Um Protocolo de Roteamento Configurável Baseado em Regras para Redes de Sensores Sem Fio*

Daniel F. Macedo[1], Luiz H. A. Correia[1,2], Aldri L. dos Santos[1], Antonio A. F. Loureiro[1], José Marcos S. Nogueira[1†]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte-MG, Brasil

[2]Departamento de Ciência da Computação
Universidade Federal de Lavras
Lavras-MG, Brasil

{damacedo,lcorreia,aldri,loureiro,jmarcos}@dcc.ufmg.br

Abstract. *Wireless sensor networks are ad hoc networks with severe resource constraints. These constraints preclude the use of traditional ad hoc protocols, and demand optimizations that incur in solutions specific to a class of applications. This article presents PROC, a protocol designed for continuous data dissemination networks, that interacts with the application to stablish routes. This mechanism allows the application to reconfigure PROC on runtime. A performance evaluation in topologies varying from 50 to 200 nodes showed that PROC increases network lifetime around 7% to 12%, and has higher throughput than EAD and TinyOS Beaconing. Furthermore, PROC presents a softer performance degradation when the number of nodes in the network increases.*

Resumo. *Redes de sensores sem fio são redes ad hoc que possuem restrições severas de recursos. Essas restrições impossibilitam o uso de protocolos ad hoc tradicionais e requerem otimizações, que implicam em soluções específicas para uma classe de aplicações. Apresentamos um protocolo para redes de disseminação contínua de dados, chamado PROC, que interage com a aplicação para estabelecer rotas. Este mecanismo permite que a aplicação reconfigure o roteamento em tempo de execução. Simulações realizadas para redes de 50 a 200 nós mostram que o PROC, em comparação aos protocolos EAD e TinyOS Beaconing, apresenta maior vazão e aumenta o tempo de vida da rede entre 7% e 12%. PROC também apresenta degradação de desempenho mais suave com o incremento de nós na rede.*

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 para comunicação, memória e bateria. Esses nós enviam os dados coletados do ambiente para um Ponto de Acesso (PA), responsável por repassar os dados ao usuário final [1]. Diferentemente das redes ad hoc tradicionais, em geral não é possível trocar ou recarregar a fonte de energia dos nós

*O presente trabalho foi realizado com apoio do CNPq, uma entidade do Governo Brasileiro voltada ao desenvolvimento científico e tecnológico. Processo 55.2111/2002-3.

†Em período sabático nas universidades de Evry e UPMC/Paris6/Lip6, França.

sensores devido à grande quantidade de nós ou às dificuldades impostas pelo ambiente. Assim, o consumo de energia é um fator crítico em RSSF, o que tem motivado a busca de protocolos específicos para essas redes que tomam decisões procurando otimizar o consumo de energia, entre outras funções. Protocolos tradicionais utilizados em redes ad hoc não são aplicáveis diretamente às RSSF, uma vez que as redes ad hoc típicas não apresentam restrições de recursos tão severas quanto as encontradas nas RSSF [1]. Busca-se economia de energia tirando proveito de características específicas da rede, tais como nos protocolos para redes dirigidas a eventos [2], nos protocolos para nós sensores que utilizam informações geográficas [3], nos protocolos para aplicações de tempo real [4], entre outros. Outra forma é o desenvolvimento de protocolos que interagem com a aplicação ou com o hardware para exercerem as suas funções de forma mais eficiente. Nesses protocolos, a separação de funções entre as camadas é violada, uma vez que a aplicação pode, por exemplo, interagir diretamente com os protocolos de acesso ao meio [5] e de roteamento [2] para evitar colisões e estabelecer rotas de forma eficiente.

Em RSSF o fluxo de dados segue um padrão. O envio de dados dos nós ao PA pode ser periódico ou esporádico, caracterizando dois tipos de fluxo de dados [6]. 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 o PA, relatando as leituras atuais dos seus sensores. Aplicações dessas redes incluem estudos ambientais, sistemas de tráfego inteligente, monitoração de plantas industriais, entre outros.

Os protocolos de roteamento geralmente são implementados para satisfazer a apenas uma classe de rede. Redes de disseminação contínua tendem a utilizar protocolos pró-ativos, pois a todo momento os nós enviam dados para o PA. Em redes dirigidas a eventos os protocolos são reativos, assim as rotas são construídas somente na ocorrência de um evento de interesse, e apenas na região onde o evento ocorreu. Como os eventos tendem a ser raros e isolados em uma região, a reconstrução periódica de rotas não é recomendável devido ao alto custo de energia associado.

Neste trabalho é proposto um protocolo de roteamento específico para redes de disseminação contínua de dados, que utiliza informações da aplicação para cumprir suas funções. O protocolo proposto, denominado PROC (*Proactive ROuting with Coordination*), interage com a aplicação para determinar os nós roteadores de dados. A interação entre o protocolo PROC e a aplicação é baseado em regras. As regras auxiliam o PROC na formação de rotas, atribuindo uma pontuação ao nó. Quanto maior o número de pontos acumulados em um nó, maior a chance do nó ser escolhido como roteador de dados. 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) em tempo de execução, de acordo com as necessidades da aplicação. Assim, a aplicação pode interagir com o roteamento, utilizando informações como posição geográfica, importância da informação sensoriada, estado atual do ambiente e do sensor, topologia da rede, entre outras, para auxiliar no estabelecimento de rotas. Por definir claramente quais são os nós roteadores de dados, o PROC possibilita que a aplicação ou protocolos de controle de topologia desliguem nós que não estão roteando dados, sem prejuízo para o fluxo de dados na rede. O PROC ainda provê mecanismos de tolerância a falhas, que detectam rotas falhas ou com enlaces de baixa qualidade.

O desempenho do protocolo foi avaliado considerando outros protocolos para RSSF. Comparamos o PROC com duas soluções existentes na literatura, o EAD [7] e o TinyOS Beaconing [8], sendo o último largamente utilizado em redes de sensores em

operação. Não fizemos comparações de desempenho com protocolos de redes ad hoc devido às especificidades dos protocolos de RSSF, não sendo possível uma comparação.

O texto está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta uma visão geral do protocolo, a Seção 4 mostra a operação do protocolo e os seus algoritmos. A Seção 5 estima a complexidade do protocolo, tanto em número de mensagens quanto em consumo de memória. Uma avaliação do desempenho do protocolo por meio de simulação é apresentada na Seção 6. Por fim, a Seção 7 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

O uso de informações da aplicação no roteamento foi introduzidos no protocolo Directed Diffusion [2]. 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. Entretanto, o Directed Diffusion mostrou-se extremamente ineficiente e de difícil adaptação quando aplicado em redes de disseminação contínua de dados [9, 10].

O protocolo Span é um protocolo de manutenção de topologia para redes ad hoc, que também estabelece quais nós da rede irão rotear dados [11]. A formação das rotas é delegada para um protocolo de roteamento qualquer. No Span, os nós roteadores, chamados de *coordenadores*, formam uma estrutura de roteamento chamada de *backbone*. Para determinar os nós coordenadores, o protocolo leva em conta a energia residual dos nós e a sua conectividade. Os nós que não pertencem ao *backbone* podem ser desligados para economizar energia. O protocolo Span não é aplicável a redes de sensores, pois consome grande quantidade de memória, sendo intensivo em processamento, realizando $O(v^6)$ operações para determinar o *backbone* (onde v é o número médio de vizinhos de um nó). Como o Span não define rotas, devemos adicionar a seu custo um protocolo de roteamento. O PROC, assim como o Span, constrói um *backbone* que permite a manutenção da topologia, e adicionalmente define rotas.

O TinyOS Beaconing é o protocolo de roteamento utilizado como padrão na plataforma de nós sensores Mica Motes [8]. Este protocolo recria periodicamente uma árvore de roteamento de menor caminho, com raiz no PA. Para criar essa árvore, o PA envia periodicamente uma mensagem *beacon* para a rede, determinando a distância em saltos dos nós até o PA. A seleção das rotas também leva em conta a confiabilidade do enlace, calculada pelo número de pacotes corretamente enviados pelos nós vizinhos. Para reduzir o número de retransmissões, apenas nós com enlaces confiáveis são usados no roteamento. Visto que o desligamento do rádio prejudica o cálculo de confiabilidade dos enlaces, o que degrada a eficiência das rotas, o TinyOS Beaconing não realiza esse procedimento. O PROC, entretanto, não utiliza o cálculo de confiabilidade, o que permite o desligamento periódico do rádio, aumentando o tempo de vida da rede.

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 [7]. Os nós folha não roteiam dados, e portanto podem manter o seu rádio desligado por períodos prolongados de tempo. O EAD também utiliza um período de *backoff* baseado na energia residual, que diminui significativamente a quantidade de colisões. O PROC, além de apresentar essas características, possibilita à aplicação influenciar na escolha dos nós de roteamento, reduzindo ainda mais o consumo de energia.

3. O Protocolo PROC

PROC é um protocolo pró-ativo desenvolvido para redes de disseminação contínua de dados. Esse protocolo considera aplicações com um padrão de tráfego *many-to-one*: os nós enviam dados ao ponto de acesso utilizando um caminho multi-saltos. Nessas aplicações, um nó comunica-se com seus nós vizinhos e com o ponto de acesso, permitindo assim que sejam construídas rotas unidirecionais dos nós sensores a um PA [7, 8, 12]. Isso simplifica o roteamento, visto que não são construídas rotas para cada nó. A construção das rotas é baseada apenas em informações da vizinhança dos nós.

O protocolo PROC constrói uma estrutura de roteamento chamada de *backbone*. Esse *backbone* é composto por um conjunto de nós, chamados *coordenadores*, que propagam informação em direção ao ponto de acesso (PA). A estrutura do *backbone* é na forma de árvore, tendo o PA como sua raiz. Nós que não fazem parte do *backbone* comunicam-se diretamente com um nó do *backbone* (nó coordenador). Os nós precisam conhecer apenas o nó pai no *backbone*, o qual repassa os dados recebidos em direção ao PA. Essa estrutura permite o uso de algoritmos de controle de topologia, onde os nós não coordenadores são colocados em modo de baixo consumo de energia. Como o *backbone* garante a cobertura de rádio da rede, protocolos de controle de topologia precisam apenas considerar a cobertura de sensoriamento [13]. 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 de tempo chamados de *ciclos*.

Criação do Backbone

A criação do *backbone* é um processo de duas fases. Inicialmente, os nós se auto-elegem coordenadores utilizando um processo probabilístico. Cada nó tem uma certa probabilidade de tornar-se coordenador. Essa probabilidade é calculada pela aplicação, por meio de regras. As regras podem utilizar informações específicas da aplicação, e são tratadas na Seção 4.4. 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 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 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 1(b) mostra um nó 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 1(c).

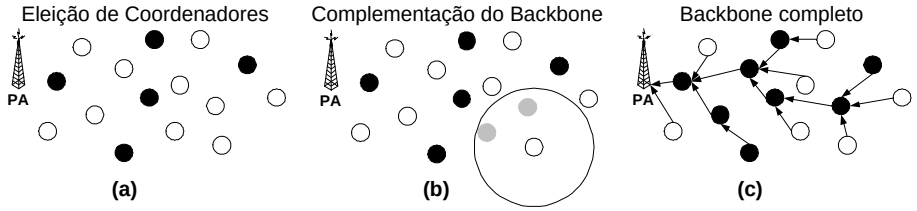

Figura 1: O processo de criação do *backbone*.

4. Operação do Protocolo

Esta seção apresenta 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/não coordenador), 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 permite que a aplicação envie dados próprios embutidos na mensagem, que serão utilizados pelas regras da aplicação. A cada pacote de roteamento recebido, o PROC extrai esta informação e a repassa para a aplicação.

4.1. Ponto de Acesso

O ponto de acesso é responsável por iniciar um ciclo, e por conseqüência 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.

Algoritmo 1 Ponto de Acesso: Manutenção do Backbone.

```
 1: procedure BaseStation(CycleTime)
 2:     nextCycle ← 0;                                  // Ciclo iniciado
 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
```

4.2. 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.$6-22), e compreende duas fases: a eleição dos coordenadores e a complementação do *backbone*. A primeira fase ocorre quando o nó recebe a mensagem M_{Sync}, executando o processo de eleição de coordenadores ($l.$10-12). Na segunda fase, os nós examinam se o *backbone* está completo e indicam, quando necessário, outros nós para se juntarem ao *backbone* ($l.$15-21).

Eleição de coordenadores: é o primeiro processo para a criação do *backbone*. Esse algoritmo utiliza regras providas pela aplicação para determinar se o nó será coordenador ou não. Para cada regra, o software de aplicação associa um valor V no intervalo $[0, V_{max}]$ para o nó ($l.$10). A soma dos valores associados a cada regra é um valor V_T, também entre $[0, V_{max}]$. A partir de V_T, o nó calcula a probabilidade de tornar-se coordenador, dada por $p = \frac{V_T}{V_{max}}$. Em seguida, um número aleatório entre 0 e 1 é gerado. Caso este seja menor que p, o nó será coordenador neste ciclo ($l.$11). Ao fim do processo, o nó envia uma mensagem M_{Sync} para a rede, reportando o seu novo estado para a vizinhança ($l.$13-14).

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.$15-21).

Caso a rota não passe por um coordenador, o nó envia uma mensagem de indicação de coordenador (M_{Coord}) para o novo nó pai escolhido ($l.$17-21). O receptor da mensagem

Algoritmo 2 Nós Sensores: Backbone de Roteamento.

```
 1: parent ← nil;                                                    // Nó pai na árvore
 2: curCycle ← integer;                                              // 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.state;
 7:   parent ← UpdateRoute(Neighbors);                               // Seleção do nó pai
 8:   if curCycle < M_Sync.cycle then
 9:     curCycle ← M_Sync.cycle;
10:     V_T ← App.ElectCoordinator(Neighbors);
11:     coordinator ← ( rand() < V_T / V_max );                      // Cálculo da prob. de ser coordenador
12:     App.NotifyNewState(coordinator);
13:     CurrentNodeState(M_Sync);
14:     send(M_Sync,BROADCAST);                                      // Propagação do estado do nó
15:     wait BackoffTime();
16:     parent ← UpdateRoute();                                      // Novo nó pai
17:     if parent.coordinator = false then                           // Força o nó pai a ser coordenador
18:       CurrentNodeState(M_Coord);
19:       send(M_Coord, parent);
20:       Neighbors_parent.coord ← true;
21:     end if
22:   end if
Require:  receive(M_Coord);                                          // Nó designado coordenador
23:   Neighbors ← Neighbors ∪ M_Coord.state;
24:   coordinator ← true;
25:   App.NotifyNewState(coordinator);
26:   CurrentNodeState(M_Sync);
27:   send(M_Sync,BROADCAST);
Require:  receive(M_Data);                                           // Envio de dados para o nó pai
28:   if App.forwardData(M_Data) = true then                         // Fusão ou descarte de dados pela aplicação
29:     send(M_Data, parent.address);
30:   end if
31: end procedure
```

M_{Coord} identifica que deverá se tornar coordenador, e envia uma mensagem M_{Sync} para os seus vizinhos, propagando seu novo estado (l.25-27). 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.

4.3. 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 é dependente do papel dos nós, coordenador ou não coordenador, e utiliza informações da vizinhança. O PROC armazena uma quádrupla $(hops, curCycle, coord, energy)$ para cada nó vizinho (representado como $Neighbors$ - conjunto de nós vizinhos - no Algoritmo 3). O campo $hops$ é a distância em saltos do nó vizinho até o PA, o campo $coord$ indica se o nó é coordenador ou não, e o campo $energy$ é a energia residual do nó. Em nossa implementação, o campo $curCycle$ é usado para garantir que a informação armazenada ainda é recente. A atualização da quádrupla é feita utilizando as informações obtidas dos pacotes de roteamento recebidos (l.6, l.23 do Algoritmo 2).

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 (l.2-12). A seleção de rotas pelo critério de menor distância evita ciclos. Caso haja empate, o nó com maior energia residual é selecionado.

Quando o nó não é coordenador, ele procura em sua vizinhança um nó coordenador com a menor distância até o PA. Se nenhum coordenador é encontrado, o nó sele-

ciona o nó vizinho com a menor distância até o PA. Se dois ou mais vizinhos possuem a menor distância até o PA, o nó com maior energia é escolhido (l.14-25).

Algoritmo 3 Nós sensores: Estabelecimento de rotas.

```
 1: procedure UpdateRoute( )
 2:     if coordinator = true then
 3:         hopsDist ← min(Neighbors, hops);
 4:         MinHops = {N ∈ Neighbors | N.hops = hopsDist};
 5:         if ∃N ∈ MinHops | N.coord = true then      // Conjunto de nós vizinhos coordenadores com menor distância
 6:             Coords ← {N ∈ MinHops | N.coord = true}
 7:             energy ← max(Coords, energy);
 8:             return {N ∈ Coords | N.energy = energy};  // Nó coordenador escolhido com menor distância e maior
                    energia
 9:         else
10:             energy ← max(MinHops, energy);
11:             return {N ∈ MinHops | N.energy = energy};      // Nó vizinho com menor distância e maior energia
12:         end if
13:     else
14:         if ∃N ∈ Neighbors | N.coord = true then                    // Conjunto de nós vizinhos coordenadores
15:             Coords ← {N ∈ Neighbors | N.coord = true}
16:             hopsDist ← min(Coords, hops);
17:             MinHops = {N ∈ Coords | N.hops = hopsDist};
18:             energy ← max(MinHops, energy);
19:             return {N ∈ MinHops | N.energy = energy};      // Nó coordenador escolhido com menor distância e
                    maior energia
20:         else
21:             hopsDist ← min(Neighbors, hops);
22:             MinHops = {N ∈ Neighbors | N.hops = hopsDist};
23:             energy ← max(MinHops, energy);
24:             return {N ∈ MinHops | N.energy = energy};      // Nó vizinho com menor distância e maior energia
25:         end if
26:     end if
27: end procedure
```

4.4. Regras de Aplicação

As regras de aplicação do PROC proporcionam uma interface entre a 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 neste ciclo, 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 (regra) em tempo de execução, de acordo com as necessidades da aplicação.

A aplicação pode usar qualquer informação disponível como entrada para o algoritmo que implementa a sua regra, tais como configuração dos nós e de outros protocolos, estado atual da rede, ou mesmo dados de sensores e atuadores. Exemplificando, é possível utilizar no estabelecimento de rotas dados como posição geográfica, topologia da rede, qualidade do sinal, energia restante, entre outros.

O PROC não exige que o software da aplicação implemente qualquer função de roteamento ou possua conhecimento do funcionamento do protocolo. Para tanto, o PROC garante que rotas serão estabelecidas mesmo que a aplicação não aponte nenhum coordenador. Além disto, o uso de regras não é mandatório no PROC. Essas aplicações podem utilizar um conjunto de regras padrão, descritas a seguir, permitindo assim que aplicações existentes possam utilizar o PROC como protocolo de roteamento sem nenhuma modificação.

A primeira regra é uma adaptação da eleição de líderes de grupo do protocolo LEACH [14], que tem como objetivo aumentar o tempo de vida da rede. Nesta regra, os nós sensores que recentemente foram coordenadores têm uma menor probabilidade de exercer o papel de coordenador nos ciclos seguintes, distribuindo a energia consumida de forma uniforme.

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 eleger um número fixo de coordenadores por região próximo 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.

4.5. Tolerância a Falhas

Em RSSF, tolerância a falhas é um requisito essencial para o projeto de protocolos e aplicações, pois falhas de hardware e comunicação são frequentes. Falhas podem ocorrer devido às condições severas do ambiente onde os nós foram depositados [1], ou devido à má qualidade dos enlaces sem fio [15, 16].

Para detectar a falha de nós ou um enlace com baixa qualidade, o PROC utiliza as mensagens de confirmação (ACKs) recebidas pelo protocolo de acesso ao meio para detectar quando o número de perdas consecutivas de pacotes é maior que um dado limite. Essa perda pode ser causada pela falha de um nó, ou por um enlace de baixa qualidade, onde é necessário um alto número de retransmissões para que o dado seja recebido corretamente. Ao detectar uma rota danificada, o PROC recalcula as suas rotas (não apresentado nos algoritmos).

Uma outra forma de aumentar a tolerância a falhas do PROC seria o uso de estimadores de qualidade de enlace, descritos em [17]. Essa informação pode ser utilizada como critério de desempate na seleção de rotas. Assim, existindo dois ou mais nós coordenadores na vizinhança com a mesma distância em saltos até o PA, o PROC selecionaria como pai o nó com o enlace de melhor qualidade.

5. Avaliação de Complexidade do Protocolo

O protocolo PROC armazena em cada nó o seu estado atual e o estado dos nós vizinhos, representado por quádruplas. A Tabela 1 mostra o formato da quádrupla, seu tamanho e descrição. Cada quádrupla consome 7 bytes. No cálculo do consumo de memória, tomamos como base o sistema operacional TinyOS [8]. Como cada nó sensor armazena o seu estado, o estado dos nós vizinhos e o identificador do seu pai, o protocolo PROC consome $((v+1) \times 7 + 2)$ bytes de memória, onde v é o número de vizinhos a um salto do nó, e os 2 bytes identificam o nó pai. Assim, o PROC escala linearmente com o número de vizinhos. Alec Woo et al. apresentam uma otimização que restringe o tamanho de dados armazenados sobre a vizinhança do nó, armazenando apenas informações sobre os nós mais favoráveis ao roteamento [17]. Pretendemos utilizar essa otimização para diminuir ainda mais o consumo de memória do PROC.

Em RSSF, uma parte significativa da energia é consumida na comunicação [18]. Assim, o número de mensagens enviadas deve ser minimizado. Para cada reconstrução do *backbone* uma rede com n nós deve enviar $3n$ mensagens no pior caso e $2n$ no melhor

Campo	Tamanho (bytes)	Descrição
$hops$	1	Distância do PA, em saltos
$curCycle$	1	Garante que o dado é atual
$coord$	1	Indica se o nó vizinho é coordenador
$energy$	2	Energia residual
$identifier$	2	Identificador do nó (endereço de rede)

Tabela 1: Formato de uma quádrupla e seu consumo de memória.

caso[1]. Em termos de processamento, o estabelecimento de rotas é a operação mais custosa do PROC, e demanda $O(v)$ operações de memória, onde v é o número de vizinhos a um salto do nó.

6. Avaliação de Desempenho do Protocolo

Avaliamos o desempenho do protocolo utilizando o simulador NS-2 [19]. Simulamos uma rede homogênea, composta por nós sensores com configuração próxima aos nós sensores da família Mica 2, rodando o sistema operacional TinyOS [8]. 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 [20]. Nessa rede, cada sensor envia mensagens de dados de 36 bytes a cada 70s. Estas mensagens são enviadas para o PA, que disponibiliza os dados através de um enlace via satélite. Neste artigo simulamos apenas a interação dos sensores com o PA.

6.1. Caracterização da Simulação

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. O PA está sempre localizado em um dos vértices da área simulada, maximizando a largura da rede. A densidade dos nós é mantida constante em 23 vizinhos por nó, em média. 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.

Os nós simulados foram ajustados para se assemelharem aos nós da arquitetura Mica2. Ajustamos a largura de banda para 12Kbps, banda máxima da arquitetura. Todos os pacotes de dados enviados possuem tamanho de 36 bytes, enquanto os pacotes de controle possuem tamanhos variados, calculados de acordo com os dados que estes carregam. 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[21]. 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. O raio de transmissão foi fixado em 15m e o consumo de energia do rádio nos estados de transmissão, envio e *idle* foi retirado de [22]. 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 [5], 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 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-

[1]Não contabilizado o consumo do PA pois este não possui restrição de energia.

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 [5], o tempo de propagação por salto para os nós Mica2 é da ordem de centenas de milisegundos. Nessas condições, o tempo de construção das rotas é muito alto. Para amenizar esse problema, utilizamos a priorização de pacotes de roteamento descrito em [17] e atualmente implementada no TinyOS. A priorização de pacotes utiliza duas filas de 16 pacotes cada. 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.

Para comparação com o PROC, dois outros protocolos foram simulados, TOSB e EAD. O TOSB é uma versão simplificada do protocolo TinyOS Beaconing, descrito na Seção 2. 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 6.2). Note que o desempenho do PROC pode variar de acordo com a aplicação, pois cada aplicação pode utilizar regras que influênciarão no desempenho do protocolo. Utilizamos no PROC as três regras propostas na Seção 4.4 como uma forma de avaliar o benefício do uso de regras no roteamento. Essas regras são genéricas e podem ser usadas em qualquer aplicação.

Devido à alta complexidade de processamento e ao número e tamanho das mensagens enviadas pelo Span (às quais devem ser acrescidas as mensagens do roteamento), 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%.

6.2. Falha de Nós

Inicialmente realizamos simulações para determinar o intervalo de recriação de rotas ideal para os protocolos avaliados em cenários onde ocorrem falhas de nós. A rede considerada é formada por 150 nós inseridos aleatoriamente em uma área quadrada, com 70m de lado. A rede funciona sem falhas durante 1500s, para que os nós cheguem ao seu estado estacionário (verificado empiricamente). Neste momento ocorre uma falha de 20 nós durante 120s, e a simulação continua por mais 1500s, permitindo que o protocolo de roteamento se recupere da falha.

Como era de se esperar, os protocolos tendem a consumir mais energia com atualizações de rotas mais frequentes (Figura 2). A taxa de entrega, entretanto, diminui quando o intervalo de recriação de rotas aumenta (Figura 3), devido à falha de nós. No PROC a redução é menor, pois o mecanismo de tolerância a falhas permite uma correção rápida das rotas. Quanto à latência média, verificamos para todos os protocolos uma diminuição nesta métrica com o aumento do tempo de recriação de rotas, que ocorre principalmente devido à menor carga imposta à rede. Para intervalos de 120s, os protocolos EAD e TOSB possuem uma bom compromisso entre taxa de entrega média, e consumo de energia. O PROC, por outro lado, possui melhor desempenho com intervalos de ciclo de 180s. Nos cenários seguintes utilizamos estes valores para o intervalo de recriação de rotas.

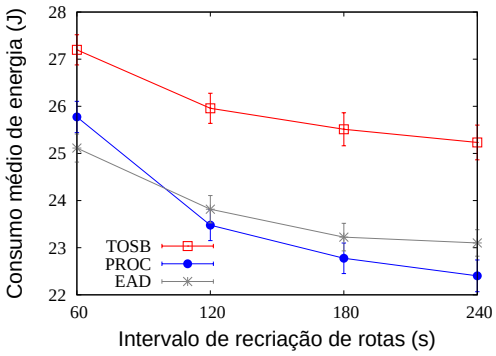

Figura 2: Consumo médio de energia.

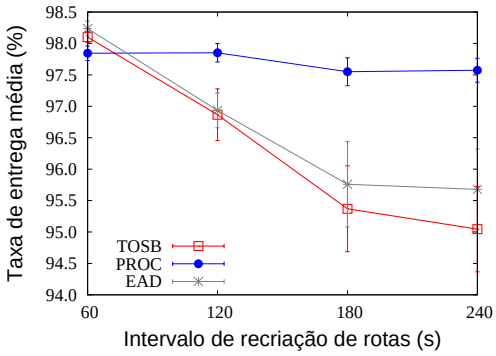

Figura 3: Taxa de entrega média.

6.3. Escalabilidade

Em seguida avaliamos a escalabilidade do protocolo, variando o número de nós da rede. A Figura 4 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.

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 5. 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.

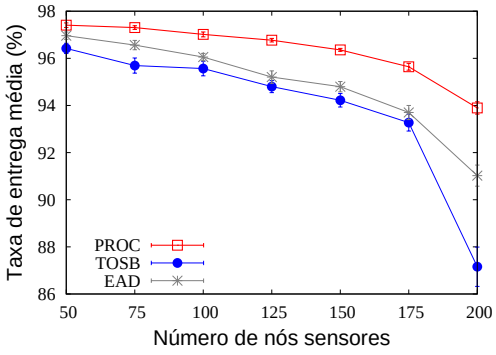

Figura 4: Taxa de entrega média.

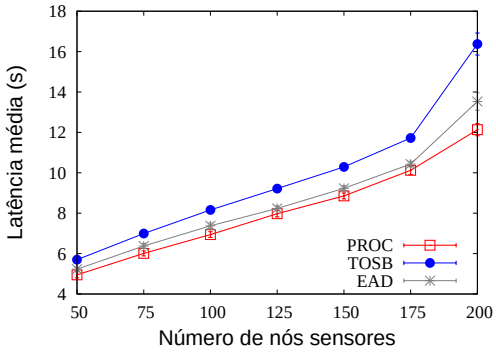

Figura 5: Latência média.

As Figuras 6 e 7 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 6 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 7 comprova este fato, visto que o número de

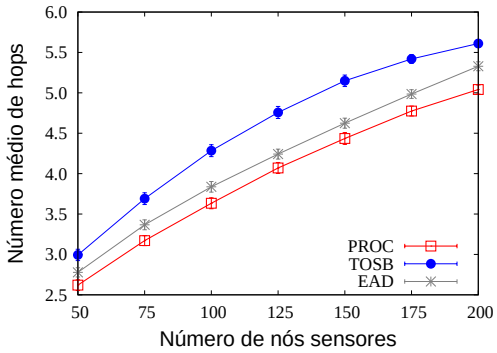

Figura 6: Saltos médios.

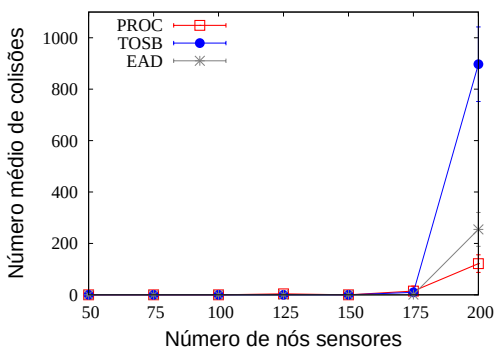

Figura 7: Média de colisões.

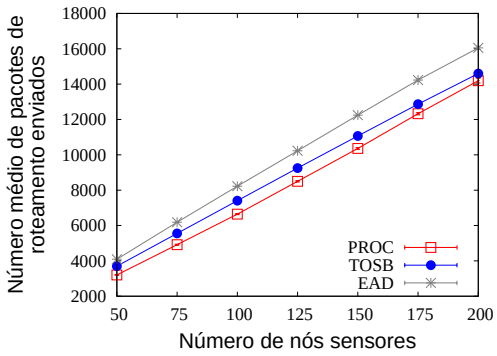

Figura 8: Média de pacotes de roteamento enviados.

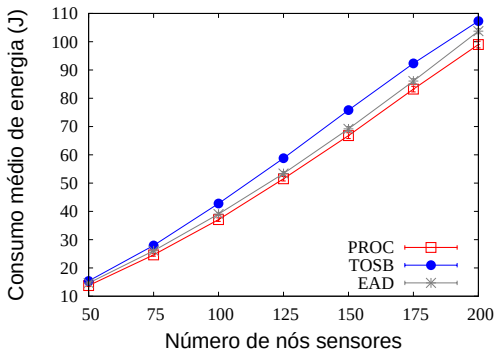

Figura 9: Consumo médio de energia.

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.

Um fator importante a ser considerado no projeto de protocolos para RSSF é o seu custo em energia e mensagens enviadas. As Figuras 8 e 9 mostram o número médio de mensagens de roteamento enviadas 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.

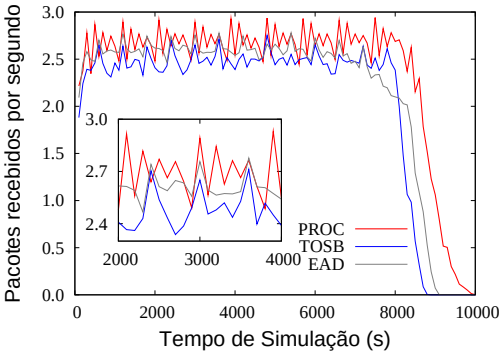

Figura 10: Vazão média.

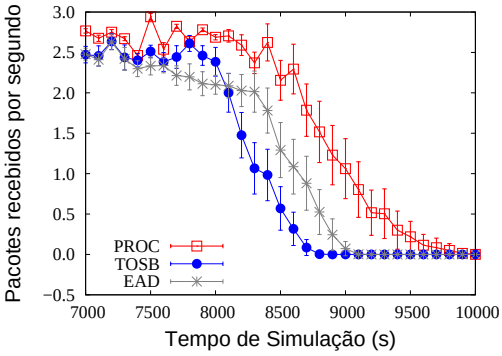

Figura 11: Vazão média (ampliado).

6.4. 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 10 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 11 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 4.4, que permitiram que o consumo de energia médio por nó se tornasse mais uniforme. Estes resultados não são mostrados neste artigo.

7. Conclusões

Este artigo apresentou um protocolo de roteamento pró-ativo, denominado PROC, para RSSF estáticas e de disseminação contínua de dados. Em RSSF a troca de informação e a mistura de funções entre camadas diferentes tem se mostrado uma necessidade. Pelo nosso conhecimento, o PROC é o primeiro protocolo de roteamento para redes de disseminação contínua de dados que utiliza a interação com a aplicação para otimizar o roteamento. Esta interação ocorre através de regras, definidas pela aplicação, a fim de permitir a escolha das melhores rotas. O protocolo consiste de duas fases, criação de um *backbone* de roteamento e estabelecimento de rotas, que foram definidas por algoritmos. O PROC pode ser estendido para mais de um PA, repetindo o processo de recriação de rotas em intervalos sincronizados.

Simulações compararam o PROC a protocolos existentes na literatura, o EAD e o TinyOS Beaconing, sendo o último amplamente utilizado em redes de sensores em operação. Apesar do aumento do número de pacotes de roteamento enviados, foi verificado que o PROC aumenta o tempo de vida da rede entre 7% e 12% e possibilita vazão superior em comparação aos protocolos avaliados. Além disto, o PROC apresentou uma degradação de desempenho mais suave com o incremento de nós na rede.

Referências

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications*, 40(8):102–114, 2002.

[2] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *ACM/IEEE Transactions on Networking*, 11(1):2–16, Feb. 2002.

[3] B. 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, 2000.

[4] T., J. A. Stankovic, C. Lu, and T. 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.

[5] 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, 2004.

[6] Linnyer Beatrys Ruiz, Antonio A. F. Loureiro, and Jose Marcos Nogueira. Functional and information models for the MANNA architecture. In *GRES03 - Colloque Francophone sur la Gestion de Reseaux et de Services*, pages 455–470, February 2003.

[7] A. Boukerche, X. Cheng, and J. 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, 2003.

[8] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. TinyOS: An operating system for wireless sensor networks. In W. Weber, J. Rabaey, and E. Aarts, editors, *Ambient Intelligence*. Springer-Verlag, 2004.

[9] 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 *22° Simpósio Brasileiro de Redes de Computadores*, pages 43–56, Gramado, Brasil, May 2004.

[10] 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 *22° Simpósio Brasileiro de Redes de Computadores*, pages 29–42, Gramado, Brasil, May 2004.

[11] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5):481–494, 2002.

[12] C. Karlof, Y. Li, and J. Polastre. Arrive: Algorithm for robust routing in volatile environments. Technical Report UCB//CSD-03-1233, University of California, Berkeley, CA, March 2003.

[13] F. Dai and J. Wu. Energy-Efficient Coverage Problems in Wireless Ad Hoc Sensor Networks. *Journal of Computer Communications on Sensor Networks*, 2004.

[14] W. R. Heinzelman and A. Chandrakasan and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, 2000.

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

[16] 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*, Oct 2004.

[17] A. Woo, T. Tong, and D. 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.

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

[19] Ns-2 simulator. http://www.isi.edu/nsnam/ns/, January, 2004.

[20] 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*, pages 307–322, Jan 2004.

[21] Chipcom Corporation. CC1000 low power FSK transceiver. http://www.chipcom.com, 2004.

[22] W. Ye and J. Heidemann and D. Estrin. An Energy-Efficient MAC protocol for Wireless Sensor Networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, USA, June 2002.