Envio de Dados de Consulta para Sinks Móveis em Alta Velocidade em Redes de Sensores Sem Fio*

Horácio A.B.F. Oliveira[1], Raimundo S. Barreto[1], Awdren L. Fontão[1], Eduardo F. Nakamura[2], Antonio A.F. Loureiro[3]

[1]Departamento de Ciência da Computação – Universidade Federal do Amazonas
[2]Centro de Análise, Pesquisa e Inovação Tecnológica – FUCAPI
Manaus – AM – Brasil
[3]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
Belo Horizonte – MG – Brasil

{horacio,rbarreto,awdren}@dcc.ufam.edu.br
eduardo.nakamura@fucapi.br, loureiro@dcc.ufmg.br

Abstract. *Sink nodes are nodes responsible for aggregating the data gathered by a Wireless Sensor Network (WSN) and transmitting them to a monitoring facility. Usually, these are static nodes that serve as gateways to infrastructured networks. In some studies, these sink nodes move around the monitoring area, usually in lower speeds (e.g., robots, dirigibles). In this work, we go further and study WSN scenarios in which sink nodes can move at higher speeds, such as an Unmanned Aerial Vehicle (UAV) or even an airplane. In these cases, propagated queries cannot be answered by using the sink's position when the query was sent, since the sink will be elsewhere due to its high speed. Thus, we propose and evaluate the performance of three algorithms for data query in WSNs when sink is moving in high speed. Our results show clearly the need for new algorithms for these scenarios as well as the good performance of our proposed algorithms.*

Resumo. *Nós sink são nós responsáveis por agregar dados coletados em uma Rede de Sensores Sem Fio (RSSF) e transmiti-los a uma central de monitoramento. Em geral, sinks são nós estáticos que operam como gateways para a rede infra-estruturada. Em alguns casos, nós sink têm a capacidade de se movimentar na área de monitoramento geralmente em baixa velocidade (e.g., robôs, dirigíveis). Este trabalho vai além e investiga cenários em RSSFs nos quais o sink pode se mover em alta velocidade como, por exemplo, um Veículo Aéreo Não-Tripulado (VANT) ou mesmo um avião. Nesses casos, consultas propagadas na rede não podem ser respondidas usando a posição do sink no momento da consulta, uma vez que no instante da resposta o sink estará em outra região e fora do alcance de comunicação dos nós que inicialmente receberam e propagaram a consulta. Desta forma, este trabalho propõe e avalia três novos algoritmos para consulta a dados em RSSFs direcionada a sinks que se movem em alta velocidade. Os resultados obtidos demonstram a necessidade de novos algoritmos para tais cenários bem como mostram o bom desempenho dos algoritmos propostos.*

*Este trabalho foi parcialmente suportado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) sob os processos 554087/2006-5, 474194/2007-8 e 575808/2008-0.

1. Introdução

Redes de Sensores Sem Fio (RSSFs) [Akyildiz et al. 2002, Estrin et al. 2001, Ilyas and Mahgoub 2004, Pottie and Kaiser 2000] são redes formadas por pequenos dispositivos computacionais capazes de monitorar o ambiente em que estão localizados, de processar os dados coletados e de se comunicarem através de comunicação sem fio. Quando implementada em uma extensa área de interesse, em um número que vai de dezenas, centenas ou milhares de nós sensores espalhados, cada elemento computacional da RSSF é responsável por monitorar o seu ambiente local e repassar os dados coletados e processados a um nó central, conhecido como nó *sink*. Tal nó é então responsável por agregar os dados recebidos e enviá-los a uma central de monitoramento por algum meio de comunicação.

Em geral, os nós sensores, incluindo o nó *sink*, são estáticos. Uma vez que comunicações de alto alcance têm a característica de consumir mais energia, em RSSFs os dados são passados ao *sink* através de múltiplos saltos, i.e., o nó envia a mensagem para o seu nó vizinho que se encontra mais perto do *sink*. Esse vizinho se encarregará, então, de retransmitir a mensagem para outro vizinho e assim por diante até que a mensagem chegue ao *sink*.

Trabalhos mais recentes [Ye et al. 2002, Fodor and Vidács 2007, Shim and Park 2006] utilizam *sinks* móveis. Nesses trabalhos, os nós *sinks* são, em geral, dispositivos de baixa mobilidade como robôs, dirigíveis, pequenos carros com controles remotos ou até mesmo pessoas, e o desafio é basicamente como manter as rotas para o *sink* atualizadas. Entretanto, alguns cenários mais atuais têm demonstrado a necessidade de formas de coleta de dados mais eficientes e ágeis. Por exemplo, em uma RSSF localizada em um terreno de difícil acesso, uma opção à utilização de um *sink* com alto poder de comunicação é a utilização de um Veículo Aéreo Não-Tripulado (VANT) ou até mesmo um avião para realizar esta tarefa de coletar dados. Tal veículo aéreo seria responsável por sobrevoar a região monitorada enviando perguntas à rede e recebendo respostas sobre as condições do ambiente.

Entretanto, ao se trabalhar com *sinks* movendo-se em alta velocidade, é necessário considerar uma série de desafios. Por exemplo, criar e manter rotas entre os nós e o *sink* móvel torna-se inviável, uma vez que as rotas mudariam mais rapidamente do que a capacidade da rede de propagar as novas informações. Colisões também precisam ser evitadas, pois poucas retransmissões podem ser suficientes para impedir que o pacote de resposta alcance o *sink* móvel antes deste sair da área de monitoramento devido à alta velocidade. Além disso, fatores diferenciados como altura do *sink* móvel, sua velocidade, atraso em cada salto na propagação de informações, etc, precisam ser avaliados em protocolos propostos nesses cenários.

Visando cenários de alta velocidade, este trabalho propõe um novo algoritmo para permitir ao *sink* realizar consultas a uma RSSF: o algoritmo RAVR (Roteamento em Alta Velocidade em Redes de Sensores Sem Fio). Com base nesse algoritmo, são propostas ainda três variantes: o *RAVR SigaMe*, o *RAVR Intercepta* e o *RAVR MenorCaminho*. A idéia principal do algoritmo RAVR é reconhecer que o *sink* não mais se encontra no local onde este fez a consulta. Assim, cada nó sensor que for repassar a resposta à consulta repassará essa resposta não para o local onde a consulta foi realizada (ver figura 1), mas para o local onde *sink* se encontra no tempo atual (*RAVR SigaMe*), no local onde

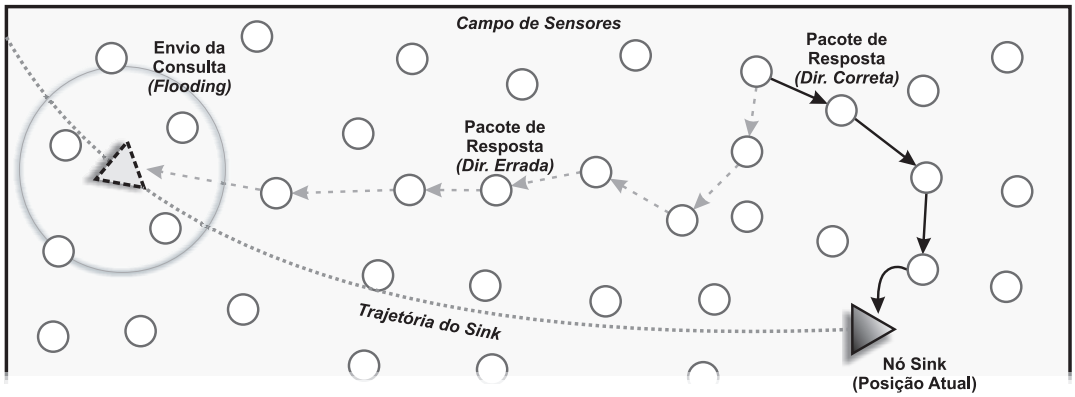

Figura 1. Uma consulta à RSSF através de um *sink* em alta velocidade.

o pacote poderá interceptar o *sink* (*RAVR Intercepta*) ou para o local onde o *sink* passará no futuro e que seja mais próximo do nó que está repassando a mensagem (*RAVR MenorCaminho*). Nesses três casos, o objetivo comum é que as respostas às consultas alcancem o seu destino. Tais algoritmos foram avaliados no simulador *network simulator-2* [McCanne and Floyd 2005] sob diversos cenários e diversos parâmetros. Os resultados demonstram claramente não só a necessidade de algoritmos novos para tais cenários, mas também a viabilidade e o bom rendimento dos algoritmos propostos.

O restante deste trabalho está organizado como segue. A seção 2 discute algumas pesquisas recentes em *sinks* móveis e outros trabalhos que avaliam o envio de pacotes em cenários de mobilidade em alta velocidade. A seção 3 apresenta algumas informações relevantes ao entendimento do presente trabalho. O algoritmo proposto RAVR bem como suas variantes são apresentados na seção 4 e as suas avaliações na seção 5. Por fim, a seção 6 discute alguns pontos a respeito da aplicação prática do RAVR e, a seção 7 apresenta as conclusões e sugestões de alguns trabalhos futuros.

2. Trabalhos Relacionados

Uma primeira tentativa de resolver o problema de roteamento com *sinks* móveis foi o algoritmo TTDD (*Two-Tier Data Dissemination*), proposto por [Ye et al. 2002]. Esse algoritmo permite uma entrega de dados escalável e eficiente para *sinks* móveis. Cada fonte de dados no algoritmo TTDD constrói uma estrutura em grade que habilita os *sinks* móveis para continuamente receber dados através de consultas por algoritmos de *flooding* somente dentro de uma célula local. Em [Fodor and Vidács 2007] é proposto um algoritmo de roteamento eficiente para *sinks* móveis em RSSFs. Nesse algoritmo, é usado *flooding* restrito para atualizar os caminhos para múltiplos *sinks* móveis na RSSF. A solução proposta tenta encontrar um ponto entre as rotas ótimas e a quantidade de mensagens necessárias para atualizar tais rotas. Por último, um trabalho que mais se assemelha a este é proposto por [Shim and Park 2006] que se baseia na utilização de localizadores para *sinks* móveis. Tais localizadores são uniformemente distribuídos no campo de sensores e localizam a atual posição do *sink* móvel. Se um sensor tenta enviar dados para o sink tardiamente, este sensor pode adquirir a nova posição do *sink* através dos localizadores.

Outros trabalhos estudaram o impacto da velocidade do *sink* na comunicação com os sensores. Em [Mergen et al. 2006], os autores realizam estudos com um dispositivo aéreo. Neste trabalho é assumido que n sensores são distribuídos aleatoriamente e uni-

formemente em um círculo de raio R. O *sink* móvel pode se posicionar em determinadas posições e enviar sinais aos sensores. Os resultados obtidos indicam que o aumento da altitude influencia na área de cobertura, bem como a diminuição de altitude diminui os gastos de energia, pois o esforço que será empregado para a transmissão de dados diminuirá. Em [Khalid et al. 2007], é estudada a habilidade de transmissão e recepção de dados. Um *sink* móvel se move pelo campo de sensores com velocidade V_i, fazendo um ângulo θ com o eixo X. Resultados indicam claramente que mesmo em alta velocidade, é possível enviar e receber dados da rede.

Conforme mostrado nesta seção, o problema de roteamento com *sinks* móveis tem sido abordado basicamente em baixas velocidades. Enquanto que trabalhos recentes, que lidam com velocidades maiores, focam-se basicamente no envio e recepção de dados. Neste trabalho, é proposta uma abordagem diferente e inovadora: roteamento de dados em altas velocidades. Similarmente aos trabalhos citados, será utilizado *flooding* para propagar consultas. Entretanto, ao contrário dos algoritmos citados, o algoritmo RAVR não se preocupa em gerar rotas, pois estas ficarão desatualizadas em milésimos de segundo, mas sim, em direcionar as respostas a uma posição atual ou futura do nó *sink*.

3. Definição do Problema

Nesta seção, serão apresentados alguns conceitos e informações necessárias para o melhor entendimento do presente trabalho.

Definição 1 (Rede de Sensores Sem Fio) *Uma RSSF pode ser representada por um grafo Euclidiano $G = (V, E)$ com as seguintes propriedades:*

- *n é o número de nós; r o raio de comunicação dos nós;*
- *$Q = [0, s] \times [0, s] \times [0, r]$ a área de sensoriamento em três dimensões;*
- *$V = \{v_0, v_1, \ldots, v_n\}$ é o conjunto de nós sensores e, sem perda de generalidade, considera-se v_0 como sendo o nó sink;*
- *$\langle i, j \rangle \in E$ se, e somente se, a distância entre v_i e v_j é menor que r, i.e., v_i alcança v_j e vice-versa;*
- *$\forall v_i \in V, (Xp_i, Yp_i, Zp_i) \in \mathbf{R}^3$ é a posição perfeita do nó v_i; e $(Xc_i, Yc_i, Zc_i) \in \mathbf{R}^3$ é a posição calculada do nó v_i (e.g., a partir de um sistema de localização).*

Definição 2 (*Sink* Móvel de Alta Velocidade) *Um sink móvel de alta velocidade, no presente trabalho, é definido como sendo o nó normal da rede v_0 com capacidade de:*

- *mobilidade acima de 100 km/h;*
- *mobilidade em padrões pré-definidos (e.g., linha reta, curva);*
- *localização contínua (e.g., equipado com GPS – Global Positioning System);*

e cujo principal objetivo é o de passar pela RSSF coletando dados.

Definição 3 (Algoritmo de Consulta a Dados) *Diferentemente dos algoritmos de roteamento normais que se preocupam em gerar e manter rotas entre origens e destinos, este trabalho foca principalmente nos algoritmos de roteamento para consulta a dados. Nestes algoritmos, o nó sink faz uma consulta à rede como se esta fosse um banco de dados distribuído (Sensor Databases [Hong and Madden 2004]). A consulta é então propagada na rede geralmente através de um flooding. Os nós que possuírem dados relativos à consulta realizada, respondem à consulta através de um*

pacote de resposta enviado para o sink. Em geral, poucos são os nós que respondem à pergunta. Como exemplos de algoritmos conhecidos de consulta a dados tem-se o Directed Diffusion [Intanagonwiwat et al. 2000] e o Received Signal Strength Routing [Boukerche et al. 2008].

4. RAVR - Envio de Dados Direcionado a *Sinks* em Alta Velocidade

Nesta seção, será proposto um novo algoritmo para consulta a dados em RSSFs: o RAVR (Roteamento em Alta Velocidade para RSSFs). Conforme mencionado (e ilustrado na figura 1), o ponto chave do RAVR está em reconhecer que o *sink* não mais se encontra no local onde este enviou a consulta à rede, mas sim em outro local da área de monitoramento devido à sua trajetória em alta velocidade. Desta forma, se o pacote de resposta for roteado em direção ao local onde a consulta foi enviada, este não mais alcançará o *sink*. Sendo assim, o algoritmo RAVR procura enviar o pacote de resposta em direção a uma posição mais atualizada do *sink* ou mesmo em direção a sua posição futura. Tal posição do *sink* é recalculada a cada salto por cada nó sensor e, então, este nó repassa a mensagem para o seu nó vizinho que se encontra mais próximo desta nova posição do *sink*. Para que isso seja possível, é necessário que cada nó conheça a sua posição calculada (ver definição 1) e a de seus vizinhos, bem como a Trajetória e o Deslocamento do *Sink* (definições 4 e 5).

Definição 4 (Trajetória do *Sink* – T_s) *Pode ser uma linha reta, uma curva ou qualquer trajetória passível de ser descrita matematicamente. Por motivos de simplicidade, mas sem perda de generalidade, neste trabalho será considerado que, enquanto estiver no campo de sensores, o sink manterá a trajetória em linha reta, uma trajetória comum no caso de objetos em alta velocidade. Logo, dado um ponto inicial (e.g., posição do sink no momento do envio da consulta à rede) e uma direção, esta reta pode ser facilmente calculada como* $y = tan(\theta)(x - x_0) + y_0$*, onde* θ *é o ângulo em relação ao eixo* x *e o ponto inicial (*x_0*,* y_0*). Tais informações são enviadas com a consulta do sink à rede.*

Definição 5 (Deslocamento do *Sink* – $S_s(t)$) *É a variação da posição do sink dentro de sua trajetória no tempo* t *(medido em segundos). Tal deslocamento também precisa ser descrito matematicamente como, por exemplo,* $S_s(t) = vt + \frac{1}{2}at^2$ *onde* a *é a aceleração constante e* v *a velocidade inicial. Novamente, por motivos de simplicidade mas sem perda de generalidade, neste trabalho a aceleração será nula, ou seja, não haverá mudança de velocidade enquanto o sink estiver dentro do campo de sensores e, então, seu deslocamento será simplesmente* $S_s(t) = v_s t$*, onde* v_s *é a velocidade do sink (medida em* km/h*).*

O Algoritmo 1 define o funcionamento do algoritmo RAVR proposto neste trabalho. Inicialmente, cada nó armazena localmente uma lista de pacotes encaminhados ($vetEnc_i$ – linha 1 – para evitar que a mesma consulta ou resposta seja encaminhada novamente, i.e., *loop*) e uma lista de vizinhos ($vetViz_i$ – linha 2 – para ser utilizado no cálculo dos próximos saltos). Esta última é atualizada quando o *sink* envia a consulta à rede via *flooding*, logo, não há custo extra para a descoberta de vizinhos. O algoritmo RAVR inicia, então, quando o *sink* entra no campo de sensores e envia uma consulta à rede (linhas 4-10). Esta consulta contém a origem da consulta (*sink* – $orig_i$), um identificador incrementado a cada nova consulta (id_i), um comando de consulta a ser avaliado por todos os sensores (cmd_i), a trajetória (T_i) e a velocidade do *sink* (S_i), a posição do último nó que enviou o pacote (Up_i, necessário para atualizar a lista de vizinhos), e por último, o Tempo de Roteamento até o momento (R_i) – definição 6.

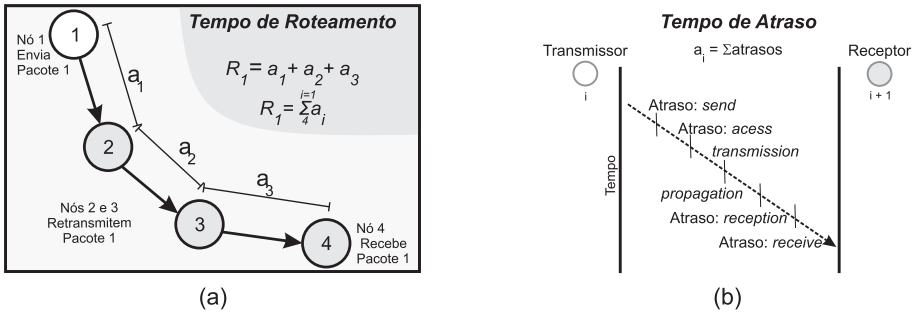

Figura 2. (a) Tempo de roteamento de um pacote; e (b) Tempo de atraso do salto.

Definição 6 (Tempo de Roteamento do Pacote – R_p**)** *Para que se possa calcular a posição mais atual do sink é necessário conhecer o seu tempo de deslocamento (i.e., o tempo* t *da definição 5). Uma solução seria considerar que os nós sensores estão sincronizados com o relógio do sink móvel, o que é uma suposição não realista devido a problemas de offsets e drifts [Maroti et al. 2004, Ping 2003, Oliveira et al. 2009]. Desta forma, para calcular* t*, o algoritmo RAVR calcula o tempo total que o pacote levou para deixar o sink e chegar ao nó atual (tempo de roteamento). Esse cálculo é possível somando-se todos os atrasos e todos os tempos de processamento dos nós intermediários (i.e., que repassaram o pacote) até o momento que este atingiu o nó atual, conforme ilustrado na figura 2(a). O cálculo do atraso de um pacote é realizado através da técnica Delay Measurement [Maroti et al. 2004]. Essa técnica é utilizada em diversos algoritmos de sincronização de nós [Ping 2003, Oliveira et al. 2009] e basicamente consiste em calcular e somar todos os atrasos individuais no envio de um pacote entre um transmissor e um receptor, conforme ilustrado na figura 2(b).*

Um nó sensor, ao receber um pacote de consulta (do *sink* ou de outro nó), irá calcular o atraso a_i desse pacote (linha 11) e atualizar o tempo de roteamento do pacote (R_i – linha 12), conforme explicado na definição 6. Irá também atualizar a sua tabela de vizinhos (linhas 13-15). Se este nó nunca encaminhou este pacote antes, ele irá encaminhá-lo e dar continuidade ao *flooding* (linhas 16-20). O nó, então, irá avaliar a consulta do *sink* e verificar se ele tem dados de resposta (linha 21). Se não tiver nada, o algoritmo acaba. Mas caso o nó tenha uma resposta a ser enviada ao *sink*, ele monta um pacote de resposta contendo sua identificação ($orig_i$), a identificação da consulta (id_k), os dados de resposta ($dados_i$), a trajetória e velocidade do *sink* (T_k e S_k) e, por último, o tempo de roteamento até o momento (R_i – linhas 21-29). Este pacote será enviado para o $proxSalto_i$ após $proxTempo_i$ segundos. Estes dois últimos valores são calculados de acordo com a variação do algoritmo RAVR, que pode ser o *RAVR SigaMe* (Seção 4.1), o *RAVR Intercepta* (Seção 4.2) ou o *RAVR MenorCaminho* (Seção 4.3).

Por último, caso o *sink* receba o pacote de resposta, ele armazena os dados (linhas 31-33). Caso um nó sensor receba o pacote de resposta, ele verifica se nunca encaminhou o pacote e, caso positivo, o encaminha usando uma das variações do RAVR (linhas 34-40), explicadas nas seções a seguir.

4.1. RAVR SigaMe

O *RAVR SigaMe* é a versão mais simples do algoritmo proposto. Basicamente, a cada salto, cada nó calcula a posição atual do *sink* ($sinkPos$) e, então, envia o pacote de

Algoritmo 1 Algoritmo geral do RAVR

▷ **Variáveis:**
1: $vetEnc_i \leftarrow \emptyset$; {Vetor de consultas e respostas encaminhadas – evitar *loops*}
2: $vetViz_i \leftarrow \emptyset$; {Vetor de vizinhos – armazena posição, id, etc}
3: $id_i \leftarrow 0$; {Identificação da última consulta enviada pelo *sink*}

▷ **Entrada:**
4: Nó *sink* 0 entra no campo de sensores e envia a $consulta\ consId_0$.
Ação:
5: $id_0 \leftarrow id_0 + 1$; {Id da nova consulta}
6: $cmd_0 \leftarrow$ '$SELECT * from\ RSSF\ where\ temperature >= 40$'; {Exemplo de consulta}
7: $orig_0 \leftarrow v_0$; $R_0 \leftarrow 0$; {Origem da consulta e Tempo de roteamento}
8: $T_0 \leftarrow trajetoriaSink()$; $S_0 \leftarrow velocidadeSink()$; {Trajetória e velocidade do *sink*}
9: $Up_0 \leftarrow (Xc_i, Yc_i)$; {Posição do último nó – que vai enviar}
10: Envia $cons(orig_0, id_0, cmd_0, T_0, S_0, R_0, Up_i)$ via $broadcast$. {Envia consulta para a rede}

▷ **Entrada:**
11: $msg_i = cons(orig_k, id_k, cmd_k, T_k, S_k, R_k, Up_k)$ tal que $a_i = delayMeasurement(msg_i)$.
Ação:
12: $R_i \leftarrow R_k + a_i$; {Atualiza o Tempo de roteamento}
13: **se** $k \neq 0$ **então** -{SE: o nó que mandou a consulta não for o *sink* . . .}-
14: $vetViz_i \leftarrow vetViz_i \cup (k, Lp_k)$; {Adiciona a posição do nó à lista de vizinhos}
15: **fim se**
16: **se** $(orig_k, id_k) \notin vetEnc_i$ **então** -{SE: este nó nunca encaminhou esta consulta ...}-
17: $vetEnc_i \leftarrow vetEnc_i \cup (orig_k, id_k)$; {Adiciona a consulta ao vetor de encaminhados}
18: $Up_i \leftarrow (Xc_i, Yc_i)$; {Posição do último nó}
19: Envia $cons(orig_k, id_k, cmd_k, T_k, S_k, R_i, Up_i)$ via $broadcast$. {Encaminha consulta}
20: **fim se**
21: **se** $avaliaConsulta(cmd_k)$ **então** -{SE: eu tiver dados da consulta realizada ...}-
22: $orig_i \leftarrow i$; {Origem da resposta}
23: $dados_i \leftarrow dadosResposta(cmd_k)$; {Dados a serem enviados ao *sink*}
24: $vetEnc_i \leftarrow vetEnc_i \cup (orig_k, id_k)$; {Adiciona a resposta ao vetor de encaminhados}
25: $P_i = distancia((Xc_i, Yc_i), T_k)/R_i$; {Velocidade de propagação da consulta}
26: $proxSalto_i \leftarrow calculaProxSalto()$; {Calcula o próximo salto da resposta}
27: $proxTempo_i \leftarrow calculaProxSalto()$; {Calcula o tempo a esperar para mandar a resposta}
28: Envia $resp(orig_i, id_k, dados_i, T_k, S_k, R_i, P_i)$ para $proxSalto_i$ em $proxTempo_i$ s; {Envia resp.}
29: **fim se**

▷ **Entrada:**
30: $msg_i = resp(orig_k, id_k, dados_k, T_k, S_k, R_k, P_k)$ tal que $a_i = delayMeasurement(msg_i)$.
Ação:
31: **se** $i = 0$ **então** -{SE: este nó for o *sink* ...}-
32: $armazena(dados_k)$; {*Sink* recebe a resposta e armazena os dados}
33: **senão**
34: **se** $(orig_k, id_k) \notin vetEnc_i$ **então** -{SE: este nó nunca encaminhou esta consulta ...}-
35: $vetEnc_i \leftarrow vetEnc_i \cup (orig_k, id_k)$; {Adiciona a resposta ao vetor de encaminhados}
36: $R_i \leftarrow R_k + a_i$; {Atualiza o Tempo de roteamento}
37: $proxSalto_i \leftarrow calculaProxSalto()$; {Calcula o próximo salto da resposta}
38: $proxTempo_i \leftarrow calculaProxTempo()$; {Calcula o tempo a esperar para mandar a resposta}
39: Envia $resp(orig_k, id_k, dados_k, T_k, S_k, R_i, P_k)$ para $proxSalto_i$ em $proxTempo_i$ s; {Resp.}
40: **fim se**
41: **fim se**

resposta para o nó vizinho mais próximo da posição atual do *sink* imediatamente. Logo,

$$\begin{aligned} sinkPos.X &= T_k.X + S_k \times cos(T_k.\theta) \times R_i; \\ sinkPos.Y &= T_k.Y + S_k \times sin(T_k.\theta) \times R_i; \quad sinkPos.Z = T_k.Z; \\ calculaProxSalto() &= vizMaisProx(sinkPos.X, sinkPos.Y, sinkPos.Z); \\ calculaProxTempo() &= 0.0; \end{aligned} \tag{1}$$

Conforme ilustrado na figura 3(a), a trajetória do pacote de resposta no *RAVR SigaMe* tende a ser uma curva, uma vez que a cada salto o *sink* está em uma posição diferente.

4.2. RAVR Intercepta

O *RAVR Intercepta* funciona de forma parecida ao *RAVR SigaMe*. A diferença é que o *RAVR Intercepta* calcula um ponto futuro no tempo em que o pacote poderá se interceptar

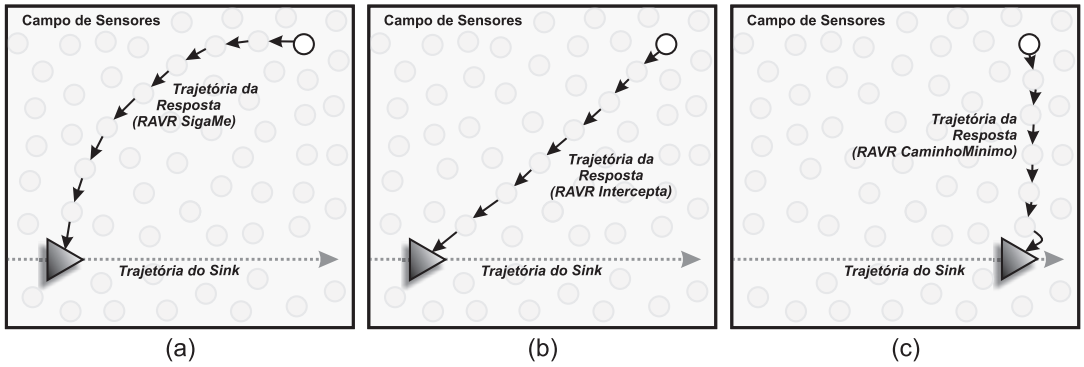

Figura 3. Variações do Algoritmo RAVR. (a) *RAVR SigaMe*; (b) *RAVR Intercepta*; e (c) *RAVR MenorCaminho*.

com o *sink*, ao invés de mandar o pacote na direção atual do *sink*. Tal cálculo é feito com base na velocidade do *sink* e na Velocidade de Propagação da Consulta.

Definição 7 (Velocidade de Propagação da Consulta – P_k) *Um nó sensor, ao receber uma consulta destinada a ele, calculará a velocidade dessa consulta, que é a distância entre o sink e o nó sensor, dividido pelo tempo de roteamento da consulta, i.e.,* $P_k = distancia((Xc_i, Yc_i), T_k)/R_i$.

Uma forma de calcular a posição de interceptação do pacote com o *sink* é calculando primeiro o tempo de interceptação:

$$
\begin{aligned}
tempoInterceptacao &= \sqrt{\frac{(T_k.Y - Yc_i)^2}{P_k^2 - (S_k \times sin(T_k.\theta))^2}} \qquad (2)\\
sinkPos.X &= T_k.X + S_k \times cos(T_k.\theta) \times (R_i + tempoInterceptacao);\\
sinkPos.Y &= T_k.Y + S_k \times sin(T_k.\theta) \times (R_i + tempoInterceptacao); \quad sinkPos.Z = T_k.Z;\\
calculaProxSalto() &= vizMaisProx(sinkPos.X, sinkPos.Y, sinkPos.Z);\\
calculaProxTempo() &= 0.0;
\end{aligned}
$$

A trajetória do pacote de resposta no *RAVR Intercepta* tende a ser uma reta, desde que todos os nós intermediários calculem o mesmo ponto de interceptação (figura 3(b)). Diversos fatores podem alterar este ponto de interceptação como, por exemplo, um atraso maior do pacote em uma região da rede. Uma vez que esse cálculo é feito a cada salto, o *RAVR Intercepta* recupera-se de mudanças na velocidade à medida que o pacote é propagado, calculando um novo ponto de interceptação quando isso ocorre. Entretanto, se a velocidade do *sink* for maior que a velocidade do pacote, o cálculo do tempo de interceptação ficará incorreto. Neste caso, o pacote é enviado usando o *RAVR SigaMe*.

4.3. RAVR MenorCaminho

A última variação do algoritmo RAVR é o *RAVR MenorCaminho*. A principal idéia deste algoritmo, ilustrada na figura 3(c), é mandar o pacote de resposta para o ponto da trajetória do *sink* que seja mais perto do nó sensor que estiver mandando ou encaminhando o pacote. Desta forma, o pacote de resposta tende a percorrer o menor caminho. Para fazer isso, o nó que estiver mandando o pacote terá que esperar até que o *sink* chegue perto da posição calculada para, então, mandar o pacote. Isso gera um atraso maior no envio da resposta, entretanto, o pacote de resposta será entregue em menos saltos.

$$
\begin{aligned}
dX = S_k \times cos(T_k.\theta) \times R_i; \quad dY = S_k \times sin(T_k.\theta) \times R_i; \\
tan = \frac{(Xc_i - T_k.X) \times dX + (Yc_i - T_k.Y) \times dY}{dX^2 + dY^2}; \\
sinkPos.X = T_k.X + tan \times dX; \\
sinkPos.Y = T_k.Y + tam \times dY; \\
sinkPos.Z = T_k.Z; \\
calculaProxSalto() = vizMaisProx(sinkPos.X, sinkPos.Y, sinkPos.Z); \\
tempoSink = distancia(sinkPos, T_k)/S_k; \\
tempoPacote = distancia((Xc_i, Yc_i), sinkPos)/P_k; \\
calculaProxTempo() = tempoSink - tempoPacote;
\end{aligned}
\tag{3}
$$

Conforme ilustrado na figura 3(c), a trajetória do pacote de resposta no *RAVR Menor-Caminho* tende a ser uma reta perpendicular à trajetória do *sink*, desde que todos os nós intermediários calculem o mesmo ponto de menor caminho. Se o tempo calculado for negativo, é porque o *sink* já passou pelo ponto de menor caminho. Nesse caso, o pacote é enviado imediatamente utilizando a técnica *RAVR Intercepta*.

5. Avaliação dos Algoritmos

Nesta seção, será avaliado e comparado o desempenho das três variações do Algoritmo RAVR: o *RAVR SigaMe*, *RAVR Intercepta* e o *RAVR MenorCaminho*.

5.1. Metodologia

A avaliação foi realizada utilizando o Network Simulator 2 [McCanne and Floyd 2005]. Em todos os resultados, as curvas representam uma média das execuções, enquanto que as barras de erro, o intervalo de confiança para 95% de confiança a partir de 33 execuções diferentes (sementes aleatórias).

A Tabela 1 apresenta valores padrões para os parâmetros de simulação. Os nós sensores são distribuídos no campo de monitoramento de acordo com uma grade perturbada, i.e., os nós tendem a ocupar a área uniformemente, mas sem formar uma grade regular. Para simular o erro de localização, a posição calculada dos nós é perturbada por uma distribuição Gaussiana com média igual ao valor da posição real do nó e desvio padrão de 2 m [Langendoen and Reijers 2003]. Erros na estimativa do atraso (*delay measurement*) são simulados com média igual ao atraso normal de um pacote e desvio padrão de 30 μs [Maroti et al. 2004]. A trajetória do *sink* considerada é uma linha reta, passando no meio do campo de sensores.

Parâmetro	*Valor*
Campo de sensores	272 m × 272 m
Numero de nós	576 nós
Densidade	0.01 nós/m^2
Raio de comunicação	20 m
Numero de vizinhos	7.3 nós
Atraso de um pacote	0.080 s
Erro de atraso	30 μs
Erro de posição	2 m
Velocidade do *Sink*	300 km/h
Trajetória do *Sink*	linha reta

Tabela 1. Valores

5.2. Impacto do Número de Nós Sensores

A escalabilidade do algoritmo foi avaliada aumentado-se o número de nós na rede de 144 a 2048 e mantendo-se uma densidade constante de 0.01 nós/m^2. Logo, o tamanho do campo de sensores aumenta de acordo com o número de nós. Primeiro, está sendo avaliada a capacidade do algoritmo de calcular a velocidade correta do pacote, uma vez que esse valor é utilizado tanto pelo *RAVR Intercepta* quanto pelo *RAVR MenorCaminho*. Como pode-se observar pelo gráfico da figura 4(a), as velocidades média calculada e perfeita de propagação da consulta são similares. Nesse e em outros gráficos similares, a curva de erro é multiplicada por 10 para facilitar a visualização, logo, no pior caso, o erro

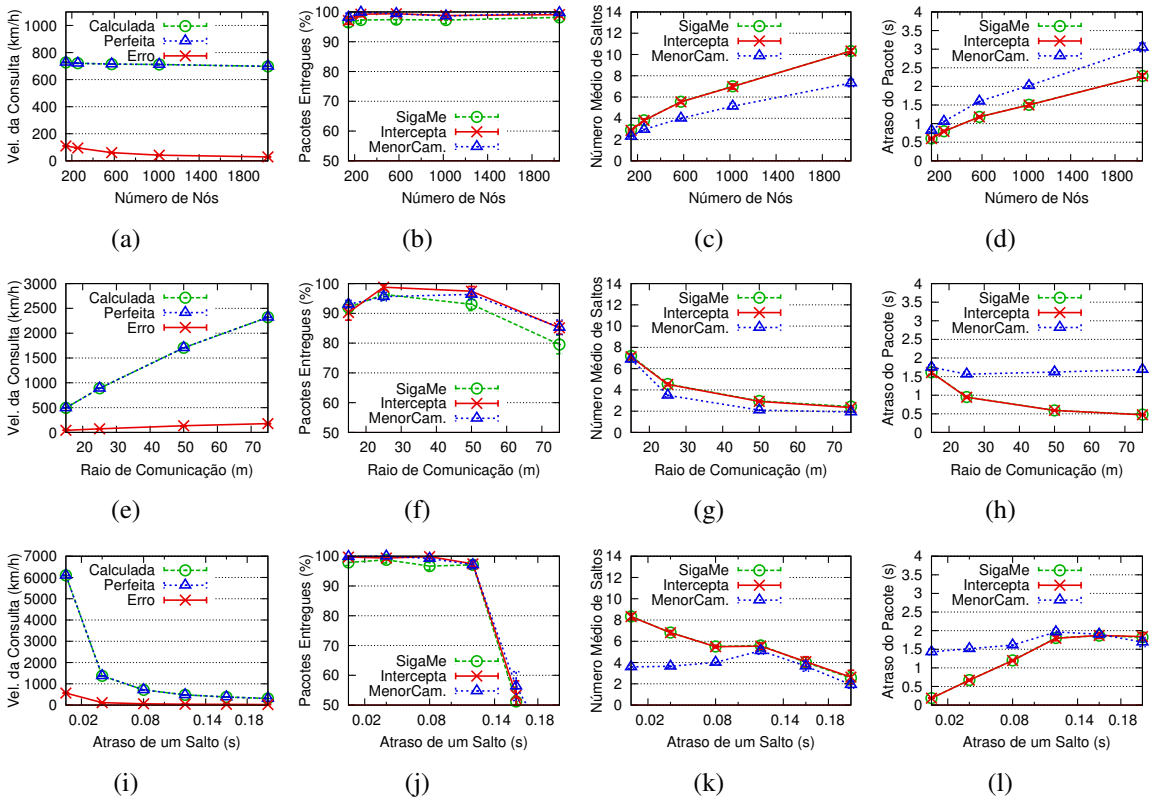

Figura 4. Resultados das simulações.

no cálculo da velocidade foi de apenas 10 km/h, o que diminui com o aumento do número de nós (e, conseqüentemente, aumento na quantidade de saltos).

A figura 4(b) mostra que todas as técnicas entregam mais de 95% dos pacotes mesmo com o aumento da rede, o que mostra a escalabilidade do RAVR. Pode-se notar ainda que o *RAVR SigaMe* tende a ser menos confiável. Isso se deve ao fato de que tanto o *RAVR Intercepta* quanto o *RAVR Menor Caminho* tendem a enviar o pacote para o *sink* no momento em que este encontra-se exatamente em cima do último nó, enquanto que no *RAVR SigaMe*, o *sink* pode estar quase deixando o raio de comunicação do último nó quando este for enviar a resposta, o que causa, algumas vezes, a perda do pacote.

Já na figura 4(c), observa-se uma característica já esperada do *RAVR MenorCaminho*: a sua quantidade menor de saltos ao enviar a resposta ao *sink*. Observa-se ainda que não há diferença significativa entre o *RAVR SigaMe* e o *RAVR Intercepta*. A principal explicação para estes resultados similares é que a curva observada pelo *RAVR SigaMe* não é suficiente para aumentar o número de saltos, ou seja, a curva existe, mas ainda assim a resposta atinge o *sink* com o mesmo número de saltos que o *RAVR Intercepta*.

Por último, a figura 4(d) mostra uma pequena desvantagem do *RAVR MenorCaminho*: um maior atraso para receber o pacote de resposta. Isso se deve ao fato dos nós esperarem até que o *sink* chegue perto deles para então para enviarem o pacote.

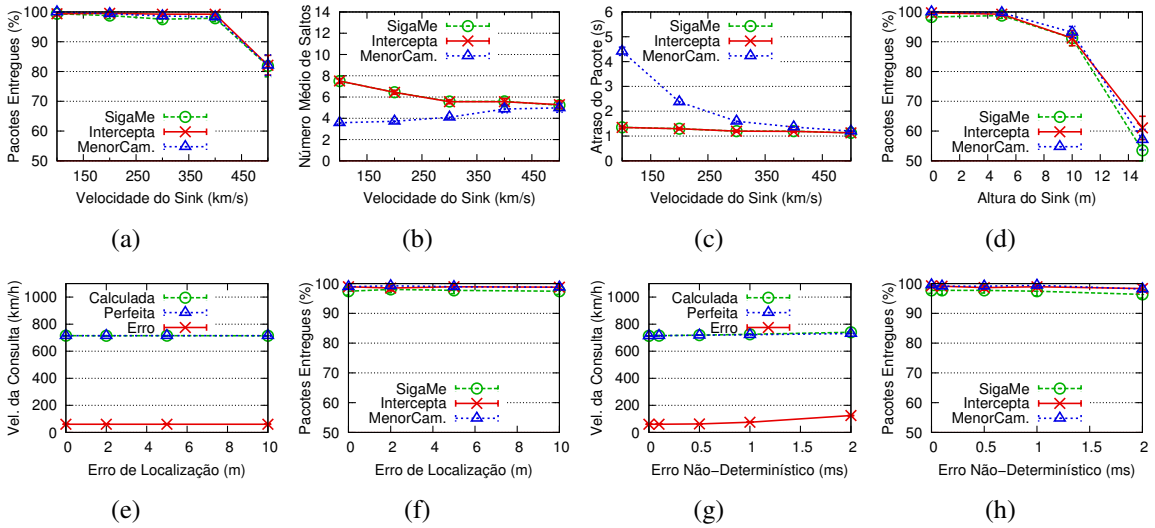

Figura 5. Resultados das simulações.

5.3. Impacto do Raio de Comunicação dos Nós

O impacto do raio de comunicação dos nós é avaliado variando-se este parâmetro de 15 m a 75 m. Em 15 m, a quantidade média de vizinhos de um nó é de 7.2, enquanto que em 75 m de raio de comunicação, a média de vizinhos é de 113 nós. A velocidade de propagação do pacote de consulta tende a crescer com o aumento do raio de comunicação, conforme observado na figura 4(e). Observa-se, no gráfico da figura 4(f), que após um raio de comunicação de 50 m, todas as variações do RAVR começam a perder pacotes, devido à alta quantidade de nós querendo acessar o meio. A figura 4(g) mostra que a quantidade média de saltos diminui com o aumento do raio de comunicação, o que já era esperado.

Um resultado interessante, ilustrado na figura 4(h), é que o atraso da resposta diminui com o aumento do raio de comunicação no *RAVR MenorCaminho* e *RAVR Intercepta*, mas permanece quase constante no RAVR MenorCaminho, uma vez que os nós ainda terão que esperar até o *sink* passar perto deles.

5.4. Impacto do Atraso Médio de um Salto

Para avaliar o impacto do atraso médio de um salto nos algoritmos RAVR, esse parâmetro foi variado de 0.005 s até 0.2 s. Conforme mostra a figura 4(i), esse atraso é o fator que mais afeta a velocidade de propagação da consulta. Entretanto, em todas as velocidades, o RAVR foi capaz de entregar mais de 95% dos pacotes (figura 4(j)), com exceção dos atrasos a partir de 0.12 s. Nesse caso, grande parte dos pacotes não alcançam o *sink* antes dele sair do campo de sensores, sendo descartados quando chegam à borda da rede.

Quanto ao número médio de saltos (figura 4(k)), tanto no *RAVR SigaMe* quanto no *RAVR Intercepta*, este tende a diminuir porque a consulta demora mais a chegar nos nós e, quando no momento da resposta, o *sink* encontra-se mais no meio da rede. Já no *RAVR MenorCaminho*, o número de saltos tende a aumentar porque, como o *sink* já estará passando do meio da rede no momento da resposta e, portanto passando do ponto de menor caminho para muitos nós, o *RAVR MenorCaminho* passará a se comportar mais

como o *RAVR Intercepta*, o que explica, também, o aumento no atraso total da resposta, conforme observado na figura 4(l).

5.5. Impacto da Velocidade e Altura do *Sink*

A velocidade do *sink* foi aumentada de 100 km/h até 500 km/h (aproximadamente duas vezes mais que a velocidade mínima de um Boeing-747). Percebe-se, pelo gráfico da figura 5(a) que os pacotes continuam sendo entregues independente da velocidade do *sink*. Após 400 km/h, este número cai porque o *sink* sai do campo de sensores antes que alguns nós tenham tempo de receber a consulta e enviar a resposta. O fato de alguns pacotes não poderem ser entregues, mesmo enviando-os na direção mais atual do *sink*, mostra a necessidade de algoritmos específicos para estes cenários em que o *sink* se move em alta velocidade. Na figura 5(b), percebe-se que o número de saltos médio tende a diminuir tanto no *RAVR SigaMe* quanto no *RAVR Intercepta*, enquanto que no *RAVR MenorCaminho*, o número de saltos tende a aumentar, pelos mesmos motivos que este comportamento foi observado na seção anterior. Por último, um resultado interessante pode ser observado na figura 5(c): o atraso da resposta diminuindo no *RAVR MenorCaminho*, uma vez que o tempo de espera pelo *sink* irá diminuir com o aumento de sua velocidade.

A avaliação do impacto da altura do *sink* foi realizada variando-se a altura de 5 m a 15 m. Como pode-se observar pela figura 5(d), com o aumento da altura, perdem-se mais pacotes. Em 15 m de altura, apenas cerca de 55% dos pacotes são entregues.

5.6. Impacto do Erro de Posição e do Cálculo de Atrasos

Como pode-se observar pelos gráficos das figuras 5(e) e 5(f), o erro de localização dos nós não afeta muito a velocidade de propagação da consulta e nem a entrega dos pacotes de resposta. Isso ocorre porque os erros de localização são pequenos se comparados às distâncias percorridas pelos pacotes, o que diminui a influência no cálculo da velocidade da consulta.

Para avaliar o impacto dos erros de cálculo de atrasos (erros da técnica *delay measurement*) no RAVR, este erro foi variado de 0 ms até 2 ms. Como pode-se observar pela figura 5(g), a velocidade de propagação da consulta é pouco afetado por este erro, uma vez que ele segue uma distribuição Gaussiana e, com o aumento do número de saltos, os erros tendem a se anular. Pelo mesmo motivo, a entrega dos pacotes também não é muito afetada, como pode-se observar pela figura 5(h).

6. Aplicabilidade da Solução Proposta

Este trabalho considera um atraso de 0.080 s em cada salto, considerado um atraso alto. Testes preliminares com sensores SunSPOTs indicam um atraso médio de apenas 0.015 s. Entretanto, um sensor SunSPOT é dotado de um processador relativamente rápido e esse atraso é sem a realização de nenhum cálculo e com a monitoração do ambiente desabilitada. Por outro lado, testes com sensores MicaZ, da Crossbow, mostram que certas operações de ponto-flutuante podem demorar até segundos inteiros para executar. Logo, tempos de 0.080 s ou mais em cada salto podem ser facilmente alcançados em RSSFs.

A comunicação em alta velocidade é outro ponto importante. Este trabalho mostra que é possível enviar dados a um nó se movendo, por exemplo, a 600 km/h sem influência na taxa de perda de dados. Trabalhos recentes [Mergen et al. 2006, Khalid et al. 2007]

indicam essa possibilidade também, mas em velocidades bem menores. Entretanto, isso é um problema a ser tratado em algoritmos de acesso ao meio, e não de roteamento, como é o caso deste trabalho.

Por último, um ponto importante a ser observado é que os algoritmos RAVR são basicamente algoritmos gulosos de encaminhamento (*Greedy Forward*) e, como tal, ainda precisam ser combinados com um algoritmo de contorno de buracos para poderem funcionar nos casos em que os sensores da RSSF não estão uniformemente distribuídos na rede. Entretanto, tais algoritmos usam uma posição dinâmica do nó *sink*, o que gera uma necessidade para novos algoritmos de contorno de buraco para tais cenários.

7. Conclusão

Este trabalho propôs uma nova classe de algoritmo de consulta a dados em RSSFs que utilizam *sinks* em alta velocidade: RAVR – Roteamento em Alta Velocidade em RSSFs. Conforme mencionado, nesses cenários, consultas propagadas na rede não podem ser respondidas usando a posição do *sink* no momento da consulta. Foram propostos três variantes para o RAVR em que, a cada salto, o pacote de resposta é enviado na direção da posição atual do *sink* (*RAVR SigaMe*), na direção de interceptação do *sink* (*RAVR Intercepta*) ou na direção do ponto mais próximo da trajetória do *sink* (*RAVR MenorCaminho*). Uma série de experimentos foram apresentados que mostram a real necessidade de novos algoritmos para tais cenários, bem como indicam que o algoritmo proposto e suas variações são capazes de funcionar em diversos cenários e condições.

O algoritmo *RAVR MenorCaminho* obteve o melhor desempenho em grande parte dos experimentos realizados, mesmo com o atraso maior na entrega das respostas ao *sink*. Esse atraso é pequeno, pois não afeta cenários em que o *sink* sobrevoa a rede, coleta os dados, e depois volta para a base, entregando os dados para a central de monitoramento.

Os resultados obtidos são promissores. Algumas vantagens e limitações serão exploradas em trabalhos futuros como, por exemplo, combinar a solução proposta com um algoritmo para contornar buracos na rede e com algoritmos de rastreamento capazes de rastrear trajetórias descritas por modelos não lineares sujeitos a ruído não Gaussiano.

Referências

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cyirci, E. (2002). Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422.

Boukerche, A., Oliveira, H. A. B. F., Nakamura, E. F., and Loureiro, A. A. F. (2008). A novel location-free greedy forward algorithm for wireless sensor networks. In *ICC'08: IEEE International Conference on Communications*, pages 2096–2101, Beijing, China.

Estrin, D., Girod, L., Pottie, G., and Srivastava, M. (2001). Instrumenting the world with wireless sensor networks. In *ICASSP'01: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 2033–2036, Salt Lake City, Utah.

Fodor, K. and Vidács, A. (2007). Efficient routing to mobile sinks in wireless sensor networks. In *WICON '07: Proceedings of the 3rd international conference on Wireless internet*, pages 1–7, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and TelecommunicationsEngineering).

Hong, W. and Madden, S. (2004). Implementation and research issues in query processing for wireless sensor networks. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 876, Washington, DC, USA. IEEE Computer Society.

Ilyas, M. and Mahgoub, I. (2004). *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter 20. CRC Press LLC.

Intanagonwiwat, C., Govindan, R., and Estrin, D. (2000). Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCom'00: 6th ACM International Conference on Mobile Computingand Networking*, pages 56–67, Boston, MA, USA. ACM Press.

Khalid, Z., Ahmed, G., and Khan, N. M. (2007). Impact of mobile speed on the performance of wireless sensor networks. *Journal of Information & Communication Technologies*, 1:7.

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.

Maroti, M., Kusy, B., Simon, G., and Ledeczi, A. (2004). The flooding time synchronization protocol. In *SenSys'04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 39–49, Baltimore, MD, USA. ACM Press.

McCanne, S. and Floyd, S. (2005). ns network simulator. [Online] Available: http://www.isi.edu/nsnam/ns/.

Mergen, G., Zhao, Q., and Tong, L. (2006). Sensor networks with mobile access: Energy and capacity considerations. *IEEE Transactions on Communications*, 54(10):1896–1896.

Oliveira, H. A. B. F., Boukerche, A., Nakamura, E. F., and Loureiro, A. A. (2009). Localization in time and space for wireless sensor networks: An efficient and lightweight algorithm. *Performance Evaluation*, 66(3-5):209–222.

Ping, S. (2003). Delay measurement time synchronization for wireless sensor networks. Technical Report IRB-TR-03-013, Intel Research.

Pottie, G. J. and Kaiser, W. J. (2000). Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58.

Shim, G. and Park, D. (2006). Locators of mobile sinks forwireless sensor networks. In *ICPPW '06: Proceedings of the 2006 International Conference Workshops on Parallel Processing*, pages 159–164, Washington, DC, USA. IEEE Computer Society.

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: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 148–159, New York, NY, USA. ACM.