Um protocolo de roteamento baseado em clusters desiguais para minimizar hot spots em RSSF

Fernando Henrique Gielow[1], **Aldri L. dos Santos**[1]

[1]NR2 – Departamento de Informática – Universidade Federal do Paraná
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brazil

{fhg07,aldri}@inf.ufpr.br

Abstract. *In order to guarantee the effective arrival of sent packets in a wireless sensor network (WSN), many techniques have been employed. A possible approach is creating clusters, in which cluster-heads are responsible for routing the data to the sink. In the approach that employs equal sized clusters, cluster-heads closer to the sink are constant parts of routes and die prematurely, causing the hot spot. This problem can not be eliminated and measures must be taken to minimize its impacts. This work proposes a routing protocol that mitigates hot spot by using clusters of proportional size to its distance to the sink, and a backbone maintenance algorithm without control packets. Simulations show that the proposed protocol was able to minimize the hot spot effects. When compared to the UCR protocol, which also employs unequal clusters, our protocol presents a considerable reduction in the number of clusters and rotations, that results in an increase of 42.18% in the network lifetime. Furthermore, higher packet delivery rates have been reached due to capability of routes maintenance.*

Resumo. *Para garantir a entrega efetiva de dados em uma rede de sensores sem fio (RSSF), várias técnicas têm sido empregadas. Uma alternativa é o agrupamento dos sensores em clusters, nos quais os líderes são responsáveis por rotear os dados até a estação-base. Na abordagem com clusters de tamanhos iguais, os líderes próximos à estação-base são partes constantes das rotas e morrem prematuramente, causando o hot spot. Tal problema não pode ser eliminado, exigindo maneiras de amenizar seu impacto. Este trabalho propõe um protocolo de roteamento para amenizar o hot spot através de clusters de tamanhos proporcionais à distância à base e da manutenção de backbone sem pacotes de controle. Simulações mostraram que o protocolo proposto consegue minimizar os efeitos do hot spot. Comparado ao UCR, que também utiliza clusters desiguais, ele apresenta uma menor quantidade de clusters e rotações, o que implicou em um aumento de 42.18% no tempo de vida da rede. Além disso, uma taxa de entrega de dados superior foi obtida devido à manutenção das rotas.*

1. Introdução

As redes de sensores sem fio (RSSF) podem ser empregadas de inúmeras maneiras, como no monitoramento ambiental remoto e em sistemas de segurança [Bulusu and Jha 2005]. Para que a comunicação entre os sensores seja realizada com eficácia é necessária a criação de rotas até uma estação-base. Assim, a perda dos dados no roteamento precisa ser minimizada, evitando que trabalho desnecessário seja realizado.

Diversos protocolos de roteamento têm sido propostos baseados em formação de clusters, como EECS [Ye et al. 2005], HEED [Younis and Fahmy 2004], LEACH [Heinzelman et al. 2000], TPC [Choi et al. 2004], VCA [Qin and Zimmermann 2005] e UCR [Chen et al. 2007]. No entanto, dentre estes, apenas o UCR se preocupa com a questão do hot spot, ou seja, áreas sobrecarregadas com maior tráfego de dados. A região mais comum para o aparecimento do hot spot é perto da estação-base, pois os sensores próximos a ela são utilizados com maior frequência como membros de uma rota.

O consumo de energia de cada sensor em uma RSSF é essencialmente diferente. Os sensores distribuídos de maneira homogênea sofrem uma espécie de efeito funil quando a abordagem multi-salto é utilizada para o envio das mensagens: na medida em que se aproximam da estação-base a quantidade de rotas possíveis diminui. O hot spot gera um processo gradual que faz um buraco de energia centrado na estação-base. Assim, com a morte dos sensores dessa região, o último salto na entrega de um dado torna-se maior, exigindo uma maior potência de transmissão, até chegar no ponto que não será possível a entrega dos dados.

Visto que o hot spot não pode ser completamente eliminado, uma técnica utilizada para aumentar o tempo de vida da rede é a criação de clusters com tamanhos que diminuem à medida que os sensores se aproximam da estação-base. Logo, existem mais líderes (cluster-heads) próximos à estação-base, oferecendo mais alternativas para a criação das rotas, como proposto no protocolo UCR [Chen et al. 2007]. Porém, o uso de clusters desiguais apenas ameniza o hot spot; Mesmo podendo estabelecer mais rotas, os clusters perto da estação-base possuem menos membros e, portanto, menos rotações de cluster-heads podem ser realizadas. Além disso, com a rotação dos cluster-heads um backbone previamente formado pode ser quebrado, pois ao eleger o novo cluster-head, ele pode estar mais distante. Assim, os sensores que eram capazes de se comunicar com o antigo cluster-head não conseguem transmitir seus dados ao novo cluster-head.

Este trabalho apresenta um protocolo de roteamento baseado em clusters desiguais para RSSF que minimiza os efeitos do hot spot ao ajustar o backbone de forma dinâmica, chamado RRUCR. Na medida que os sensores enviam seus dados à estação-base o backbone é atualizado, não exigindo uma fase ou operação específica do protocolo, e isso evita gastos desnecessários de energia. O protocolo foi avaliado via simulação e seu desempenho comparado com o do UCR. Os resultados mostram que o hot spot foi efetivamente amenizado, e a manutenção dinâmica das rotas sem pacotes de controle possibilitou uma maior taxa de entrega dos dados.

O artigo está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve o funcionamento do protocolo UCR. A Seção 4 detalha o funcionamento do RRUCR. A Seção 5 mostra a avaliação do desempenho do protocolo proposto. Finalmente, a Seção 6 apresenta as conclusões do trabalho.

2. Trabalhos relacionados

Diversos protocolos de roteamento têm sido propostos baseados em formação de clusters. O LEACH [Heinzelman et al. 2000], utilizado para aplicações que coletam dados periodicamente, realiza a rotação de líderes randômica dentro dos clusters para balancear o consumo de energia interno. Na fase de transmissão de dados, os líderes os agregam e enviam diretamente para a base, tendo um grande consumo de energia. Existe uma

variação do LEACH [Heinzelman et al. 2005] na qual os sensores que possuem mais energia têm maior preferência para serem escolhidos como líderes. Esta medida auxilia no balanceamento inicial da energia, mas não garante que a rede tenha um tempo de vida muito superior.

O uso da potência de transmissão necessária para fazer os sensores se comunicarem é levado em consideração pelo HEED [Younis and Fahmy 2004], e a probabilidade inicial de um sensor se candidatar a líder depende de sua energia residual. Os líderes são eleitos também com base no custo de energia para comunicação dentro do cluster formado. Uma melhoria do HEED foi proposta em [Qin and Zimmermann 2005], aonde os sensores votam em seus líderes para selecionar os sensores mais aptos, levando em conta diversos fatores.

[Ye et al. 2005] propõem o protocolo EECS, que introduz uma abordagem competitiva sem iterações de troca de mensagens. Ele estende os protocolos LEACH e HEED ao escolher como líderes os sensores com as maiores energias residuais, conseguindo uma boa distribuição inicial de energia na rede. Clusters de tamanhos diferentes são criados, estando os maiores clusters perto da estação-base e diminuindo ao se distanciar, para preservar energia e enviar mensagens à longa distância. No entanto, como os líderes se comunicam diretamente com a base, é necessário um consumo elevado de energia e não se pode determinar a área na qual o hot spot surge por causa da ausência de um backbone.

Protocolos que levam em consideração o problema do hot spot têm sido propostos. O protocolo HUNT [Sesay et al. 2006] define parâmetros como ocupação do buffer, perda de pacotes, e contenção na camada de enlace, que devem ser monitorados e combinados a fim de detectar um hot spot. Porém, ele é focado para aplicações multimídia. O protocolo [Liu et al. 2008] se baseia no problema de otimização por enxame de partículas. Ao redefinir as regras de vôo das partículas é obtido um problema de otimização para roteamento no qual se busca um gradual balanceamento da energia na RSSF, amenizando o hot spot.

A otimização do balanceamento de energia de uma RSSF foi estudada em [Shu et al. 2005]. Eles demonstraram que a rede como um todo deve ser avaliada, e não apenas o interior dos clusters, visto que o hot spot surge quando um sensor é excessivamente selecionado como integrante de rotas até a estação-base. [Perillo et al. 2005] concluíram que, embora o tempo de vida da rede possa ser prolongado com algoritmos que tenham controle da potência de transmissão, o hot spot em si não é solucionado apenas com tal abordagem. Outra maneira para tentar minimizar este problema é o uso de clusters de tamanhos desiguais, como utilizado em [Chen et al. 2007].

3. UCR: Roteamento baseado em clusters desiguais para RSSF

Uma maneira de amenizar o problema do hot spot em RSSF organizadas em clusters é a criação dos clusters com tamanhos desiguais. O UCR [Chen et al. 2007] é um protocolo de roteamento baseado em formação de clusters desiguais em RSSF. Diferente do proposto em EECS [Ye et al. 2005], que diminui o tamanho dos clusters à medida que os sensores se distanciam da estação-base, no UCR o tamanho dos clusters diminui ao se aproximar da estação-base, ilustrado à direita na Figura 1.

O funcionamento do UCR é divido em fases distintas. Inicialmente a distância de cada sensor até a estação-base é calculada através do RSSI (*Received Signal Strength*

Indicator), na Figura 2(a), e através disso o raio de atuação de cada sensor é definido. Baseando-se em uma probabilidade pré-estipulada, os sensores candidatos a líder são aleatóriamente escolhidos. Eles então enviam uma mensagem de anúncio *COMPETE_HEAD(1)* contendo seu identificador e o valor de sua energia residual a todos os sensores dentro de seu raio de atuação, Figura 2(b). Ao receberem essa mensagens, os sensores vizinhos adicionam este sensor à uma lista de vizinhos candidatos.

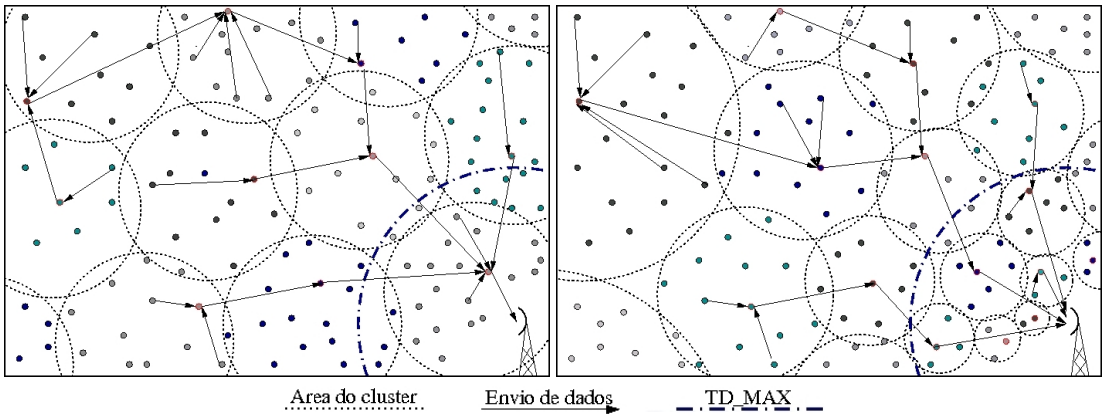
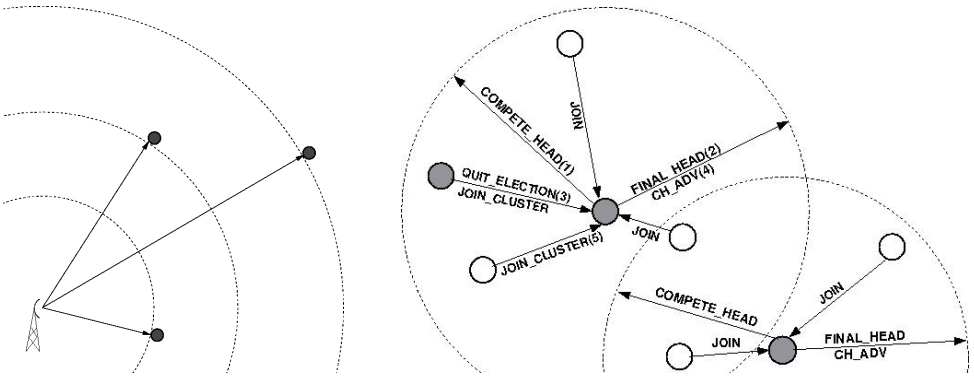


Figura 1. Organização da rede em clusters de tamanhos iguais/desiguais

Após terminada a fase de anúncio, cada sensor candidato verifica se sua energia residual é maior do que a de todos em sua lista, e caso seja, ele é definido como um líder definitivo. Em seguida, o líder envia uma mensagem *FINAL_HEAD(2)* aos demais sensores vizinhos, informando sobre sua posição de líder. Os sensores que receberem esta mensagem sairão da eleição enviando uma mensagem *QUIT_ELECTION(3)*, que implica na remoção deste sensor da lista de vizinhos concorrentes dos demais líderes. Os candidatos que continuarem na fase de eleição irão verificar novamente se suas energias são maiores do que as dos sensores vizinhos ainda dentro da lista de concorrentes, podendo também se tornarem líderes definitivos. O processo é repetido dentro de um tempo destinado à definição dos cluster-heads.



(a) Distância à base - RSSI (b) Raios de atuação & formação de clusters

Figura 2. Operações da fase inicial

Ao final do processo de eleição, os líderes finais enviarão uma mensagem de anúncio *CH_ADV(4)*, e os sensores comuns escolhem (*JOIN_CLUSTER(5)*) o sensor líder

dentro de seu alcance com o qual a comunicação de melhor qualidade é estabelecida (RSSI). Após a criação dos clusters, estes estão sujeitos à rotação de líderes, medida necessária para balancear o consumo de energia interno dos clusters. A comunicação dentro de um cluster é similar ao proposto no protocolo LEACH [Heinzelman et al. 2000].

A comunicação entre os clusters ocorre por meio dos líderes, que transmitem mensagens uns aos outros. A escolha de um cluster-head para transmitir os dados é feita com base na qualidade de sinal entre ambos e na distância do cluster-head escolhido até a estação-base. Caso o líder se encontre dentro de uma área denominada TD_MAX centrada na base (Figura 1), o cluster-head retransmite suas mensagens diretamente para a base[1]. O UCR utiliza para a entrega de dados até a estação base o backbone de menor custo em questão de energia.

Entretanto, o UCR não se preocupa com a manutenção do backbone. Assim, com o decorrer do tempo, os clusters estão sujeitos à rotação de cluster-heads, e isto pode levar a quebra do enlace de comunicação entre dois cluster-heads. Dois clusters adjacentes, por exemplo, podem rotacionar seus cluster-heads escolhendo sensores mais distantes em relação ao outro cluster, impossibilitando a entrega efetiva de dados enviados devido à distância superior ao alcance.

4. Protocolo de roteamento baseado em clusters desiguais reativo à rotações

O protocolo proposto, denominado RRUCR (*a Rotation Reactive Unequal cluster-based Routing Protocol*) se preocupa em minimizar a ocorrência do hot spot através da utilização de clusters de tamanhos desiguais, da rotação de cluster-heads, e da integração dos fluxos de dados na manutenção do backbone sem a necessidade de mensagens de controle. O RRUCR é dividido em 5 operações: definição do raio de atuação dos sensores, formação e divulgação dos clusters, formação do backbone inicial, rotação dos cluster-heads, e coleta de dados, que serve também para a manutenção do backbone.

As operações de definição de raio e da formação dos clusters acontecem apenas uma vez no início do tempo de vida da rede. Portanto, o número de clusters não aumentará com o decorrer do tempo, diminuindo com a morte de sensores. Logo, tanto a distribuição de energia inter-cluster quanto intra-cluster deve ser considerada. Após a formação dos clusters, um backbone inicial é estabelecido. As demais operações ocorrem diversas vezes, garantindo melhor distribuição de energia e maior taxa de entrega de pacotes. As subseções adiante descrevem as operações do RRUCR e os algoritmos utilizados. É apresentado um pseudo-código em cada subseção para facilitar o entendimento.

4.1. Definição do raio de atuação

A estação-base inicialmente envia um mensagem *INCR_POT* (linha 9 do Algoritmo 1; por simplicidade, as linhas dos algoritmos serão referidas como *l* ao longo do texto) em broadcast para cada uma das 21 potências de transmissão[2] (similar à Figura 2(a)), informando a potência utilizada no envio. Ao receber esta mensagem pela primeira vez, os sensores guardam a potência utilizada em uma variável *RBase* e retornam uma resposta *POT_ANS* (*l*.10-13). Através da resposta dos sensores a base saberá quais foram a menor *(RFMin)*

[1] esta área é definida em [Ye et al. 2006] como a região do hot spot.

[2] são as 21 potências mapeadas no datasheet do rádio CC1000, utilizado por sensores do tipo MICA2.

e maior *(RFMax)* potências utilizadas para que eles fossem alcançados (*l*.14-21). Em seguida a estação-base envia em broadcast uma mensagem *SETUP_CONFIG* na potência máxima contendo o valor de *RFMin* e *RFMax* (*l*.23).

Visto que os sensores podem não ser alcançados por esta mensagem[3], os sensores mais distantes(aqueles cujo $RFMax = RFMin$) retransmitem a mensagem *SETUP_CONFIG* e todos que ainda não haviam sido alcançados por ela também a retransmitem (*l*.24-29). Esta mensagem tem um contador *cont* que informa quantas vezes ela foi reenviada até atingir o sensor atual.

Algoritmo 1 - Definição dos raios de atuação & Inicialização

```
 1: procedure Initialize
 2:     for ∀si do                                                    ▷ Sensores da rede
 3:         si.RFMax ← 0;
 4:         si.RFMin ← ∞;
 5:         si.makeNewRoute ← false;          ▷ Variável que diz quando devo procurar uma nova rota
 6:         si.routeCost ← 0;                                         ▷ Custo da rota até a base
 7:     end for
 8:     for k in {1..21} do
 9:         BASE_STATION broadcast INC_POT(BASE_STATION.ID,i) na potência k   ▷ Potências baseadas no rádio CC1000
10:         on si receber INC_POT(sj.ID,k) de sj do
11:             si.RBase ← i;
12:             si envia POT_ANS(si.RBase) para sj;                   ▷ Responde apenas uma vez
13:         end on
14:         on si receber POT_ANS(sj.RBase) de sj do                  ▷ Base atualiza os valores máximos e mínimos
15:             if sj.RBase > si.RFMax then
16:                 si.RFMax ← sj.RBase;
17:             end if
18:             if sj.RBase < si.RFMin then
19:                 si.RFMin ← sj.RBase;
20:             end if
21:         end on
22:     end for
23:     BASE_STATION broadcast SETUP_CONFIG(BASE_STATION.{RFMin,RFMax},0);   ▷ Tudo "ok"
24:     on si receber SETUP_CONFIG(sj.{RFMin,RFMax},count) de sj do   ▷ Atualiza para calcular o raio
25:         si.RFMin ← sj.RFMin;
26:         si.RFMax ← sj.RFMax;
27:         si define seu raio de atuação   ▷ Fórmula exata apresentada adiante ainda na subseção 4.1
28:         si broadcast SETUP_CONFIG(si.{RFMin,RFMax},count+1);      ▷ Repassa apenas uma vez
29:     end on
30: end procedure
```

Dois limites são pré-estipulados para a potência de transmissão: *pot_limit*, que é a potência máxima que pode ser utilizada pelos sensores que foram atingidos na primeira onda de mensagens *SETUP_CONFIG*, e *pot_max_global*, que é a potência máxima que pode ser utilizada pelos demais sensores. Estes limites existem pois com a rotação de líderes é possível que alguns sensores antes alcançáveis se distanciem além da cobertura da potência utilizada para a comunicação inter-clusters. Desta forma, os clusters terão diâmetro inferior à cobertura da potência utilizada para a comunicação inter-clusters, dificultando a quebra de enlaces.

Os sensores alcançados pela primeira onda da mensagem *SETUP_CONFIG* têm seus Raios (*l*.27), representados por potências de transmissão, calculados da seguinte maneira:

$$Raio = (int)((1 - \frac{(RFMax - RBase)}{(RFMax - RFMin)}) * pot\_limit)$$

[3]a maior potência cobre aproximadamente 266 metros.

E o dos demais sensores:

$Raio = (pot\_limit + cont)$, se $pot\_limit + cont \leq pot\_max\_global$,

Senão:

$Raio = pot\_max\_global$

4.2. Formação dos clusters

Após definidos os raios de atuação dos sensores começa a operação de formação de clusters (Algoritmo 2). A partir de uma probabilidade pré-estipulada *pBeTHead*, alguns sensores são selecionados e se tornam candidatos a cluster-heads (*l*.2-4). Esses então enviam uma mensagem de competição *(COMPETE_HEAD)* (*l*.5) para todos dentro do seu raio de atuação, contendo sua energia residual *(ER)*. Os sensores que não tenham recebido nenhuma mensagem *COMPETE_HEAD* também se candidatam a cluster-head (*l*.3). Essa medida evita que existam áreas sem nenhum cluster-head.

Algoritmo 2 - Formação de clusters

```
1:  procedure Clustering
2:      T ← RAND(0, 1);
3:      if (T < pBeTHead) or (si não recebeu nenhum COMPETE_HEAD) then
4:          si.beTentativeHead ← 1;                          ▷ Selecionado para entrar na competição
5:          si broadcast COMPETE_HEAD(si.{ID,ER});           ▷ Este broadcast é feito em tempo aleatório
6:      end if
7:      on si receber COMPETE_HEAD de sj do
8:          add sj.{ID,ER} to si.SCH;                        ▷ Lista de vizinhos competidores
9:      end on
10:     while tempo destinado à competição e candidatura não acabou do
11:         if (si.beTentativeHead = 1) and (si.ER > sj.ER; ∀sj ∈ SCH) then   ▷ Maior energia da vizinhança
12:             si broadcast FINAL_HEAD;        ▷ Este broadcast é feito em tempo aleatório, respeitando a duração do while
13:             si.beTentativeHead ← ∞;         ▷ Evita que o valor seja decrescido e este cluster-head perca sua posição
14:         end if
15:         on si receber FINAL_HEAD de sj do          ▷ Para criar ou excluir cluster-heads, balanceando a rede
16:             si.finals ← si.finals + 1;                                  ▷ Contabilização
17:             si.beTentativeHead ← si.beTentativeHead − 1;   ▷ Alguns candidatos podem perder seu posto
18:         end on
19:         if finals = 0 then      ▷ Esta condicional é checada em tempo aleatório, respeitando a duração do while
20:             si broadcast FINAL_HEAD;
21:             si.beTentativeHead ← 7;                  ▷ Valor a ser decrementado ocasionando a perca do posto
22:         end if
23:     end while
24:     if (si.beTentativeHead = 4 and si.finals > 1) or (si.beTentativeHead ≤ 6 and si.finals > 2) then
25:         si.beTentativeHead ← −1;                  ▷ Definitivamente fora da eleição para evitar áreas densas
26:     end if
27:     if si.beTentativeHead ≥ 4 then
28:         si é cluster definitivo
29:         si broadcast CH_ADV;          ▷ Este broadcast é feito em tempo aleatório,respeitando a duração do while
30:     end if
31:     if si.beTentativeHead < 4 then                                  ▷ Sensor comum
32:         si broadcast JOIN_CLUSTER para sj |sj.RSSI > sk.RSSI, ∀sk ∈ SCH;   ▷ Este broadcast é feito em tempo
    aleatório,respeitando a duração do while
33:         si.next ← sj.id;                                          ▷ Salva a id do lider
34:     end if
35: end procedure
```

Os sensores ao receberem a mensagem *COMPETE_HEAD* (*l*.7-9) adicionam a *ER* do sensor que a enviou em uma lista (*l*.7-9), e após o término do tempo destinado à

candidatura eles verificam se sua *ER* é superior à energia dos sensores de sua vizinhança. Caso seja, eles se tornam cluster-heads definitivos e enviam uma mensagem *FINAL_HEAD* para todos os sensores no seu raio de atuação (*l*.11-14).

Todos os sensores contam o número de mensagens *FINAL_HEAD* recebidas (*l*.15-18). Os sensores que não tenham recebido nenhuma destas mensagens em um dado tempo enviarão uma *FINAL_HEAD* (*l*.19-22), também se candidatando. Após o término do tempo de candidatura, os sensores candidatos que não se enquadram nos limites estabelecidos em *l*.24 deixam de ser candidatos (*l*.25). Isso possibilita balancear o número de clusters e ampliar a cobertura na rede, sem criar clusters em excesso.

Os sensores consolidados como cluster-heads definitivos se anunciam com uma mensagem *CH_ADV* (*l*.27-30). Através desta mensagem os sensores comuns selecionam aquele cluster-head com o qual consigam estabeler uma comunicação de maior intensidade de sinal. Em seguida guardam o id deste cluster-head e enviam a ele uma mensagem *JOIN_CLUSTER* contendo sua ER (*l*.31-34). Ao receber estas mensagens, o cluster-head guarda o maior valor de ER para as operações de rotação.

4.3. Formação do backbone inicial

O Algoritmo 3 descreve a criação do backbone inicial. O funcionamento é simples e se preocupa em criar um backbone válido que permita que a estação-base seja alcançada sem muitos saltos. Formados os clusters, a estação-base envia uma mensagem *BEACON_ROUTE* para todos sensores dentro da região definida como *TD_MAX*, definida por uma potência de transmissão (*l*.3). Nesta região estão os cluster-heads que se comunicam diretamente com a estação-base, normalmente onde o hot spot ocorre.

Algoritmo 3 - Criação do backbone inicial

```
1:  procedure Backbone
2:      sj.wave ← ∞, ∀sj
3:      BASE_STATION broadcast BEACON_ROUTE(BASE_STATION.id,0) dentro de TD_MAX   ▷ Wave 0 - Se comunica
    diretamente com a estação-base
4:      on si receber BEACON_ROUTE(sj.ID,wave) de sj do
5:          if (si.wave > wave) or (si.wave = wave and sj.RSSI > next.RSSI) then   ▷ Melhor conexão
6:              si.wave ← wave;
7:              if si.beTentativeHead ≥ 4 then                                    ▷ Sou um cluster-head
8:                  si.next ← sj.{ID, RSSI};
9:              end if
10:         end if
11:         si broadcast BEACON_ROUTE(si.ID, si.wave + 1);  ▷ Repassa a mensagem incrementando o valor de wave
12:     end on
13: end procedure
```

Na mensagem *BEACON_ROUTE* existe um contador que informa quantos saltos se passaram desde o seu primeiro envio, para que cada sensor mantenha informação de há quantos saltos ele está da estação-base (*l*.2). Quando um sensor recebe uma mensagem *BEACON_ROUTE*, a sua quantidade de saltos é atualizada se o valor indicado na mensagem for menor (*l*.5-6). Além disso, se o sensor for um líder, o id do próximo sensor na rota é também atualizado com o id de quem enviou a mensagem (*l*.7-9). Este id também será atualizado se o salto da mensagem recebida for igual ao valor do seu salto, mas a intensidade do sinal for melhor do que a intensidade do sinal da mensagem que anteriormente causou a atualização da quantidade de saltos (*l*.5). A mensagem será então retransmitida incrementando 1 no campo que indica a quantidade de saltos passados (*l*.11).

4.4. Rotações de cluster-heads

As rotações dos cluster-heads ocorrem sempre que a energia de um líder se torne inferior à uma porcentagem da maior energia (ER) do cluster (Algoritmo 4). Quando isso acontecer, o cluster-head envia uma mensagem *ROTATE_CH*, contendo o valor de sua energia atual (*l*.2-4). Ao receberem esta mensagem, os sensores pertencentes ao cluster que possuam maior energia do que a informada enviam para o atual cluster-head suas ER através da mensagem *ENERGY_DATA* (*l*.6-9).

O antigo cluster-head seleciona o sensor que tenha informado maior ER como novo cluster-head e envia uma mensagem *DENOMINATE_CH* (*l*.18) para ele. O novo cluster-head envia então uma mensagem *INFORM_NEW_CH* em broadcast para anunciar sua nova função, também informando qual era o id do antigo cluster-head (*l*.20). Os sensores que receberem esta mensagem e que se comunicavam com o antigo cluster-head, cujo id foi informado na mensagem, passarão a se comunicar com o novo cluster-head. O novo cluster-head continuará utilizando em sua rota o mesmo líder com o qual o antigo se comunicava. Tanto o novo cluster-head quanto aqueles cluster-heads que se comunicavam com o antigo cluster-head atualizarão suas rotas, para evitar quebras de enlaces (*l*.10-12 e *l*.23-28).

Algoritmo 4 - Rotação de cluster heads

```
1:  procedure Rotate
2:      if si.ER < (pRotate * maior.ER) then        ▷ Menor que certa porcentagem da energia do sensor com mais energia
3:          si broadcast ROTATE_CH(si.{ID,ER}
4:      end if
5:      while tempo destinado à informar as energias do
6:          on si receber ROTATE_CH(sj.{ID,ER} de sj do
7:              if (si nao é CH) and (si.ER > sj.ER) and (si.next.id = sj.id) then   ▷ Responde se pertencer ao cluster
8:                  si envia ENERGY_DATA(si.{ID, ER}) para sj;
9:              end if
10:             if (si é CH) and (si.next.id = sj.id) then
11:                 makeNewRoute ← true;                                  ▷ Atualiza a rota assim que possível
12:             end if
13:         end on
14:         on si receber ENERGY_DATA(sj.{ID,ER} de sj do
15:             adiciona sj.{ID, ER} em listaEnergia;
16:         end on
17:     end while
18:     si envia DENOMINATE_CH(si.{ID,next.id} para sj |sj.ER > sk.ER ∀ sk ∈ listaEnergia;
19:     on si receber DENOMINATE_CH(sj.{ID,next.id}) de sj do
20:         si broadcast INFORM_NEW_CH(si.{ID,next.id});
21:         maior.ER ← si.ER;          ▷ Rotacionará de novo quando atingir certa porcentagem de sua energia atual
22:         makeNewRoute ← true;                                          ▷ Atualiza a rota assim que possível
23:         next.id ← sj.next.id;                                  ▷ Atualiza ID do sensor para o próximo salto
24:     end on
25:     on si receber INFORM_NEW_CH(sj.{ID,next.id}) de sj do
26:         if si.next.id = sj.next.id then                                   ▷ Mesmo antigo cluster-head
27:             next.id ← sj.ID;                                              ▷ Atualiza novo cluster-head
28:         end if
29:     end on
30: end procedure
```

4.5. Coleta de dados & Reconstrução de rotas

O Algoritmo 5 descreve a operação de coleta de dados e como ela implica na reconstrução das rotas. Quando um sensor precisa mandar algum dado coletado até a estação-base ele envia uma mensagem *DATA_GATERED* contendo os dados e o id que aponta para o

próximo salto (*l*.2-4). O cluster-head cujo id foi apontado como próximo salto automaticamente reenvia esta mensagem até que a estação-base seja atinginda (*l*.15-17). Junto com a mensagem, o cluster-head envia um valor que identifica sua distância até a estação-base e ajudará na atualização das rotas.

Algoritmo 5 - Dados coletados e manutenção do backbone

```
1:  procedure DataGathered
2:      if sensor si precisa enviar dados à base then
3:          si broadcast DATA_GATHERED(si.{dados_lidos,ID,next.id},∞) ▷ ∞ pois não posso rotacionar com um sensor comum
4:      end if
5:      on si receber DATA_GATHERED(sj.{dados_lidos,ID,next.id},wr) de sj do
6:          if (si.{wave + RBase/100} > wr) and (makeNewRoute) then ▷ Preciso atualizar rota
7:              si.next.id ← sj.ID; ▷ Atualizo rota para a primeira válida encontrada
8:              routeCost ← wr; ▷ Atualizo custo da rota
9:              makeNewRoute ← false; ▷ Já criei nova rota válida
10:         end if
11:         if si.routeCost > wr then ▷ sj é melhor que o antigo sensor usado como parte de rota
12:             si.next.id ← sj.ID; ▷ Atualizo rota para a melhor
13:             routeCost ← wr; ▷ Atualizo custo da rota do melhor encontrado para novas comparações
14:         end if
15:         if (si.id = sj.next.id) and (si.id ≠ BASE_STATION.id) then ▷ Devo retransmitir
16:             si broadcast DATA_GATHERED(si.{dados_lidos, ID, next.id}, wave + RBase/100);
17:         end if
18:     end on
19: end procedure
```

A manutenção do backbone, embora inicialmente não garanta a menor rota, economiza energia e também diminui o overhead por não utilizar pacotes de controle e ser um processo dinâmico. Sempre que um cluster-head receber uma mensagem *DATA_GATHERED* ele poderá atualizar sua rota. Caso a manutenção seja obrigatória em razão da rotação (*l*.6-10), o cluster-head verificará apenas se o cluster-head que enviou a mensagem está entre ele e a base, e caso esteja, o adotará imediatamente como sendo o próximo salto. Caso a manutenção não seja obrigatória (*l*.11-14), ele atualizará a rota apenas se esta nova possibilidade de rota tiver um custo menor do que o atual.

5. Avaliação de desempenho

Os dois protocolos foram implementados no ambiente de simulação NS-2 e simulados operando com o protocolo IEEE 802.11 na camada MAC. Uma rede homogênea foi criada, composta de sensores com características similares aos Mica2 rodando o sistema operacional TinyOS. Os parâmetros do rádio foram ajustados para simular os do CC1000, utilizado na arquitetura Mica2. A aplicação gera uma leitura aleatória de dados, onde cada sensor tem probabilidade de 0.1% de gerar um dado a cada segundo, o qual será transmitido até a estação-base sem nenhuma agregação.

A rede simulada opera por 5000 segundos e consiste de 700 sensores espalhados em uma área quadrada, que mede 1000m em cada lado. A localização de todos os sensores, incluindo a da estação-base (EB), é aleatória em cada simulação; Assim como suas energias iniciais, geradas individualmente com valores entre 0.9 e 1.1 Joules. Foi utilizada em ambos os protocolos uma probabilidade de 35% para que um sensor entre na eleição de cluster-head e uma potência de transmissão inter-cluster que cobre 266m (a potência de maior cobertura). A área denominada TD_MAX, dentro da qual todos os sensores se comunicam diretamente com a EB, tem 149m centrados nela. No UCR foi considerada como 140m a área máxima de atuação de cada sensor. No RRUCR foram

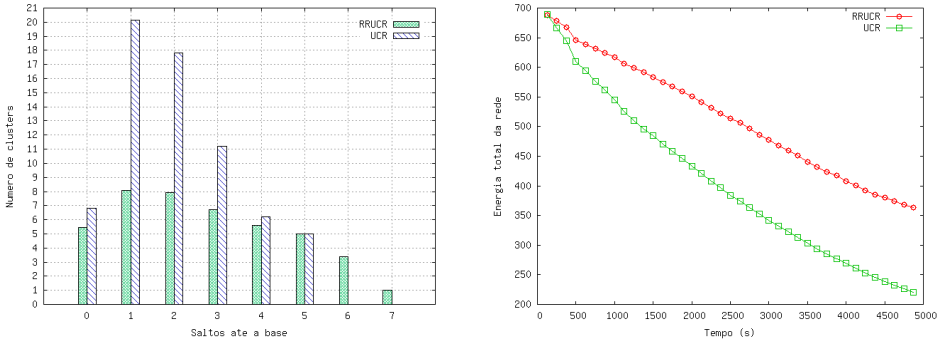
utilizados *pot_limit*=11 (cobertura de 141m), *pot_max_global*=14 (cobertura de 178m), e *pRotate*=65%.

Foram realizados três tipos de simulações para cada protocolo: operação da rede sem falhas, com falhas perto da EB, e com falhas longe da EB (a distância é quantificada em saltos). No caso de falhas perto, são desativados aleatóriamente 8 sensores que levem de 0 a 2 saltos para se comunicar com a EB, e mais 8 sensores que levem de 1 a 5 saltos. No caso da falha ser longe da estação-base são desativados 25 sensores, sendo deles 12 que levem de 2 a 5 saltos para se comunicar com a EB, e mais 13 que levem de 3 a 6 saltos. Em ambas as situações de falha, ela ocorre aos 400s da simulação.

As métricas avaliadas são distância em saltos de cada cluster-head até a EB, energia total da rede, tempo de vida da rede, número de sensores mortos considerando suas distâncias até a EB, taxa de entrega de dados (ela mede sempre a porcentagem de entrega dos últimos 30 dados enviados), e número de rotações. Todos resultados foram obtidos com base em 35 simulações realizadas para cada protocolo e para cada tipo de rede (sem falhas, com falhas perto da EB, e com falhas longe da EB).

5.1. Distribuição dos clusters & Consumo de energia

O número de clusters é um fator importante em uma RSSF: uma quantidade grande demais implica em consumos elevados de energia, uma quantidade pequena resulta em maior overload e maior potência de transmissão necessária para se realizar a comunicação inter-clusters. Como o UCR não realiza manutenção de suas rotas, seus clusters devem ser menores, para que na rotação a probabilidade de se selecionar um novo cluster-head muito distante do atual diminua.



(a) Distribuição dos clusters (b) Energia total da rede

Figura 3. Número de clusters & Consumo de energia

Na Figura 3(a), observa-se que embora o RRUCR tenha clusters que precisem de mais saltos para alcançar a estação-base, ele cria um número significantemente menor de clusters. No RRUCR são criados em média 43 clusters, enquanto o UCR cria em média 67. Essa diferença de 35.82% no número de clusters gera um consumo de energia muito elevado, como mostrado na Figura 3(b). A maior queda de energia ocorre até os 500s (segs) pois este é o tempo aonde está compreendido o processo de formação dos clusters, no qual grande quantidade de mensagens são trocadas. No caso do UCR a queda é ainda maior, devido aos pacotes *QUIT_ELECTION*, que são enviados em grande quantidade, tendo como objetivo evitar áreas sem cobertura na rede.

Após os 500s, a queda de energia do UCR é muito mais rápida do que no RRUCR. Isso acontece pois com um número superior de clusters ocorrerão mais rotações em clusters distintos. Em cada rotação são trocadas diversas mensagens, como visto no Algoritmo 4 (O mesmo algoritmo de rotação foi utilizado para o UCR).

5.2. Tempo de vida da rede

O tempo de vida da rede é definido como o tempo decorrido até que o primeiro sensor morra. Uma medida para prolongar este tempo em RSSF em clusters é a rotação dos cluster-heads, distribuindo assim o consumo interno de energia. Além disso, a criação de mais alternativas de rotas e sua manutenção consegue balancear também a energia da rede como um todo. O protocolo RRUCR conseguiu estender o tempo de vida da rede em 42.18% em redes sem falhas. Em redes com falhas perto da estação-base se teve uma melhoria de 35.08%, e em redes com falhas perto da estação-base 21.80% (Figura 4).

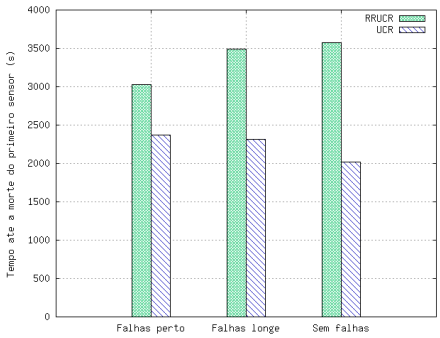
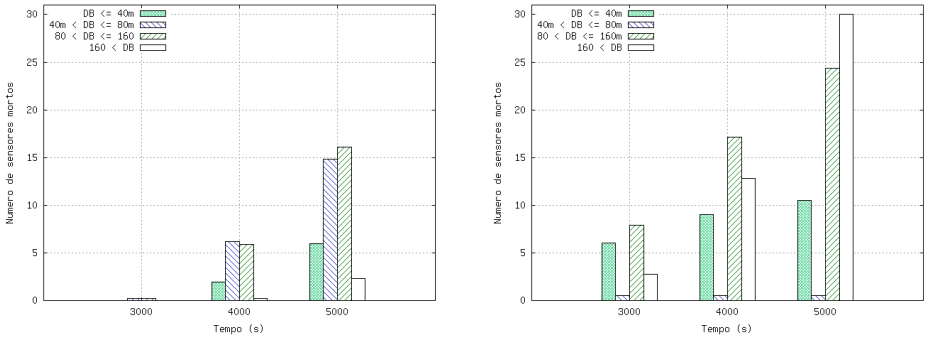


Figura 4. Tempo de vida da rede

Para avaliar o desempenho de ambos os protocolos na minimização do efeito do hot spot, a quantidade de sensores mortos foi avaliada, levando em consideração sua distância até a estação-base (DB). Ambos os protocolos conseguiram minimizar os efeitos do hot spot: ao diminuírem o número de sensores mortos perto da estação-base, mais sensores podem ser utilizados no último salto (Figura 5). Porém, devido ao grande número de clusters, ocorrem muito mais mortes de sensores no UCR.

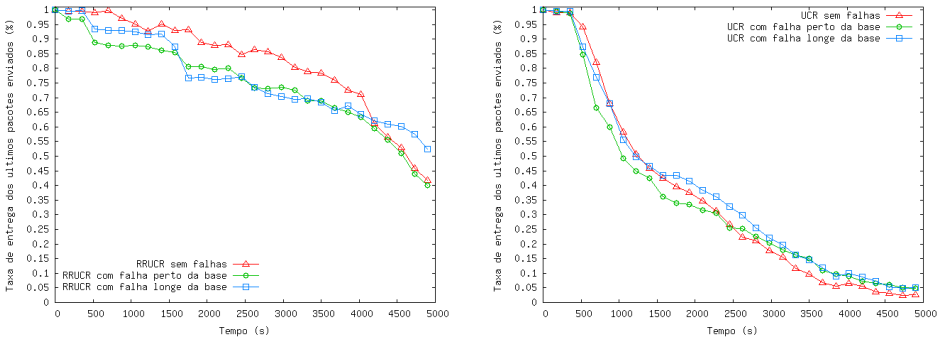


(a) RRUCR (b) UCR

Figura 5. Quantidade de sensores mortos X distância à base X tempo

5.3. Taxa de entrega dos dados

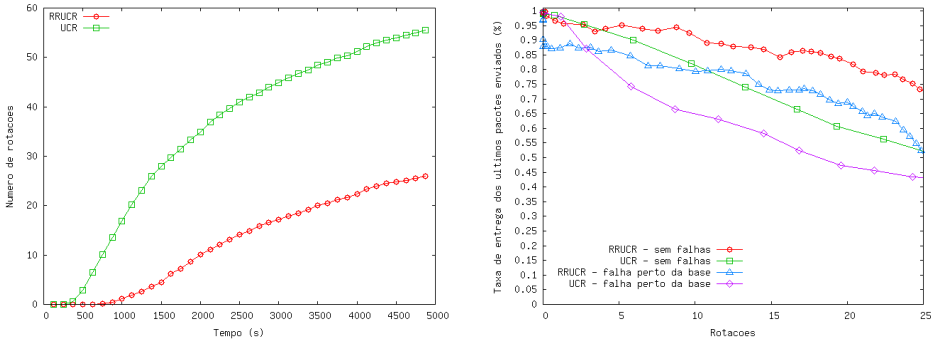
Apenas se preocupar com a distribuição de energia e existência de uma rota inicial não garante uma taxa de entrega de dados satisfatória. A Figura 6 mostra que o UCR teve uma taxa de entrega muito inferior ao RRUCR, se equiparando com ele apenas no começo das simulações, antes que qualquer rotação ou falha tenha acontecido.



(a) RRUCR (b) UCR

Figura 6. Taxa de entrega dos dados em redes com ou sem falhas

Devido a grande quantidade de clusters criados (Figura 3(a)), o UCR precisa realizar mais rotações (Figura 7(a)), e essas rotações geram quebras de enlaces, prejudicando sua taxa de entrega (Figura 7(b)). No UCR quando dois cluster-heads que se comunicavam rotacionam e perdem o alcance um do outro, o enlace entre eles é definitivamente perdido. Isso acontece porque mesmo que novas rotações em ambos os clusters ocorram e os novos cluster-heads estejam em uma distância coberta pela transmissão, eles não saberão que devem se comunicar. A informação da troca de cluster-heads é passada pelos cluster-heads antigos, mas como suas transmissões não alcançavam um ao outro, a atualização da rota não foi realizada.



(a) Número de rotações (b) Taxa de entrega dos dados

Figura 7. Relação entre rotações e entrega de dados

6. Conclusões

Este trabalho propôs um protocolo de roteamento para minimizar os efeitos do hot spot baseado na criação de clusters de tamanho proporcional à sua distância até a base, e na manutenção das rotas sem a necessidade de pacotes de controle. O protocolo é composto

de cinco fases, das quais se destacam as operações de rotação de cluster-heads e coleta de dados, que possibilitam o reparo dinâmico das rotas e uma melhor distribuição de energia.

Simulações mostraram que o RRUCR aumenta o tempo de vida da rede em 42.18%, quando comparado ao UCR. Além disso, o número de clusters criados é 35.82% inferior ao UCR, tendo menores gastos de energia decorrentes da operação de rotação. A tolerância à falhas dos protocolos também foi testada, e embora a operação da manutenção das rotas do RRUCR seja simples, ela mostrou eficácia satisfatória em relação ao UCR. Como um trabalho futuro, operações que verificam a integridade dos enlaces da RSSF e realizem reparos serão implementadas.

Referências

Bulusu, N. and Jha, S. (2005). *Wireless sensor networks - A system perspective*. Artech House.

Chen, G., Li, C., Ye, M., and Wu, J. (2007). An unequal cluster-based routing protocol in wireless sensor networks. *Wireless Networks*.

Choi, W., Shah, P., and Das, S. K. (2004). A framework for energy-saving data gathering using two-phase clustering in wireless sensor networks. *Proceedings of International Conference on Mobile and Ubiquitous Systems*, pages 203–212.

Heinzelman, W., Chandrakasan, A., and Balakrishnan, H. (2000). Energy-efficient communication protocols for wireless microsensor networks. *Proceedings of the 33rd Hawaiian International Conference on Systems Science*.

Heinzelman, W., Chandrakasan, A., and Balakrishnan, H. (2005). An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, pages 660–670.

Liu, A., Ma, M., Chen, Z., and Gui, W. (2008). Energy-hole avoidance routing algorithm for wsn. *Proceedings of the 4th IEE International Conference on Natural Computation*.

Perillo, M., Cheng, Z., and Heinzleman, W. (2005). An analysis of strategies for mitiating the sensor network hot spot problem. *Proceedings of the 2nd International Conference on Mobile and Ubiquitous Systems*.

Qin, M. and Zimmermann, R. (2005). An energy-efficient voting algorithm for sensor networks. *Procedings of 1st ACIS Workshop on Self-Assembling Wireless Networks*.

Sesay, S., Xiang, J., He, J., Yang, Z., and Cheng, W. (2006). Hotspot mitigation with measured node throughput in mobile ad hoc networks. *Proceedings of the 6th International Conference on ITS Telecommunications*.

Shu, T., Krunz, M., and Vrudhula, S. (2005). Power based coverage-time optimization for clustered wireless sensor networks. *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*.

Ye, M., Chan, E., and Chen, G. (2006). On mitiating hot spots for clustering mechanisms in wireless sensor networks. *Proceedings of the 3rd IEE International Conference on Mobile Ad Hoc and Sensor Systems*.

Ye, M., Li, C., Chen, C., and Wu, J. (2005). An energy efficient clustering scheme in wireless sensor networks. *Ad Hoc & Sensor Networks*.

Younis, O. and Fahmy, S. (2004). Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Wireless Communications*, pages 660–669.