Roteamento e Agregação de Dados Usando *Sinks* em Alta Velocidade em Redes de Sensores Sem Fio

Leandro N. Balico[1], Horácio A.B.F. Oliveira[1], Eduardo F. Nakamura[12], Raimundo S. Barreto[1], e Antonio A.F. Loureiro [3]

[1]Instituto de Computação – Universidade Federal do Amazonas

[2]Centro de Analise, Pesquisa e Inovação Tecnológica – FUCAPI
Manaus – AM – Brasil

[3]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
Belo Horizonte – MG – Brasil

{balico,horacio,rbarreto}@icomp.ufam.edu.br

eduardo.nakamura@fucapi.br, loureiro@dcc.ufmg.br

Abstract. *In this work, we study the impact of data aggregation in WSNs with mobile sink nodes that can move at higher speeds. In these cases, propagated queries cannot be answered by using the sink's position when the query was sent, since the sink node will be elsewhere. Therefore, conventional data aggregation schemes may not be appropriate due to this high speed. Thus, we propose and evaluate the performance of three new algorithms for data routing and aggregation in WSNs when the sink is moving at a high speed. These algorithms explore the sink movement to create a routing graph composed by the union paths that intersect the sink's trajectory. At each hop, the sink speed and current network delays are used as metric to guide in-network data aggregation. Our results show clearly significant energy savings and efficient data delivery achieved by the proposed algorithms, as well as provide significant results about their behavior in different scenarios.*

Resumo. *Neste trabalho, avaliamos o impacto da agregação de dados em RSSFs utilizando nós sinks movendo-se em altas velocidades. Nestes casos, consultas propagadas na rede não podem ser respondidas utilizando a posição do nó sink quando a consulta foi enviada, uma vez que o sink estará em outro lugar na rede. Por esse motivo, esquemas convencionais de agregação de dados podem não ser apropriados devido à alta velocidade do nó sink. Portanto, neste trabalho, propomos e avaliamos o desempenho de três novos algoritmos para roteamento e agregação de dados usando sinks em alta velocidade em RSSFs. Estes algoritmos exploram o movimento do sink para criar um grafo de roteamento composto pela união de caminhos de entrega que interseccionam-se com a trajetória do sink. Em cada salto, a velocidade do sink e atrasos vigentes na rede são utilizados como métrica para guiar a agregação de dados durante o roteamento. Os resultados obtidos neste trabalho demonstram uma economia de energia significante bem como uma entrega de dados eficiente obtida pelos algoritmos propostos, assim como fornecem resultados relevantes sobre seu comportamento em diferentes cenários.*

1. Introdução

Redes de Sensores Sem Fio (RSSFs) [Akyildiz et al. 2002, Estrin et al. 2001] são tipicamente compostas por dispositivos computacionais autônomos (nós sensores) depositados em uma área de interesse. Esses sensores, de forma colaborativa, monitoram condições ambientais físicas e químicas em diferentes locais. Dados coletados pelos sensores são normalmente processados e enviados para o nó *sink* de forma distribuída, utilizando comunicação sem fio em múltiplos saltos.

Estudos recentes [Oliveira et al. 2010, Luo and Hubaux 2010, Kim et al. 2003, Ye et al. 2002] demonstraram o uso de *sinks* móveis em RSSFs como forma de prolongar o tempo de vida da rede. Neste contexto, decisões em relação à mobilidade do *sink* são tomadas visando equilibrar o consumo energético na rede [Faheem et al. 2009]. Nesses trabalhos, o nó *sink* tem em geral uma mobilidade baixa e o principal desafio para o roteamento é manter as tabelas de roteamento atualizadas e encaminhar os dados coletados em direção ao nó *sink* (como ilustrado na figura 1).

Cenários recentes de RSSFs, demonstram a necessidade de novos algoritmos para encaminhar os dados para *sinks* movendo-se em altas velocidades [Oliveira et al. 2010]. Nestes cenários, o nó *sink* pode ser um Veículo Aéreo Não Tripulado (VANT) ou até mesmo um avião. Entretanto, devido à alta velocidade do nó *sink*, os pacotes de resposta não podem ser enviados para a posição do *sink* quando ele enviou o pacote de consulta, uma vez que ele estará em outro lugar na rede (como mostra a figura 1).

Neste trabalho, propomos um novo algoritmo de Roteamento e Agregação de Dados para o envio de dados em direção a um *sink* em alta velocidade em RSSFs: HISPEAR (*HIgh SPEed Aggregation and Routing*). Com base neste algoritmo, propomos três diferentes variantes: HISPEAR *Shortest*, HISPEAR *Intercept* e HISPEAR *Hybrid*. A ideia principal dos algoritmos HISPEAR é calcular posições futuras do nó *sink* utilizando sua trajetória e efetuar roteamento e agregação de dados em direção a essas posições calculadas. O desempenho dos algoritmos propostos foi avaliado utilizando o simulador NS-2. Apresentamos um extenso conjunto de experimentos que demonstram a eficiência e significante economia de energia obtida pelos algoritmos propostos em diferentes cenários.

O restante deste trabalho é organizado como segue. A seção 2 descreve os trabalhos relacionados no contexto da mobilidade do *sink*. A seção 3 apresenta o algoritmo HISPEAR e suas variantes, enquanto que a seção 4 descreve sua avaliação de desempenho. Finalmente, a seção 5 apresenta as conclusões e sugestões de trabalhos futuros.

2. Trabalhos Relacionados

A primeira abordagem ao problema de entregar dados de nós sensores para um *sink* móvel em RSSF foi o algoritmo TTDD (*The Two-Tier Data Dissemination*) [Ye et al. 2002]. Nesse algoritmo, ao detectar um novo evento, o nó sensor constrói de maneira proativa uma estrutura de grade virtual que permite aos *sinks* móveis receberem dados através de consultas disseminadas de maneira controlada às células locais em que se encontram. Kim et al. [Kim et al. 2003] propuseram o algoritmo SEAD (*Scalable Energy-efficient Asynchronous Dissemination*) que utiliza a distância e a quantidade de tráfego entre os sensores para criar uma aproximação da Árvore de Steiner mínima para encaminhar dados e interconectar a redes aos *sinks* móveis sem disseminar a posição desses nós pela árvore.

Shim e Park [Shim and Park 2006] propuseram um algoritmo baseado em localizadores para *sinks* móveis. Esses nós localizadores são distribuídos uniformemente pelo campo de sensores e rastreiam a posição atual do *sink* móvel. Se um nó sensor tentar enviar dados para uma posição desatualizada do *sink*, esse nó pode obter uma posição mais atualizada utilizando os localizadores. Finalmente, Oliveira et al. [Oliveira et al. 2010] propuseram o algoritmo WHISPER (*Wireless High Speed Routing*) para cenários de RSSFs onde o *sink* movimenta-se em alta velocidade. Assim como o algoritmo HISPEAR proposto neste trabalho, o algoritmo WHISPER também encaminha os pacotes para posições futuras do nó *sink*. Entretanto, assim como todas as soluções citadas anteriormente, esses estudos não consideram a agregação de dados a fim de aperfeiçoar a eficiência energética de RSSFs com sinks movimentando-se em altas velocidades, a qual é a motivação principal deste trabalho.

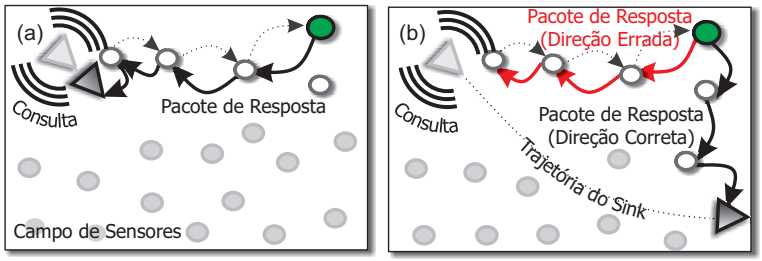

Figura 1. Consulta enviada por: (a) um *sink* em baixa velocidade; e (b) um *sink* em alta velocidade.

3. *High Speed Aggregation and Routing* – (HISPEAR) em RSSFs

Nesta seção, propomos um novo algoritmo para encaminhar dados para *sinks* em alta velocidade efetuando agregação de dados em cada salto da rede: o algoritmo HISPEAR (*High Speed Aggregation and Routing*). O algoritmo HISPEAR explora o movimento do *sink* para criar um grafo de roteamento composto pela união de caminhos de entrega que interseccionam-se com a trajetória do *sink*. Em cada salto, a velocidade de *sink* e atrasos vigentes na rede são utilizados como métrica para guiar a agregação de dados durante o roteamento. Este algoritmo calcula o melhor tempo disponível para agregar dados em cada salto da rede, a fim de entregar dados menos redundantes para uma posição mais atualizada, ou mesmo uma posição futura, do nó *sink*. Cada nó encaminha pacotes agregados para o nó vizinho mais próximo da nova posição calculada do *sink*. Dessa forma, é importante que cada nó conheça a sua própria posição, a posição dos seus vizinhos, a Trajetória e Deslocamento do *sink* bem como a Velocidade de Propagação dos Pacotes [Oliveira et al. 2010].

3.1. Definições Preliminares

Nesta seção, apresentaremos formalmente os conceitos utilizados neste trabalho.

Definição 1 (Redes de Sensores Sem Fio): nós definimos uma RSSF como um grafo Euclidiano $G = (V, E, r)$, onde $|V| = n$ é o número de nós e r é o raio de comunicação; $V = \{v_0, v_1, v_2, \ldots, v_n\}$, onde v_0 é o *sink* móvel e $\{v_1, v_2, \ldots, v_n\}$ o conjunto de nós sensores; $\langle i, j \rangle \in E$ se e somente se v_i alcança v_j, em outras palavras, v_i está dentro do raio de comunicação r de um nó v_j; e $\forall v_i \in V$, $(Xp_i, Yp_i, Zp_i) \in \mathbb{R}^3$ é a posição real do nó v_i; enquanto $(Xc_i, Yc_i, Zc_i) \in \mathbb{R}^3$ é a posição estimada do nó v_i (i.e., utilizando um sistema de localização).

Definição 2 (Alta Velocidade do Sink Móvel): neste trabalho, consideramos o nó *sink* como estando em alta velocidade caso não seja possível que o pacote de resposta de uma consulta enviada pelo *sink* alcance-o usando o caminho reverso em que a consulta percorreu na rede, devido a uma grande distância percorrida pelo *sink* desde a posição onde a consulta foi originada (ver figura 1).

Definição 3 (Algoritmo de Consulta de Dados): diferentemente de tradicionais esquemas de roteamento fim-a-fim que tentam manter rotas entre nós de origem e destino, este trabalho está focado em algoritmos de roteamento para consulta de dados [Boukerche and Nikoletseas 2004], objetivando dar suporte à agregação de dados *in-network*. Neste algoritmo, o nó *sink* envia uma consulta para a rede de sensores, como se esta fosse um sistema de banco de dados distribuído (e.g., *Sensor Databases* [Hong and Madden 2004]). Esta consulta é disseminada na rede por *flooding* e, nós sensores com dados relacionados montam um pacote de resposta que é enviado para o nó *sink*. Em cada salto, os nós sensores podem olhar o conteúdo dos pacotes e efetuar agregação de dados em múltiplos pacotes de entrada para encaminhar um único pacote agregado, de acordo com a função de agregação especificada na consulta. Neste trabalho, consideramos funções simples de agregação de dados (tais como *máximo, mínimo, média*). Trabalhos relacionados deste tipo incluem [Krishnamachari et al. 2002] and [Intanagonwiwat et al. 2003].

Definição 4 (Trajetória do Sink – T_s): esta trajetória pode ser uma linha, uma curva ou qualquer outra trajetória que possa ser expressa matematicamente. Por uma questão de simplicidade, e sem qualquer perda de generalidade, consideramos que, enquanto no interior do campo de sensores, o *sink* manterá uma trajetória em linha reta, que é comum para objetos em alta velocidade. Portanto, dado um ponto inicial (e.g., a posição do *sink* quando a consulta foi enviada) e uma direção, esta linha pode ser facilmente calculada como $y = \tan(\theta)(x - x_0) + y_0$, onde θ é o ângulo em relação ao eixo x e (x_0, y_0) é a posição inicial. Esta trajetória do *sink* é enviada junto com o pacote de consulta.

Definição 5 (Tempo de Espera Para Agregação de Dados – A_t): um nó sensor, ao receber um pacote de consulta, define uma linha perpendicular entre a sua posição (Xc_i, Yc_i) e a trajetória do *sink* pela equação $y = \frac{-1}{\tan(\theta)}(x - Xc_i) + Yc_i$. Dois pontos são calculados nesta linha para servir como referência para duas métricas. O primeiro ponto A_S, está localizado na intersecção da linha e a trajetória do *sink*, como mostra a figura 2(a). Para esse ponto, definimos a diferença entre o tempo para o nó *sink* atingir o ponto A_S e o tempo para um pacote do nó atual chegar a esse ponto como primeira métrica de tempo de espera para agregação de dados (A_t). Portanto, pela equação $A_t = dist(T_k, A_S) * v - \frac{dist((Xc_i, Yc_i), A_S)}{P_k}$ é estabelecida uma relação entre as velocidades do *sink* e pacote. O segundo ponto A_B, está localizado no ponto de intersecção da linha com a fronteira do campo de sensores, conforme ilustrado na figura 2(b). Para esse ponto, o tempo de espera para agregação de dados é definido como o tempo para uma consulta ser propagada a partir do nó atual e retornar como um pacote de resposta do nó mais distante na linha. Logo, pela equação $A_t = 2 \times \frac{dist((Xc_i, Yc_i), A_B)}{P_k}$ o nó atual estima o tempo necessário para os pacotes de resposta dos nós intermediários chegam à sua posição.

3.2. Algoritmo HISPEAR

O HISPEAR, mostrado e explicado no algoritmo 1, começa quando o nó *sink* entra no campo de sensores e envia um pacote de consulta (linhas 2-6). Cada nó que recebe a

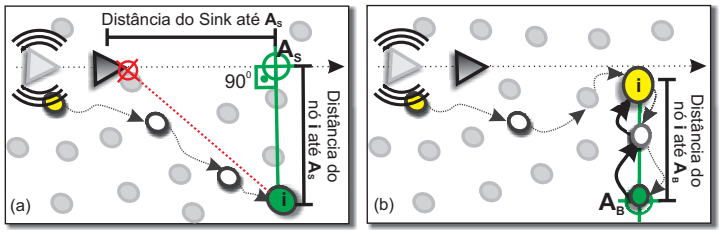

Figura 2. Tempo de Espera para Agregar Dados: (a) ponto de referência baseado nas velocidades do *sink* e pacotes; e (b) ponto de referência para os pacotes dos nós mais distantes.

consulta (diretamente do *sink* ou por meio de um nó intermediário) estima o atraso desse pacote, atualiza sua tabela de vizinhos e encaminha o pacote de consulta para os seus vizinhos (linhas 8–16). Em seguida, o tempo de espera para agregação de dados é calculado (linhas 17–21) e, um temporizador é então inicializado com o tempo escolhido (A_t , linha 22) para agregar os pacotes que chegam ao nó enquanto esse temporizador estiver ativo.

Algorithm 1 Algoritmo HISPEAR

▷ **Input:**
1: Nó *sink* 0 entra no campo de sensores e envia o pacote de consulta *query* com id id_0.
Action:
2: $id_0 + +; src_0 \leftarrow v_0; R_0 \leftarrow 0;$ {Id da consulta, origem e tempo de roteamento}
3: $T_0 \leftarrow traj(); S_0 \leftarrow speed();$ {Trajetória do *sink* e velocidade}
4: $Lp_0 \leftarrow (Xc_i, Yc_i);$ {Última posição do *sink*}
5: $cmd_0 \leftarrow 'SELECT\ MAX(temperature)\ FROM\ wsn';$ {Especificação da função de agregação de dados}
6: Envie $qry(src_0, id_0, cmd_0, T_0, S_0, R_0, Lp_i);$ {Dissemina a consulta}
▷ **Input:**
7: $msg_i \leftarrow qry(src_k, id_k, cmd_k, T_k, S_k, R_k, Lp_k); d_i = delay(msg_i);$
Action:
8: $R_i \leftarrow R_k + d_i;$ {Atualiza o tempo de roteamento}
9: $P_i \leftarrow dist((Xc_i, Yc_i), Tk)/R_i;$ {Velocidade de propagação da consulta}
10: **if** $k \neq 0$ **then** {SE: o pacote não veio diretamente do nó *sink* }
11: $Neig_i \leftarrow Neig_i \cup (k, Lp_k)$ {Atualiza a tabela de vizinhos}
12: **end if**
13: **if** $(src_k, id_k) \notin Fwd_i$ **then** {SE: o nó ainda não propagou a consulta }
14: $Fwd_i \leftarrow Fwd_i \cup (src_k, id_k)$ {Atualiza a tabela de encaminhamento}
15: $Lp_i \leftarrow (Xc_i, Yc_i)$ {Posição deste nó}
16: Envie $qry(src_k, id_k, cmd_k, T_k, S_k, R_i, Lp_i);$ {Encaminha a consulta}
17: $A_t \leftarrow dist(T_k, A_S) \times S_k - dist((Xc_i, Yc_i), A_S)/P_i;$ {Tempo para agregar os dados em relação ao *sink* }
18: $tmp \leftarrow 2 \times dist((Xc_i, Yc_i), A_B)/P_i;$ {Tempo para agregar os dados dos nós mais distantes}
19: **if** $tmp < A_t$ **then** {SE: Escolhe o menor tempo para agregar dados}
20: $A_t \leftarrow tmp;$
21: **end if**
22: $Timer_i.schedule(A_t)$ {Inicializa o temporizador para agregação de dados}
23: **end if**
24: **if** $data_i \leftarrow evaluate(cmd_k)$ **then** {SE: o nó tem dados para responder ao *sink*}
25: $src_i \leftarrow i;$ {Id de origem da resposta}
26: $Fwd_i \leftarrow Fwd_i \cup (ori_k, id_k);$ {Atualiza a tabela de encaminhamento}
27: $forward(src_i, id_k, data_i, T_k, S_k, R_i, P_i)$; {Chama o algoritmo de encaminhamento}
28: **end if**
▷ **Input:**
29: $msg_i \leftarrow reply(src_k, id_k, data_k, T_k, S_k, R_k, P_k); d_i = delay(msg_i);$
Action:
30: **if** $k = 0$ **then** {SE: este nó é o *sink*}
31: $store(data_k)$ {Recebe o pacote de resposta e o armazena}
32: **else**
33: **if** $(src_k, id_k) \notin Fwd_i$ **then** {SE: o este nó nunca encaminhou um pacote de resposta}
34: $Fwd_i \leftarrow Fwd_i \cup (src_k, id_k)$ {Atualiza a tabela de encaminhamento}
35: $R_i \leftarrow R_k + d_i$ {Atualiza o tempo de roteamento}
36: $forward(src_i, id_k, data_i, T_k, S_k, R_i, P_k)$; {Chama o algoritmo de encaminhamento}
37: **end if**
38: **end if**

Para encaminhar esses pacotes, o algoritmo HISPEAR chama a função de encaminhamento ($forward$, linhas 27 e 36), a qual irá escolher uma das três variantes do algoritmo. Neste trabalho, propomos três diferentes variantes do algoritmo HISPEAR: o algoritmo HISPEAR *Intercept* (seção 3.3), o algoritmo HISPEAR *Shortest* (seção 3.4) e o algoritmo HISPEAR *Hybrid* (seção 3.5). Nas próximas seções vamos explicar a operação destes três algoritmos propostos.

3.3. HISPEAR *Intercept*

O algoritmo HISPEAR *Intercept* calcula o primeiro ponto de intercepção entre a trajetória do nó *sink* e do pacote de resposta. Neste algoritmo, para encaminhar um pacote agregado ($data_{Ak}$), conforme descrito no algoritmo 2 (linha 2), primeiro é verificado se o pacote é capaz de interceptar o nó *sink* antes que ele saia do campo de sensores. Nesse caso, o pacote é transmitido imediatamente para o seu nó vizinho que está mais próximo da posição atual do nó *sink* (linhas 3–9). Caso contrário, o ponto de intercepção na trajetória do *sink* é calculado com base nas velocidades do *sink* e da velocidade de propagação da consulta [Oliveira et al. 2010] para encaminhar o pacote (linhas 11–15). Se o temporizador de agregação de dados ($Timer_k$) está ativo, os dados agregados são agendados para ser encaminhados para o nó vizinho que está mais próximo do ponto calculado (linhas 16-17), caso contrário, os dados são enviados imediatamente (linhas 18–21). Como mostra a figura 3(a), os caminhos criados por este algoritmo para encaminhar e agregar dados tendem a ser inclinados em relação à trajetória do nó *sink*.

Algorithm 2 Algoritmo de encaminhamento HISPEAR *Intercept*

▷ **Input:**
1: $forward(src_k, id_k, data_k, T_k, S_k, R_k, P_k)$;
Action:
2: $data_{Ak} = aggregate(data_{Ak}, data_k)$ {Efetua a agregação de dados }
3: **if** $S_k > P_k$ **then** {SE: se a velocidade do *sink* é maior que a do pacote}
4: $sinkPos.X = T_k.X + S_k \times \cos(T_k.\theta) \times R_k$;
5: $sinkPos.Y = T_k.Y + S_k \times \sin(T_k.\theta) \times R_k$;
6: $nextHope_k = closestNeigh(sinkPos.X, sinkPos.Y, sinkPos.Z)$; {Vizinho mais próximo do *sink*}
7: $Timer_k.cancel()$; {Cancela o temporizador}
8: Envie $reply(src_k, id_k, data_{Ak}, T_k, S_k, R_k, P_k)$ para $nextHop_k$ {Encaminhe o pacote imediatamente}
9: $data_{Ak} \leftarrow \emptyset$;
10: **else**
11: $timeIntercept = \sqrt{\frac{(T_k.Y - Y_{ci})^2}{P_k^2 - (S_k \times \sin(T_k.\theta))^2}}$
12: $sinkPos.X = T_k.X + S_k \times \cos(T_k.\theta) \times (R_k + timeIntercept)$;
13: $sinkPos.Y = T_k.Y + S_k \times \sin(T_k.\theta) \times (R_k + timeIntercept)$;
14: $sinkPos.Z \leftarrow T_k.Z$;
15: $nextHop_k = closestNeigh(sinkPos.X, sinkPos.Y, sinkPos.Z)$; {Ponto de interceptação}
16: **if** $Timer_k.isActive()$ **then** {SE: ainda há tempo para agregar dados}
17: $msg_k \leftarrow (src_k, id_k, data_{Ak}, T_k, S_k, R_k, P_k)$; {Atualiza o pacote agendado}
18: **else** {SENÃO: encaminhe o pacote agora}
19: $Timer_k.cancel()$; {Cancela o temporizador}
20: Envie $reply(src_k, id_k, data_{Ak}, T_k, S_k, R_k, P_k)$ para $nextHop_k$ {Encaminhe os pacotes agregados agora}
21: $data_A \leftarrow \emptyset$;
22: **end if**
23: **end if**

3.4. HISPEAR *Shortest*

O algoritmo *HISPEAR Shortest*, em vez de calcular o primeiro ponto de interceptação, calcula o ponto mais próximo de interceptação, como explicado no algoritmo 3 (linhas 2–7). Uma vez que o *sink* pode estar distante desse ponto mais próximo calculado, também é calculado um tempo de espera para que o *sink* esteja mais perto desse ponto ($timesink$,

linhas 8-10) para encaminhar os pacotes. Assim, antes de enviar o pacote de resposta, nós sensores utilizam esse tempo para efetuar agregação de dados enquanto esperam que o nó *sink* aproxime-se do ponto de entrega. Entretanto, também é verificado se o nó sink já cruzou o ponto mais próximo de interceptação ($nextTime$ negativo). Nesse caso, o pacote de resposta é enviado imediatamente utilizando o algoritmo HISPEAR *Intercept* (linhas 11–12).

Algorithm 3 Algoritmo de Encaminhamento HISPEAR *Shortest*

▷ **Input:**
1: $forward(src_k, id_k, data_k, T_k, S_k, R_k, P_k)$;.
Action:
2: $dX \leftarrow S_k \times \cos(T_k.\theta) \times R_k$;
3: $dY \leftarrow S_k \times \sin(T_k.\theta) \times R_k$;
4: $tan \leftarrow \frac{(Xc_i - T_k.X) \times dX + (Yc_i - T_k.Y) \times dY}{dX^2 + dY^2}$;
5: $sinkPos.X \leftarrow T_k.X + tan \times dX$
6: $sinkPos.Y \leftarrow T_k.X + tan \times dY$
7: $sinkPos.Z \leftarrow T_k.Z$;
8: $timePkt \leftarrow dist((Xc_i, Yc_i), sinkPos)/P_k$; {Tempo para o *sink* estar próximo}
9: $timeSink \leftarrow dist(sinkPos, T_k)/S_k$; {Tempo para o pacote alcançar o *sink*}
10: $nextTime \leftarrow timeSink - timePkt$;
11: **if** $nextTime < 0$ **then** {SE: o nó *sink* ultrapassou o ponto}
12: encaminhe o pacote imediatamente pelo algoritmo HISPEAR *Intercept*;
13: **else**
14: $nextHop_k = closestNeigh(sinkPos.X, sinkPos.Y, sinkPos.Z)$; {Ponto mais próximo}
15: $data_{Ak} = aggregate(data_A, data_k)$ {Efetua a agregação de dados}
16: $msg_k \leftarrow (src_k, id_k, data_{Ak}, T_k, S_k, R_k, P_k)$; {Atualiza o pacote agendado}
17: **if** $Timer_k.isNotActive()$ **then** {SE: o timer para agregação de dados não está ativo}
18: $Timer_A.schedule(nextTime)$ {Inicializa o temporizador para agregação de dados}
19: **else** {SENÃO: Atualiza o temporizador para agregação de dados }
20: $Timer_A.reschedule(nextTime)$ {Atualiza o temporizador para agregação de dados}
21: **end if**
22: **end if**

Um aspecto interessante deste algoritmo é que os nós sensores calculam aproximadamente o mesmo ponto mais próximo em um caminho de entrega. O tempo para o nó *sink* estar próximo do ponto mais próximo é recalculado em cada salto e utilizado para atualizar o temporizador da agregação de dados (linhas 17–20). Como ilustra a figura 3(b), a trajetória do pacote de resposta tende a ser uma linha perpendicular à trajetória do *sink*.

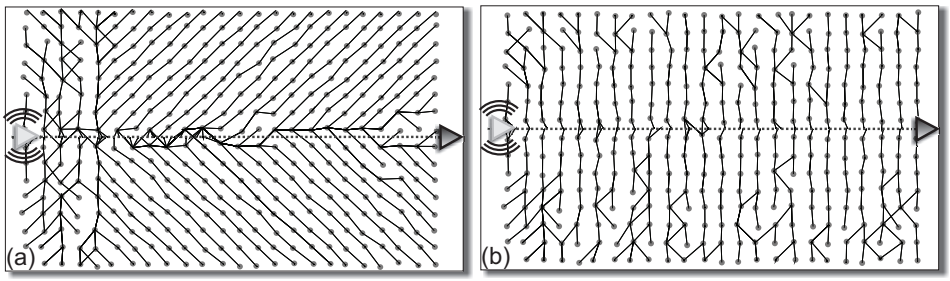

Figura 3. Grafos de roteamento HISPEAR: (a) HISPEAR *Intercept*; e (b) HISPEAR *Shortest*.

3.5. HISPEAR *Hybrid*

O algoritmo HISPEAR *Hybrid* foi projetado para explorar um número reduzido de transmissões, a partir de caminhos mais curtos de encaminhamento e efetuar agregação de dados mesmo quando a velocidade do *sink* é alta em comparação com a velocidade dos pacotes. O algoritmo proposto escolhe, a cada salto, a melhor opção entre o ponto de

menor distância (HISPEAR *Shortest*) e o primeiro ponto de intercepção (*HISPEAR Intercept*). Como mostrado no algoritmo 4, primeiro é calculado o ponto mais próximo de interceptação na trajetória do *sink* (linha 2), da mesma forma que é calculado no HISPEAR *Shortest* (algoritmo 3, linhas 2–7). O primeiro ponto de intercepção entre as trajetórias do nó *sink* e do pacote de resposta é também calculado (linha 3), assim como no HISPEAR *Intercept* (algoritmo 2, linhas 11–15).

Algorithm 4 Algoritmo de encaminhamento HISPEAR *Hybrid*

▷ **Input:**
1: $forward(src_k, id_k, data_k, T_k, S_k, R_k, P_k)$;
Action:
2: Calcula o ponto mais próximo de interceptação (HISPEAR *Shortest*);
3: Calcula o primeiro ponto de interceptação (HISPEAR *Intercept*);
4: **if** $timePkt < timeIntercept$ **then** { SE: o tempo do ponto mais próximo é melhor}
5: Encaminhe o pacote pelo algoritmo HISPEAR *Shortest*;
6: **else** { SENÃO: o tempo do primeiro ponto de interceptação é melhor}
7: Encaminhe o pacote pelo algoritmo HISPEAR *Intercept*;
8: **end if**

Com base nesses dois pontos calculados, o algoritmo HISPEAR *Hybrid* escolhe aquele que resulta no menor tempo para entrega de dados (linhas 4–8) e encaminha o pacote de resposta usando o algoritmo respectivo. No algoritmo HISPEAR *Hybrid*, a trajetória do pacote de resposta também tende a ser uma linha perpendicular à trajetória do *sink*, da mesma forma que no algoritmo HISPEAR *Shortest*, como ilustra a figura 3(b).

4. Avaliação de Desempenho

4.1. Metodologia

A avaliação de desempenho foi realizada através de simulações utilizando o simulador NS-2 (versão 2.34). Os parâmetros de simulação são baseados na plataforma MicaZ, os padrão de valores destes parâmetros são mostrados na Tabela 1. Em todos os resultados, as curvas representam valores médios, enquanto as barras de erro representam intervalos de confiança para 95 % de confiança a partir de 33 instâncias independentes (sementes aleatórias). Quanto à topologia da rede, assumimos que a implantação dos nós obedece a uma grade perturbada, na qual a localização de cada nó é perturbada por um erro aleatório Gaussiano de média zero. Portanto, os nós tendem a ocupar uniformemente o campo de sensores sem formar uma grade regular.

Parâmetro	Valor
Campo de sensores	728 m × 728 m
Número de nós	576
Densidade dos nós	0,001 nós/m^2
Raio de Comunicação	50 m
Número de Vizinhos	7.6 nós
Atraso em um salto	0.1 s
Erro não determinístico	30 µs
Erro de Localização	2 m
Altura do *sink*	40 m
Velocidade do *sink*	200 km/h
Trajetória do *sink*	linha

Tabela 1. Parâmetros de simulação.

Para simular imprecisões de cálculo de posição, perturbamos a posição dos nós utilizando uma distribuição de Gaussiana com média igual ao valor da posição real do nó e um desvio padrão de 2 m [Langendoen and Reijers 2003]. Para simular imprecisões na medição de atrasos, perturbamos o atraso médio em um desvio padrão de

30 µs [Maróti et al. 2004]. Finalmente, a trajetória do nó *sink* é definida como uma linha que passa pelo meio do campo de sensores por ser uma trajetória comum para aeronaves em alta.

Em todas as simulações, cada nó sensor gera um pacote de dados de eventos detectados. A economia de transmissões é a razão entre o número de transmissões de pacotes de dados que não foram necessárias devido à agregação de dados e o numero total de pacotes de dados que seriam transmitidos sem a utilização da agregação de dados. Como o cenário deste trabalho é focado em sinks em alta velocidade (como VANTs), é realístico afirmar que a coleta de dados não será frequente e uma grande quantidade de dados será detectada. Neste caso, optamos por analisar o pior cenário, onde todos os nós da rede têm dados coletados.

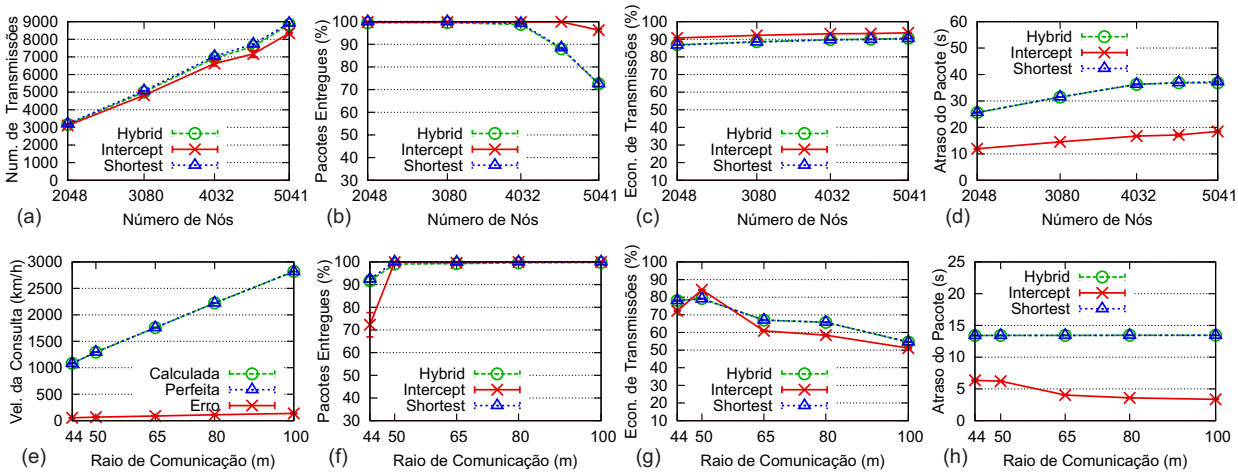

Figura 4. Escala da Rede e raio de comunicação.

4.2. O Impacto da Escala da Rede

A escalabilidade é avaliada através do aumento da quantidade de nós da rede de 2048 até 5041, mantendo mesma densidade. Assim, o campo de sensores é redimensionado de acordo com o número de nós. Quando se aumenta o tamanho da rede, a sobrecarga na rede aumenta na mesma proporção, uma vez que todos os nós sensores geram um pacote de dados para transmitir. A figura 4(a) compara o número de pacotes transmitidos enquanto aumenta o número de nós. Nesta avaliação, podemos perceber um aumento no número de pacotes transmitidos, onde o algoritmo HISPEAR *Intercept* tem a vantagem, especialmente para valores maiores 4032 nós. Para este cenário, os pacotes de resposta enviados pelos algoritmos HISPEAR *Shortest* e HISPEAR *Hybrid* têm os menores caminhos de comunicação, no entanto, o algoritmo HISPEAR *Intercept* tem um maior grau de agregação de dados resultando em menos pacotes transmitidos.

A figura 4(b) mostra que o HISPEAR *Intercept* é capaz de entregar mais que 95 % dos pacotes quando o número de nós aumenta para 5041. Entretanto, os algoritmos HISPEAR *Shortest* e HISPEAR *Hybrid* são altamente afetados pelo aumento da escala da rede, especialmente para valores maiores que 4032 nós, quando o *sink* deixa o campo de sensores antes de receber todos os pacotes de resposta destes algoritmos. Na figura 4(c), podemos notar que o algoritmo HISPEAR *Intercept* tem uma maior taxa de economia de transmissões enquanto não há diferença estatística entre os algoritmos HISPEAR *Shor-*

test e HISPEAR *Hybrid*. Relacionado com este resultado, a economia de transmissão alcançada por todas as técnicas pela agregação de dados contribui para reduzir a sobrecarga na rede, reduzindo em mais de 90% a quantidade de pacotes, contribuindo assim com a eficiência na entrega de dados e throughput. É importante destacar que que as vantagens da agregação de dados são proporcionais à quantidade de dados coletados, uma vez que diminuem proporcionalmente com a redução de eventos detectados.

Finalmente, a figura 4(d) mostra uma desvantagem dos algoritmos HISPEAR *Shortest* e HISPEAR *Hybrid*: um aumento no atraso dos pacotes. Esse comportamento era esperado, uma vez que o algoritmo HISPEAR *Shortest* espera que o nó *sink* esteja mais próximo do ponto de entrega, antes de enviar o pacote de resposta e, para este cenário, o algoritmo HISPEAR *Hybrid* foi projetado para ter o mesmo comportamento. Estes resultados, mostram claramente que o tempo de espera para que o nó *sink* esteja próximo do ponto de entrega introduz um atraso maior em relação ao tempo de espera necessário para agregar os dados coletados.

4.3. O Impacto do Raio de Comunicação

Para avaliar o impacto do raio de comunicação, aumentamos esse parâmetro de 44 m para 100 m. Ao aumentar o raio de comunicação, a velocidade dos pacotes também aumenta, conforme mostra a figura 4(e). Nesse caso, podemos notar que o algoritmo HISPEAR foi capaz de calcular com precisão esta velocidade, o que é muito importante para o funcionamento do algoritmo proposto. Depois de um raio de comunicação de 50 m, em torno de 100 % dos pacotes são entregues, como mostrado na figura 4(f). Considerando 44 m de raio de comunicação (e com o *sink* a uma altura de 40 m), podemos notar uma diminuição no número de pacotes entregues. Entretanto, os algoritmos HISPEAR *Shortest* e HISPEAR *Hybrid* ainda são capazes de entregar mais de 90 % dos pacotes, o que indica que estas variantes do algoritmo HISPEAR são mais confiáveis para menores diferenças entre a altura do *sink* e raio de comunicação dos nós.

Como ilustrado na figura 4(g), podemos notar uma diminuição na economia de transmissões quando o raio de comunicação aumenta. Esta diminuição ocorre devido ao o número de saltos para entregar os dados para o nó *sink* diminuir e, consequentemente, o número de oportunidades para efetuar agregação de dados diminui na mesma maneira. Além disso, um resultado interessante pode ser visto na figura 4(h): o atraso dos pacotes diminui no algoritmo HISPEAR *Intercept* enquanto em ambos os algoritmos HISPEAR *Shortest* e HISPEAR *Hybrid* o atraso dos pacotes permanece praticamente igual, já que os nós terão de esperar para que o nó *sink* esteja mais perto do ponto de entrega.

4.4. O Impacto do Atraso Médio de Um Salto

O atraso de um salto refere-se ao tempo de processamento exigido por nó sensor, antes de encaminhar um pacote (e.g., para calcular a posição do *sink*, o próximo salto, para agregar dados). Para avaliar o impacto desse atraso, variamos esse parâmetro de 0.1 s para 0.5 s. Como mostrado na figura 5(a), o aumento do atraso afeta o número de pacotes transmitidos, uma vez que os pacotes de resposta precisam percorrer uma distância maior para alcançar o nó *sink*. Os algoritmos HISPEAR *Shortest* e HISPEAR *Hybrid* tiveram um desempenho melhor, resultando em menores caminhos de comunicação. Como mostra a figura 5(b), quando o atraso é maior que 0.3 s, podemos notar que ambos os algoritmos HISPEAR *Intercept* e *HISPEAR Hybrid* são capazes de entregar mais pacotes, enquanto

o algoritmo HISPEAR *Shortest* é altamente afetado pelo aumento do atraso, uma vez que os pacotes de resposta de nós distantes não chegam a tempo ao nó *sink*, antes que ele saia do campo de sensores, de modo que o pacote é descartado.

Na figura 5(c), podemos ver uma desvantagem do algoritmo HISPEAR *Intercept* na economia de transmissões, uma vez que o atraso maior causa um aumento no número de pacotes de resposta enviados imediatamente para interceptar o nó *sink* sem serem agregados. Podemos notar também que economia de transmissões em ambos os algoritmos HISPEAR *Shortest* e HISPEAR *Hybrid* não é afetada pelo atraso nos pacotes. Como pode ser visto na figura 5(d), temos novamente outro interessante resultado para estes dois algoritmos em relação ao atraso introduzido, uma vez que o atraso nos pacotes permanece praticamente o mesmo. Esse comportamento também é explicado pelo tempo necessário para aguardar que o nó *sink* esteja mais perto do ponto de entrega

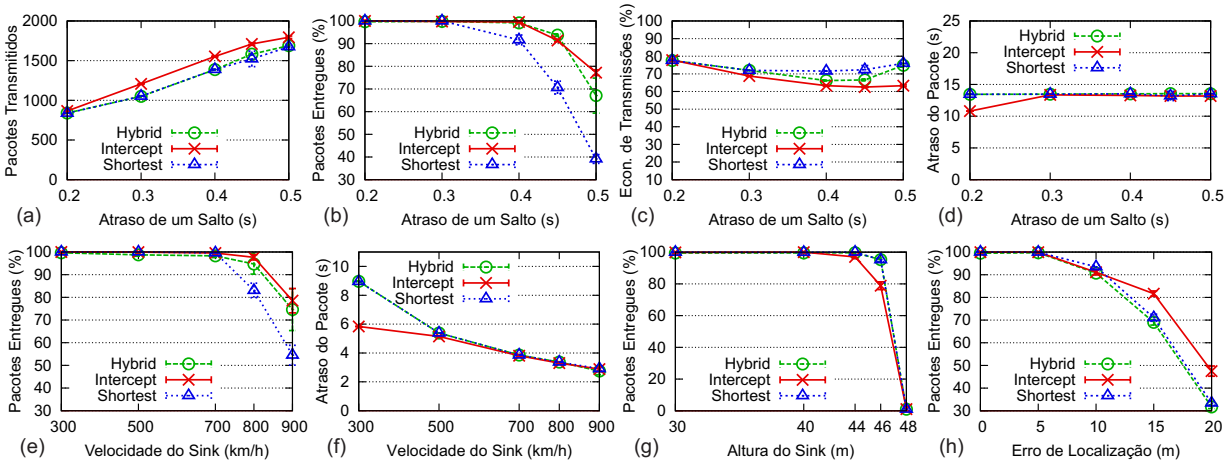

Figura 5. Atraso de um salto, velocidade do sink, altura do sink e erro de localização.

4.5. O Impacto da Velocidade e Altura do Nó *Sink*

Para avaliar o impacto da velocidade do nó *sink*, aumentamos esse parâmetro de 300 km/h (velocidade de um VANT) para 900 km/h (velocidade de um Boeing-737). Como pode-se ver na figura 5(e), em torno de 95 % dos pacotes são entregues até 700 km/h por todas as técnicas. Depois de 800 km/h, podemos notar uma diminuição no número de pacotes entregues devido ao nó *sink* deixar o campo de sensores antes de receber todos os pacotes de resposta. O algoritmo HISPEAR *Intercept* tem um melhor desempenho, enquanto o algoritmo HISPEAR *Shortest* é afetado pela velocidade do nó *sink*. Como esperado, o algoritmo HISPEAR *Hybrid* teve um comportamento semelhante ao algoritmo HISPEAR *Intercept* com uma pequena desvantagem. A figura 5(f) mostra que o atraso do pacote de resposta cai em todas as técnicas com uma vantagem para o algoritmo HISPEAR *Intercept* até 500 km/h. Esta queda no atraso dos pacotes também é explicada pelo aumento do número de pacotes enviados imediatamente em virtude do aumento da velocidade do nó *sink*.

Avaliamos o impacto da altura do nó *sink* aumentando esse parâmetro de 30 m para 48 m (nesse caso ficando com uma diferença de somente 2 m em relação ao raio de comunicação dos nós sensores). Como ilustra a figura 5(g), quando a altura do nó *sink* é superior a 44 m (apenas 6 m a menos do que o raio de comunicação dos nós) ocorre uma

queda acentuada no número de pacotes entregues, uma vez que a altura do nó *sink* fica muito próxima do limite do raio de comunicação dos nós sensores.

4.6. O Impacto do Erro de Localização

Para avaliar o impacto do erro de localização, aumentamos esse parâmetro de 0 m até 20 m. Como mostra a figura 5(h), o algoritmo HISPEAR algoritmo não é afetado de forma significativa por erros de localização, especialmente para grandes valores de erro, tais como erros até 10 m (o qual é 20% do raio de comunicação). Para esse valor, todas as técnicas são capazes de entregar mais de 90% dos pacotes.

5. Conclusão

Neste trabalho, propomos três novas técnicas de roteamento combinadas com agregação de dados, para um nó *sink* em alta velocidade, definidas como o algoritmo HISPEAR *(High Speed Aggregation and Routing)*. A ideia principal do algoritmo HISPEAR é calcular o melhor tempo disponível para agregar dados em cada salto, de maneira a entregar dados menos redundantes em direção a uma posição mais atualizada, ou mesmo uma posição futura do nó *sink*. Propomos três variantes do algoritmo HISPEAR: HISPEAR *Intercept*, HISPEAR *Shortest* e HISPEAR *Hybrid*.

O algoritmo HISPEAR *Intercept* obteve os melhores resultados em termos de escala da rede e velocidade do *sink* com um atraso menor nos pacotes. Embora não resultando nos menores caminhos de entrega de dados, ao responder as consultas para o *sink*, um elevado número de pacotes são agregados. Os algoritmos HISPEAR *Shortest* e HISPEAR *Hybrid* têm um atraso maior nos pacotes, porém ambos resultam nos menores caminhos de comunicação, com uma vantagem do algoritmo HISPEAR *Hybrid* para o número de pacotes agregados e entregues. Ainda, todas as técnicas foram capazes de alcançar significativa economia de energia e eficiente entrega de dados em diferentes cenários de RSSFs.

Os resultados são muito promissores, mas algumas limitações ainda precisam ser exploradas como trabalhos futuros. Primeiro, precisamos combinar nossa solução com um algoritmo de roteamento por superfície/face. Também pretendemos avaliar o desempenho dos algoritmos propostos utilizando algoritmos de localização reais, tais como [Boukerche et al. 2007]. Finalmente, iremos avaliar o custo computacional da solução proposta em relação modelos não lineares de trajetória do *sink* assim como o gasto energético de funções complexas de agregação de dados.

Agradecimentos

Este trabalho foi parcialmente suportado pelo CNPq, FAPESP e FAPEAM sob os processos 573963/2008-8, 08/57870-9 e 01135/2011.

Referências

Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). A survey on sensor networks. *IEEE, Communications Magazine*, 40:102–114.

Boukerche, A. and Nikoletseas, S. (2004). Protocols for data propagation in wireless sensor networks. pages 23–51. Plenum Press, New York, NY, USA.

Boukerche, A., Oliveira, H. A. B. F., Nakamura, E. F., and Loureiro, A. A. F. (2007). Localization systems for wireless sensor networks. *Wireless Communications, IEEE*, 14:6–12.

Estrin, D., Girod, L., Pottie, G., and Srivastava, M. (2001). Instrumenting the world with wireless sensor networks. In *ICASSP'01*, pages 2033–2036, Salt Lake City, Utah.

Faheem, Y., Boudjit, S., and Chen, K. (2009). Data dissemination strategies in mobile sink wireless sensor networks: A survey. In *2009 2nd IFIP, Wireless Days (WD)*, pages 1–6.

Hong, W. and Madden, S. (2004). Implementation and research issues in query processing for wireless sensor networks. In *ICDE '04*, page 876, Washington, DC, USA. IEEE.

Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. (2003). Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11:2–16.

Kim, H. S., Abdelzaher, T. F., and Kwon, W. H. (2003). Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 193–204.

Krishnamachari, L., Estrin, D., and Wicker, S. (2002). The impact of data aggregation in wireless sensor networks. In *ICDCSW'02*, pages 575–578.

Langendoen, K. and Reijers, N. (2003). Distributed localization in wireless sensor networks: a quantitative comparison. volume 43, pages 499–518, New York, NY, USA. Elsevier North-Holland, Inc.

Luo, J. and Hubaux, J. P. (2010). Joint sink mobility and routing to maximize the lifetime of wireless sensor networks: The case of constrained mobility. *IEEE/ACM Transactions on Networking*, 18:871–884.

Maróti, M., Kusy, B., Simon, G., and Lédeczi, A. (2004). The flooding time synchronization protocol. In *SenSys'04*, pages 39–49, New York, NY, USA. ACM.

Oliveira, H. A. B. F., , Barreto, R. S., Fontao, A. L., Loureiro, A. A. F., and Nakamura, E. F. (2010). A novel greedy forward algorithm for routing data toward a high speed sink in wireless sensor networks. In *ICCCN'10*, pages 1–7.

Shim, G. and Park, D. (2006). Locators of mobile sinks forwireless sensor networks. In *ICPPW'06*, pages 159–164, Washington, DC, USA. IEEE.

Ye, F., Luo, H., Cheng, J., Lu, S., and Zhang, L. (2002). A two-tier data dissemination model for large-scale wireless sensor networks. In *MobiCom '02*, pages 148–159, New York, NY, USA. ACM.