. Para solucionar este problema, são utilizados protocolos de controle de topologia. Estes protocolos desligam os nós sensores que não realizam funções críticas na rede (por exemplo, não participam do repasse de dados ou não acrescentam informações relevantes à observação atual da rede) [4, 13, 56]. Como veremos no capítulo 3, o protocolo proposto procura minimizar o número de nós que repassam dados, para que protocolos de controle de topologia possam desligar um maior numero de nós sensores.

Outra maneira de tornar um sistema robusto é a prevenção de falhas. Esta abordagem é pouco considerada em RSSF, pois os dispositivos são de baixo custo e possuem capacidades restritas, dificultando o uso de mecanismos para diagnóstico da saúde de um nó. Szewczyk et al. identificaram que a falha de nós é frequentemente precedida de leituras errôneas no sensor de humidade [71]. Esta constatação mostra que algumas falhas podem ser previstas utilizando regras simples.

2.5 Conclusão

Este capítulo apresentou uma revisão da literatura de roteamento em redes ad hoc sem fio e redes de sensores sem fio. Mostramos que as MANETs são redes sem fio onde não há uma infra-estrutura prévia e que em geral cada nó da rede possui os seus interesses. Assim, os protocolos em MANETs são desenvolvidos para suportar qualquer tipo de tráfego. Nas RSSF, por outro lado, os nós cooperam para atender a uma única aplicação. Desta forma, podemos empregar certas otimizações em RSSF que levam em consideração características do tráfego da rede, o que não é possível em MANETs.

As RSSF podem ser classificadas quanto a características como o modelo de entrega de dados e os fluxos de dados empregados. O primeiro caracteriza a frequência do envio de dados ao PA e quem decide que o dado deve ser enviado ao PA; o próprio PA ou os nós sensores. Já o fluxo de entrega de dados identifica como os pacotes fluem na rede classificando os fluxos de dados quanto ao número e natureza dos destinatários. Em seguida apresentamos os principais protocolos de roteamento para RSSF, identificando quais são os fluxos de dados e o modelo de entrega de dados empregados. Verificamos que nenhum protocolo que suporte fluxos de dados baseado em disseminação contínua de dados apresenta mecanismos simples de configuração.

Devido à diversidade de falhas silenciosas a que as RSSF estão expostas, construímos um modelo de falhas que classifica as diversas falhas levantadas em apenas três tipos, simplificando enormemente a avaliação do protocolo proposto. Utilizamos este modelo no capítulo seguinte para avaliar o comportamento do PROC frente a falhas.

A seguir discutimos como os protocolos existentes procuram contornar a ocorrência de falhas. O primeiro método para contornar a falha é a recriação periódica das rotas existentes. Outra alternativa é o uso de múltiplas rotas, para que os pacotes possuam uma maior probabilidade de serem recebidos corretamente. Como o custo de envio por múltiplas rotas pode ser muito alto, podemos utilizar somente uma rota, entretanto a sua construção deve considerar a qualidade dos enlaces utilizados. Finalmente, podemos utilizar a escuta promíscua para detectar falhas em nós que compõem uma rota. Os nós próximos à rota escutam o meio para verificar se os nós da rota estão operando normalmente. Quando ocorre uma falha, os vizinhos do nó falho detectam o evento e o substituem.

Utilizamos a classificação e a análise dos protocolos apresentada neste capítulo para guiar a implementação do PROC. A partir desta, determinamos os fluxos de dados que o PROC deve suportar e definimos quais seriam os mecanismos de tolerância a falhas utilizados, como mostramos no capítulo seguinte.

Capítulo 3

O protocolo PROC

Neste capítulo descrevemos o protocolo proposto, denominado PROC (*Proactive ROuting with Coordination*). A Seção 3.1 apresenta os requisitos funcionais e não funcionais do protocolo, as premissas utilizadas na especificação do protocolo e as suas aplicações alvo. A Seção 3.2 apresenta em detalhes o funcionamento do PROC e descreve os algoritmos que o compõem. Em seguida, apresentamos na Seção 3.3 as regras de aplicação, exemplificando como estas podem ser utilizadas por *softwares* de aplicação ou outros protocolos para otimizar o funcionamento do PROC. Em seguida, a Seção 3.4 apresenta os mecanismos de tolerância a falhas utilizados no PROC para aumentar a sua resiliência a falhas. Finalmente, a Seção 3.5 apresenta uma modelagem em grafos do problema de roteamento resolvido pelo *backbone* do PROC, bem como uma análise do custo assintótico do protocolo.

3.1 Premissas e Requisitos do Protocolo

Para o desenvolvimento de um protocolo otimizado para redes de disseminação contínua de dados, definimos um conjunto de premissas sobre características da rede e o tipo de tráfego que esta carrega. Estas premissas, que são apresentadas e justificadas a seguir, foram definidas com o objetivo de simplificar o protocolo e otimizar o funcionamento do PROC para os cenários mais comumente encontrados em redes de disseminação contínua de dados.

O Ponto de acesso (PA) é fixo: Embora existam aplicações de RSSF onde o PA é móvel, por exemplo em um robô ou dirigível se movimentando pela região recolhendo informações produzidas pelos nós, estas redes são pouco frequentes. Por esta razão, o PROC assume que o PA é fixo, simplificando o processo de criação e manutenção de rotas.

A mobilidade dos nós é baixa: Pode ser entendida como a movimentação ocasional e por pequenas distâncias, que pode ocorrer em situações de tremores de terra, movimentação ocasionada por animais e deslizamentos, por exemplo. Como estes eventos tendem a ocorrer com uma frequência insignificante, os nós serão estáticos na maioria do tempo. Optamos por

considerar apenas redes estáticas, pois protocolos de roteamento que permitem a mobilidade dos nós são mais custosos e complexos. Além disso, a mobilidade dos nós ocorre em poucas aplicações de RSSF. A adição de mecanismos que tratam da mobilidade de nós diminuiria o desempenho do PROC no tipo mais frequente de RSSF, que são as redes estáticas. Assim, optamos por implementar um protocolo que opera somente em redes estáticas.

Nós trocam dados apenas com o PA e vizinhos a um salto de distância: Por motivos de escalabilidade, os nós sensores em geral comunicam-se apenas com os seus vizinhos a um salto de distância [26]. Esta característica da comunicação é necessária já que a largura de banda dos rádios é limitada e a memória restrita impede a criação de grandes tabelas de roteamento. Além disso, devido ao emprego de um grande número de nós, protocolos específicos para RSSF tendem a limitar a comunicação entre nós somente aos nós vizinhos a um salto para aumentar a sua escalabilidade [35]. Ao limitar comunicação dos sensores aos seus vizinhos e ao PA, o PROC diminui o tamanho da tabela de rotas drasticamente, pois é necessário somente armazenar a rota até o PA.

A comunicação do PA para os nós sensores ocorre por fundação: Assumimos que toda a comunicação entre o PA e os nós sensores será feita por inundação. Essa opção impede o uso de protocolos que requerem a comunicação individual do PA com os nós sensores. Como o PA usualmente se comunica com regiões da rede ou com toda a rede [26, 35], acreditamos que esta restrição não limita a aplicabilidade do protocolo proposto.

Em seguida, levantamos os requisitos funcionais e não funcionais que um protocolo de roteamento para RSSF deve atender em redes de disseminação contínua de dados. Devido à existência de poucas informações sobre RSSF em produção, realizamos o levantamento de requisitos pela análise da literatura de RSSF. A partir deste levantamento, obtivemos os requisitos descritos a seguir.

Formação de rotas do PA para os nós sensores: O protocolo deve formar rotas de um nó até o PA, pois os nós irão, em geral, enviar os dados produzidos diretamente ao PA.

Comunicação com os nós vizinhos: A comunicação com os nós a um salto de distância é necessária para que haja processamento colaborativo na rede e para a implementação de protocolos de camadas superiores.

Fusão ou agregação de dados: O protocolo deve permitir que um pacote seja descartado ou modificado a cada salto da comunicação, para que seja realizada a fusão ou agregação de dados [5]. Dessa forma, o PROC deve prover mecanismos para que protocolos ou o *software* da aplicação armazenem, processem, concatenem e descartem pacotes.

Mecanismos de configuração simples, alteráveis em tempo de execução: O protocolo deve permitir que o *software* da aplicação ou outros protocolos alterem o comportamento do protocolo de roteamento, para que este opere de acordo com as necessidades da aplicação. Como o comportamento da rede e da aplicação pode variar devido às mudanças no ambiente, os mecanismos de configuração devem permitir que o protocolo seja reconfigurado durante a sua

operação. A configuração ainda deve ser simples, para que a interação entre protocolos ocorra de forma explícita e livre de erros de programação.

Tolerância a falhas: Falhas são uma constante em RSSF. Sendo assim, o protocolo de roteamento deve ser tolerante a falhas, diminuindo ou eliminando o efeito das falhas na sua operação. Devido à grande variedade de falhas a que as RSSF estão sujeitas, faz-se necessário delimitar um conjunto de falhas, englobando as falhas mais corriqueiras.

Bom Desempenho: O protocolo deve apresentar um desempenho aceitável em um conjunto de métricas como taxa de entrega, latência fim a fim (tempo de propagação da mensagem do seu produtor até o PA) e consumo médio de energia dos nós. Estas métricas são usualmente utilizadas para mensurar o desempenho de protocolos de roteamento em redes sem fio,

Tendo definido os requisitos e premissas sobre as quais o protocolo proposto se baseia, iremos agora descrever o seu funcionamento.

3.2 Funcionamento do Protocolo

O PROC é um protocolo de roteamento pró-ativo, que constrói uma estrutura de roteamento chamada de *backbone*. O *backbone* é composto por um conjunto de nós, chamados *coordenadores*, que propagam a informação em direção ao ponto de acesso (PA). O *backbone* é uma árvore, tendo o PA como sua raiz. Os nós que não fazem parte do *backbone*, chamados de *nós folha*, comunicam-se diretamente com um nó do *backbone* (chamados de *nós coordenadores*). Os nós precisam conhecer apenas o nó pai no *backbone*, o qual repassa os dados recebidos em direção ao PA. Essa em árvore estrutura permite o uso de algoritmos de controle de topologia, onde os nós folha são colocados em modo de baixo consumo de energia. Como o *backbone* garante a cobertura de rádio na rede, protocolos de controle de topologia precisam apenas considerar a cobertura de sensoriamento [56]. Para aumentar a economia de energia, o processo de criação do *backbone* busca minimizar o número de nós coordenadores. O *backbone* é reconstruído periodicamente, em intervalos regulares de tempo chamados de *ciclos*.

A criação do *backbone* é um processo de duas fases. Inicialmente, os nós se auto-elegem coordenadores utilizando um processo probabilístico. Cada nó possui uma certa probabilidade de tornar-se coordenador, que é calculada pelo *software* de aplicação por meio de regras. As regras podem utilizar informações específicas da aplicação, e são tratadas na Seção 3.3. No entanto, a eleição dos nós coordenadores pode não estabelecer um *backbone* completo. A segunda fase da criação do *backbone* complementa a estrutura formada, adicionando mais nós ao *backbone*. A Figura 3.1 mostra as fases da criação do *backbone*. Os nós pretos representam coordenadores e os nós brancos representam nós comuns. Inicialmente, a eleição de coordenadores seleciona um conjunto de nós a partir das regras da aplicação, conforme mostra a Figura 3.1(a). Esta primeira fase é chamada de "Eleição de coordenadores". Em seguida, é executada a segunda fase, denominada "Complementação do *backbone*", que garante a existência de uma rota de cada nó até

o PA. A Figura 3.1(b) mostra um nó folha (nó branco) e sua área de alcance do rádio. Como não existe nenhum nó coordenador na sua vizinhança, o nó deve selecionar um dos seus vizinhos (nós cinzas) para se tornar um coordenador. Em seguida, as rotas são estabelecidas, como mostrada na Figura 3.1(c).

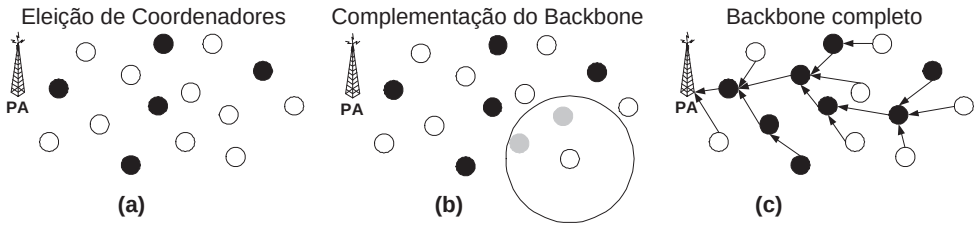

Figura 3.1: O processo de criação do *backbone*.

A seguir apresentamos os algoritmos e a operação do protocolo PROC. Os algoritmos desenvolvidos utilizam apenas informações da vizinhança do nó e o seu estado atual, reduzindo assim a quantidade de memória utilizada. As mensagens de roteamento carregam dados sobre o estado atual do nó emissor, entre eles, o papel do nó na rede (coordenador ou nó folha), o ciclo atual, a energia residual e a distância em saltos até o PA. Esses dados atualizam as informações sobre a vizinhança do nó. O PROC utiliza algoritmos diferentes para o PA e os nós sensores.

3.2.1 Ponto de Acesso

O ponto de acesso é responsável por iniciar um ciclo, e por consequência iniciar o processo de criação do *backbone*. A criação do *backbone* permite que o consumo de energia dos nós seja balanceado, e corrige rotas danificadas. O início do ciclo é sinalizado com o envio de uma mensagem de sincronização (M_{Sync}) pelo PA (linha 7 do Algoritmo 1; por simplicidade, nos referimos às linhas dos algoritmos como l ao longo do texto). O *backbone* é construído tendo como métrica a distância dos nós ao PA. Assim, o PA envia uma mensagem anunciando a sua distância como zero, como mostrado no Algoritmo 1.

```
 1: procedure BaseStation(CycleTime)
 2:     nextCycle ← 0;                          // Número do ciclo atual
 3:     loop                                    // Iniciando um novo backbone
 4:         M_Sync.cycle = nextCycle;
 5:         M_Sync.hops = 0;
 6:         M_Sync.coord = true;
 7:         send(M_Sync, BROADCAST);
 8:         nextCycle ← nextCycle + 1;
 9:         wait CycleTime seconds;
10:     end loop
11: end procedure
```

Algoritmo 1: Criação de rotas a partir do PA.

3.2.2 Nós Sensores

O algoritmo empregado pelos nós sensores utiliza uma abordagem baseada em eventos; as mudanças no estado do nó ocorrem quando os nós recebem uma mensagem de roteamento. O funcionamento dos nós sensores é mostrado no Algoritmo 2.

A construção do *backbone* é iniciada com o recebimento de uma mensagem de sincronização M_{Sync} pelos nós (l.6-24), e compreende duas fases: a eleição dos coordenadores e a complementação do *backbone*. A primeira fase ocorre quando o nó recebe a mensagem M_{Sync}, executando o processo de eleição de coordenadores (l.8-15). Na segunda fase, os nós examinam se o *backbone* está completo e indicam, quando necessário, outros nós para se juntarem ao *backbone* (l.16-23).

Eleição de coordenadores: É a primeira fase da criação do *backbone*. Esse algoritmo utiliza regras providas pela aplicação para determinar se o nó será coordenador ou não. O *software* de aplicação associa um valor V no intervalo $[0, V_{max}]$ para o nó (l.10). A partir de V, o nó calcula a probabilidade de tornar-se coordenador, dada por $p = \frac{V}{V_{max}}$. Em seguida, um número aleatório entre 0 e 1 é gerado. Caso este seja menor que p, o nó será coordenador neste ciclo (l.11). Ao fim desta fase, o nó envia uma mensagem M_{Sync} para a rede, reportando o seu novo estado para a vizinhança (l.13-15). Esta informação é preenchida pela função $CurrentNodeState$ (l.35-37).

Complementação do Backbone: Após o recebimento da primeira mensagem M_{Sync} de um ciclo, os nós aguardam por um período aleatório e determinam se o *backbone* está completo. Para isto, os nós verificam se a rota escolhida pelo algoritmo de estabelecimento de rotas passa por um nó coordenador (l.16-23). Caso a rota não passe por um coordenador, o nó envia uma mensagem de indicação de coordenador (M_{Coord}) para o seu novo nó pai (l.18-23). O receptor da mensagem M_{Coord} se torna coordenador, e envia uma mensagem M_{Sync} para os seus vizinhos, propagando seu novo estado (l.28-30). O tempo aleatório de espera antes da verificação dos nós pais evita que mais de um nó seja indicado coordenador, permitindo assim que menos nós coordenadores sejam indicados.

3.2.3 Estabelecimento de Rotas

Sempre que um nó recebe uma mensagem M_{Sync}, o algoritmo de estabelecimento de rotas é executado, elegendo um nó pai utilizando o Algoritmo 3. Para economizar energia, apenas uma mensagem M_{Sync} é propagada por ciclo de formação do backbone.

O algoritmo de estabelecimento de rotas utiliza informações da vizinhança, que são armazenados em uma tabela, representada como $Neighbors$ nos algoritmos. Para cada nó vizinho i, o PROC armazena uma quádrupla $Neighbors_i = (hops, curCycle, coord, energy)$. Estes campos são descritos na Tabela 3.1. O campo *hops* armazena a distância em saltos do nó vizinho até o PA, o campo *coord* indica se o nó é coordenador ou não, e o campo *energy* armazena a energia residual do nó. O campo *curCycle* é usado para garantir que a informação armazenada ainda

```
1:  parent ← nil;                                                    // Nó pai na árvore
2:  curCycle ← −1;                                                   // Ciclo atual
3:  Neighbors ← ∅;                                                   // Estado dos vizinhos
4:  coordinator ← false;
5:  procedure SensorNodes( )
Require: receive(M_Sync);                                            // Eleição de coordenadores
6:      Neighbors ← Neighbors ∪ M_Sync.{hops, cycle, coord, energy}; // Adicionando novos dados da topologia.
7:      parent ← UpdateRoute();                                      // Seleção do nó pai
8:      if curCycle < M_Sync.cycle then
9:          curCycle ← M_Sync.cycle;
10:         prob ← App.computeAppRules(Neighbors, parent);           // Calculando a prob. do nó ser coordenador
11:         coordinator ← ( rand() < prob );
12:         App.newCycle(curCycle, coordinator);                     // Início de um novo ciclo
13:         CurrentNodeState(M_Sync);
14:         App.sendRoutePacket(M_Sync,BROADCAST);                   // Adicionando dados da aplicação
15:         send(M_Sync,BROADCAST);                                  // Disseminação do novo estado
16:         wait BackoffTime();
17:         parent ← UpdateRoute();                                  // Definindo o nó pai
18:         if parent.coordinator = false then                       // Força o nó pai a ser coordenador
19:             CurrentNodeState(M_Coord);
20:             App.sendRoutePacket(M_Sync,BROADCAST);               // Inserindo dados da aplicação
21:             send(M_Coord, parent);
22:             Neighbors_parent.coord ← true;
23:         end if
24:     end if
Require: receive(M_Coord);                                           // Nó designado coordenador
25:     Neighbors ← Neighbors ∪ M_Sync.{hops, cycle, coord, energy}; // Adicionando novos dados da topologia.
26:     coordinator ← true;
27:     App.forcedCoordinator();
28:     CurrentNodeState(M_Sync);
29:     App.sendRoutePacket(M_Sync,BROADCAST);                       // Adicionando dados da aplicação
30:     send(M_Sync,BROADCAST);
Require: receive(M_Data);                                            // Envio de dados para o nó pai
31:     if App.forwardData(M_Data) = true then                       // Fusão ou descarte de dados pela aplicação
32:         send(M_Data, parent.address);
33:     end if
34: end procedure
35: procedure CurrentNodeState(Message)
36:     Message.{hops, cycle, coord, energy} ← {parent.hops + 1, curCycle, coordinator, getEnergy()};
37: end procedure
```

Algoritmo 2: Algoritmo empregado nos nós sensores.

é recente. A atualização da quádrupla é feita utilizando as informações obtidas dos pacotes de roteamento recebidos ($l.6$, $l.25$ do Algoritmo 2).

A operação do algoritmo de estabelecimento de rotas depende do papel de cada nó (coordenador ou folha). Se o nó é coordenador, ele seleciona seu nó pai entre os nós vizinhos coordenadores com a menor distância até o PA. Caso não existam nós coordenadores na vizinhança, é selecionado o nó com a menor distância até o PA. A seleção de rotas pelo critério de menor distância

Campo	Descrição
hops	Distância do PA, em saltos
curCycle	Garante que o dado é atual
coord	Indica se o nó vizinho é coordenador
energy	Energia residual

Tabela 3.1: Campos armazenados pelo PROC em cada quádrupla da tabela de vizinhos.

evita ciclos, como exemplificado na Figura 3.2. Essa figura mostra dois nós coordenadores (círculos pretos) e um nó folha (círculo branco), e suas respectivas tabelas de distância até o PA. Na Figura 3.2(a) mostramos as rotas que seriam selecionadas se priorizássemos o uso de coordenadores ao invés da distância ao PA. Os nós 2 e 3 anunciam que estão à distância $i+1$ do PA, por existir um caminho pelo nó 1 com distância ao PA igual a i. Entretanto, como os nós priorizam o repasse entre nós coordenadores, o nó 2 escolhe o nó 3 como seu pai, enquanto o nó 3 escolhe o nó 2 como seu pai, o que gera um ciclo. Na Figura 3.2(b), mostramos as rotas que o PROC seleciona, que utilizam o critério de menor distância. Como visto, esta escolha evita a criação de ciclos no roteamento. Caso dois ou mais nós possuam a mesma distância ao PA, o nó com maior energia residual é selecionado como nó pai. Somente nós coordenadores podem selecionar um nó folha como seu pai. O nó folha, entretanto, é imediatamente indicado a se juntar ao *backbone*, tornando-se assim um nó coordenador.

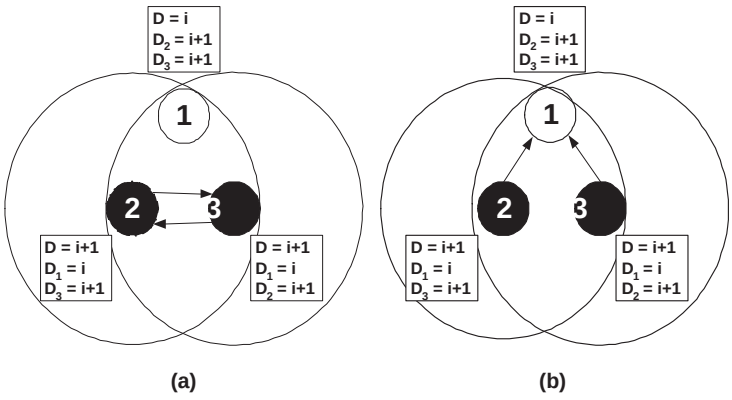

Figura 3.2: Evitando ciclos no roteamento utilizando o critério de menor distância.

Nós folhas, por outro lado, selecionam rotas de forma diferente. Esses nós procuram em sua vizinhança um nó coordenador com a menor distância até o PA. Se nenhum vizinho coordenador é encontrado, o nó seleciona o seu vizinho com a menor distância até o PA, e envia uma mensagem M_{Coord}, que o força a se tornar um nó coordenador. Nós folhas podem se dar ao luxo de escolher um vizinho coordenador como pai, e não o seu vizinho de menor distância ao PA pois, ao contrário dos nós coordenadores, nós folha não repassam dados de outros nós. Caso haja dois nós com a mesma distância ao PA, o nó com maior energia é escolhido.

```
1: procedure UpdateRoute( )
2:     if coordinator = true then
3:         return min(Neighbors, {hops, coord, 1/energy});
                    // Vizinho com menor distância em saltos e maior energia residual
4:     else
5:         return min(Neighbors, {coord, hops, 1/energy});
       // Vizinho, de preferência coordenador, com menor distância em saltos e maior
       energia residual
6:     end if
7: end procedure
```

Algoritmo 3: Algoritmo de estabelecimento de rotas.

3.3 Regras de Aplicação

As regras de aplicação do PROC proporcionam uma interface entre o *software* da aplicação e o protocolo de roteamento, permitindo que o PROC se adapte às características da aplicação e da rede. Essa estratégia otimiza o processo de estabelecimento de rotas. Cada regra é implementada como uma função, e o seu valor de retorno definirá a probabilidade do nó rotear dados no ciclo atual, o que possibilita à aplicação direcionar o algoritmo de estabelecimento de rotas. Logo, o uso de regras confere ao PROC um mecanismo de adaptação e economia de energia, permitindo o ajuste da política de estabelecimento de rotas (a regra de aplicação) em tempo de execução, de acordo com as necessidades da aplicação.

O *software* de aplicação pode usar qualquer informação disponível como entrada para o algoritmo que implementa as regras, tais como configuração dos nós e de outros protocolos, estado atual da rede, ou mesmo dados de sensores e atuadores. As regras podem ser altamente especializadas, podendo até se aproveitar de características de uma instância específica de uma aplicação, por exemplo, "A rede de monitoração ambiental instalada no Maine". Fica a cargo do projetista definir quais serão as características a serem consideradas na regra da aplicação. O PROC permite que o *software* de aplicação envie dados embutidos nos pacotes de roteamento, que podem ser utilizados para trocar informações úteis ao cálculo das regras.

O uso de regras não é mandatório no PROC. Assim, caso o projetista não deseje utilizar regras, serão utilizadas as regras genéricas descritas a seguir, que procuram minimizar o consumo de energia e a latência do envio de dados. Para que o projetista não tenha que se preocupar com aspectos de roteamento na implementação das regras, o PROC garante que rotas serão estabelecidas mesmo que a aplicação não aponte nenhum coordenador. Esta garantia é provida pela fase de complementação do *backbone*.

3.3.1 Regras Genéricas de Aplicação

A primeira regra é uma adaptação da eleição de líderes de grupo do protocolo LEACH [79], que tem como objetivo aumentar o tempo de vida da rede. Nesta regra, os nós sensores que

recentemente foram coordenadores possuem uma menor probabilidade de exercer o papel de coordenador nos ciclos seguintes, distribuindo a energia consumida de forma uniforme. No LEACH, a probabilidade que os nós são definidos líderes de grupo, chamada de $T(n)$, é calculada pela fórmula abaixo:

$$T(n) = \begin{cases} \frac{P}{1-P\times(r\times(mod\frac{1}{P}))} & \text{se } n \in G \\ 0 & \text{caso contrário} \end{cases}$$

Onde r indica o número de ciclos desde a última vez que este nó foi líder de grupo, P é a porcentagem desejada de líderes de grupo e G é o conjunto de nós que não foram coordenadores nos últimos $\frac{1}{P}$ ciclos do LEACH. Esta fórmula apresenta um comportamento exponencial em relação a valores crescentes de r. Como a fórmula apresenta operações de multiplicação e divisão de valores de ponto flutuante, realizamos diversas simplificações à fórmula, mantendo apenas o seu comportamento exponencial, como mostramos abaixo.

$$T(n) = \begin{cases} 2^{r-B} - 1 & \text{se } r < B \\ 1 & \text{caso contrário} \end{cases}$$

Nesta fórmula, B indica o tamanho em bits da variável utilizada para registrar a probabilidade de que o nó será coordenador neste ciclo. Na implementação do TinyOS, por exemplo, B é definido como 16, pois o gerador de número aleatórios da plataforma gera valores de 16 bits. Assim, caso o nó tenha acabado de ser eleito coordenador, ele terá a probabilidade de 2^{-B} de ser coordenador no próximo ciclo, 2^{-B+1} no próximo ciclo, e assim em diante. Esta fórmula pode ser facilmente implementada com implementações de deslocamento de bits e operações lógicas OU e E. Se representamos a probabilidade de 100% como todos os bits da variável em 1, para reduzir a probabilidade para aproximadamente 50% devemos definir o bit mais significativo da variável como 0 e os bits restantes em 1. Para diminuir esta probabilidade pela metade, devemos mudar o bit mais significativo que esteja em 1 para 0, e assim sucessivamente, até conseguirmos a probabilidade desejada.

A segunda regra visa ajustar o número de coordenadores de acordo com a densidade de cada área da rede. Estatisticamente, em áreas mais densas, a probabilidade de se ter mais coordenadores é maior que em áreas menos densas. Para manter a densidade de coordenadores por região próxima de um valor ideal, medido empiricamente para cada rede, aplicamos a seguinte regra: A probabilidade de um nó se tornar coordenador é inversamente proporcional ao número de nós sensores na sua vizinhança. Verificamos empiricamente que a densidade de coordenadores na rede se torna mais uniforme com o uso dessa regra.

Na terceira regra os nós sensores próximos do PA possuem maior probabilidade de se tornarem coordenadores. Os nós próximos do PA repassam mais dados que os nós mais distantes, pois suportam todo o tráfego da rede, consumindo mais energia. Uma forma de amenizar o problema

é aumentar o número de nós que repassam dados na vizinhança do PA, distribuindo o fluxo de dados entre mais rotas. Para tanto, mais nós coordenadores são eleitos em regiões próximas ao PA.

Além das regras descritas, cada aplicação pode possuir características específicas, que podem ser utilizadas no PROC via regras de aplicação para otimizar o seu funcionamento. A seguir, apresentamos algumas situações onde o uso de regras do PROC poderia contribuir para simplificar o funcionamento da rede ou para aumentar o seu desempenho.

3.3.2 Exemplos de Regras de Aplicação

As regras podem aproveitar a heterogeneidade do *hardware* e da capacidade da bateria. Podemos, por exemplo, utilizar nós com maior energia ou mesmo nós fixos, que não possuem restrições de energia, para repassar os dados. Nós com maior alcance de comunicação também seriam ótimos candidatos a coordenadores, pois permitem menor latência na comunicação.

As regras também podem ser utilizadas para modificar a topologia da rede de acordo com a especialidade de cada nó. Em aplicações de detecção de intrusos, é natural imaginar que alguns nós irão executar sistemas de detecção de nós maliciosos (ou IDS, de *Intrusion Detection System*). A tarefa de um nó com IDS seria facilitada se este fosse coordenador, uma vez que mais tráfego passaria por ele, permitindo assim diagnósticos mais precisos.

O mesmo ocorre em redes onde o *software* de aplicação emprega o paradigma de fluxos infinitos de dados (*data streams*). Neste paradigma, os dados são tratados como um fluxo infinito de dados, que deve ser tratado em tempo real [7]. Uma maneira de facilitar a análise dos dados é o uso de técnicas de pré-processamento nos nós sensores, e os melhores candidatos para este papel são os nós coordenadores, por lidarem com uma maior quantidade de dados.

Os nós coordenadores ainda podem ser utilizados para implementar protocolos de controle de topologia. O problema do controle de topologia em RSSF consiste em desligar nós que não produzem informação útil à rede, pois existem um ou mais nós que geram a mesma informação dos nós desligados [56]. O controle de topologia deve considerar dois fatores, a cobertura de sensoriamento e a cobertura de rádio. A primeira garante que toda a área monitorada está coberta por um sensor ativo, e a segunda garante que existe a comunicação entre os nós operantes. Caso o controle de topologia desligue nós que estão sendo utilizados no roteamento, as rotas deverão ser recriadas. Assim, em geral, os protocolos de controle de topologia determinam quais nós irão operar, e em seguida o roteamento é executado sobre os mesmos. O PROC poderia simplificar o desenvolvimento de protocolos de controle de topologia com a utilização das regras. O algoritmo de controle de topologia definiria como coordenadores os nós que garantem a cobertura de sensoriamento. O PROC então garantiria a cobertura de rádio, ao adicionar mais nós ao *backbone* durante a fase de complementação do *backbone*. Finalmente, os nós folhas seriam desligados, pois o *backbone* criado garante tanto a cobertura de sensoriamento quanto a cobertura

de comunicação.

3.4 Mecanismos de Tolerância a Falhas

O PROC implementa dois mecanismos de tolerância a falhas, que são a reconstrução periódica de rotas e a monitoração ativa do nó pai. Estes mecanismos procuram minimizar o efeito das falhas silenciosas (discutidas na Seção 2.4.2) na operação do protocolo proposto.

Reconstrução periódica de rotas: A cada nova recriação do *backbone*, as rotas são completamente reconstruídas, utilizando somente os nós ativos. Isto ocorre pois somente os nós que repassam as mensagens M_{Sync} no ciclo corrente são considerados na formação de rotas. Este é o principal mecanismo de tolerância a falhas do PROC.

O intervalo entre cada recriação de rotas pode ser ajustado de acordo com o grau de tolerância a falhas e o consumo de energia desejados. Por ser um processo que envolve o envio de mensagens para toda a rede, a recriação de rotas envolve o envio de um grande número de mensagens. Como a energia é um recurso escasso em RSSF, este processo deveria ocorrer com a menor frequência possível. Entretanto, quanto mais frequente a atualização de rotas, mais rapidamente as falhas serão contornadas. Analisamos este compromisso nas simulações mostradas na Seção 4.5.1.

Monitoração ativa do nó pai: O PROC utiliza um mecanismo de monitoração de atividade equivalente a mensagens de *ping-pong*. Os quadros de confirmação (ACK) do protocolo de enlace são utilizados para monitorar o funcionamento do próximo salto da rota até o PA. Ao enviar um quadro de dados (equivalente à mensagem de *ping*) ao nó pai, o PROC aguarda que o protocolo MAC acuse o recebimento bem-sucedido do pacote, indicado pela recepção de uma confirmação (equivalente à mensagem de *pong*). A falha na recepção da mensagem é indicada pela não recepção de uma mensagem de confirmação após um tempo máximo pré-determinado, ou *timeout* da confirmação. Este mecanismo, mostrado na Figura 3.3 por uma máquina de estados, registra quantas mensagens seguidas de confirmação foram perdidas. Quando o número de mensagens perdidas ultrapassa um limite determinado, chamado de limiar, o protocolo considera que o nó pai está falho. O nó pai é retirado da lista de vizinhos e as rotas são recalculadas. Este mecanismo permite que o nó identifique rotas falhas antes da próxima reconstrução de rotas, aumentando a resiliência do protocolo. Como este mecanismo opera como suporte à recriação periódica das rotas, o número de mensagens não confirmadas é zerado a cada novo ciclo.

Em RSSF, devido às limitações de banda e energia, protocolos MAC evitam o uso de quadros de confirmação. Verificamos, entretanto, que os principais protocolos MAC para RSSF utilizam a confirmação de quadros [66, 74, 78, 81]. Estudos de Polastre et al. mostraram que o aumento da latência e do consumo de energia são mínimos para o protocolo B-MAC quando os quadros de confirmação são ativados [66].

A não recepção de um ACK se dá por dois motivos: A mensagem original não foi recebida pelo emissor, ou esta foi recebida com erros na transmissão. Assim, devemos determinar se um

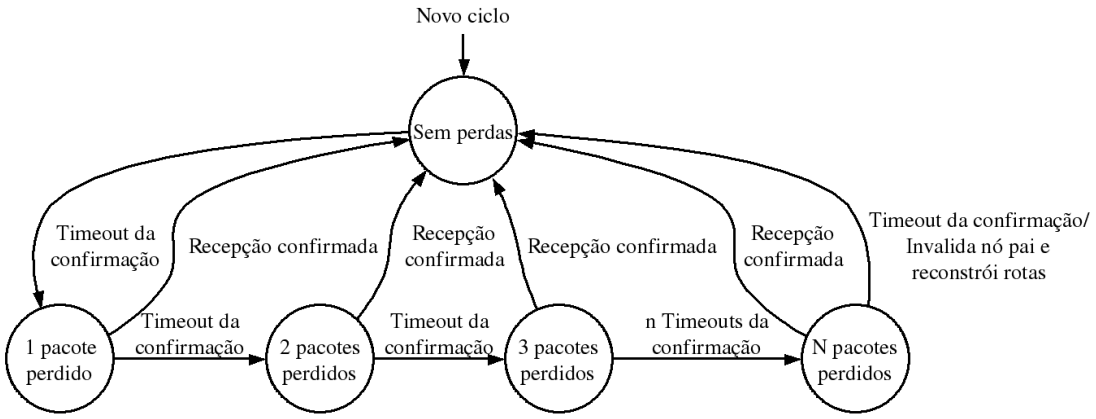

Figura 3.3: Máquina de estados finitos para o mecanismo de monitoração ativa do nó pai.

ACK não foi recebido devido a uma falha no destinatário ou devido a um erro de transmissão. Utilizamos a probabilidade de que um quadro seja perdido (PER – *Packet Error Rate*) para determinar o número mínimo de quadros consecutivos não recebidos que irão indicar a falha de um nó. O PER é função da taxa de erro por bit (BER –*Bit Error Rate*)[1], que é comumente encontrada nos manuais dos transceptores. Devemos escolher um valor para o limiar tal que a probabilidade da ocorrência de falsos positivos (erros de retransmissão considerados como falhas de nós) seja próxima de zero. Esta probabilidade, que chamamos de P, é dada pela Equação 3.1, onde T_{Quadro} indica o tamanho do quadro em bits.

$$P = PER(T_{Quadro})^{limiar} \tag{3.1}$$

A escolha de um limiar alto faz com que uma grande quantidade de dados precise ser recebida até que a falha seja percebida, mas ao mesmo tempo a certeza da falha é maior, enquanto um limiar baixo aumenta a quantidade de falsos positivos, entretanto o tempo necessário para a detecção de falhas é menor. Utilizamos esta equação na avaliação do protocolo PROC para determinar o valor mínimo do limiar.

3.5 Análise do Protocolo

Esta seção apresenta uma análise matemática da complexidade e do custo do PROC. Primeiro, modelamos a construção do *backbone* como um grafo e em seguida apresentamos como o *backbone* construído no PROC soluciona este problema. A segunda parte desta seção os algoritmos em termos de consumo de memória, número de operações do processador e quantidade de mensagens enviadas, utilizando a notação "big-Oh".

[1] $PER(t) = 1 - (1 - BER)^t$, onde t é o tamanho do quadro enviado.

3.5.1 Modelagem do Backbone

O protocolo PROC procura minimizar o número de nós que participam do roteamento, para diminuir o consumo de energia na rede. Nós que não repassam dados podem ser desligados a qualquer momento, pois não há nenhum nó que dependa deles para o repasse de dados. Assim, ao diminuir o número de nós que repassam dados, o PROC permite que uma maior quantidade de nós possa ser desligada.

Podemos analisar uma rede como um grafo, onde vértices representam nós sensores e arestas entre vértices indicam que dois nós conseguem comunicar entre si. Este problema é semelhante ao problema *Minimum Connected Dominant Subset* (ou MCDS). O MCDS é o menor subgrafo conexo onde nós fora do subgrafo estão diretamente conectados a um ou mais nós do subgrafo. Os nós do subgrafo formam um grafo conectado, ou seja, todos os nós do subgrafo possuem um caminho a todos os outros nós do subgrafo.

O MCDS é utilizado em vários protocolos de roteamento e difusão de dados em redes ad hoc [19, 77, 88]. Além de minimizar o número de nós que participam do roteamento, o MCDS simplifica a formação de rotas. Podemos utilizar o MCDS como um "backbone". Nós fora do MCDS, por construção, possuem pelo menos um nó pertencente ao MCDS como vizinho. Estes repassam as mensagens a um dos nós pertencentes ao MCDS, que se encarrega de repassar a mensagem ao destinatário. Assim, simplificamos o processo de estabelecimento de rotas, pois rotas devem ser criadas apenas entre os nós que compõem o MCDS.

Como visto anteriormente, o PROC utiliza o conceito de *backbone* na formação das rotas. O *backbone* criado, entretanto, não é um MCDS, pois o PROC não permite a comunicação entre todos os sensores. Podemos modificar o problema do MCDS para modelar o comportamento do PROC.

Primeiramente, iremos considerar um grafo direcionado. Uma aresta a_{ij}, segundo a nossa terminologia, é uma aresta que se origina no vértice i e termina no vértice j, modelando que existe uma rota do nó i ao nó j. Como o PROC permite apenas um nó "pai" por nó sensor, o grau de saída (ou *out-degree*) dos vértices do grafo é sempre igual a um. Como mencionado anteriormente, o grafo não será conexo, entretanto todos os nós devem possuir um caminho (rota) até um vértice central (o PA), que consideramos nesta modelagem como o vértice v_0. Assim, podemos modelar o problema de construção do *backbone* do PROC como o problema a seguir.

Dado um conjunto de vértices V e o conjunto A de todas as possíveis arestas entre os vértices em V, construa um grafo direcionado $G(V, A)$ com as seguintes propriedades:

1. Todo vértice contido em V deve possuir um caminho até v_0;
2. O grau de saída de todo vértice exceto o vértice v_0 deve ser um;
3. O número de vértices em V com o grau de entrada (*in-degree*) deve ser maximizado;

4. Nenhuma aresta deve ser originada em v_0.

Neste grafo, vértices com grau de entrada não nulo serão nós coordenadores, e formam o *backbone* de roteamento. Os nós restantes serão nós ordinários, e poderão ser desligados a qualquer momento. As condições 1 e 2 implicam que o grafo será acíclico, logo será uma árvore. Note que, ao contrário do MCDS, não haverá uma rota de todos os nós para qualquer outro nó. O grafo gerado somente garante que, para qualquer vértice v_i, existe um caminho $v_i \twoheadrightarrow v_0$ que passa por um ou mais vértices do grafo.

Devemos salientar que o PROC não procura somente minimizar o número de nós coordenadores. Tendo em vista um maior desempenho e economia de energia, o PROC pode selecionar um número de coordenadores maior que o mínimo, caso a aplicação assim deseje, utilizando as regras de aplicação. Outro protocolo que constrói uma estrutura semelhante ao *backbone* do PROC é o EAD (descrito no capítulo 2), que também tem como objetivo minimizar o número de nós que roteiam dados.

3.5.2 Análise de Complexidade

Esta seção apresenta uma análise da complexidade do protocolo PROC. Para esta análise, iremos considerar uma densidade média de v vizinhos, ou seja, em média um nó consegue se comunicar diretamente com v vizinhos. A título de ilustração, iremos comparar o consumo do protocolo PROC com o protocolo de controle de topologia Span, desenvolvido para redes ad hoc [13]. O protocolo Span, como o PROC, forma um *backbone*, que também é uma aproximação de um CDS (*Connected Dominating Set*), que entretanto tem por objetivo maximizar a capacidade da rede. O protocolo Span seleciona os nós que participarão do *backbone* de acordo com a sua energia residual, e implementa um rodízio de nós coordenadores para que o consumo de energia na rede seja uniforme. Utilizamos os pseudo-algoritmos apresentados em [13] na análise da complexidade do protocolo. A comparação do Span com o PROC permitirá levantar as diferenças entre protocolos de RSSF e redes ad hoc.

O primeiro aspecto que analizaremos é o consumo de memória. O protocolo PROC armazena um conjunto de $n^{\smile}$*tuplas* (descritas na Seção 3.2.3), que provêm ao protocolo uma visão do estado corrente dos nós vizinhos. Os campos que compõem as $n^{\smile}$*tuplas* e suas respectivas funções são apresentadas na Tabela 5.1. Como o PROC armazena somente a informação necessária sobre os seus vizinhos a um salto de distância, o consumo de memória do protocolo é de $O(v)$ bytes. A escolha pelo uso somente de informações locais se dá à quantidade limitada de memória disponível nos nós sensores. O protocolo Span, por outro lado, necessita de $O(V^2)$ bytes de memória, pois cada nó armazena informações de nós a até dois saltos de distância. Esta quantidade de dados armazenados é inaceitável em RSSF, como mostraremos a seguir.

Considere que cada nó sensor possui dez vizinhos. Vamos supor que a quantidade de dados armazenados para cada nó seja 7 bytes, como na implementação do PROC no sistema operaci-

onal TinyOS (veja Tabela 5.1). Nesse caso, a tabela de roteamento do PROC (composta pelas $n\check{}tuplas$) irá utilizar 70 bytes, enquanto o Span irá utilizar 700 bytes da memória. Este consumo equivale a 1% e 18% da memória disponível nos nós da plataforma Mica2 [16], respectivamente. Assim, soluções desenvolvidas para RSSF que consumam uma quantidade quadrática de memória ou superior devem ser evitadas, pois não são escaláveis. Caso se deseje diminuir ainda mais o consumo de memória do PROC, podemos armazenar um número limitado de $n\check{}tuplas$. Somente as $n\check{}tuplas$ relativas aos nós mais favoráveis ao roteamento serão armazenadas, como sugerido por Woo et al. [82].

Outra métrica de consumo que deve ser considerada em RSSF é o número de mensagens enviadas. Como mostramos no capítulo 2, o rádio é o maior consumidor de energia no *hardware* empregado pelos nós sensores. Assim, o número e tamanho das mensagens enviadas deve ser minimizado. Para cada reconstrução do *backbone*, os nós enviam no pior caso $3v$ mensagens. Cada nó envia uma mensagem M_{Sync} para propagar o início do ciclo aos seus vizinhos. Suponha que o nó não seja selecionado como coordenador, e este não possua vizinhos coordenadores. O nó, então, irá enviar uma mensagem M_{Coord} a um dos seus vizinhos, para que este mude seu estado para coordenador. Por fim, um dos vizinhos do nó decide que este é a melhor rota, e o elege coordenador. Ao ser eleito coordenador, o nó envia uma segunda mensagem M_{Sync}, para notificar seus vizinhos da sua nova função. No caso ótimo, nenhum novo coordenador deve ser indicado, assim cada nó irá enviar somente uma mensagem M_{Sync}. Desta forma, a complexidade do protocolo em termos de mensagens enviadas para cada recriação das rotas é linear com o número de nós na rede.

O Span, assim como o PROC, envia $O(n)$ mensagens para a criação do seu *backbone*, onde n é o número total de nós na rede. O Span, entretanto, realiza a substituição incremental dos nós, enquanto o PROC realiza a substituição completa dos nós a cada ciclo.

Por fim, analisamos o custo dos protocolos PROC e Span em relação a ciclos do processador. A atividade que mais demanda recursos de processamento no PROC é a função *UpdateRoute*, que demanda $O(v)$ operações de CPU. Esta função varre as $n\check{}tuplas$ à procura do nó com menor distância ao PA, verificando se este é coordenador e qual é a sua energia residual[2]. Esta operação é realizada no recebimento de pacotes M_{Sync} e M_{Coord}. O repasse de dados demanda somente a escrita do endereço do próximo salto no pacote, pois o mesmo é armazenado entre execuções da função *UpdateRoute*. Como o PROC interage com a aplicação pelas regras da aplicação, uma regra muito complexa ou com erros de programação pode deteriorar o desempenho do protocolo. Assim, as regras de aplicação implementadas pelo *software* de aplicação devem ser compatíveis com os limites de processamento dos nós sensores.

Nós não coordenadores no Span periodicamente executam a função $check - announce - coordinator$, para verificar se a sua adição ao *backbone* irá conectar um par de nós por uma rota

[2]A operação do algoritmo varia se o nó é coordenador ou não, entretanto sua complexidade se mantém inalterada.

mais curta. Esta verificação é determinada pela função $connect - pairs$, que verifica se um par qualquer de vizinhos (A e B) seria beneficiado pela entrada do nó corrente no *backbone*. Esta operação, como descrita no algoritmo apresentado em [13], é uma operação que realiza $O(v^2)$ chamadas à função $share - other - coordinators$. A função $share - other - coordinators$ verifica se a rota A e B comunicam entre si por uma rota que envolve um ou dois coordenadores. A função $share - other - coordinators$ tem custo $O(v^2)$, pois verifica a lista dos vizinhos dos seus vizinhos para realizar tal verificação. Assim, o processo de verificação realizado pela função $check - announce - coordinator$ realiza $O(v^4)$ operações, sendo v o número médio de nós vizinhos.

A comparação do Span com o PROC mostra que, apesar de operarem em ambientes restritos, protocolos para redes ad hoc podem ser mais intensivos em memória e processamento que os protocolos utilizados em RSSF. Isto ocorre por dois motivos: primeiro, porque a capacidade de memória e processamento dos nós utilizados em redes ad hoc é, em geral, dezenas a centenas de vezes superior à provida pelos nós sensores, permitindo assim o uso de soluções complexas. O segundo fator é a quantidade de nós empregados em RSSF, que em geral ordens de magnitude superiores ao número de nós empregados em uma rede ad hoc tradicional. Assim, os algoritmos desenvolvidos para RSSF devem ser mais escaláveis quanto ao número de nós do que os algoritmos empregados em redes ad hoc.

3.6 Conclusão

Neste capítulo apresentamos o protocolo PROC, um protocolo pró-ativo desenvolvido para RSSF estáticas que possuem um fluxo contínuo de dados dos nós sensores ao PA. Iniciamos o capítulo mostrando na Seção 3.1 os requisitos do protocolo proposto. Estes requisitos capturam os fluxos de dados e operações normalmente realizadas nas redes de disseminação contínua de dados. Em seguida apresentamos o funcionamento do protocolo, que foi dividido em nós sensores e PA. Apresentamos na Seção 3.2 os algoritmos e estruturas de dados utilizados no PROC, bem como descrevemos passo a passo o processo de formação do *backbone*.

Em seguida discorremos sobre as as regras de aplicação, que são o diferencial do PROC em relação aos protocolos existentes para redes de disseminação contínua. As regras de aplicação permitem que o PROC utilize informações providas pelo *software* da aplicação ou outros protocolos de comunicação na formação de rotas. Esta interação permite que o comportamento do PROC seja modificado dinamicamente durante a operação da rede.

O PROC procura manter um bom desempenho mesmo quando ocorrem falhas na rede. Para tanto, o protocolo implementa os dois mecanismos de tolerância a falhas descritos na Seção 3.4. Estes mecanismos procuram contornar falhas silenciosas, que ocorrem quando um nó da rede não recebe ou transmite mensagens por um período de tempo.

Na Seção 3.5 apresentamos um problema de grafos que aproxima a estrutura de roteamento formada pelo PROC. Verificamos que, por construção, as rotas no PROC não possuem ciclos.

Em seguida determinamos o consumo típico de memória e o número de mensagens enviadas pelos protocolos PROC e Span utilizando a notação "big-Oh". Mostramos que algoritmos que requerem complexidade assintótica superior a $O(v)$ em função do número de nós vizinhos v, como é o caso do Span, são ineficientes em RSSF mesmo para valores pequenos de v. O PROC, por outro lado, requer quantidades de memória e mensagens dentro deste limite assintótico.

Capítulo 4

Avaliação de Desempenho

Neste capítulo avaliamos o desempenho do PROC utilizando o simulador NS-2 [61], um simulador extensivamente usado na literatura para a simulação de redes. Na época do início do trabalho, os simuladores em geral possuíam pouco suporte para a avaliação de redes de sensores sem fio. Sendo assim, foi necessário desenvolver módulos para implementar tal funcionalidade no simulador NS-2. O texto deste capítulo se organiza da seguinte maneira. A Seção 4.1 descreve as modificações feitas no simulador NS-2 e as características da aplicação utilizada na avaliação. As seções seguintes apresentam a avaliação do protocolo, que foi dividida em cenários. Cada cenário avalia um requisito esperado do protocolo, como as regras de aplicação, avaliadas na Seção 4.2, a escalabilidade, tratada na Seção 4.3, o tempo de vida da rede, que é avaliado na Seção 4.4, e a tolerância a falhas, apresentada na Seção 4.5.

4.1 Caracterização da Simulação

A aplicação simulada consiste em uma rede homogênea, composta por nós sensores com configuração próxima aos nós sensores da família Mica 2, rodando o sistema operacional TinyOS [52]. Simulamos uma aplicação multi-saltos de coleta ambiental, com as características de tráfego similares à rede empregada na ilha de Great Duck para estudos do ecosistema e comportamento de aves [71]. Nessa rede, cada sensor envia para o ponto de acesso mensagens de dados de 36 bytes, em períodos regulares de 70s. O PA, ao receber as mensagens, as repassa para processamento por um enlace via satélite. Avaliamos somente a interação entre os nós sensores e o PA, uma vez que a conexão do PA com outras redes não influi no desempenho do protocolo de roteamento.

A topologia simulada consiste em uma rede de nós estacionários, dispostos em uma área retangular, com 50 a 200 nós localizados em posições aleatórias seguindo uma distribuição uniforme, como mostra a Figura 4.1. O PA está sempre localizado em um dos vértices da área simulada, maximizando a largura da rede. A densidade dos nós (círculos pretos) é mantida constante em 23 vizinhos por nó, em média. Para tanto, aumentamos a área da rede ao aumentarmos o número

de nós simulados. Como mostra a figura, entretanto, podem ocorrer regiões onde a concentração de nós está abaixo da densidade de nós esperada, uma vez que esta é calculada como a densidade média para toda a topologia medida.

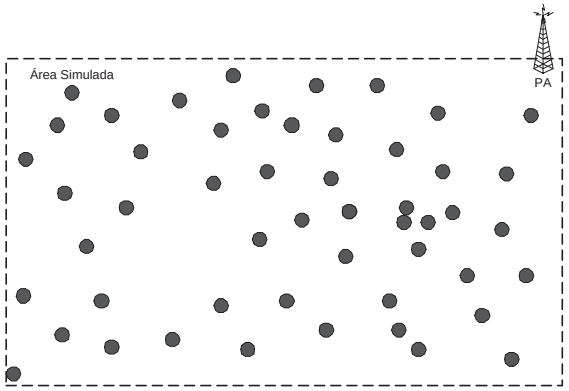

Figura 4.1: Exemplo de uma topologia de rede empregada na avaliação dos protocolos.

Limitamos o número de nós sensores a 200 nós por duas razões. Redes acima de 200 nós se mostraram congestionadas com o número de mensagens, degradando o desempenho da rede em métricas como taxa de entrega e atraso. Isto ocorre devido às limitações do meio físico e do protocolo de controle e acesso ao meio (MAC) empregado, que limita a taxa de transmissão de dados a 12kbps. Acreditamos que simulações com novos protocolos MAC, como o IEEE 802.15.4 [39], permitirão um número maior de nós na rede. Entretanto, este não foi utilizado na avaliação por ter se tornado disponível apenas recentemente. O segundo motivo para o limite de nós se deve aos limites computacionais do ambiente de simulação empregado. O tempo requerido para simulações para redes acima de 200 nós, feitas em um computador Pentium IV 3.0GHz, ultrapassa uma semana. Como testes iniciais mostraram que os resultados não seriam relevantes nestes cenários, decidimos por não realizar tais experimentos.

Os nós simulados foram ajustados para se assemelharem aos nós da arquitetura Mica2. Ajustamos a largura de banda para 12Kbps, banda máxima da arquitetura. Os pacotes de dados enviados possuem tamanho de 36 bytes, enquanto os pacotes de controle possuem tamanhos variados, de acordo com os dados que estes carregam. O tamanho dos pacotes pode ser visto na Seção 5.2.4. Nas simulações, 4% dos pacotes são perdidos por erros, que é equivalente a uma taxa de erros por bit (BER) de 10^{-3}, taxa de referência para o rádio empregado[12]. Apesar da taxa de erro variar com as condições do ambiente, utilizamos uma taxa de erro fixa devido à dificuldade de se modelar de forma precisa o comportamento do meio de transmissão.

Nós sensores da arquitetura Mica2 são capazes de transmitir dados a distâncias de até 100m, dependendo de condições como potência de transmissão, temperatura, existência de fontes de interferência, tipo do solo e elevação [15]. Definimos o raio de transmissão em 15m, que é a distância máxima de recepção quando utilizamos a potência padrão do rádio (0 dBm), como

verificado em estudo posterior [15]. O consumo de energia do rádio nos estados de transmissão, envio e *idle* foi retirado de [78]. Simulamos apenas o consumo de energia do rádio, pois seu consumo é o mais significativo dentre todos os componentes [18].

Implementamos uma versão do protocolo MAC descrito pelo padrão IEEE 802.11, adaptado para RSSF, para aumentar a fidelidade da simulação à pilha de protocolos do TinyOS. Esta versão apresenta características semelhantes ao B-MAC [66], um protocolo CSMA/CA implementado no TinyOS. No B-MAC, os pacotes possuem no máximo 36 bytes, sendo 6 destes utilizados para cabeçalhos. Retiramos as mensagens RTS/CTS do IEEE 802.11, diminuímos a banda do protocolo e ajustamos parâmetros como DIFS, SIFS e a janela de contenção do IEEE 802.11 com valores adequados à banda utilizada pelo B-MAC. Também diminuímos o número de retransmissões para 1, simulando uma aplicação que retransmite dados ao identificar um pacote não confirmado.

De acordo com as medições realizadas em [66], o tempo de propagação por salto para os nós Mica2 é da ordem de centenas de milissegundos. Nessas condições, o tempo de construção das rotas é muito alto. Para amenizar esse problema, utilizamos a priorização de pacotes de roteamento descrito em [82] e atualmente implementada no TinyOS. A priorização de pacotes utiliza duas filas de tamanho fixo de 16 pacotes cada, que é o tamanho padrão da fila de pacotes no TinyOS. A primeira, com maior prioridade, é a fila de pacotes de roteamento; a segunda, de menor prioridade, armazena os pacotes da aplicação, que são enviados após o envio de todos os pacotes de roteamento. Verificamos para as nossas simulações que este tamanho de fila é adequado para os cenários avaliados, já que a quantidade de pacotes descartados devido a filas cheias foi ínfimo. Entretanto, cenários diferentes dos avaliados podem requerer filas menores ou maiores. Veja a Seção 5.3.3 para mais detalhes.

Para comparação com o PROC, dois outros protocolos foram simulados, TOSB (*TinyOS Beaconing*) e EAD (*Energy-Aware Distributed routing*). O TOSB é uma versão simplificada do protocolo TinyOS Beaconing, descrito na Seção 2.3. A implementação do TinyOS Beaconing baseia-se na descrição encontrada em [44]. Uma característica importante do TinyOS Beaconing, que não foi simulada no protocolo, é o uso de estimadores de confiabilidade das rotas, pois seria necessária a implementação de um modelo fiel de qualidade de canal, o que não é o escopo deste trabalho. O TinyOS Beaconing foi escolhido para a comparação pois é um protocolo muito difundido em RSSF em operação. O EAD, por sua vez, é um protocolo simples e eficiente que utiliza os mecanismos mais frequentes dos protocolos de roteamento em RSSF: seleção de rotas de acordo com a energia, consumo de energia homogêneo entre os nós e recriação periódica das rotas.

Ajustamos o tempo de ciclo dos protocolos PROC, EAD e TOSB para 180s, 120s e 120s, respectivamente. Esses valores foram utilizados por apresentarem o melhor desempenho em um cenário com falhas (mostrados na Seção 4.5). Os valores utilizados na avaliação foram encontrados empiricamente para o cenário avaliado. Outros cenários, com número de nós e característica de tráfego diferentes, irão requerer intervalos de recriação de rotas diferentes. Uma forma de

calcular o melhor intervalo de recriação de rotas é conduzir um estudo similar ao mostrado na Seção 4.5, determinando qual é o intervalo de recriação de rotas que possui o melhor compromisso entre entrega de dados e energia consumida. Note que o desempenho do PROC pode variar de acordo com a aplicação, pois cada aplicação pode utilizar regras que influenciarão no influenciarão protocolo. Utilizamos no PROC as três regras propostas na Seção 3.3 como uma forma de avaliar o benefício do uso de regras no roteamento. Essas regras são genéricas e podem ser usadas em qualquer aplicação.

Utilizamos a Equação 3.1, mostrada na Seção 3.4, para definir o número de perdas consecutivas necessárias para o PROC recalcular o nó pai (chamado de limiar). Esta equação apresenta a probabilidade que ocorram perdas consecutivas de dados ocasionadas por erros de transmissão (ou falsos positivos, pois um evento que não é considerado uma falha é diagnosticado como uma falha). Com este valor podemos calcular com qual probabilidade a reconstrução das rotas do nó será desnecessária, pois ocorreu devido a flutuações passageiras na qualidade do enlace, e não a uma falha de nós. A Tabela 4.1 mostra a probabilidade esperada de reconstruções desnecessárias de rotas para a rede simulada. Utilizamos para o cálculo um pacote de tamanho de 36 bytes com preâmbulo de 4 bytes, que é o tamanho máximo dos pacotes na plataforma Mica2 [52]. Esta arquitetura utiliza o rádio CC1000, que possui taxa de erro por bit típica de 10^{-3} [12]. Escolhemos o limiar igual a dois, pois para este valor a probabilidade de reconstruções desnecessárias é pequena (um pouco maior que 1%). Além disso, o benefício de valores superiores a dois é negligível, assim procuramos escolher um valor pequeno para o limiar para permitir uma resposta rápida a falhas.

Valor do limiar	**Probabilidade de falsos positivos**
1	0.238
2	0.057
3	0.014
4	0.003
5	0.001
6	1.829^{-4}
7	4.357^{-5}
8	1.038^{-5}

Tabela 4.1: Probabilidade esperada de falsos positivos para alguns valores de limiar.

Devido à alta complexidade de processamento e ao número e tamanho das mensagens enviadas pelo Span (vide Seção 3.5.2), que mostram a sua inadequação aos requisitos de RSSF, decidimos não incluí-lo na avaliação de desempenho por simulação.

As métricas utilizadas nas simulações são: *taxa de entrega média fim a fim* (porcentagem do total de pacotes recebidos corretamente pela PA); *latência média*; *distância média em saltos até o PA*; *número médio de colisões durante a simulação*; *vazão*; *número médio de pacotes de*

roteamento enviados; *consumo médio de energia*. O consumo médio é calculado somente para os nós que estão ativos ao fim da simulação. Todos os resultados são obtidos pela média de 33 simulações, e apresentados com intervalo de confiança de 95%.

O código desenvolvido para a avaliação do PROC encontra-se disponível no endereço `http://www.dcc.ufmg.br/~damacedo/proc.html`. Disponibilizamos a implementação do PROC, TREE e EAD, bem como as modificações realizadas no NS-2 para modelar RSSF com maior fidelidade. Este código é distribuído utilizando a licença LGPL (*Lesser General Public License*), que permite o uso do código em produtos comerciais e não-comerciais [32].

4.2 Regras de Aplicação

Neste cenário avaliamos o desempenho do PROC utilizando regras de aplicação. Simulamos o PROC em cinco configurações, que são: sem regras de aplicação, utilizando as três regras descritas na Seção 3.3 em conjunto e utilizando cada uma das três regras em separado. Variamos o número de nós da rede de 50 a 200 nós em incrementos de 25 nós, mantendo a sua densidade constante, ao aumentar a área da rede proporcionalmente ao aumento do número de nós. Além disso, simulamos um modelo de falhas onde os nós falham aleatoriamente, tornando-se indisponíveis durante 120s.

Apresentamos somente os resultados para redes que possuem de 100 a 200 nós pois, para redes contendo 50 e 75 nós, os resultados foram muito próximos em todas as métricas avaliadas. O PROC se comporta de maneira similar para todas as configurações analisadas quando a rede contém 50 ou 75 nós, devido ao pequeno tráfego na rede e ao baixo número médio de saltos, que variou entre 2 e 3. Nestas condições, a rede opera com baixa carga, assim a perda de dados é pequena.

Ao aumentarmos o tamanho da rede, verificamos que o uso das regras permite um aumento de desempenho. A Figura 4.2 mostra a taxa de entrega para redes com 100 a 200 nós. O uso de regras aumenta a taxa de entrega média, como mostra a figura. A diferença de desempenho do PROC sem o uso de regras para o PROC utilizando regras aumenta com o número de nós na rede, chegando a até 5% para redes com 200 nós. Dentre as regras propostas, a regra número um, que procura revezar os nós coordenadores, se mostrou a mais eficiente. As regras dois e três e as três regras combinadas apresentaram um desempenho levemente inferior ao obtido pela regra um, como mostra a figura.

A Figura 4.3 mostra a latência média dos pacotes de dados. Novamente, a diferença no desempenho do PROC diminui ao diminuirmos o número de nós na rede. A menor latência foi obtida ao combinarmos as três regras. As regras dois e três, que elegem mais nós coordenadores para distribuir o tráfego, obtiveram ganhos marginais de latência. Entretanto, quando combinamos as três regras, obtivemos uma latência até 8% inferior à obtida pelo PROC sem regras de aplicação.

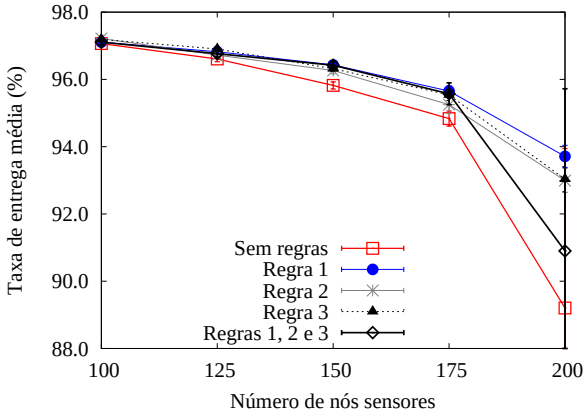

Figura 4.2: Taxa de entrega média variando as regras de aplicação.

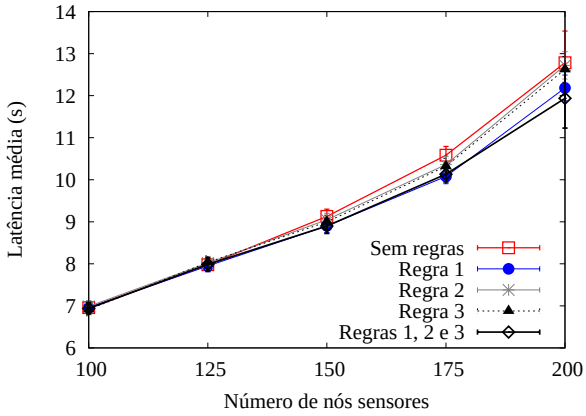

Figura 4.3: Latência média variando as regras de aplicação.

A seguir avaliamos o consumo médio de energia por nó. À excessão do cenário com 200 nós, nos cenários restantes não verificamos nenhuma variação significativa na energia consumida nas diferentes configurações do PROC. Para 200 nós, por outro lado, verificamos que o uso de regras permitiu uma economia de energia de aproximadamente 21%. Entretanto, o desempenho das três regras combinadas foi inferior, sendo igual ao encontrado para o PROC quando não utilizamos regras. Acreditamos que isso ocorre devido a um conflito entre as regras, que procuram otimizar parâmetros diferentes. Ao operarem em conjunto, as ações de uma regra podem prejudicar as ações de outra regra.

Além de procurar minimizar o consumo de energia e a latência da rede, as regras propostas na Seção 3.3 procuram selecionar os coordenadores conforme a topologia da rede. Esta estratégia foi adotada com o intuito de diminuir o número de coordenadores adicionados na segunda fase da operação do PROC. Como a adição de um nó coordenador requer duas mensagens, uma para demandar a entrada do nó no *backbone* e outra para comunicar a sua nova função, o número

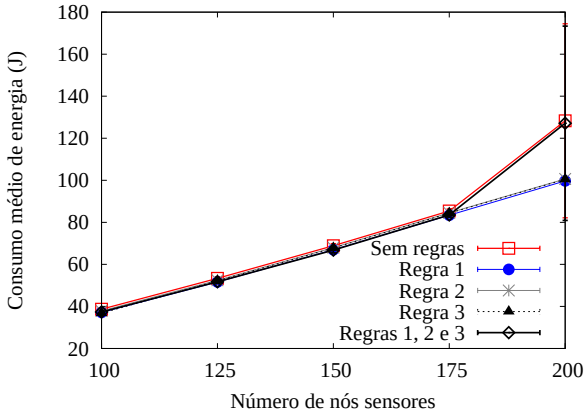

Figura 4.4: Consumo médio de energia variando as regras de aplicação.

de mensagens de roteamento enviadas pela rede é reduzido ao diminuirmos o número de coordenadores adicionados na segunda fase da operação do PROC. A Figura 4.5 apresenta o número médio de mensagens de roteamento enviadas por nó durante as simulações. Como esperado, o uso de regras diminuiu o número de pacotes de roteamento enviado, pois a primeira fase do PROC identifica com mais eficiência quais nós devem ser utilizados no roteamento. Novamente, a regra 1 e a combinação das três regras apresentaram os melhores resultados. Também vemos pela figura que o número de mensagens de roteamento enviadas por cada nó independe do número de nós na rede.

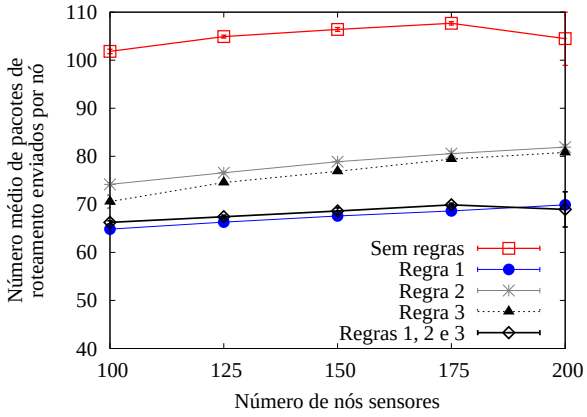

Figura 4.5: Média de pacotes de roteamento enviados por nó.

Utilizamos um segundo cenário de simulações para avaliar o tempo de vida da rede de acordo com o uso de regras. Neste cenário depositamos 200 nós em uma rede de área quadrada, com a mesma densidade de nós utilizada nos cenários anteriores. Neste cenário, entretanto, os nós possuem apenas 100J de energia, para que possamos avaliar o tempo de vida dos mesmos. A

simulação foi executada por 11000s pois, após este tempo, verificamos que a vazão da rede se encontra muito próxima de zero. Verificamos neste cenário pequenas variações na taxa de entrega média dos protocolos avaliados, da ordem de menos de 0.5%. Obtivemos resultados para a latência e número médio de saltos condizentes aos encontrados na análise anterior para redes de 200 nós.

O tempo de vida da rede, entretanto, apresentou uma variação significativa. Quando empregamos o PROC sem regras da aplicação, a vazão da rede é nula a partir de 9600 segundos de simulação. Ao utilizarmos a regra 1, o PROC consegue entregar pacotes até 9800s de simulação, o que mostra que o rodízio dos nós coordenadores de fato aumenta o tempo de vida da rede. A segunda regra apresentou um resultado ainda melhor, permitindo que a rede funcionasse por 9900 segundos. A regra 3 obteve os melhores resultados, permitindo que os nós operasse até o fim dos 11000 segundos avaliados. Como era de se esperar, a partir de 9900 segundos a vazão se manteve baixa, uma vez que poucos nós se mantiveram ativos.

4.3 Escalabilidade

Neste cenário variamos o número de nós da rede de 50 a 200 nós para avaliar sua escalabilidade. A duração de cada simulação foi 9000s. Além disso, simulamos falhas de nós, considerando um modelo onde os nós falham aleatoriamente, tornando-se indisponíveis durante 120s.

A Figura 4.6 mostra a taxa de entrega média fim a fim. Para redes com menos de 150 nós, vemos que os protocolos possuem comportamento idêntico, entregando quase a totalidade de mensagens. Para redes acima de 150 nós, entretanto, a quantidade de mensagens enviadas pelos nós satura a rede, e a diferença entre os protocolos é pronunciada. Para 175 nós, o PROC se mostra superior ao TOSB e EAD em 2%. Já para 200 nós, o PROC obteve 6% e 3% de melhora sobre o TOSB e o EAD, respectivamente.

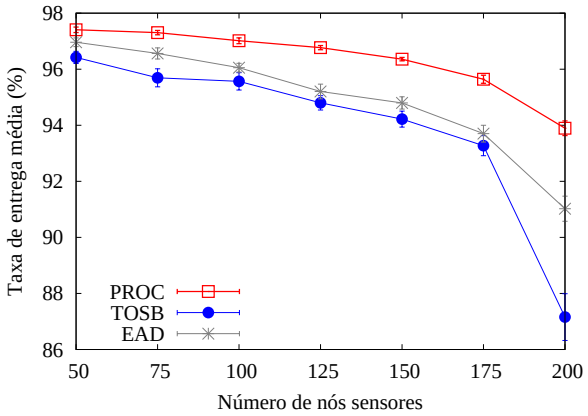

Figura 4.6: Taxa de entrega média.

Era esperado que o PROC obtivesse uma latência média superior à dos outros protocolos

avaliados por entregar mais mensagens. Entretanto, o PROC obteve latências menores para todas as simulações, como mostra a Figura 4.7. Isto se deve principalmente a uma das regras de eleição de coordenadores empregadas, que induz a existência de mais coordenadores na vizinhança do PA (regra 3). Isso provê um número maior de rotas, permitindo que o tráfego seja distribuído de forma mais homogênea entre os nós. A latência do PROC é, em média, 1,3s menor que a do TOSB e 0,3s menor que a do EAD para cenários com menos de 150 nós. Para 200 nós, entretanto, a latência aumenta significativamente para todos os protocolos. Nesse tamanho, o PROC possui latência média de 12,1s, enquanto o EAD e o TOSB obtiveram valores 11% e 34% superiores, respectivamente. Note que, à medida que o número de nós aumenta, o comportamento da rede torna-se cada vez mais irregular, como mostrado pelo intervalo de confiança.

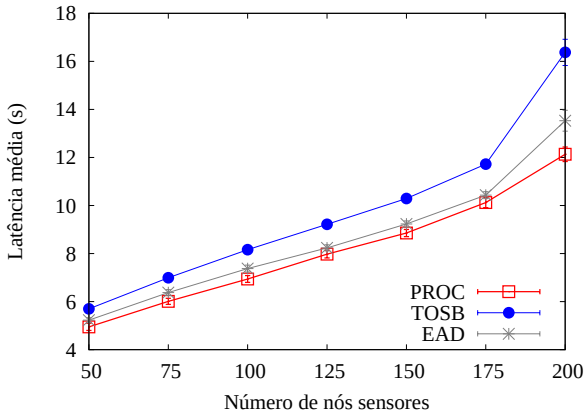

Figura 4.7: Latência média.

As Figuras 4.8 e 4.9 apresentam o número médio de saltos até o PA e o número médio de colisões, respectivamente. Estas métricas indicam quais são as fontes de atraso da rede. A Figura 4.8 mostra que o número de saltos aumenta gradativamente com o aumento do número de nós da rede, em redes até 150 nós. Para redes com 175 e 200 nós, entretanto, o aumento da latência é desproporcional ao aumento do número de nós, o que indica um aumento da contenção na rede. A Figura 4.9 comprova este fato, visto que o número de colisões aumenta significativamente para redes acima de 175 nós. O PROC apresentou o melhor desempenho para cenários com mais de 175 nós, visto que o espalhamento do tráfego em várias rotas diminuiu o número de colisões na rede. O EAD e o TOSB, ao contrário, utilizam o menor número possível de rotas. Esta concentração de tráfego em um número reduzido de nós, entretanto, aumentou significativamente a latência e prejudicou a taxa de entrega, devido ao maior número de colisões. Vemos por estes dados que a diferença de desempenho entre o PROC e outros protocolos tende a se acentuar em redes maiores, uma vez que o PROC possui mecanismos para cooperar com situações de tráfego acentuado, que diminuem o número de colisões e aumentam o número de rotas utilizadas.

Vemos pelos resultados anteriores que, devido ao grande número de mensagens enviadas,

redes com mais de 200 nós irão apresentar um desempenho deteriorado em métricas como taxa de entrega e latência. Assim, caso a aplicação necessite de um número maior de nós, duas alternativas são possíveis. A primeira seria o aumento da banda da rede, utilizando protocolos MAC como o IEEE 802.15.4. Outra opção é a adição de um ou mais PAs em outros pontos da rede, dividindo o tráfego entre os mesmos. O protocolo PROC pode ser utilizado em redes com mais de um PA, desde que os PAs operem em sincronia, ou seja, operem no mesmo número de ciclo.

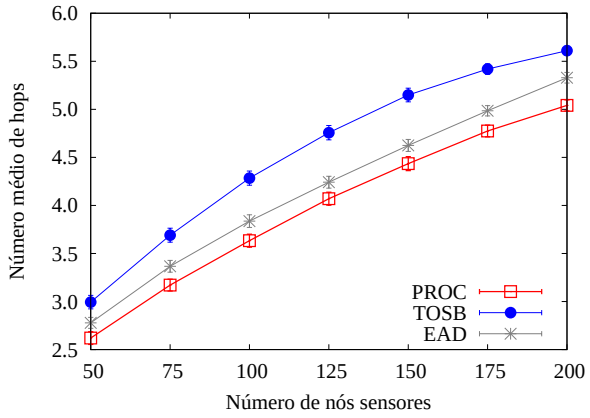

Figura 4.8: Saltos médios.

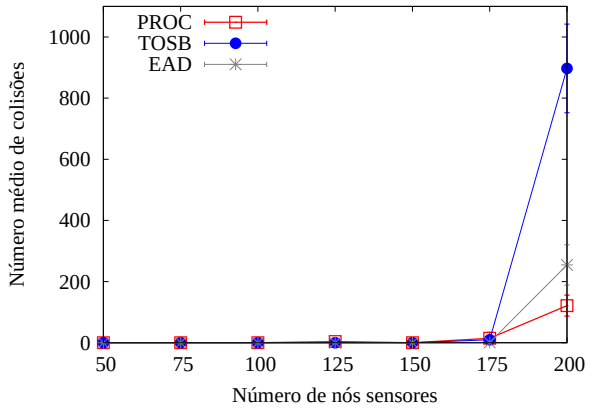

Figura 4.9: Média de colisões.

Um fator importante a ser considerado no projeto de protocolos para RSSF é o seu custo em energia e mensagens enviadas. As Figuras 4.10 e 4.11 mostram o número médio de mensagens de roteamento enviadas pela rede e o consumo médio de energia por nó, respectivamente. Verificamos que o PROC envia em torno de 13% menos mensagens que o EAD, e 4% menos que o TOSB. Foi observado que o consumo de energia aumenta de forma linear com o número de nós

na rede, aumentando de 15J para 104J, em média. Verificamos que os pacotes de roteamento correspondem a 19% dos pacotes enviados para o PROC em simulações com 50 nós, contra 24% para o EAD e 20% para o TOSB. À medida que aumentamos o número de nós simulados, entretanto, os pacotes de roteamento são responsáveis por 11%, 12% e 10% do tráfego total para o PROC, EAD e TOSB, respectivamente.

Note que no modelo de rádio adotado a quantidade de energia consumida na transmissão é muito próxima à energia consumida na recepção e no modo de repouso. Em rádios onde esta diferença é mais acentuada, os ganhos do PROC seriam maiores. Além disto, as simulações não empregaram um protocolo de acesso ao meio que desliga o rádio em períodos de inatividade. Por enviar menos dados de controle em relação aos protocolos comparados, o PROC se beneficiaria com tais esquemas, aumentando assim a sua economia em relação aos protocolos avaliados.

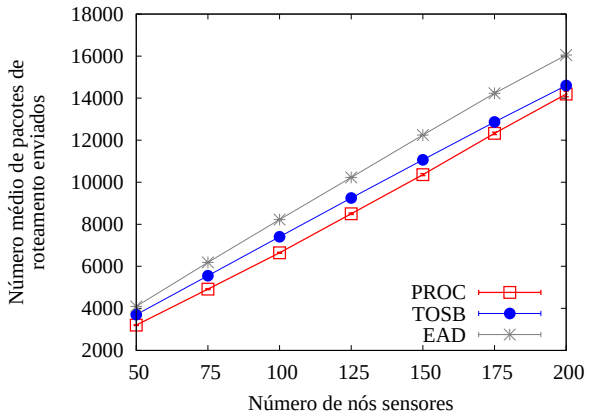

Figura 4.10: Média de pacotes de roteamento enviados em cada simulação.

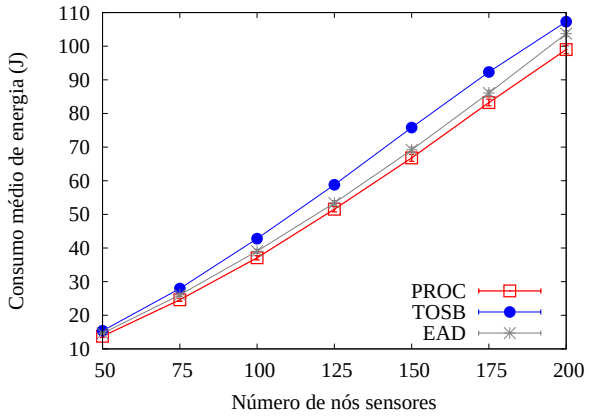

Figura 4.11: Consumo médio de energia.

4.4 Tempo de Vida da Rede

No terceiro cenário de simulação foi avaliado o tempo de vida de uma rede de 200 nós. Fixamos a quantidade de energia na bateria de cada nó em 100J. A Figura 4.12 mostra a variação do número médio de pacotes recebidos por segundo (denominado de vazão) durante a simulação. Observamos que os protocolos possuem uma vazão irregular, oscilando em torno de 2,6 pacotes por segundo (pps) para o PROC, 2,4 pps para o EAD, e 2,26 pps para o TOSB. O desvio médio da vazão obtida foi, respectivamente: 0,2, 0,3 e 0,36pps. A Figura 4.13 mostra a vazão para os últimos 3000s da simulação. O PROC aumenta o tempo de vida da rede em 12,5% e 7,6% em comparação ao TOSB e ao EAD, respectivamente, aumentando a vida da rede para 9900s. Além disto, o PROC apresentou maior vazão durante toda a simulação, e uma degradação de desempenho mais suave que os outros protocolos avaliados. Verificamos que o aumento no tempo de vida da rede é devido ao uso das regras expostas na seção 3.3, que permitiram que o consumo de energia médio por nó se tornasse mais uniforme. Estes resultados não são mostrados neste artigo.

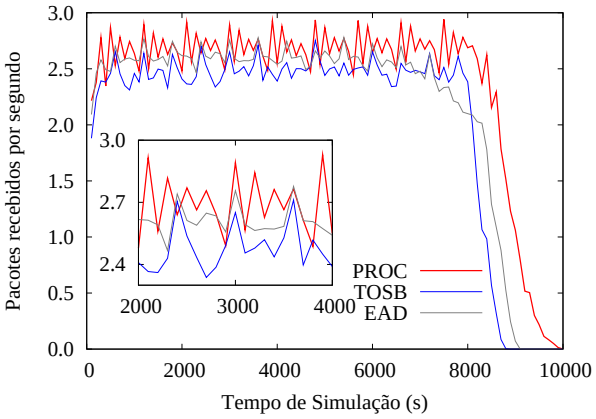

Figura 4.12: Vazão média.

4.5 Falha de Nós

A rede considerada é formada por 150 nós inseridos aleatoriamente em uma área quadrada, com 70m de lado. O PA se encontra no canto da rede em todos os cenários de simulação, maximizando o número de saltos. A rede funciona sem falhas durante 1500s, para que os nós cheguem ao seu estado estacionário (verificado empiricamente). Neste momento ocorre uma falha, e a simulação continua por mais 1500s, permitindo que o protocolo de roteamento se recupere da falha. As falhas são modeladas de acordo com o modelo apresentado na Seção 2.4.2. Desta forma, avaliamos as falhas de acordo com a sua extensão e persistência. Nos cenários em que ocorrem

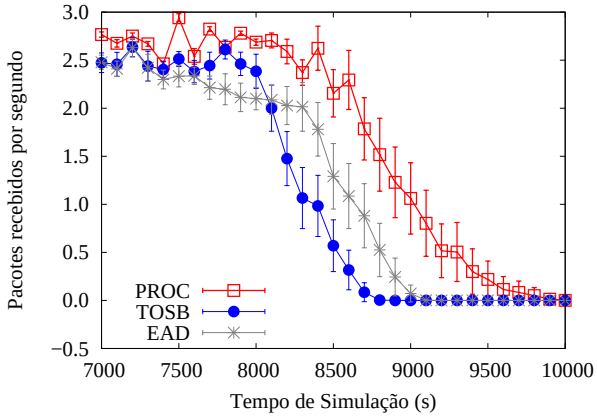

Figura 4.13: Vazão média (ampliado).

falhas isoladas, os nós que irão falhar são escolhidos aleatoriamente. Para falhas agrupadas, é definido um ponto central, e todos os nós que estão a uma distância máxima r_{max} do ponto irão falhar.

As métricas avaliadas são: *taxa de entrega média fim a fim* (porcentagem do total de pacotes recebidos corretamente pelo PA); *latência média*; *distância média em saltos até o PA*; *vazão*; e *consumo médio de energia*. O consumo médio é calculado somente para os nós que estão ativos ao final da simulação. A análise dos resultados enfatiza a energia média consumida, taxa de entrega média e vazão média, pois estas são as métricas mais importantes quando consideramos tolerância a falhas em RSSF. Todos os resultados são obtidos pela média de 33 simulações, e apresentados com intervalo de confiança de 95%.

4.5.1 Falhas Transientes e Isoladas

As falhas transientes foram avaliadas sobre três dimensões: intervalo de recriação de rotas, tempo da falha e número de nós falhos. A frequência da atualização de rotas influirá no grau de tolerância a falhas do protocolo, pois define o tempo de reação a falhas em protocolos como o EAD e TOSB, que dependem da reconstrução de rotas para normalizar o funcionamento da rede. Neste cenário 20 nós falharam durante 120s, e o tempo de ciclo variou de 120 a 300s. Como era esperado, os protocolos tendem a consumir mais energia com atualizações de rotas mais frequentes (Figura 4.14). A taxa de entrega, mostrada na Figura 4.15, diminui ao aumentarmos o intervalo de recriação de rotas. No PROC a redução é menor, devido ao uso de ACKs para identificar falhas, pois as falhas são corrigidas mais rapidamente. Verificamos um aumento na taxa de entrega para intervalos de 300s, que se deve a uma diminuição na carga da rede, diminuindo o número de pacotes perdidos devido a falta de espaço na fila. Quanto à latência média, todos os protocolos mostraram uma diminuição nesta métrica com o aumento do tempo de recriação de rotas, que ocorre principalmente devido à menor carga imposta à rede.

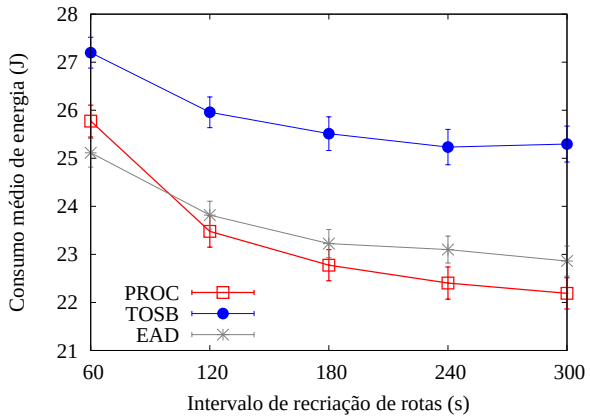

Figura 4.14: Energia consumida variando o intervalo de recriação de rotas.

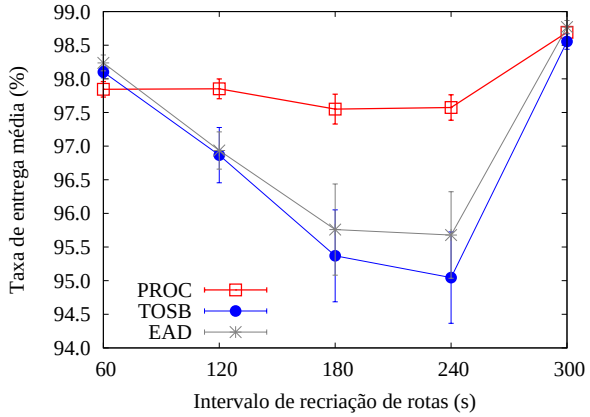

Figura 4.15: Taxa de entrega média variando o intervalo de recriação de rotas.

Em seguida avaliamos como o tempo de falha afeta o desempenho dos protocolos. Para todos os protocolos, verificamos que existe uma queda na vazão durante o tempo de falha, que é recuperada quase completamente após um intervalo de recriação de rotas (Figura 4.16). Isto ocorre porque os protocolos utilizam como mecanismo de tolerância a falhas a recriação completa das rotas em intervalos regulares de tempo. Novamente, o PROC apresentou uma taxa de entrega média levemente superior (em torno de 0.5%), devido ao uso de ACKs para detectar falhas, como mostrado na Figura 4.17. O intervalo de recriação de rotas mostrou-se adequado, uma vez que os protocolos EAD e TOSB, mesmo sem possuírem mecanismos ativos de detecção de falha como o PROC, obtiveram bons resultados. Como anteriormente, a quantidade de energia consumida por nó diminuiu, uma vez que estes tiveram que rotear menos dados. Dentre os protocolos avaliados, o PROC consumiu menos energia, consumindo em torno de 22 a 23J, enquanto o EAD e TOSB consumiram em torno de 4% a 14% mais energia que o PROC, respectivamente.

Finalmente, variamos o número de nós falhos. Simulamos falhas de 25 a 100 nós, com in-

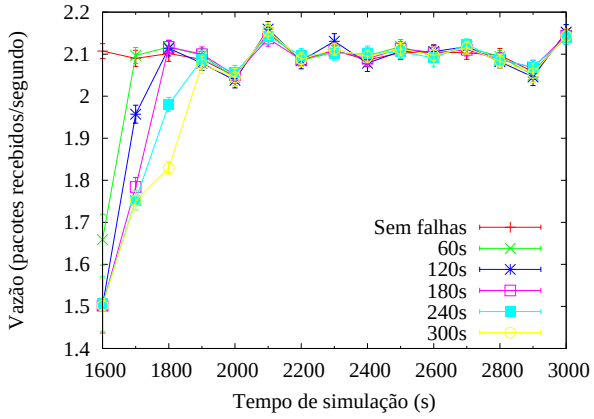

Figura 4.16: Vazão do PROC variando o tempo de falha.

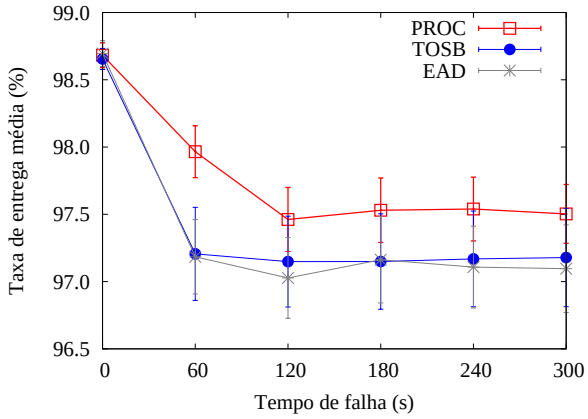

Figura 4.17: Taxa de entrega variando o tempo de falha.

crementos de 25 nós. Identificamos que todos os protocolos possuem um comportamento similar quando variamos o número de nós falhos. Os protocolos apresentam um declínio de vazão, que é restaurada em até 200s após a falha. A queda na vazão é determinada pelo número de nós falhos, exemplificada pelo comportamento do PROC, mostrado na Figura 4.18. O mecanismo de identificação de falhas empregado pelo PROC permitiu que este protocolo recuperasse mais rapidamente da falha, desta forma aumentando a sua taxa de entrega em relação aos outros protocolos em torno de 0.5%. Como a falha tem um intervalo de tempo pequeno em relação ao tempo total de simulação, o ganho por uma recuperação mais rápida da falha é amenizado no resultado final. A latência e o número de saltos médios não sofrem alterações com a falha de nós, entretanto a energia consumida decresce. Isto ocorre devido à diminuição do número de pacotes enviados ao PA, correspondente ao tráfego originado dos nós falhos. A Figura 4.19 mostra o gráfico do consumo médio de energia dos protocolos. A pequena variação do consumo médio e da taxa de entrega média mostram que o número de nós falhos não influi de forma significativa

na rede. Como as falhas são distribuídas pela rede, novas rotas são facilmente encontradas.

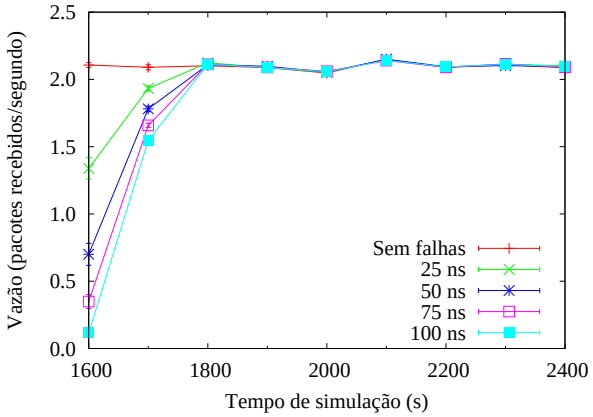

Figura 4.18: Vazão do PROC variando o número de nós falhos.

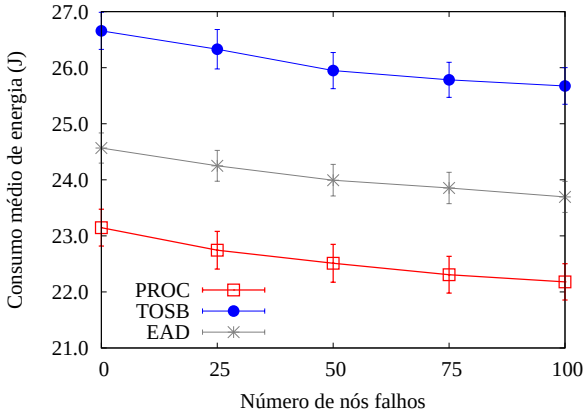

Figura 4.19: Consumo de energia variando o número de nós falhos.

4.5.2 Falhas Permanentes e Isoladas

Neste cenário avaliamos o comportamento dos protocolos na ocorrência de falhas permanentes e isoladas. Realizamos simulações variando o número de nós falhos em 20, 40 e 60 nós. Verificamos que os protocolos se recuperam rapidamente, entretanto a vazão cai após a ocorrência da falha, pois os nós falhos deixam de enviar dados. A desativação dos nós fez com que o número de saltos até o PA aumentasse levemente, em torno de 0.1 saltos para cada 20 nós falhos. A latência média se manteve quase inalterada, mostrando que a diminuição do tráfego de dados compensou o aumento do número de saltos médios. Quanto à taxa de entrega média, verificamos uma queda com o aumento do número de nós falhos, mostrado na Figura 4.20.

Em comparação com as falhas transientes isoladas, verificamos que as falhas permanentes são mais prejudiciais à rede do que as falhas transientes, pois acarretam uma queda mais significativa no consumo médio de energia e na taxa de entrega média.

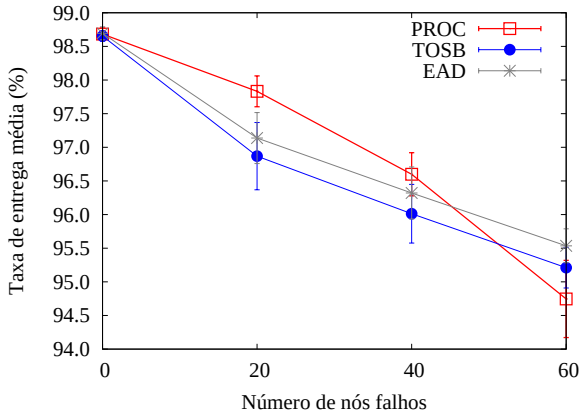

Figura 4.20: Taxa de entrega média para falhas permanentes isoladas.

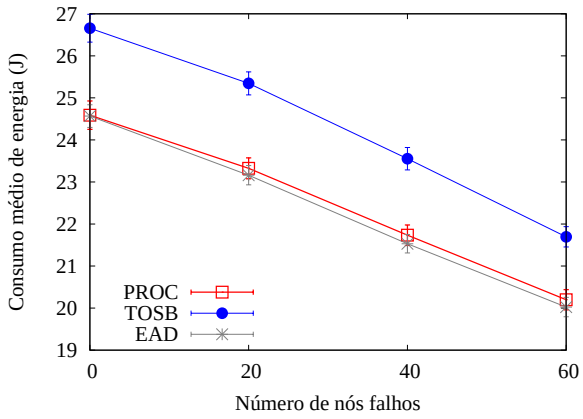

Figura 4.21: Consumo de energia para falhas permanentes isoladas.

4.5.3 Falhas Permanentes e Agrupadas

Neste cenário avaliamos o efeito de falhas permanentes agrupadas. O número de nós falhos depende do raio de falha, que variou de 5 a 40m. Os resultados obtidos mostram que este tipo de falha é o mais severo dentre os avaliados, assim realizamos uma análise mais completa.

A taxa de entrega média cai até 9% com o aumento do raio de falha, como mostra a Figura 4.22. Como nos cenários anteriores, os protocolos se recuperam da falha após a recriação completa das rotas. Neste cenário, entretanto, verificamos uma grande variação da taxa de entrega média, causada por cenários onde ocorrem partições na rede. É sabido que nós próximos ao PA repassam

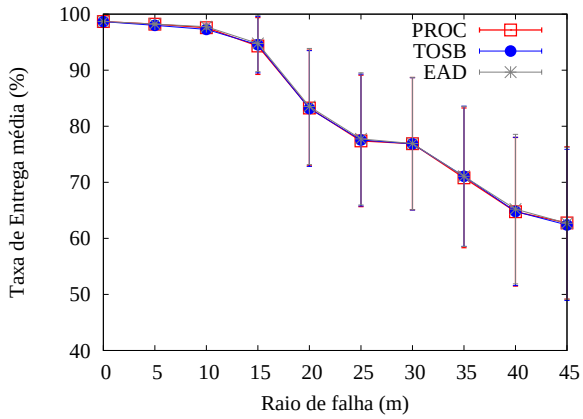

Figura 4.22: Taxa de entrega para falhas permanentes agrupadas.

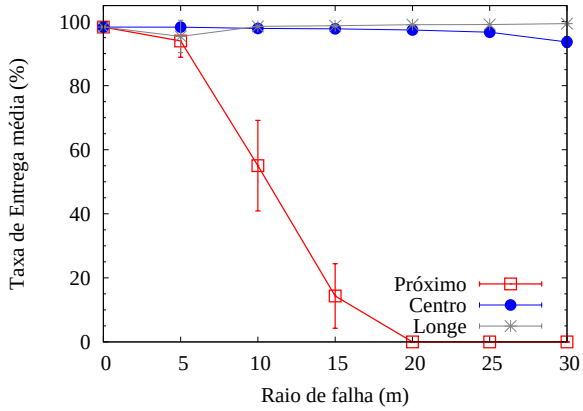

Figura 4.23: Taxa de entrega para falhas em pontos distintos da rede.

mais dados que os nós distantes, assim falhas nestes nós irão degradar significativamente o serviço da rede. A Figura 4.23 exemplifica a taxa de entrega para falhas em pontos distintos da rede. A curva "Próximo" apresenta a taxa de entrega média para falhas que possuem o seu centro em até 17m do PA. A curva "Centro" mostra falhas na região central da rede, enquanto a curva "Longe" apresenta falhas em até 17m do canto oposto ao PA. Os resultados apontam que falhas de nós em pontos não muito próximos do PA pouco afetam a taxa de entrega. Entretanto, falhas próximas ao PA podem degradar substancialmente o desempenho da rede.

Encontramos um intervalo de confiança de até 10% para a taxa de entrega na curva "Próximo" da Figura 4.23, que verificamos ser ocasionado por partições na rede. Uma análise do histograma da taxa de entrega para a curva "Próximo" (Figura 4.24), em grupos de 5%, mostra que os resultados estão aglomerados próximo de 5% e de 95%, o que explica o alto intervalo de confiança, e identifica o papel crucial da partição da rede no desempenho dos protocolos neste cenário. As partições ocorrem em cenários como o mostrado na Figura 4.25, onde a ocorrência de falhas

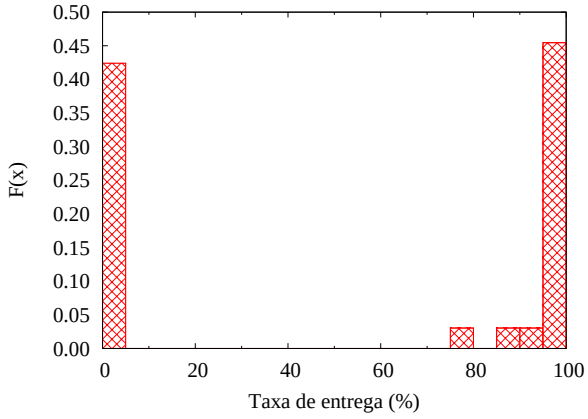

Figura 4.24: Histograma da taxa de entrega em falhas perto do PA.

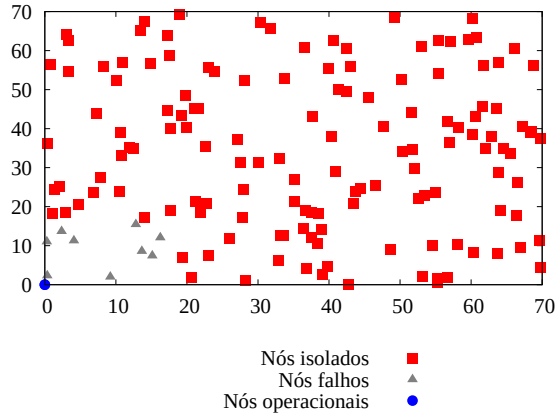

Figura 4.25: Exemplo de rede particionada.

isolou todos os nós operacionais da rede, impedindo a comunicação com o PA. A figura classifica os nós em "Nós falhos" (nós que sofreram falhas), "Nós operacionais" (nós que estabelecem rotas para o PA) e "Nós isolados" (nós que não conseguem estabelecer rotas para o PA). Uma forma para recuperar partições na rede, que necessita de suporte no MAC, é o aumento da potência de transmissão. Com a negociação de uma nova potência de transmissão entre os nós próximos à região de falhas, a comunicação poderia ser reestabelecida. O roteamento também pode contribuir para amenizar a severidade de uma partição da rede detectando a ocorrência de falhas e adotando medidas de economia de energia.

Verificamos que a latência média (Figura 4.26) e a distância média em saltos até o PA aumentam quando o raio da falha é pequeno, pois é necessário que as rotas evitem a região falha, para tanto aumentando o caminho percorrido. Para falhas mais extensas, devido a partições na rede, verificamos uma diminuição em ambas as métricas. Como apenas os nós próximos do PA conseguem enviar dados, o número de saltos médios e a latência média diminuem.

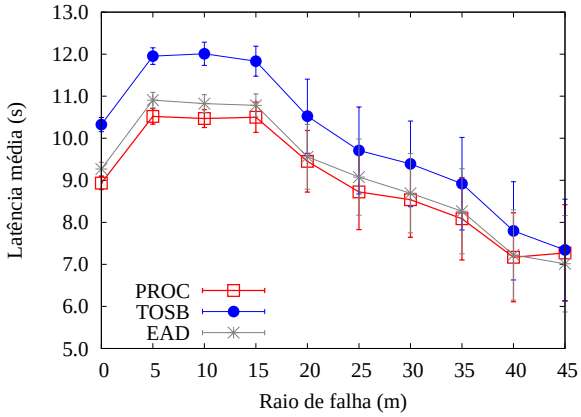

Figura 4.26: Latência média para falhas permanentes agrupadas.

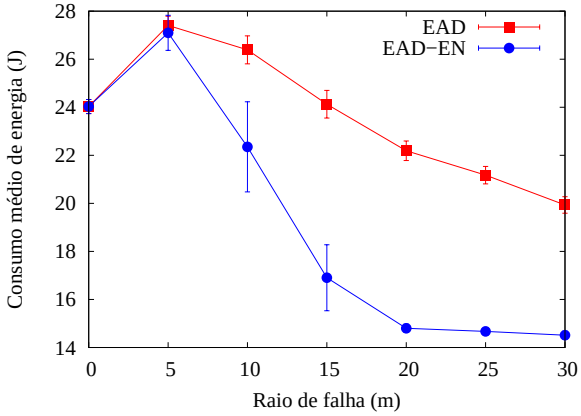

Figura 4.27: Consumo de energia com mecanismos de desligamento do rádio.

A Figura 4.27 mostra mecanismos de economia de energia que podem ser utilizados na ocorrência de uma partição na rede para diminuir o consumo dos nós. Comparamos o EAD ao EAD-EN, uma versão modificada do EAD, que recusa o envio de mensagens da aplicação caso o nó não tenha recebido mensagens de recriação de rotas em um intervalo de tempo equivalente a dois períodos entre a recriação de rotas (curva "EAD-EN" na figura). Como visto na figura, o uso de técnicas de economia de energia permite uma diminuição substancial no consumo (de 16% a 33% apenas evitando o envio de mensagens), não interferindo em nenhuma outra métrica da rede. Verificamos resultados semelhantes para os outros protocolos avaliados, justificando a implementação destas técnicas em protocolos de roteamento.

4.6 Conclusão

Este capítulo apresentou uma avaliação por simulação do protocolo PROC. As simulações procuraram aproximar as condições encontradas em RSSF reais. Para tanto, modelamos o fluxo de dados e definimos as configurações dos nós de acordo com as características de uma rede real, utilizada pela Universidade de Berkeley para a coleta de informações ambientais e estudo de pássaros. Avaliamos aspectos do protocolo como o desempenho propiciado pelo uso de regras, a escalabilidade e a tolerância a falhas. Além disso, comparamos o desempenho do PROC com os protocolos TOSB e EAD, sendo o TOSB baseado em um protocolo empregado em RSSF reais.

Identificamos que as regras de aplicação proporcionam um aumento de desempenho ao protocolo. Utilizamos para esta avaliação as três regras propostas na Seção 3.3. Em geral, o desempenho utilizando regras foi igual ou superior ao desempenho do PROC sem a utilização de regras. Cada regra proposta aumentou o desempenho do protocolo em uma determinada métrica. A primeira regra proposta reduziu a latência e o consumo de energia dos nós ao revezar os nós utilizados no *backbone*. Já a segunda regra diminuiu o consumo de energia e a latência média ao minimizar o número de nós que são indicados a integrar o *backbone*, diminuindo assim o número de mensagens de controle enviadas. A terceira regra, por sua vez, aumentou o tempo de vida da rede ao distribuir o tráfego entre mais nós, o distribuindo o consumo de energia. A combinação de duas ou mais regras de aplicação deve ser feita de forma cautelosa, pois o uso de múltiplas regras pode diminuir o desempenho da rede.

Comparamos a escalabilidade do PROC com os protocolos EAD e TOSB para redes compostas por 50 a 200 nós. Verificamos que o PROC permite um aumento na taxa de entrega em todos os cenários avaliados. Ao mesmo tempo, o protocolo proposto apresentou um menor consumo médio de energia por nó e uma menor latência média. Além disto, o PROC forma rotas menores, diminuindo o número de repasses de dados requeridos. Devido ao seu menor consumo de energia, o PROC possibilitou que as redes simuladas operassem por até 12,5% mais tempo, em comparação com os protocolos avaliados.

Tendo em vista a frequente ocorrência de falhas em RSSF, avaliamos o desempenho dos protocolos PROC, EAD e TOSB em cenários com falhas de diversas características. Utilizamos o modelo de falhas apresentado na Seção 2.4.2, o que simplificou enormemente as simulações ao limitar o número de cenários representativos a somente três. A monitoração ativa do nó pai permitiu que o PROC se recuperasse de falhas mais rapidamente que os outros protocolos avaliados. Este mecanismo ainda apresentou custo desprezível, pois não requer o envio de mensagens ou bytes adicionais. Dentre as falhas avaliadas, identificamos que as mais prejudiciais são as falhas agrupadas. Caso ocorram em nós perto do PA, estas falhas podem interromper a operação de toda a rede.

Capítulo 5

Implementação do PROC no TinyOS

Neste capítulo apresentamos a implementação do PROC no sistema operacional TinyOS, que é amplamente utilizado em RSSF. A Seção 5.1 apresenta uma visão geral do TinyOS, descrevendo a sua linguagem de programação e apresentando a arquitetura de comunicação utilizada. A Seção 5.2 mostra a implementação do PROC no TinyOS, descrevendo as *interfaces* utilizadas, o formato dos pacotes e as estruturas de dados. A Seção 5.3 mostra o protocolo em operação.

5.1 O Sistema Operacional TinyOS

O sistema operacional TinyOS foi desenvolvido pela Universidade de Berkeley para simplificar a programação dos nós Mica [52]. Este sistema operacional está se tornando um padrão *de facto* em RSSF, sendo utilizado nas arquiteturas mais populares de RSSF. São exemplos de arquiteturas de nós sensores que utilizam o TinyOS: a família Mica (Mica, Mica2, MicaZ) [16], Telos [57], BTNode [25] e CodeBlue [62]. O TinyOS é um sistema operacional modular e compacto, que fornece abstrações para o acesso ao *hardware* e implementa protocolos de comunicação.

Para diminuir o consumo de memória e processamento, o TinyOS é compilado em conjunto com o *software* da aplicação em um único binário, que contém somente os módulos necessários ao funcionamento da aplicação. Isto evita que o nó sensor carregue consigo bibliotecas e módulos não utilizados. Além disso, a criação de um único executável aumenta o desempenho do nó sensor, pois evita a carga dinâmica de bibliotecas e todos os problemas associados, como a checagem de compatibilidade de bibliotecas com o *software* de aplicação e a alocação de memória para o código a ser carregado, dentre outros.

O TinyOS é implementado em uma linguagem própria, chamada NesC [31], que é derivada do C. No NesC, os programas são compostos de módulos, que comunicam entre si por *interfaces*. As *interfaces* são direcionais, ou seja, certos módulos implementam a *interface*, enquanto outros módulos as utilizam. A Figura 5.1 mostra um programa de exemplo, onde os quadrados indicam os módulos e as setas representam as *interfaces*. No exemplo, o módulo BLINKM utiliza os serviços

providos pelo módulo SINGLETIMER, que fornece abstrações para acesso aos temporizadores do processador. Na figura, os módulos onde as setas se originam utilizam os serviços fornecidos pela *interface* que dá nome à seta. Estes serviços são implementados pelos módulos apontados pelas setas. O módulo SINGLETIMER, por exemplo, implementa a *interface* TIMER, enquanto o módulo BLINKM utiliza a implementação da *interface* TIMER provida pelo módulo SINGLETIMER.

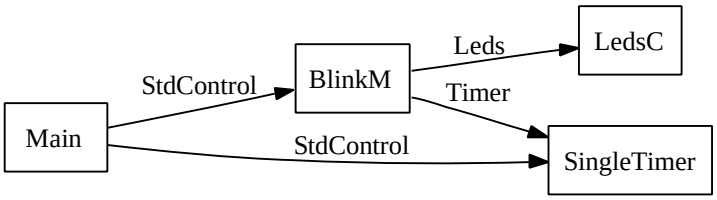

Figura 5.1: Exemplo de uma aplicação no TinyOS e seus módulos.

As *interfaces* definem comandos e eventos que permitem a comunicação entre módulos. Os módulos que utilizam a *interface* executam os comandos definidos na mesma. Os módulos que implementam a *interface*, por outro lado, notificam a ocorrência de situações de importância aos módulos que as utilizam ao disparar eventos. Na figura, por exemplo, o módulo BLINKM utiliza comandos da *interface* TIMER para requerer serviços do módulo SINGLETIMER, enquanto o módulo SINGLETIMER notifica o término da execução utilizando os eventos definidos na *interface*. As *interfaces* também são utilizadas no TinyOS para o tratamento de eventos de *hardware*.

Cada módulo, por sua vez, pode ser constituído por um conjunto de módulos. Isto ocorre com componentes complexos, como por exemplo com as implementação das camadas de roteamento e de controle e acesso ao meio. Esta divisão permite que quebremos a implementação de um grande serviço em partes menores, ao mesmo tempo isolando a divisão interna dos outros módulos da aplicação. Assim, um serviço pode ser visto como um grande módulo, apesar de ser composto internamente por vários módulos. A camada de roteamento, por exemplo, define um módulo chamado MULTIHOPROUTER, que é composto por dois módulos internos, chamados MULTIHOPENGINEM e MULTIHOPLEPSM.

5.1.1 Modelo de Programação

Alguns eventos, como os eventos que tratam de interrupções de *hardware*, devem ser tratados imediatamente. Assim, os eventos e comandos devem executar rapidamente, para que a interrupção seja tratada a tempo. Para suportar cálculos que demandam maior tempo de processamento, o NesC utiliza *tarefas*. As tarefas podem ser interrompidas a qualquer momento por eventos, e retornam a sua execução após o término da execução do evento que lhe interrompeu. Esta operação é similar ao que ocorre em *mainframes*, por exemplo, que executam tarefas pesadas como

a geração de um relatório somente quando a carga de serviços prioritários, como por exemplo o acesso ao banco de dados, está baixa.

Como os eventos e comandos devem responder rapidamente devido aos eventos de *hardware*, o TinyOS emprega um modelo de programação orientado a eventos, onde as funções dos módulos são executados de forma assíncrona. Tomemos como exemplo o envio de mensagens para a rede, que é implementado pela comando SEND e pelo evento SENDDONE. O primeiro é chamado para sinalizar que um módulo deseja enviar um novo pacote. Para evitar que a execução fique parada até que o pacote seja enviado, o comando SEND retorna imediatamente, sinalizando apenas se o pacote foi colocado na fila para envio ou se a fila está cheia. O fim da transmissão de dados é notificado ao módulo que requisitou a comunicação utilizando o evento SENDDONE. Da mesma forma, qualquer operação no TinyOS que não pode ser realizada instantaneamente é implementada de forma assíncrona, geralmente utilizando uma tarefa. Um comando irá escalonar a execução de uma tarefa, que notifica o fim da sua execução por um evento. Como veremos a seguir, a implementação do PROC segue esta filosofia de projeto.

5.1.2 Comunicação no TinyOS

A pilha de protocolos do TinyOS possui características peculiares que lhe diferenciam dos modelos de referência comumente utilizados em redes de computadores. Como exemplo, o TinyOS permite que aplicações utilizem apenas os componentes da pilha que são necessários para uma dada aplicação. Caso todos nós consigam ouvir uns aos outros, por exemplo, pode-se optar por retirar o módulo de roteamento, uma vez que toda a comunicação ocorrerá em um único salto.

Podemos dividir a comunicação no TinyOS em cinco camadas, da superior para a inferior: Aplicação, roteamento, AM, controle e acesso ao meio (MAC) e camada física. Toda comunicação ocorre em um modelo orientado a pacotes, onde não é garantida a entrega dos dados. A função de cada camada se encontra detalhada abaixo.

Aplicação: Implementa a lógica da aplicação, requisitando a transmissão de mensagens diretamente à camada de roteamento ou à camada de controle e acesso ao meio. No TinyOS, os programas não são obrigados a seguirem uma arquitetura rígida de comunicação. Assim, a aplicação pode requisitar o envio de pacotes diretamente à camada MAC ou à camada AM. Neste caso, entretanto, a aplicação deve reimplementar alguns serviços das camadas que foram retiradas.

Roteamento: Constrói as rotas e fornece primitivas para envio de pacotes para o PA. Por uma decisão de projeto, o módulo de roteamento do TinyOS permite que as aplicações enviem dados somente para o PA. Caso o nó deseje enviar dados para um dos seus vizinhos, por exemplo, este deverá requisitar tal transmissão diretamente para as camadas inferiores. A camada de roteamento ainda provê mecanismos para fusão e agregação de dados durante os saltos da comunicação, bem como uma fila de pacotes a serem transmitidos. Esta é a camada que mais

consome memória (devido à fila de pacotes), e por isso é geralmente removida quando a aplicação requer apenas transmissões em um único salto.

AM (Active Message): A camada AM implementa um classificador, que direciona os pacotes para módulos específicos de acordo com um cabeçalho de tipo. Ao contrário dos modelos ISO/OSI e TCP/IP, onde o despacho dos dados para a aplicação correta ocorre acima do nível de roteamento utilizando o conceito de portas, no TinyOS esta tarefa é feita abaixo do roteamento. Em geral, os dados são despachados para o módulo que implementa o roteamento para que este decida o destino do pacote. Entretanto, certos módulos podem optar por gerenciar alguns tipos de pacotes de forma particular.

Camada MAC: Esta camada implementa um protocolo CSMA/CA, com o uso opcional de mensagens de confirmação (ACK). Esta camada não implementa nenhum tipo de fila de pacotes, pois o roteamento é responsável por tal tarefa. Outra característica marcante do MAC no TinyOS é o emprego de ciclos de operação, onde o rádio é desligado em intervalos regulares de tempo, tendo em vista a economia de energia. A camada MAC ainda é responsável pelo enquadramento dos dados, detecção de erros e retransmissão de quadros.

Camada física: A camada física provê um fluxo contínuo de bytes (nas arquiteturas Mica2, MicaZ e Telos) ou de bits (na arquitetura Mica) para a camada MAC, que deve processá-los e discernir dados úteis do ruído do meio. Para tanto, a camada física também provê leituras da potência do sinal recebido.

Em geral, a implementação do TinyOS utiliza abstrações que permitem que o código de protocolos e aplicações seja independente do *hardware* utilizado. Assim, a grande maioria do código escrito para este SO é independente de plataforma. As camadas MAC e física, entretanto, possuem implementações diferentes para cada plataforma suportada no TinyOS por serem fortemente relacionadas ao *hardware*. A camada de roteamento, por outro lado, é em geral independente de plataforma. O PROC, assim, pode ser utilizado em todas as plataformas suportadas pelo TinyOS. Somente uma característica do PROC, a monitoração ativa do nó pai, é dependente de código específico para cada plataforma, pois requer uma interação direta com o protocolo MAC. Apesar da adaptação deste recurso para outras plataformas requerer uma modificação simples de poucas linhas, decidimos adicionar ao código do PROC uma opção de compilação que desativa a monitoração ativa do nó pai. Com esta opção, o PROC pode ser compilado para qualquer plataforma suportada pelo TinyOS.

5.1.3 A Arquitetura de Roteamento do TinyOS

O TinyOS provê uma arquitetura de roteamento, descrita em [65], que facilita a criação de novos protocolos de roteamento e define interfaces que estes devem implementar. A arquitetura também simplifica a codificação da aplicação. Para que troquemos o protocolo de roteamento utilizado em uma aplicação, por exemplo, é necessário somente substituir o nome do módulo em

um arquivo de configuração.

A arquitetura de roteamento é composta de dois módulos. O primeiro, chamado MultiHopSend, gerencia as filas de pacotes, escalona a transmissão de dados e interage com a aplicação. O Segundo módulo, chamado MultiHopRoute, implementa os protocolos e algoritmos de manutenção da tabela de roteamento e estabelecimento de rotas. Com essa separação, a implementação de novos protocolos de roteamento no TinyOS é simplificada, pois o desenvolvedor precisa somente codificar as tarefas únicas do seu protocolo, que na arquitetura se encontram no módulo MultiHopRoute. O módulo MultiHopSend, por outro lado, tende a ser igual para qualquer protocolo de roteamento empregado.

Os protocolos de roteamento provêm três *interfaces* (primitivas) de comunicação com a aplicação, chamadas Send, Receive e Intercept, que são utilizadas para enviar pacotes, receber pacotes e modificar pacotes em trânsito, respectivamente. A *interface* Intercept ainda permite que o módulo que implementa a aplicação ou outros módulos específicos realizem a agregação, fusão e descarte de dados em cada salto da comunicação.

As mensagens de roteamento são do tipo AM_MULTIHOPMSG, e são enviadas pelo AM para o módulo MultiHopRoute. Mensagens de outros tipos são enviadas para o módulo MultiHopSend, que decide se estas serão entregues para um módulo no nó atual ou serão repassadas para outro nó.

5.2 O Protocolo PROC

Nesta seção discutimos como as estruturas de dados e os pacotes de roteamento foram implementados e descrevemos as novas *interfaces* criadas pelo PROC, que são utilizadas na interação com o *software* de aplicação. O PROC procura seguir a arquitetura de roteamento definida pelo TinyOS para manter a compatibilidade com programas e módulos existentes, como mostraremos a seguir.

5.2.1 Estrutura do PROC

A estrutura de módulos utilizada pelo PROC é baseada na arquitetura de roteamento apresentada na seção anterior, como mostra a Figura 5.2. Por motivos de compatibilidade, decidimos manter inalterados os nomes das *interfaces* exportadas pelo PROC e o nome do módulo de roteamento (MultiHopRouter), para que programas que utilizam o roteamento padrão do TinyOS operem com o PROC sem a necessidade de re-escrita de código. O PROC é composto por dois módulos internos, como descreve a arquitetura de roteamento do TinyOS.

Módulo MultiHopEngineM: Implementa as funções do módulo MultiHopSend descritos anteriormente. Este módulo interage diretamente com os módulos que implementam as funções da aplicação, fornecendo os serviços de envio, recebimento e fusão agregação de pacotes. Devido

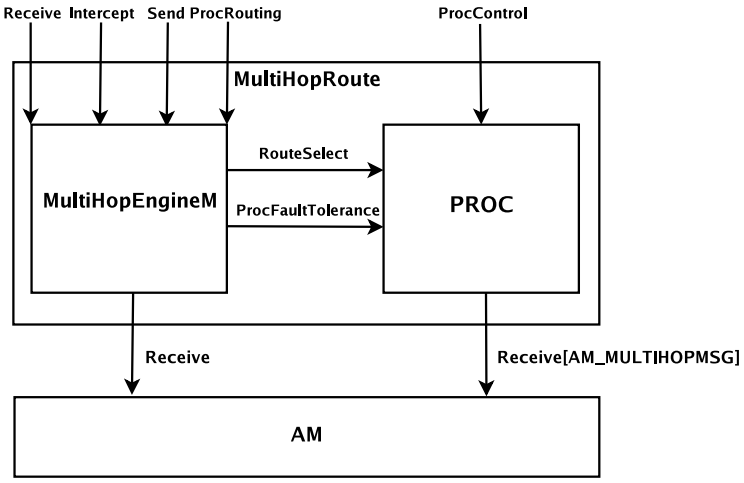

Figura 5.2: Diagrama de módulos e *interfaces* da implementação do PROC.

a limitações do TinyOS, que permite que o nó envie dados apenas para o PA, adicionamos ao módulo MULTIHOPSEND a *interface* PROCROUTING, que permite que os nós enviem mensagens para os seus vizinhos. O MULTIHOPENGINEM recebe do módulo AM todos os pacotes de aplicação recebidos pelo rádio e, de acordo com o destino do pacote, decide se irá repassá-los para um módulo local ou enviá-los para outro nó. No caso de um repasse de dados, o módulo utiliza a *interface* ROUTESELECT para requisitar uma rota. Por receber todos os pacotes de aplicação vindos da rede, este módulo detecta a falha de nós utilizando o mecanismo de monitoração ativa do nó pai, descrito na Seção 3.4. Quando o módulo detecta a falha do nó pai, este comunica ao módulo chamado PROC que as rotas devem ser recalculadas.

Módulo PROC: Implementa as funções do módulo MULTIHOPROUTE na arquitetura de roteamento. Este módulo gerencia a tabela de vizinhos e interage com a aplicação pelas regras de aplicação. O módulo se liga ao módulo AM para receber os pacotes de roteamento, que no TinyOS são do tipo AM_MULTIHOPMSG. O módulo exporta a *interface* PROCCONTROL, que sinaliza quando as regras da aplicação devem ser executadas e permite à aplicação controlar o funcionamento do PROC, como mostramos a seguir.

5.2.2 Interfaces Exportadas

O PROC exporta duas *interfaces* que provêm funcionalidades como as regras de aplicação e o envio de mensagens para todos os vizinhos ou para apenas um dos vizinhos dos nós. O PROC não exige que sejam definidas regras de aplicação, assim o protocolo pode ser utilizado em aplicações desenvolvidas previamente sem que nenhum módulo seja modificado. Caso nenhum módulo implemente regras de aplicação, o PROC utiliza uma regra de aplicação padrão em que os nós possuem 50% de chance de se tornarem coordenadores. Caso algum módulo deseje utilizar as

funções específicas do PROC, este deve empregar as *interfaces* providas pelo protocolo, descritas a seguir.

A primeira *interface* que o PROC provê se chama ProcControl, mostrada na Figura 5.3, que permite que o *software* de aplicação implemente as regras de aplicação. A *interface* ProcControl define eventos que informam a aplicação quando um novo ciclo foi começado (evento newCycle) ou sinalizam que o nó foi forçado a se tornar coordenador (evento forcedCoordinator). O PROC também utiliza eventos para que outros módulos possam inserir seus dados nos pacotes de roteamento, utilizando o evento sendRoutePacket. Da mesma forma, o PROC notifica outros módulos que um pacote de roteamento foi recebido usando o evento receiveRoutePacket, permitindo que outros protocolos ou o *software* de aplicação recuperem os dados embutidos no pacote de roteamento. A interface ainda provê o comando getStatus, que informa se o nó atual é um coordenador ou nó folha.

```
interface ProcControl {
  /* Informa o estado (coordenador ou no folha) do no. */
  command status_t getStatus();
  /* Sinaliza ao software de aplicacao que deve calcular as regras da aplicacao.
     Ao fim do calculo, o modulo deve informar o resultado ao PROC utilizando o
     comando returnAppRulesResult. */
  event void computeAppRules(Node_t neighb[], uint8_t parent);
  /* Informa ao PROC o resultado das regras de aplicacao. */
  command void returnAppRulesResult(uint16_t value);
  /* Notifica o software de aplicacao que um novo ciclo acaba de comecar. */
  event void newCycle(uint16_t cnum, status_t cStatus);
  /* Notifica que o no foi forcado a se tornar coordenador. */
  event void forcedCoordinator();
  /* Este evento comunica ao software de aplicacao que um novo pacote
     de roteamento foi recebido. */
  event TOS_MsgPtr receiveRoutePacket(TOS_MsgPtr msg, void *pload,
        uint16_t plen);
  /* Permite que o software de aplicacao insira dados nos pacotes de roteamento. */
  event void sendRoutePacket(TOS_MsgPtr msg, void *pload, uint16_t destination,
        uint16_t *plen);
}
```

Figura 5.3: *Interface* utilizada para a interação com o *software* da aplicação, em NesC.

As regras de aplicação são implementadas pelo evento computeAppRules e pelo comando returnAppRulesResult. O *software* de aplicação é notificado que deve computar as regras de aplicação pelo evento computeAppRules, e informa ao PROC o resultado das regras utilizando o comando returnAppRulesResult. O processo é dividido em duas fases, pois o cálculo das regras de aplicação pode ser muito custoso. Caso o PROC esperasse o resultado das regras como o valor de retorno do evento, o nó sensor ficaria bloqueado até o término do cálculo das regras, não respondendo a outros eventos de sistema. O PROC permite que o algoritmo de cálculo das regras tenha acesso à tabela de roteamento (um arranjo de estruturas do tipo Node_t, como

mostraremos a seguir) para diminuir a replicação de informações entre o PROC e as estruturas de dados das regras. Como os nós sensores não possuem suporte a números de ponto flutuante, implementamos a probabilidade de um nó se tornar coordenador como um número inteiro sem sinal de 16 bits.

A *interface* PROCROUTING, mostrada na Figura 5.4, permite que os nós enviem dados para os seus vizinhos, uma vez que o TinyOS não provê suporte para comunicação entre nós vizinhos. Como o desenvolvimento de outros protocolos depende desta capacidade, implementamos estas duas funções. O nó envia mensagens para todos os seus vizinhos com o comando SENDBROADCAST, ou para somente um vizinho utilizando o comando SEND. O PROC notifica o término da transmissão da mensagem pelo comando SENDDONE.

```
interface ProcRouting {
  /* Envia dados para um vizinho do no. */
  command result_t send(TOS_MsgPtr msg,uint16_t PayloadLen, uint16_t neighbor);
  /* Envia dados para todos os vizinhos do no em broadcast. */
  command result_t sendBroadcast(TOS_MsgPtr msg, uint16_t PayloadLen);
  /* Notifica ao no que a mensagem terminou de ser enviada. */
  event result_t sendDone(TOS_MsgPtr msg, result_t success);
}
```

Figura 5.4: *Interface* para envio de dados aos nós da vizinhança.

5.2.3 Estruturas de Dados

O PROC utiliza um arranjo de estruturas do tipo NODE_T, que armazenam dados relativos aos vizinhos do nó corrente (as *n˘tuplas* descritas no Capítulo 3). Este arranjo, que é utilizado no algoritmo de estabelecimento de rotas, consome a maior parte da memória utilizada pelo PROC. Como o NesC não permite alocação dinâmica de memória, definimos como 20 o número máximo de vizinhos que o nó pode armazenar, entretanto este valor pode ser modificado de acordo com a densidade de cada rede.

Campo	**Tamanho (bytes)**	**Descrição**
hops	1	Distância do PA, em saltos
curCycle	1	Garante que o dado é atual
flags	1	Indica se o vizinho é coordenador e se os dados são válidos
energy	2	Energia residual
addr	2	Identificador do nó (endereço de rede)

Tabela 5.1: Campos da estrutura NODE_T e seu consumo de memória.

Os campos da estrutura Node_t e a sua função são mostrados na Tabela 5.1 (veja a Seção 3.2.3 para mais detalhes). Como os nós sensores não executam operações de ponto flutuante em *hardware*, a energia residual dos nós vizinhos, que é armazenada no campo *energy*, é definida como um número inteiro de 16 bits. Três campos foram adicionados à implementação. O primeiro, chamado *addr*, armazena o identificador do nó. Este campo determina a qual nó vizinho as informações pertencem. O segundo campo, *curCycle*, armazena o maior valor de ciclo reportado pelo nó vizinho, e garante que a informação armazenada é atual. Com isto, os nós identificam vizinhos falhos, inativos ou que moveram para fora do alcance do rádio ao verificar se o valor de *curCycle* está dentro de valores aceitáveis. Esta verificação, entretanto, deve ser feita de forma cautelosa, pois o valor de *curCycle* sofre de estouro[1]. Nestas situações, o algoritmo detecta e contorna o estouro, que ocorrerá várias vezes durante a operação da rede. Uma opção para contornar o estouro seria o uso de uma variável longa, por exemplo com 64 ou 128 bits, que não sofreria estouro durante o tempo de operação da rede. Entretanto, qualquer aumento do campo *curCycle* em cada entrada da tabela aumentaria significativamente o consumo de memória do protocolo. Assim, optamos por modificar o algoritmo para comportar o estouro deste campo. Por último, o campo *flags* armazena um conjunto de variáveis de um bit, que indicam se o vizinho é um nó coordenador ou nó folha e se os dados armazenados ainda são válidos.

5.2.4 Cabeçalhos das Mensagens

Utilizamos mensagens do tipo AM_MULTIHOPMSG para implementar as mensagens M_{Sync} e M_{Coord}. Identificamos se a mensagem é do tipo M_{Sync} ou M_{Coord} de acordo com um cabeçalho de tipo, como descrito a seguir. Os pacotes de dados e roteamento enviados pelo PROC possuem tamanho máximo de 36 bytes, incluindo cabeçalhos da camada MAC, que utilizam 7 bytes. Empregamos este valor para manter a compatibilidade do PROC com outros módulos do TinyOS.

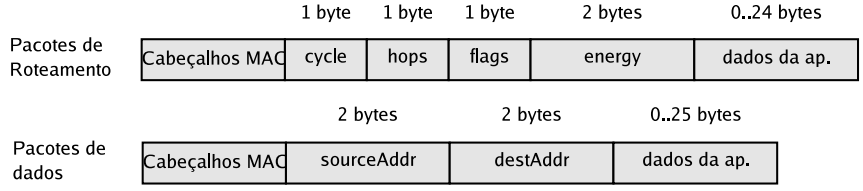

Figura 5.5: Cabeçalhos do PROC nas mensagens de dados e controle.

O pacote de roteamento possui tamanho variável, pois permite que o *software* da aplicação envie dados nos pacotes de controle do PROC, como por exemplo dados utilizados no cálculo das regras de aplicação. Além disso, outros protocolos como controle de topologia, por exemplo,

[1]Em um inteiro de 16 bits sem sinal, por exemplo, o incremento de uma unidade a 65535 é igual a 0.

podem ser implementados utilizando o espaço disponibilizado nos pacotes de roteamento, minimizando a quantidade de mensagens na rede. O PROC utiliza 5 bytes para seus cabeçalhos. O cabeçalho *flags* da mensagem de controle carrega um conjunto de informações de um bit, como na estrutura de dados NODE_T, apresentadas na Seção 5.2.3. O cabeçalho define se a mensagem é do tipo M_{Sync} ou M_{Coord}, e se o emissor da mensagem é coordenador ou nó folha. O campo *hops* indica o menor número de saltos requerido para que o emissor da mensagem alcance o PA e o campo *energy* informa a energia residual do emissor.

O pacote de dados é composto por apenas dois cabeçalhos de 2 bytes cada, que complementam as informações da camada MAC. O cabeçalho *destAddr* é utilizado pelo PROC para determinar o destinatário da mensagem, que pode ser o PA ou um dos nós vizinhos. O cabeçalho *sourceAddr* contém o endereço do emissor da mensagem, para que o destinatário ou nós que repassam a mensagem identifiquem quem lhes enviou os dados. Não adicionamos no cabeçalho o próximo salto da mensagem pois esta informação é fornecida ao protocolo MAC no momento de envio do pacote.

5.3 Exemplos de Uso do PROC

Esta seção apresenta os testes executados sobre o código implementado. Não apresentamos resultados experimentais nesta seção devido a limitações encontradas para a montagem e medição de dados em cenários com um número razoável de nós. Assim, nos limitamos a medir a memória requerida pelo PROC e exemplificar o seu funcionamento em um ambiente real.

5.3.1 Descrição do Cenário

O cenário avaliado possui dimensões reduzidas, contendo apenas cinco nós, devido às dificuldades encontradas na construção de redes maiores. Redes compostas por dezenas de sensores requerem um espaço somente encontrado em ambientes ao ar livre, onde é difícil garantir a segurança dos equipamentos e manter o PA operando por várias horas (uma vez que o PA geralmente é ligado a um *notebook*, que possui tempo de operação limitado pela sua bateria). Outra opção analisada foi a interação entre uma rede de pequena escala e o simulador TOSSIM [50]. Nesta configuração, um computador pessoal simula dezenas ou centenas de nós sensores em tempo real e interage com uma rede real utilizando um nó sensor ligado ao computador. Desta forma, os nós na rede real percebem um tráfego similar ao encontrado em uma rede composta por um grande número de nós. O TOSSIM, entretanto, mostrou-se extremamente lento e de difícil operação, o que impediu a realização dos experimentos.

O cenário de teste consistiu de cinco nós, sendo que um deles opera como PA, reconstruindo as rotas a cada dois segundos. Este nó é ligado a um computador pessoal, que registra todos os pacotes enviados ao PA. Os quatro nós restantes transmitem dados para o PA. Utilizamos as

regras de aplicação descritas na Seção 3.3, implementadas como um módulo separado do *software* de aplicação. O teste foi realizado em um ambiente fechado, com equipamentos eletrônicos em funcionamento e com o trânsito de pessoas. Devido às limitações de espaço existentes, o raio de comunicação foi diminuído a uma média de 2m, o que permitiu a montagem do cenário em um espaço de poucos metros quadrados.

Desenvolvemos uma aplicação de teste para demonstrar a funcionalidade do código implementado. Testamos o PROC utilizando uma aplicação que envia dados em intervalos regulares de 1s para o PA. Os dados são compostos por um contador, o endereço do nó emissor da mensagem e o endereço do nó pai do emissor da mensagem, totalizando uma carga (*payload*) de apenas 5 bytes. O TinyOS fornece uma aplicação simples de visualização dos dados, que os imprime na tela, em formato hexadecimal. Esta aplicação não identifica nomes de campos ou diferencia entre tipos de pacotes, o que torna a análise dos dados uma tarefa extremamente árdua. Por essa razão, desenvolvemos uma aplicação gráfica para apresentar os dados de forma legível e construir uma representação da topologia da rede.

5.3.2 A Aplicação de Depuração

A aplicação de visualização da operação da rede, que chamamos de *PROC interactive network debugging interface*, foi desenvolvida em Java. Utilizamos as bibliotecas fornecidas juntamente ao TinyOS, que realizam a conexão com a porta serial e constroem automaticamente um objeto em Java para cada pacote recebido. Para tanto, o TinyOS emprega um compilador chamado MIG (*Message Interface Generator*), que gera o código-fonte em Java do objeto que representa um dado tipo de pacote a partir da sua estrutura de dados codificada em NesC.

Apesar do TinyOS possuir uma aplicação gráfica para exibição de dados chamada *Surge*, decidimos implementar uma nova aplicação pois o código do Surge é extremamente complexo e pouco modular, o que dificulta a sua extensão para a operação com o PROC. Assim, desenvolvemos uma nova aplicação, baseada no padrão de desenho *Modelo-Visão-Controlador* [34], separando assim o código em módulos que possuem funções claras e bem definidas.

O módulo que responde pela funcionalidade de *Visão*, como o seu nome indica, implementa a visualização dos dados, que na aplicação desenvolvida consiste em uma tela gráfica, descrita a seguir. O *Modelo*, por sua vez, é responsável pela manutenção e armazenamento dos dados que são exibidos na tela. Na aplicação desenvolvida, optamos por armazenar os dados relativos aos nós sensores ao invés de cada pacote de dados. Finalmente, o módulo que comunica com a porta serial assume o papel de *Controlador*, uma vez que este atualiza os dados armazenados.

A aplicação gráfica desenvolvida possui somente uma tela, que apresenta três informações: uma construção gráfica da topologia da rede, uma listagem dos pacotes de dados recebidos e uma listagem dos pacotes de roteamento recebidos. O posicionamento de cada um dos ítens é mostrado na Figura 5.6. A topologia da rede é construída utilizando ícones. Os nós sensores são

representados por imagens reduzidas dos nós sensores Mica2 Dot, enquanto o PA é representado por uma antena. Setas representam as rotas, onde o nó apontado pela seta indica o nó pai do nó de onde a seta se origina. As listagens de pacotes de roteamento e de dados apresenta o nome de cada campo seguido pelo seu valor em hexadecimal, ao contrário da aplicação de visualização padrão do TinyOS, que não separa os dados contidos no pacote em campos. A aplicação ainda permite que os dados recebidos sejam gravados em um arquivo de texto seguindo a formatação apresentada na tela.

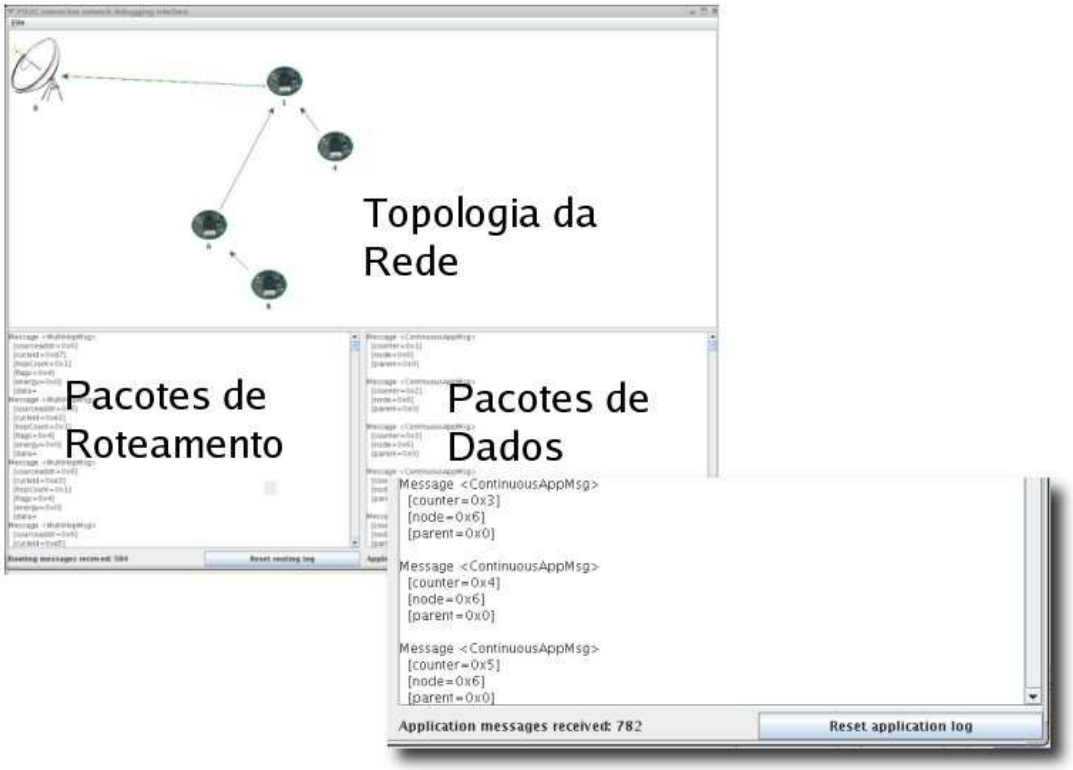

Figura 5.6: A Aplicação de depuração do PROC. A ampliação mostra o formato em que os pacotes de dados são apresentados.

5.3.3 Avaliação do Consumo de Memória

A Tabela 5.2 mostra o consumo de memória do PROC quando compilado para a plataforma Mica2. Comparamos o consumo do PROC com o TinyOS Beaconing, o protocolo padrão de roteamento do TinyOS. Utilizamos nesta avaliação a aplicação descrita anteriormente, rodando sobre a versão 1.1.13 do TinyOS. Apresentamos resultados para duas configurações do PROC. Na primeira, o PROC foi compilado sem regras de aplicação, que é o modo padrão do código implementado. Na segunda configuração do PROC utilizamos as regras da aplicação mostradas na

Seção 3.3. O consumo em memória RAM mede o consumo com variáveis para toda a compilação (sistema operacional, roteamento e aplicação), enquanto o consumo em memória ROM indica o tamanho do código. A título de comparação, também mostramos a capacidade das memórias RAM e ROM na arquitetura Mica2.

Protocolo	**Consumo de ROM (bytes)**	**Consumo de RAM (bytes)**
PROC sem regras	14438	2088
PROC com regras	14646	2089
TinyOS Beaconing	15722	2196
Capacidade total [16]	131072	4096

Tabela 5.2: Consumo de memória RAM e ROM para o PROC e o TinyOS Beaconing na plataforma Mica2.

A tabela mostra que o uso de regras aumentou o consumo de memória RAM do PROC em apenas um byte. Isto ocorre pois as regras de aplicação implementadas utilizam somente os dados armazenados na tabela de roteamento do PROC. O aumento mais significativo foi no tamanho do código, que aumentou em 208 bytes. O protocolo TinyOS Beaconing, por outro lado, utiliza 107 bytes a mais que o PROC em RAM, e 1334 bytes a mais em ROM. O maior consumo de RAM do TinyOS Beaconing é devido ao uso de estimadores de qualidade de enlace, que dobram o tamanho das entradas da tabela de roteamento. Os estimadores ainda requerem contas complexas que utilizam divisão e multiplicação de números de 16 bits, operações que são emuladas em *software* no Mica2. Para tanto, cada operação de divisão e multiplicação é substituída por um pequeno programa, aumentando o tamanho do código.

Em ambos os protocolos, a principal causa de consumo de memória RAM se dá pela fila de pacotes de roteamento. Como o TinyOS não permite a alocação dinâmica de memória, não é possível codificar uma fila de pacotes que cresça ou diminua de acordo com a sua demanda. Desta forma, o desenvolvedor deve escolher um tamanho para a fila de pacotes que seja suficientemente grande para que a frequência de perda de dados devido a filas cheias seja pequena, e ao mesmo tempo o consumo de memória desta fila seja satisfatório. Tal escolha depende de cada cenário e aplicação, fugindo do escopo do nosso trabalho.

5.3.4 Testes em uma Topologia Multi-Saltos

Em seguida testamos o PROC em uma topologia multi-saltos. Utilizamos cinco nós sensores, sendo quatro transmitindo dados ao PA e o quinto operando como PA, como mostra a Figura 5.7. Realizamos um conjunto de testes para nos certificarmos que a implementação do PROC foi feita corretamente.

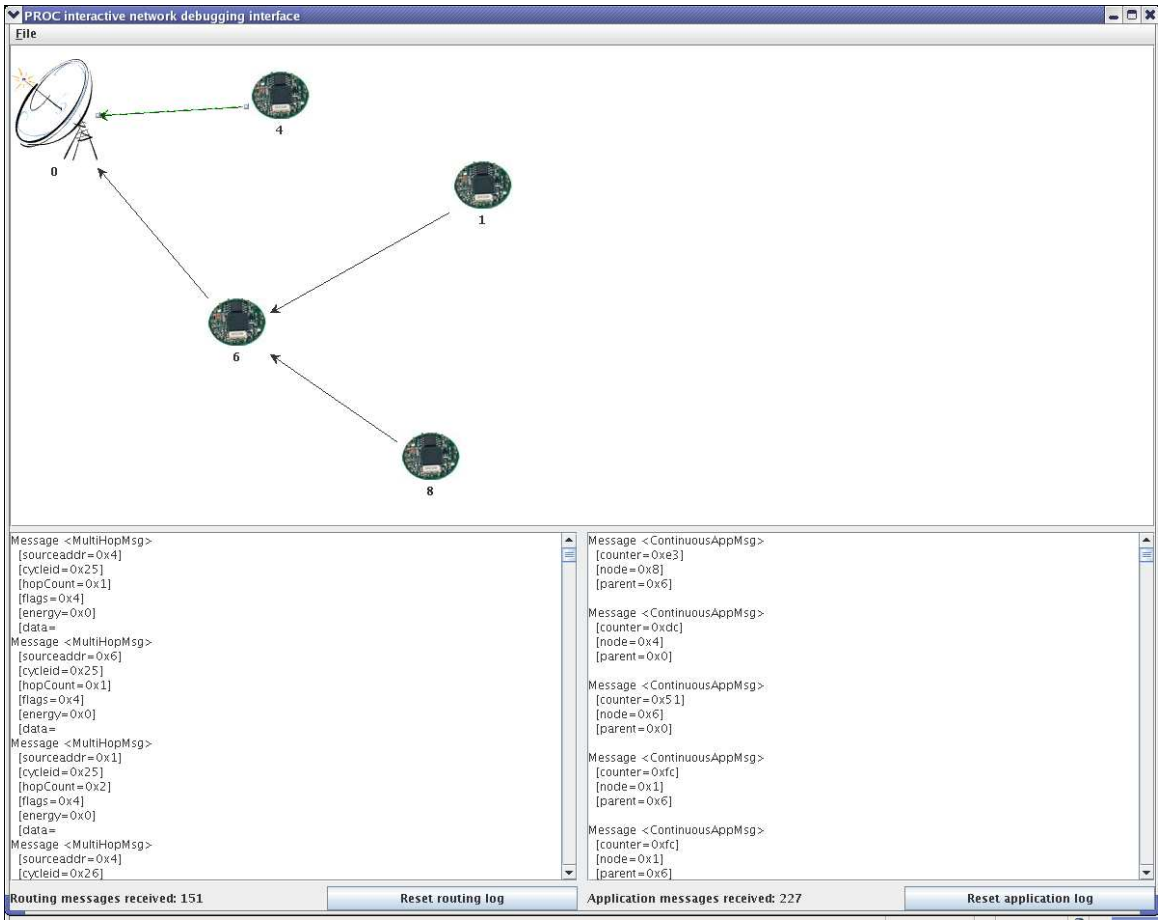

Figura 5.7: Visualização da topologia de teste construída.

Verificamos se os nós conseguem encontrar novas rotas ao desligarmos o seu nó pai. Como esperado, os nós reconstruíram as suas rotas em menos de dois segundos, que é o intervalo entre cada recriação completa das rotas. Em um segundo teste, construímos uma topologia onde um certo nó, que iremos chamar de N, possui somente um caminho possível até o PA. Ao desligarmos o nó pai de N, verificamos que o nó N enviou dados três vezes e interrompeu o seu funcionamento. Ao religarmos o nó pai de N, o nó voltou a funcionar. Este comportamento é devido ao mecanismo de monitoração ativa do nó pai, descrito na Seção 3.4, que reconstrói as rotas após um certo número de transmissões mal-sucedidas (três tentativas na versão implementada no TinyOS). Quando o nó detectou que não existia mais nenhuma rota para o PA, este demandou para a aplicação, que interrompesse o envio de dados enquanto uma nova rota não fosse estabelecida.

No último teste, verificamos a operação das regras de aplicação e o algoritmo de construção do *backbone*. Neste teste, os nós coordenadores permaneciam com o LED vermelho ligado, enquanto os nós folha mantinham o LED vermelho desligado. A partir da topologia apresentada pela aplicação de depuração, foi possível inspecionar visualmente se os nós coordenadores sele-

cionados pelo PROC construíram um *backbone* completo ao verificar quais nós apresentavam o LED vermelho ligado. Como esperado, os nós construíram *backbones* bem formados.

5.4 Conclusão

Este capítulo apresentou a implementação do PROC para o sistema operacional TinyOS. A implementação do PROC neste sistema operacional, que é quase um padrão *de facto* em RSSF, permite que o protocolo proposto seja executado nas plataformas mais utilizadas em RSSF. Mostramos a arquitetura de módulos do PROC e as decisões de implementação, como as modificações feitas nas estruturas de dados e nas mensagens para que o protocolo se adequasse às restrições do TinyOS. A implementação encontra-se disponível para uso acadêmico ou comercial no endereço `http://www.dcc.ufmg.br/~damacedo/proc.html`.

A implementação do PROC procurou, sempre que possível, minimizar a quantidade de código necessária para que projetistas utilizem o PROC. Para tanto, procuramos sempre que possível manter inalteradas as *interfaces* e nomes de módulos que o PROC exporta. Caso deseje utilizar o PROC, o projetista deve apenas mudar um parâmetro em um arquivo de configuração do TinyOS para trocar o protocolo de roteamento padrão da plataforma pelo PROC. Por outro lado, o PROC provê *interfaces* para que o projetista possa otimizar a operação da rede utilizando as regras de aplicação. A implementação do PROC para a plataforma Mica2 mostrou que o consumo de memória e o tamanho do código do protocolo são inferiores aos utilizados pelo TinyOS Beaconing, que é o protocolo de roteamento padrão do TinyOS.

Devido a limitações das aplicações de depuração do TinyOS, encontramos dificuldades para visualizar os dados e a topologia da rede. Solucionamos este problema com o desenvolvimento de uma nova aplicação, que apresenta graficamente ambas as informações. Esta aplicação se mostrou extremamente útil na depuração do protocolo, pois agilizou o processo de análise dos dados obtidos do PA.

Capítulo 6

Conclusões

Em RSSF, a troca de informação entre as camadas da pilha de comunicações para otimizar o desempenho da rede tem se mostrado uma necessidade. Tal característica, que não é usual nas redes tradicionais, se faz necessária para que a rede ofereça um serviço confiável e eficiente mesmo diante das restrições severas de recursos encontrados nas RSSF. Assim, utilizamos protocolos configuráveis, que provêm *interfaces* simples de interação entre protocolos. Estes protocolos permitem que otimizemos o funcionamento da rede, sem que para tanto o desenvolvedor conheça profundamente o funcionamento de cada protocolo empregado.

Este trabalho apresentou um protocolo de roteamento específico para as redes de sensores sem fio que empregam um fluxo contínuo de dados que procura otimizar o seu funcionamento de acordo com as necessidades de cada aplicação. O protocolo proposto, denominado PROC, é o primeiro protocolo de roteamento para redes de disseminação contínua de dados que provê mecanismos simples de interação com a aplicação tendo em vista a otimização do roteamento. Esta interação ocorre através das *regras de aplicação*, que permitem a escolha das melhores rotas, baseado nas peculiaridades de cada aplicação. Como as regras da aplicação podem ser modificadas dinamicamente, o PROC se adapta em tempo de execução a mudanças ocorridas no ambiente.

As rotas são criadas em um processo de duas fases. O PROC determina na primeira fase quais nós irão repassar dados a partir das informações fornecidas pela aplicação via regras de aplicação. A segunda fase garante que todos os nós da rede possuem rotas. Assim, a segunda fase do protocolo garante o seu funcionamento correto mesmo que as regras de aplicação não definam corretamente as rotas. Com isso, o PROC permite que programadores que não conhecem o funcionamento do protocolo possam otimizar o processo de criação de rotas. O protocolo ainda implementa mecanismos simples de tolerância a falhas que aumentam a sua resiliência frente a falhas silenciosas. Estes mecanismos são eficientes em energia, pois não requerem o envio de mensagens adicionais.

Avaliamos o protocolo por simulações em um cenário que emula uma rede de sensores real. Verificamos que o PROC aumenta o tempo de vida da rede e permite que os nós se recuperem mais

rapidamente de falhas, devido à monitoração contínua do próximo salto. Além disso, o PROC é altamente escalável, pois a quantidade de mensagens de controle enviadas por nó e o consumo de memória independem do tamanho da rede. Implementamos o protocolo na plataforma Mica2, onde verificamos que a quantidade de memória RAM e ROM requerida pelo PROC é inferior à utilizada pelo protocolo de roteamento padrão da plataforma.

6.1 Contribuições

A principal contribuição desta dissertação é o desenvolvimento de um protocolo de roteamento para RSSF, chamado PROC. Este protocolo provê mecanismos de tolerância a falhas e possui um desempenho superior ao de protocolos encontrados na literatura. Avaliamos o protocolo por simulações, que compararam o seu desempenho ao de dois outros protocolos.

O código utilizado na simulação do PROC se encontra disponível na Internet, permitindo que outros pesquisadores o utilizem para seus próprios trabalhos. Disponibilizamos a implementação dos protocolos avaliados (PROC, EAD e TREE) e as modificações realizadas ao NS-2 para que este simule uma RSSF com maior fidelidade.

Implementamos o PROC na plataforma Mica2, que é a plataforma mais utilizada na pesquisa em RSSF. A partir desta implementação, mostramos o funcionamento do PROC em um ambiente real e avaliamos o seu consumo de recursos. A implementação do PROC e os *softwares* auxiliares, como a *interface* gráfica de depuração e a aplicação de testes, se encontram disponíveis para *download*.

Desenvolvemos um modelo de falhas que engloba as falhas silenciosas que ocorrem no roteamento. Este modelo facilita a avaliação dos protocolos em cenários onde ocorrem falhas, pois aglutina múltiplas falhas de acordo com as suas principais características. Com isso, reduzimos o número de cenários a serem avaliados a três, sendo que as falhas em cada cenário possuem no máximo dois parâmetros a serem variados.

Resultados parciais foram publicados em conferências e periódicos nacionais e internacionais. Obtivemos uma publicação em periódico internacional, duas publicações em conferências internacionais e dois artigos completos em conferências nacionais, sendo que um dos artigos internacionais obteve o prêmio *Best Paper Award on Simulation*, oferecido pelo *IEEE Communications Society Technical Committee on Simulation*. Vide o Apêndice A para a lista completa de publicações.

6.2 Trabalhos Futuros

Ao contrário das MANETs e redes cabeadas, a função de roteamento em RSSF requer a frequente interação de camadas da pilha de comunicação para realizar o seu trabalho de forma satisfatória. Pretendemos continuar o desenvolvimento do PROC, adicionando interações do mesmo com a camada de controle e acesso ao meio e com a camada física. Estas alterações terão

como objetivo aumentar desempenho e tolerância a falhas do protocolo. Também pretendemos aumentar a aplicabilidade do PROC ao adicionar funcionalidades que permitam a sua utilização em cenários com mobilidade. Além disso, pretendemos disponibilizar o código em outras plataformas, sejam reais ou de simulação.

Comparação entre Técnicas de Tolerância a Falhas

Como visto na Seção 2.4, protocolos que utilizam escuta promíscua, como o ExOR e o CBS, são empregados em MANETs para aumentar a resiliência das rotas. Esta técnica ainda não foi avaliada em RSSF e desta forma não podemos precisar certamente qual é o seu custo em comparação ao uso de rotas redundantes.

Assim, pretendemos implementar ambas as técnicas no PROC para avaliar se o maior consumo de energia proveniente da escuta promíscua é compensado por uma maior tolerância a falhas. Com o uso das regras do PROC, poderíamos variar o grau de tolerância a falhas que o ExOR e o CBS provêm. Caso os nós que operam em modo promíscuo sejam coordenadores, o grau de tolerância a falhas pode ser ajustado dinamicamente pela densidade de coordenadores na região.

Construção de Rotas Empregando Técnicas de Controle de Potência

Tradicionalmente, a construção de rotas em RSSF considera que a energia empregada no repasse de dados é a mesma para qualquer vizinho. Quando empregamos um protocolo MAC que utiliza técnicas de controle de potência de transmissão (CPT), a energia consumida varia de acordo com a qualidade do enlace e a distância entre o nó emissor e receptor [15]. Além disso, a vazão do enlace é variável, uma vez que os protocolos MAC podem modificar sua codificação e modulação quando a taxa de erro do canal é muito grande [46]. Assim, os protocolos de roteamento existentes devem ser modificados para que operem corretamente quando empregados em conjunto com protocolos MAC que utilizam técnicas de CPT.

Pretendemos estender o PROC para que o algoritmo de criação de rotas considere a potência de transmissão, qualidade e vazão de cada enlace na construção das rotas. Este projeto se encontra em curso, estando no momento com dois artigos em fase de submissão (vide Apêndice A, artigos 1 e 2 da lista de trabalhos em submissão).

Suporte Para um PA Móvel

Em regiões extremamente inóspitas, o uso de um PA fixo pode ser impossível ou muito custoso. Assim, faz-se necessário o uso de um PA móvel, por exemplo montado sobre um dirigível, que sobrevoa a região para coletar os dados produzidos pela rede. Este PA então repassa os dados por um enlace via satélite ou por uma tecnologia de rede sem fio de longa distância.

Quando o PA se move lentamente, a reconstrução de rotas pode ser pouco frequente. Por outro lado, quando o PA se move rapidamente, as rotas devem ser reconstruídas frequentemente. Caso as rotas não sejam atualizadas a tempo, o desempenho da rede irá se degradar, pois as rotas poderão estar inválidas ou serão mais longas que o necessário. Assim, o intervalo entre cada reconstrução de rotas deve ser adaptativo, refletindo a velocidade do movimento do PA. Como a reconstrução das rotas pode ser frequente, o protocolo de roteamento deve procurar minimizar o número de nós que devem ser comunicados da mudança nas rotas.

Integração do PROC ao Simulador MannaSim

O simulador MannaSim adiciona módulos ao simulador NS-2 para que este modele RSSF com maior fidelidade. A integração do PROC ao MannaSim aumentaria a precisão dos resultados obtidos e diminuiria o trabalho requerido por pesquisadores que desejem utilizar o PROC em suas pesquisas.

Referências Bibliográficas

[1] Kemal Akkaya and Mohamed Younis. A survey of routing protocols in wireless sensor networks. *Elsevier Ad Hoc Network Journal*, 3(3):325–349, 2005.

[2] Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless sensor and actor networks: Research challenges. *Elsevier Ad Hoc Networks Journal*, 2(4):351–367, October 2004.

[3] Ian F. Akyildiz, Dario Pompili, and Tommaso Melodia. Underwater acoustic sensor networks: research challenges. *Ad Hoc Networks*, 3(3):257–279, 2005.

[4] Alberto Cerpa and Deborah Estrin. ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. In *IEEE INFOCOM 2002*, New York, NY, June 23–27 2002.

[5] Antonio Alfredo F. Loureiro and José Marcos S. Nogueira and Lynnier Beatrys Ruiz and Raquel Aparecida de Freitas Mini and Eduardo Freire Nakamura and Carlos Maurício Seródio Figueiredo. Redes de Sensores Sem Fio. In *21º Simpósio Brasileiro de Redes de Computadores*, pages 179 – 226, Maio 2003.

[6] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, 2004.

[7] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM Press, 2002.

[8] Sanjit Biswas and Robert Morris. Opportunistic routing in multi-hop wireless networks. *SIGCOMM Computer Communications Review*, 34(1):69–74, 2004.

[9] Gary Boone. Reality mining: Browsing reality with sensor networks. *Sensors Magazine*, September, 2004.

[10] Azzedine Boukerche, Xiuzhen Cheng, and Joseph Linus. Energy-aware data-centric routing in microsensor networks. In *Proceedings of the 6th international workshop on Modeling*

analysis and simulation of wireless and mobile systems, pages 42–49. ACM Press, September 2003.

[11] Jenna Burrell, Tim Brooke, and Richard Beckwith. Vineyard computing: sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1):38–45, March 2004.

[12] CC1000. Cc1000 – single chip very low power RF transceiver. http://www.chipcon.com/files/CC1000_Data_Sheet_2_3.pdf, Aug 2005.

[13] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5):481–494, 2002.

[14] Luiz H. A. Correia, Daniel F. Macedo, Aldri L. dos Santos, José Marcos S. Nogueira, and Antonio A. F. Loureiro. A taxonomy for medium access control protocols in wireless sensor networks. *Annales des Télécommunications (Annals of Telecommunications)*, 60(7-8):944–969, July/August 2005.

[15] Luiz H. A. Correia, Daniel F. Macedo, Daniel A. C. Silva, Aldri L. dos Santos, Antonio A. F. Loureiro, and José; Marcos S. Nogueira. Transmission power control in mac protocols for wireless sensor networks. In *MSWiM '05: Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 282–289, New York, NY, USA, 2005. ACM Press.

[16] Crossbow. Mica2 Series (MPR4x0). http://www.xbow.com/Products/productsdetails.aspx?sid=72, January 2006.

[17] Crossbow, Inc. Crossbow technology releases security solution for perimeter monitoring, intrusion
detection, and identification.
http://www.xbow.com/General_info/Info_pdf_files/Security_Press_Release.pdf, March 2005.

[18] Crossbow Technology, Inc. MPR/MIB mote user manual, October 2003.

[19] Bevan Das and Vaduvur Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *IEEE International Conference on Communications*, pages 376–380, June 1997.

[20] Swades De, Chunming Qiao, and Hongyi Wu. Meshed multipath routing with selective forwarding: an efficient strategy in wireless sensor networks. *IEEE Network*, 43(4):481–497, November 2003.

[21] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*, San Diego, California, September 2003.

[22] Budhaditya Deb, Sudeept Bhatnagar, and Badri Nath. Reinform: reliable information forwarding using multiple paths in sensor networks. In *Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks*, pages 406–415. IEEE Computer Society, October 2003.

[23] Deepak Ganesan and Ramesh Govindan and Scott Shenker and Deborah Estrin. Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks. *SIGMOBILE Mobile Computing and Communications Review*, 5(4):11–25, 2001.

[24] Eduardo H. B. Maia and Daniel Camara and Antonio A. F. Loureiro. ICA: Um novo algoritmo de roteamento para redes de sensores. In *22º Simpósio Brasileiro de Redes de Computadores*, May 2003.

[25] Computer Engineering and ETH Zurich Networks Laboratory. Btnodes - a distributed environment for prototyping ad hoc networks. http://btnode.ethz.ch/, January 2006.

[26] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: scalable coordination in sensor networks. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 263–270. ACM Press, August 1999.

[27] Laura M. Feeney. A taxonomy for routing protocols in mobile ad hoc networks. Technical Report T99–07, Swedish Institute of Computer Science, 1, 1999.

[28] Carlos M. S. Figueiredo, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Protocolo adaptativo híbrido para disseminação de dados em redes de sensores sem fio auto-organizáveis. In *22º Simpósio Brasileiro de Redes de Computadores (SBRC 2004)*, pages 43–56, Gramado, RS, Brasil, May 2004.

[29] Carlos Maurício S. Figueiredo, Aldri L. dos Santos, and José Marcos S. Nogueira Antonio A. F. Loureiro. Policy-based adaptive routing in autonomous wsns. In *16th IFIP/IEEE Distributed Systems: Operations and Management*, October 2005.

[30] Gregor Gaertner and Vinny Cahill. Understanding link quality in 802.11 mobile ad hoc networks. *IEEE Internet Computing*, 8(1):55–60, 2004.

[31] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03:*

Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, pages 1–11, New York, NY, USA, 2003. ACM Press.

[32] GNU Lesser General Public License. http://www.gnu.org/copyleft/lesser.html, February, 2006.

[33] Tian He, John A. Stankovic, Chenyang Lu, and Tarek F. Abdelzaher. SPEED: A stateless protocol for real-time communication in sensor networks. In *23rd International Conference on Distributed Computing Systems*, pages 46–57, May 2003.

[34] Cay S. Horstmann, Gary Cornell, and Cay S. Forstmann. *Core Java 2: Fundamentals*. Prentice Hall, 2002.

[35] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A Survey on Sensor Networks. *IEEE Communications*, 40(8):102–114, 2002.

[36] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the sixth annual international conference on Mobile computing and networking*, pages 56–67. ACM Press, 2000.

[37] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *ACM/IEEE Transactions on Networking*, 11(1):2–16, February 2002.

[38] Intel Inc. New computing frontiers – the wireless vineyard. http://www.intel.com/technology/techresearch/research/rs01031.htm, October, 2005.

[39] Jianliang Zheng and Lee, M.J. Will IEEE 802.15.4 make ubiquitous networking a reality?: a discussion on a potential low power, low bit rate standard. *IEEE Communications Magazine*, 42(6):277–288, June 2004.

[40] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Charles E. Perkins, editor, *Ad Hoc Networking*, pages 139–172. Addison-Wesley, 2001.

[41] K. Sohrabi and J. Gao and V. Ailawadhi and G. Pottie. Protocols for Self-Organization of a Wireless Sensor Network. *IEEE Personal Communications*, 7(5):16–27, 2000.

[42] Chris Karlof, Yaping Li, and Joseph Polastre. ARRIVE: Algorithm for robust routing in volatile environments. Technical Report UCB//CSD-03-1233, University of California, Berkeley, CA, March 2003.

[43] Chris Karlof, Naveen Sastry, and David Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175. ACM Press, 2004.

[44] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In *IEEE International Workshop on Sensor Network Protocols and Applications*, pages 113–127, February 2003.

[45] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254. ACM Press, August 2000.

[46] V. Kawadia and P. R. Kumar. A cautionary perspective on cross layer design. *IEEE Wireless Communications*, 12(1):3–11, February 2005.

[47] Brendan I. Koerner. Intel's tiny hope for the future. *Wired*, 11.12, December 2003.

[48] Jiejun Kong, Xiaoyang Hong, Yunjung Yi, Joon-Sang Park, Jun Liu, and Mario Gerla. A secure ad-hoc routing approach using localized self-healing communities. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 254–265, New York, NY, USA, 2005. ACM Press.

[49] Vijay Kumar, Daniela Rus, and Sanjiv Singh. Robot and sensor networks for first responders. *IEEE Pervasive Computing*, 3(4):24–33, October-December 2004.

[50] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137. ACM Press, 2003.

[51] Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in TinyOS. In *First Symposium on networked system design and implementation (NSDI04)*, pages 1–14, San Francisco, California, USA, March 2004.

[52] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. TinyOS: An operating system for wireless sensor networks. In W. Weber, J. Rabaey, and E. Aarts, editors, *Ambient Intelligence*. Springer-Verlag, New York, NY, 2004.

[53] Stephanie Lindsey, C. S. Raghavendra, and K. Sivalingam. Data Gathering Algorithms in Sensor Networks Using Energy Metrics. IEEE Transactions on Parallel and Distributed Systems, September 2002.

[54] Juan Liu, Feng Zhao, and Dragan Petrovic. Information-directed routing in ad hoc sensor networks. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2003. ACM Press.

[55] Konrad Lorincz, David J. Malan, Thaddeus R. F. Fulford-Jones, Alan Nawoj, Antony Clavel, Victor Shnayderk, Geoffrey Mainland, Matt Welsh, and Steve Moulton. Sensor networks for emergency response: Challenges and opportunities. *IEEE Pervasive Computing*, 3(4):16–23, 2004.

[56] Mihaela Cardei and Jie Wu. Energy-Efficient Coverage Problems in Wireless Ad Hoc Sensor Networks. *Journal of Computer Communications on Sensor Networks*, 2004.

[57] moteiv. moteiv: wireless sensor networks. http://www.moteiv.com/products-reva.php, January 2006.

[58] Shree Murthy and J. J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mob. Netw. Appl.*, 1(2):183–197, 1996.

[59] Eduardo F. Nakamura, Carlos M. S. Figueiredo, and Antonio A.F. Loureiro. Disseminação adaptativa de dados em redes de sensores sem fio auto-organizáveis. In *22º Simpósio Brasileiro de Redes de Computadores (SBRC 2004)*, pages 29–42, Gramado, RS, Brasil, May 2004.

[60] National Science Foundation. Report of the National Science Foundation Workshop on Fundamental Research in Networking. http://www.cs.virginia.edu/~jorg/workshop, January 2004.

[61] NS-2 simulator. http://www.isi.edu/nsnam/ns/, November, 2005.

[62] Division of Engineering and Harvard University Applied Sciences. Codeblue: Wireless sensor networks for medical care. http://www.eecs.harvard.edu/~mdw/proj/codeblue/, January 2006.

[63] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244, New York, NY, USA, 1994. ACM Press.

[64] Charles E. Perkins and Elizabeth M. Royer. Ad hoc on-demand distance vector routing. In *2nd IEEE Workshop on Mobile Computing System and Applications*, pages 90–100, New Orleans, LA, February 1999.

[65] Ad hoc routing component architecture. http://www.tinyos.net/tinyos-1.x/doc/ad-hoc.pdf, February, 2003.

[66] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107. ACM Press, November 2004.

[67] Ananth Rao, Christos Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108, New York, NY, USA, 2003. ACM Press.

[68] Niels Reijers, Gertjan Halkes, and Koen Langendoen. Link layer measurements in sensor networks. In *1st IEEE Int. Conference on Mobile Ad hoc and Sensor Systems (MASS '04)*, pages 224–234, October 2004.

[69] Vineet Srivastava and Mehul Motani. Cross-layer design: A survey and the road ahead. *IEEE Communications Magazine*, 43(12):112–119, December 2005.

[70] Stephanie Lindsey and Cauligi S. Raghavendra. PEGASIS: Power-Efficient Gathering in Sensor Information Systems. In *Proceedings of IEEE Aerospace Conference*, volume 3, pages 1125–1130. IEEE, March 2002.

[71] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, pages 307–322, January 2004.

[72] Andrew S. Tanenbaum. *Computer networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 4 ed. edition, 2002.

[73] Sameer Tilak, Nael B. Abu-Ghazaleh, and Wendi Heinzelman. A taxonomy of wireless micro-sensor network models. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(2):28–36, 2002.

[74] Tijs van Dam and Koen Langendoen. An Adaptive Energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 171–180. ACM Press, 2003.

[75] Marcos Augusto M. Vieira, Luis Filipe M. Vieira, Linnyer Beatrys Ruiz, Antonio A. F. Loureiro, Antônio O. Fernandes, José Marcos S. Nogueira, and Diógenes C. da Silva Jr. Como obter o mapa de energia em redes de sensores sem fio? uma abordagem tolerante a falhas. In *Anais do 5o. Workshop de Comunicação sem Fio (WCSF)*, pages 183–189, 2003.

[76] Bernhard Walke, Norbert Esseling, Jörg Habetha, Andreas Hettich, Arndt Kadelka, Stefan Mangold, Jörg Peetz, and Ulrich Vornefeld. IP over Wireless Mobile ATM - Guaranteed Wireless QoS by HiperLAN/2. *Proceedings of the IEEE*, 89:21–40, Jan 2001.

[77] Peng-Jun Wan, Khaled M. Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9(2):141–149, 2004.

[78] Wei Ye and John Heidemann and Deborah Estrin. An Energy-Efficient MAC protocol for Wireless Sensor Networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, USA, June 2002. USC/Information Sciences Institute, IEEE.

[79] Wendi Rabiner Heinzelman and Anantha Chandrakasan and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, 2000.

[80] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *Proceedings of the 2nd European Workshop on Sensor Networks (EWSN'05)*, January 2005.

[81] Alec Woo and David E. Culler. A transmission control scheme for media access in sensor networks. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 221–235, 2001.

[82] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 14–27. ACM Press, 2003.

[83] Anthony D. Wood and John A. Stankovic. Denial of service in sensor networks. *Computer*, 35(10):54–62, 2002.

[84] Fan Ye, Alvin Chen, Songwu Lu, and Lixia Zhang. A scalable solution to minimum cost forwarding in large sensor networks. In *Proceedings of the 10th IEEE International Conference on Computer Communications and Networks*, pages 304–309, 2001.

[85] Fan Ye, Haiyun Luo, Jerry Cheng, Songwu Lu, and Lixia Zhang. A two-tier data dissemination model for large-scale wireless sensor networks. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 148–159, New York, NY, USA, 2002. ACM Press.

[86] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang. Gradient broadcast: A robust data delivery protocol for large scale sensor networks. *Wireless Networks*, 11(3):285–298, 2005.

[87] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, UCLA Computer Science Department, May 2001.

[88] Di Yuan. Energy-efficient broadcasting in wireless ad hoc networks: performance benchmarking and distributed algorithms based on network connectivity characterization. In *MSWiM '05: Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 28–35, New York, NY, USA, 2005. ACM Press.

Apêndice A

Lista de Publicações no Período

Durante o mestrado, foram publicados artigos em conferências e periódicos nacionais e internacionais relatando resultados parciais dos estudos realizados. A lista abaixo contém publicações resultantes de trabalhos paralelos ao tema da dissertação, que foram desenvolvidos durante o mestrado. São diretamente referentes à dissertação as publicações 3, 5, 7, 9, 12, 14 e 15.

1. Luiz H. A. Correia, Daniel F. Macedo, Aldri Luiz dos Santos, Antonio A. Loureiro, José M. Nogueira. Ajustando a Potência de Transmissão em Protocolos MAC. *24º Simpósio Brasileiro de Redes de Computadores*, 2006.

2. Daniel F. Macedo, Pedro F. Macedo, Luiz H. A. Correia, Aldri Luiz dos Santos, Antonio A. Loureiro, José M. Nogueira. Um Protocolo de Roteamento para Redes Ad Hoc com QoS Baseado no Controle da Potência de Transmissão. *24º Simpósio Brasileiro de Redes de Computadores*, 2006.

3. Daniel F. Macedo, Luiz H. A. Correia, Aldri L. dos Santos, Antonio A. F. Loureiro e José M. Nogueira. A Rule-based Adaptive Routing Protocol for Continuous Data Dissemination in WSNs. *Journal of Parallel and Distributed Computing*, 66(4):542-555, Abril 2006.

4. Luiz H. A. Correia, Daniel F. Macedo, Aldri L. dos Santos, José M. S. Nogueira e Antonio A. F. Loureiro. A taxonomy for medium access control protocols in wireless sensor networks. *Annales des Télécommunications (Annals of Telecommunications)*, 60(7-8):944–969, Julho/Agosto 2005.

5. Daniel F. Macedo, Luiz H. A. Correia, Aldri L. dos Santos, Antonio A. F. Loureiro e José M. Nogueira. A pro-active routing protocol for continuous data dissemination wireless sensor networks. In *10th IEEE Symposium on Computer and Communications (ISCC)*, páginas 361–366, Junho 2005 **(Premiado com o "Best Paper Award on Simulation", oferecido pelo IEEE Computer Society Technical Committee on Simulation)**.

6. Luiz Henrique A. Correia, Daniel F. Macedo, Daniel A. C. Silva, Aldri L. dos Santos, Antonio A. F. Loureiro e José Marcos S. Nogueira. Transmission Power Control in MAC Protocols for Wireless Sensor Networks. In *ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, páginas 282–289, Outubro 2005.

7. Leonardo B. Oliveira, Isabela G. Siqueira, Daniel F. Macedo, Antonio A. F. Loureiro e José M. Nogueira. Evaluation of peer-to-peer network content discovery techniques over mobile ad hoc networks. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, páginas 51–56, Junho 2005.

8. Daniel F. Macedo, Luiz H. A. Correia, Aldri L. dos Santos, Antonio A. F. Loureiro, José M. S. Nogueira e Guy Pujolle. Evaluating Fault Tolerance Aspects in Routing Protocols for Wireless Sensor Networks. In *Fourth Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, Junho 2005.

9. Daniel F. Macedo, Luiz H. A. Correia, Aldri L. dos Santos, Antonio A. F. Loureiro e José M. S. Nogueira. Um protocolo de roteamento configurável baseado em regras para redes de sensores sem fio. In *23º Simpósio Brasileiro de Redes de Computadores (SBRC)*, páginas 89–102, Maio 2005.

10. Leonardo B. Oliveira, Isabela G. Siqueira, Daniel F. Macedo, Antonio A. F. Loureiro, Hao Chi Wong e José M. S. Nogueira. Avaliação de técnicas de descoberta de conteúdo em redes peer-to-peer sobre redes móveis ad hoc. In *23º Simpósio Brasileiro de Redes de Computadores (SBRC)*, páginas 553–564, Maio 2005.

11. Luiz H. A. Correia, Daniel F. Macedo, Daniel A. C. Silva, Aldri L. dos Santos, Antonio A. F. Loureiro e José Marcos S. Nogueira. Controle de Potência de Transmissão em Protocolos MAC para Redes de Sensores Sem Fio. In *XXII Simpósio Brasileiro de Telecomunicações - SBrT*, páginas 709–714, Setembro 2005.

12. Daniel F. Macedo, Luiz H. A. Correia, Aldri L. dos Santos, Antonio A. Loureiro e José M. S. Nogueira. Avaliando aspectos de tolerância a falhas em protocolos de roteamento para redes de sensores sem fio. In *VI Workshop de Testes e Tolerância a Falhas (WTF)*, Maio, páginas 15–26, 2005.

13. Daniel F. Macedo, Leonardo B. Oliveira e Antonio A. F. Loureiro. Integrando redes overlay e redes de sensores sem fio. *Telecomunicações*, 7(1):36–43, 2004. Editor: José Marcos Camara Brito.

14. Linnyer B. Ruiz, Luiz H. A. Correia, Luiz F. M. Vieira, Daniel F. Macedo, Eduardo F. Nakamura, Carlos M. S. Figueiredo, Marcos A. M. Vieira, Eduardo H. B. Maia, Daniel

Camara, Antonio A. F. Loureiro, José Marcos S. Nogueira, Diógenes C. da Silva Jr. e Antônio O. Fernandes. Arquiteturas para Redes de Sensores Sem Fio. In *22º Simpósio Brasileiro de Redes de Computadores (SBRC)*, páginas 167–218, Maio 2004.

15. Daniel F. Macedo, Luiz H. A. Correia, Antonio A. F. Loureiro e José Marcos S. Nogueira. PROC: Um Protocolo Pró-ativo com Coordenação de Rotas em Redes de Sensores sem Fio. In *22º Simpósio Brasileiro de Redes de Computadores (SBRC)*, páginas 571–574, Maio 2004.