ROTEAMENTO BASEADO EM CENTRALIDADE

PARA

REDES DE SENSORES SEM FIO

EDUARDO MUCELLI REZENDE OLIVEIRA

ROTEAMENTO BASEADO EM CENTRALIDADE PARA REDES DE SENSORES SEM FIO

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

Orientador: Antônio Alfredo Ferreira Loureiro
Co-orientador: Heitor Soares Ramos Filho

Belo Horizonte
28 de julho de 2011

Rezende Oliveira, Eduardo Mucelli
Roteamento Baseado em Centralidade para Redes de Sensores sem Fio / Eduardo Mucelli Rezende Oliveira. — Belo Horizonte, 2011
xx, 59 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Antônio Alfredo Ferreira Loureiro
Co-orientador: Heitor Soares Ramos Filho

1. redes de sensores sem fio. 2. medidas de centralidade. 3. roteamento. I. Título.

Agradecimentos

Só tenho a agradecer.

> All that I am, or hope to be, I owe to my angel mother
>
> Abraham Lincoln

Agradeço a minha mãe por tudo e por ser a pessoa mais importante na minha vida.

> A happy family is but an earlier heaven.
>
> George Bernard Shaw

À minha família por ter dado o apoio fundamental e necessário para que hoje eu possa concluir este trabalho. Agradeço a minha irmã pelos momentos divertidos que passamos juntos, valeu, Isabele *"Bibilz"* Mucelli R. de Oliveira. Ao meu pai, por sempre me motivar com sua sabedoria e jovialidade, valeu, José *"Gordon"* de Oliveira.

> Truly great friends are hard to find, difficult to leave, and impossible to forget.
>
> G. Randolf

Laboratório 3027, minha segunda casa, ou as vezes até a primeira, lugar de grandes amigos, de muitos momentos memoráveis. Antônio *"Loureiro"* Alfredo Ferreira Loureiro, Daniel *"Guidoni"* Guidoni, Emanuel *"Mano"* Vianna, Felipe *"Big Head"* Domingos Cunha, Fernando Afonso Santos, Guilherme *"Guigui"* Maia, Heitor *"Hector"*

Soares Ramos Filho, Leandro *"Joselito"* Villas, Marcelo G. Almiron, Pedro *"Pedrão"* Olmo, Rafael *"Colares"* Colares, Rafael *"Odon"* Odon de Alencar, Rafael *"Santin"* Santin, Thiago *"Pedpano"* Henrique Silva.

Agradeço também aqueles que não estavam no dia-a-dia do laboratório, mas que estavam comigo. Andre *"Jesus"* Luiz Durão, Dinaldo, Emerson *"Mersão"* Melo Almeida, Gilberto Melo Almeida, Huerlie, Mário Henrique, Max *"Nihilista"* Mendel.

Agredeço aquilo que vivi e aprendi e, acredito que o maior resultado deste curso não são as páginas que virão a seguir com fórmulas e gráficos, mas ser uma pessoa melhor por ter compartilhado um tempo de vida com vocês.

"There is no substitute for hard work."
(Thomas Edison)

Resumo

O uso de características topológicas, mais especificamente, a importância de um elemento relacionado com sua posição é um assunto amplamente estudado. Por sua vez, a teoria das redes complexas provê algumas métricas de centralidade de uso geral que foram aplicadas a uma imensa variedade de campos do conhecimento. Este trabalho tem o objetivo de usar a informação de centralidade no projeto de algoritmos de roteamento para Redes de Sensores sem Fio (RSSFs). Neste contexto, foram propostas duas novas métricas de centralidade, o *Sink Stress* e o *Sink Betweenness*, algoritmos distribuídos para o cálculo de ambas e algoritmos de roteamento em árvore que tiram proveito das mesmas. Simulações comparando a abordagem proposta com alguns dos algoritmos de fusão de dados mais eficientes da literatura mostram que os algoritmos propostos geram árvores de qualidade compatível, utilizando-se de um número significativamente menor de mensagens na construção da árvore.

Abstract

The use of topological features, more specifically, the importance of an element related to its structural position is a subject widely studied. For instance, complex networks theory provides some general use centrality metrics that have been applied in a large variety of knowledge fields. This work aims to use centrality information in the design of routing algorithms for Wireless Sensor Networks (WSN). In this context, we propose two new topological metrics, the Sink Stress, and Sink Betweenness, distributed algorithms to calculate them, and devise tree-based routing algorithms that take advantage of them. Simulations comparing the proposed approach with some of the most efficient data fusion algorithms from the literature, show that the proposed tree algorithms generate consistent quality trees, using a significantly smaller number of messages in the construction of the tree.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

Toda grande caminhada começa pelo primeiro passo.

Mao Tsé-Tung

Capítulo 1

Introdução

1.1 Motivação

Desde o começo da década passada, as Redes de Sensores sem Fio (RSSFs) [Akyildiz et al., 2002; Pottie and Kaiser, 2000; Yick et al., 2008] têm recebido bastante atenção da comunidade científica devido à sua capacidade de realizar monitoramento de ambientes e coleta de diversos tipos de informação [Loureiro et al., 2003]. A principal característica que distingue o projeto de RSSFs das demais redes é que os nós sensores possuem recursos bastante restritos, tais como capacidade limitada de memória, de largura de banda [Chong and Kumar, 2003], computacional e de energia [Marin et al., 2005].

O nó sensor é tipicamente pequeno, o que tende a limitar a sua capacidade de recursos, se comparado a um sensor tradicional usado apenas para coleta de dados [Yick et al., 2008]. Além disso, a sua energia é provida por uma bateria integrada e, em muitas aplicações, esse nó é depositado em áreas remotas, dificultando ou mesmo impossibilitando o acesso a esse elemento para sua manutenção [Loureiro et al., 2003]. Nesse cenário, o tempo de vida da rede depende da quantidade de energia disponível nos nós sensores e, por isso, esses nós devem balancear seus recursos limitados com o objetivo de aumentar o tempo de vida da rede. Portanto, a conservação de energia é um dos aspectos mais importantes a serem considerados no projeto das RSSFs [Anastasi et al., 2009].

Numa RSSF, os nós sensores coletam os dados de interesse, processam localmente ou coordenadamente entre os nós vizinhos e enviam essa informação resultante para um nó especial, chamado de estação base (*sink*). O projeto do algoritmo distribuído para estabelecer e manter rotas numa RSSF depende de vários fatores, dentre eles, a

frequência com que os dados devem ser enviados para a estação base, qualidade de serviço esperada nessa comunicação (e.g., rotas com menor latência ou menor consumo de energia) e outras funções que podem ser executadas conjuntamente com o roteamento como a agregação de dados.

Geralmente, algoritmos de roteamento para RSSFs têm o objetivo de levar os dados por rotas que minimizam o número de retransmissões e, potencialmente, o gasto de energia da rede. Para tanto, a estrutura de roteamento precisa ser concebida de tal forma que todos os nós fonte, e.g., nós que detectaram um evento, possam escoar suas leituras até o *sink* da maneira mais eficiente, ou seja, requisitando o menor número de retransmissões possível. Além disto, ao rotear pacotes, o algoritmo pode utilizar técnicas como agregação de dados [Nakamura et al., 2009], que explora a correlação dos dados e realiza a agregação em nós intermediários reduzindo o número de mensagens trocadas na rede.

A agregação de dados visa combinar informações de fontes diferentes e, através de uma estrutura de roteamento, geralmente em árvore [Nakamura et al., 2009], esta informação combinada é enviada em direção ao *sink*. A fusão das informações contidas nos pacotes é feita por certos nós conhecidos como agregadores, presentes em pontos de junção de duas, ou mais rotas em direção ao *sink*. Desta forma, quanto mais rotas passam por um nó, maior sua importância sobre a agregação e transmissão dos dados.

A importância de um nó na estrutura de um grafo pode ser mensurada através de métricas de centralidade, ou índices de centralidade desde a metade do século passado [Bavelas, 1948]. O objetivo destas métricas é classificar os vértices segundo sua importância na rede utilizando-se, por exemplo, a proximidade do mesmo em relação aos demais, ou a quantidade de caminhos que passam por ele. Vértices posicionados em áreas centrais costumam ter maior importância estrutural do que aqueles localizados nas margens da rede, e, nos casos em que há fluxo de dados, nós centrais são os difusores naturais de informação. De maneira geral, a importância do vértice cresce diretamente proporcional à sua participação nos caminhos [da F. Costa et al., 2007]. Consequentemente, a importância de um elemento computacional em uma rede pode ser calculada com base na sua centralidade topológica.

As métricas de centralidade têm sido amplamente estudadas, mas uma vez que uma RSSF é um ambiente computacionalmente restrito e o cálculo de diversas métricas de centralidade é impraticável para redes muito grandes [Brandes and Pich, 2007], poucas propostas, para este contexto, são encontradas na literatura. Portanto, é importante que se tenha uma forma de calcular a centralidade dos nós de maneira pouco custosa e distribuída, devido às restrições computacionais e o caráter distribuído das RSSFs. Neste trabalho, são propostas duas novas métricas de centralidade para o con-

texto das RSSFs e algoritmos distribuídos para o cálculo das mesmas. Além disto, foram propostos dois algoritmos de roteamento em árvore que fazem proveito destas métricas de centralidade a fim de aumentar a ocorrência de sobreposição de rotas e, consequentemente, melhorar a fusão de dados.

1.2 Objetivos

O objetivo deste trabalho é investigar a centralidade dos nós como elemento base no projeto de algoritmos de roteamento em árvore para RSSFs. Neste contexto, é necessário a criação de métricas de centralidade que levem em consideração características das RSSFs e, além disto, algoritmos distribuídos para o cálculo destas métricas. Por fim, criar algoritmos de roteamento que selecionem os nós retransmissores segundo sua centralidade de maneira a maximizar a sobreposição de rotas e, consequentemente, maximizar a fusão de dados.

Desta forma, os objetivos específicos deste trabalho são:

1. Propor métricas de centralidade compatíveis com as RSSFs;
2. Propor algoritmos distribuídos para o cálculo destas métricas de centralidade;
3. Propor algoritmos de roteamento em árvore que façam uso dos valores de centralidade com o intuito de maximizar a fusão de dados.

1.3 Organização do Trabalho

O presente trabalho está organizado da seguinte forma: o capítulo 2 apresenta os trabalhos relacionados. Neste capítulo, são discutidas várias métricas de centralidade em diversas áreas do conhecimento e, além disto, são apresentadas propostas de algoritmos de roteamento em árvore para RSSFs. O capítulo 3 apresenta as métricas de centralidades propostas, os algoritmos distribuídos para o cálculo das mesmas e os algoritmos de roteamento em árvore. O capítulo 4 apresenta, inicialmente, uma discussão sobre a precisão dos algoritmos distribuídos elaborados para o cálculo das métricas de centralidade propostas e, posteriomente, os resultados de simulação dos algoritmos de roteamento. Por fim, o capítulo 5 apresenta as conclusões e o 6 indica algumas possíveis direções futuras.

Every new beginning comes from some other beginning's end.

Seneca

Capítulo 2

Trabalhos Relacionados

Este trabalho propõe o uso do conceito de centralidade em RSSFs e, para tal, são propostas duas métricas de centralidade, formas de calculá-las de maneira distribuída e algoritmos de roteamento em árvore que fazem uso das mesmas com intuito de maximizar a fusão de dados.

Este capítulo discute algumas propostas encontradas na literatura que abordam os principais conceitos deste trabalho, centralidade e algoritmos de roteamento em RSSFs. A seção 2.1 apresenta os desafios de se aplicar métricas de centralidade num contexto de RSSFs, métricas de centralidade *clássicas*, métricas propostas para contextos específicos e suas aplicações em diversas áreas do conhecimento. Posteriormente, na seção 2.2 serão abordados os desafios e diversos trabalhos no campo dos algoritmos de roteamento em RSSFs.

2.1 Centralidade

2.1.1 Desafios

Apesar dos diversos contextos nos quais o conceito centralidade foi amplamente estudado Koschützki et al. [2005], existem poucos trabalhos na literatura que abordam roteamento de dados em RSSFs, principalmente porque este é um ambiente computacionalmente restrito e, portanto, impraticável para o cálculo de muitas métricas clássicas de centralidade. O custo para computar, de maneira centralizada, uma métrica de centralidade clássica conhecida como *Betweeness* [Anthonisse, 1971; Freeman, 1977, 1979] é $O(nm + n^2 \log n)$ para grafos ponderados e $O(nm)$ para grafos não ponderados [Brandes, 2001], onde n é o número de nós e m é o número de arestas. Observe que

esta computação requer o cálculo de todos os caminhos mínimos que passam pelos vértices. O custo computacional para calcular algumas métricas de centralidade em redes com milhares de nós pode ser impraticável [Brandes and Pich, 2007]. Sendo possível adequar métricas para o contexto das RSSFs e calcular a centralidade de cada nó da rede de maneira distribuída, pode-se aproveitar tal característica em funções da rede como, por exemplo, no roteamento. A próxima seção apresenta diversas métricas de centralidade e suas aplicações nos mais variados contextos.

2.1.2 Métricas

Na teoria dos grafos, a idéia de classificar um vértice por sua importância foi introduzida por [Bavelas, 1948] e [Leavitt, 1951], que resultou na proposição do primeiro índice de centralidade para redes conectadas, o índice de Bavela.

Medidas de centralidade [Koschützki et al., 2005; da F. Costa et al., 2007] têm o intuito de estimar a importância de um dado vértice [Yiwei et al., 2006], ou seja, ranqueá-lo segundo sua importância topológica [Wasserman and Faust, 1994]. Nós em posições centrais geralmente têm grande importância estrutural e, em casos onde há fluxo de dados, são agentes de escoamento vitais. Geralmente, quanto maior for a participação de um vértice ou de uma aresta em caminhos no grafo, maior será sua importância [da F. Costa et al., 2007]. Desta forma, pode-se saber o quão importante é, por exemplo, um elemento computacional para uma rede, ou uma pessoa em uma rede social, inferindo-se esta importância a partir da posição desta entidade frente às demais.

Existem diversas métricas *clássicas* de centralidade baseadas em diferentes características do grafo considerando, tanto aspectos simples como o grau do nó, quanto conceitos mais elaborados como a relação mútua de importância entre os nós vizinhos. A métrica de centralidade mais simples é o *Degree Centrality* [Koschützki et al., 2005], sendo, para grafos não direcionados $C_D(v) = g(v)$ onde v é um nó qualquer do grafo e g é seu grau. No caso de grafos direcionados, duas variações são adequadas: *In-Degree Centrality*, $C_{In-Degree}(v) = g^-(v)$, que corresponde ao grau de entrada do nó v e o *Out-Degree Centrality*, $C_{Out-Degree}(v) = g^+(v)$, que corresponde ao grau de saída do nó v. Em [Hage and Harary, 1995] é proposta a métrica *Eccentricity Centrality* definida, para um nó v qualquer, como $C_{Eccentricity}(v) = \frac{1}{max\{d(u,v):v \in V\}}$, ou seja, v será o nó mais central se possuir a menor distância máxima entre qualquer vértice.

Conceitos de distância entre nós também são amplamente exploradas pelas métricas de centralidade. Possivelmente, a métrica mais conhecida que faz uso da distância é o *Closeness* [Beauchamp, 1965; Sabidussi, 1966]. Esta métrica avalia o somatório

das distâncias de um nó qualquer até os demais e, atribui o inverso deste valor aos nós, desta forma, o nó cujo somatório for o menor, será o mais central, ou o nó que está mais *perto* dos demais. Outras métricas menos difundidas que também fazem uso da centralidade são *Information Centrality* [Stephenson and Zelen, 1989] e o *Centroid* [Koschützki et al., 2005].

Além disto, outra característica base usada pelas métricas de centralidade é o caminho mínimo, utilizado, por exemplo, pelo *Stress Centrality* [Shimbel, 1953], *Shortest-path Betweeness Centrality* que, neste trabalho será chamado apenas de *Betweenness* e *Reach* [Gutman, 2004; Goldberg et al., 2006]. O *Stress* de um nó v qualquer pode ser definido como:

$$\mathrm{C}_{\mathrm{Stress}}(v) = \sum_{a \neq v \in V} \sum_{t \neq v \in V} \sigma_{at}(v) \tag{2.1}$$

onde a e t são todos os pares de nós distintos do grafo e $\sigma_{at}(v)$ é o número de caminhos mínimos que partem de a até t que passam pelo vértice v. Já o *Betweenness* de um nó v qualquer pode ser definido como se segue:

$$\mathrm{C}_{\mathrm{Betweenness}}(v) = \sum_{a \neq v \in V} \sum_{t \neq v \in V} \frac{\sigma_{at}(v)}{\sigma_{at}} \tag{2.2}$$

onde a e t são todos os pares de nós distintos do grafo, σ_{at} é o número de caminhos mínimos que partem de a até t e $\sigma_{at}(v)$ é o número de caminhos mínimos que partem de a até t, mas que passam pelo vértice v. Em [Freeman, 1977] esta métrica foi proposta como forma de se contornar o problema que existe ao se aplicar o *Closeness* em grafos desconexos. A distância entre dois vértices não conexos é, usualmente, definida como infinita e, uma vez que o *Closeness* soma o inverso destas distâncias, nenhuma informação útil poderia ser extraída do cálculo. Já o *Betweenness* não possui tal problema, pois a inexistência de um caminho entre dois nós adicionará zero na parcela do somatório específica referente aquele nó. *Stress* e *Betweenness* são métricas similares, mas o primeiro usa a contagem absoluta dos caminhos mínimos enquanto o segundo usa a soma relativa do número de caminhos para todos os pares de vértices. Uma forma de se interpretar o *Betweenness*, é que esta métrica mensura o controle de um nó sobre o fluxo de informação entre os demais nós.

Há ainda a utilização da importância dos vizinhos para o cálculo da importância própria, como é o caso da métrica *Eigenvector* [Bonacich, 1972], ou ainda as métricas correlatas de classificação de páginas Web *Hub Score*, *Authority* [Kleinberg, 1999] e *PageRank* [Brin, 1998].

Devido à versatilidade das métricas de centralidade, estas são aplicadas em diver-

sas áreas do conhecimento. Na biologia, as métricas de centralidade foram aplicadas em redes biomoleculares [Mason and Verwoerd, 2007], cadeias alimentares [Jordan et al., 2007; Jordan, 2009] sendo que, como resultado deste último, foi proposta a métrica R de centralidade específica para tal contexto. Áreas como urbanismo e sociologia também estudam e aplicam métricas de centralidade. Por exemplo, Crucitti et al. [2006] fazem um estudo sobre a distribuição da centralidade em grafos que representam a malha viária de diversas cidades no mundo. Nesse trabalho são descritas possíveis aplicações e a influência de estudos da centralidade sobre ambientes urbanos como, por exemplo, no planejamento do fluxo de carros, ou pessoas [Porta et al., 2008]. Em [Fu and Chen, 2008], é feito um estudo de caso sobre o ativismo virtual pela independência do Tibet através de métricas de centralidade

A partir dos trabalhos relatados na literatura, ficou evidente que nem toda métrica de centralidade é apropriada para todo contexto. Com o tempo, variações foram sendo propostas, sendo a maioria baseada nas métricas descritas anteriormente. Por ser uma métrica amplamente estudada, diversas variações do *Betweenness* foram propostas na literatura. Por exemplo, o *Flow Betweenness Centrality* (FBC) [Freeman et al., 1991] leva em consideração todos os caminhos, entre fonte e destino, pelos quais as informações trafegam quando um fluxo máximo é inserido na rede. Tanto a métrica original quanto o FBC fazem suposições de otimalidade, ou seja, para a primeira, apenas caminhos geodésicos são computados, enquanto na segunda, apenas caminhos referentes ao fluxo máximo. Como resultado, Newman [2005] propôs uma versão do *Betweenness* que considera todos os caminhos entre um par de nós, uma vez que esta é uma suposição razoável para ambientes cuja troca de dados nem sempre está restrita aos caminhos mínimos. Essa métrica indica, em essência, o quão frequente um vértice intermediário i será visitado quando se caminha de um vértice j até outro k. Freeman [1977] propôs a métrica *Central Point Dominance* que faz uso do *Betweenness* máximo e aplica a normalização do valor de importância dos vértices entre 0, para vértices de um grafo completo e 1, para o vértice central de um grafo estrela. Krause et al. [2006] propuseram o *Cumulative Betweeness* que atribui ao nó i uma importância baseada nos valores do *Betweenness* local e dos nós que têm uma aresta incidente à i. Em [Wang et al., 2008], é proposta uma métrica de centralidade que calcula a importância de um nó através de seu valor de *Betweenness*, grau próprio e de seus vizinhos. Nesse trabalho é feito um estudo de caso em uma rede sexual de relacionamentos da AIDS na qual, segundo os resultados apresentados, a métrica proposta foi mais discriminativa do que o *Closeness*, *Betweenness* e o grau do nó, individualmente.

Além de adaptações sobre o *Betweenness*, existem outros trabalhos que adaptam várias das demais métricas citadas. Geralmente, o foco está em descobrir o vértice

mais central que, quase sempre, é o elemento mais importante. Por outro lado, Coromina et al. [2008] propõem versões das métricas *Closeness*, *Degree Centrality*, *Density* e *Diversity* para redes com duas entidades igualmente centrais e que sejam mais centrais que as demais. Valente and Foreman [1998] propõem duas métricas baseadas no *Closeness*: *Integration*, que mede o quão bem conectado um membro da rede está e *Radiality*, que mensura o quanto as ligações de um nó expandem-se pela rede. Vértices com *integration* alto ficam, mais cedo, cientes de informações trafegadas pela rede pois, na média, eles estão mais próximos dos demais vértices, enquanto vértices com valor alto de *radiality* deveriam ser bons emissores de informação. van den Brink and Gilles [2000] propõem duas métricas para digrafos baseadas no *Closeness*, β-*measure* e *source-measure*, que também possuem variação aplicável para grafos não direcionados. Em [Dangalchev, 2006], é proposta uma métrica mais sensível para verificar a resistência a ataques de um rede, mesmo que a mesma seja desconexa. Em [Hage and Harary, 1995], é proposta uma variação do *Eccentricity* com a incorporação do caminho central do grafo no cálculo. Em [Kretschmer and Kretschmer, 2007], é proposta uma métrica similar ao *Degree Centrality* como uma forma de fazer inferências sobre a co-autoria e citações de artigos científicos.

2.1.3 Roteamento em redes ad-hoc

Apesar dos estudos de métricas de centralidade em diversos contextos, existem poucos trabalhos em roteamento para RSSFs que usam essas métricas, principalmente por causa das restrições computacionais que inviabilizam a computação de várias das métricas clássicas de centralidade.

Krause et al. [2006] propõem uma estrutura de roteamento para redes *ad-hoc* de vários saltos que visa maximizar a vazão fim-a-fim com a adição de enlaces na periferia da rede. O objetivo é modificar a estrutura da rede como forma de minimizar a quantidade de nós com centralidade muito superior aos demais pois, esses nós provavelmente serão pontos de congestionamento. Além disto, é proposta uma métrica baseada no *Betweenness* que é computada de maneira centralizada com o intuito de avaliar o ganho da modificação na estrutura da rede. Posteriormente, Yu et al. [2007] utilizaram a mesma métrica como forma de evitar o congestionamento em nós mais centrais, sem modificações estruturais na rede.

Uma abordagem semelhante ao trabalho aqui proposto é estudada em [Souihli et al., 2009]. Esse trabalho estabelece um mecanismo de balanceamento de carga que evita o tráfego em regiões centrais da rede que são identificadas pela métrica proposta ϵ (*Epsilon*). De acordo com essa métrica, uma rota será tão boa quanto maior for o

número de nós entre origem e destino, ou seja, rotas centrais, que geralmente possuem poucos nós, serão evitadas.

2.2 Algoritmos de roteamento em RSSFs

2.2.1 Desafios

O roteamento em RSSFs é tarefa fundamental e desafiadora, principalmente pela possibilidade de haver milhares a centenas de milhares de nós sensores e, consequentemente, a inviabilidade de se criar uma tabela de endereçamento global [Al-karaki and Kamal, 2004]. Geralmente, algoritmos de roteamento para RSSFs têm o objetivo de levar os dados por rotas que minimizam o número de retransmissões e, consequentemente, o gasto de energia da rede. Para tanto, a estrutura de roteamento precisa ser concebida de tal forma que todos os nós fonte, e.g., nós que detectaram um evento, possam escoar suas leituras até o *sink* da maneira mais eficiente, ou seja, requisitando o menor número de retransmissões possível. Além disto, ao rotear pacotes, o algoritmo pode utilizar técnicas como agregação de dados [Nakamura et al., 2009], que explora a correlação dos dados e realiza a agregação em nós intermediários reduzindo o número de mensagens trocadas na rede.

A construção da árvore de agregação ótima é um problema NP-Difícil [Al-Karaki et al., 2004] que é equivalente ao problema da Árvore de *Steiner* [Krishnamachari et al., 2002]. Este problema é amplamente conhecido e, formalmente, ele pode ser apresentado como se segue: seja uma rede representada como um grafo $G = (V, E)$, onde $V = \{v_1, v_2, v_3, ..., v_n\}$ é o conjunto de nós sensores e $E = \{e_1, e_2, e_3, ..., e_i\}$ representa o conjunto de arestas, com um peso associado, que fazem as conexões entre os nós de V. Como construir uma árvore de custo mínimo que conecte todos os nós fonte $F = \{f_1, f_2, f_3, ..., f_n\}$ e o *sink* sendo que $F \subseteq V$. O custo da árvore de *Steiner* resultante será o somatório do custo das arestas da árvore.

Na literatura, diversas heurísticas e algoritmos aproximados foram propostos para a resolução do problema da Árvore de *Steiner*. Em [Hougardy and Prömel, 1999] e [Robins and Zelikovsky, 2000] são propostos algoritmos com fator de aproximação pequeno, igual a 1.55. O principal problema destas soluções é que, se implementadas como algoritmo distribuído, as mesmas não serão adequadas a um contexto restrito em termos de energia, como é o caso das RSSFs, pois seria necessário um grande número de troca de mensagens para a construção da árvore.

Por outro lado, diversos protocolos adequados para as RSSFs foram propostos na literatura, e estes podem ainda ser classificados, segundo sua estrutura, em baseados

em *cluster* [Huang et al., 2009; Jia et al., 2008], baseados em árvore e não estruturados. Algoritmos baseados em árvore e em *cluster* são fundamentalmente distintos. Protocolo baseado em *cluster* cria grupos de nós e escolhe um líder para o mesmo, portanto, estes protocolos tem o potencial de agregar os dados dentro do *cluster*, sem a necessidade de transmitir dados redundantes até o *sink*. Algoritmos baseados em árvore procuram construir estruturas de roteamento em formato de árvore que possa escoar as leituras dos nós que sensorearam eventos até o *sink*. Desta forma, a principal possibilidade de agregação destes algoritmos está nos pontos de agregação, que são, geralmente, nós que tem mais de um descendente na árvore. É importante ressaltar que estas duas abordagens não são mutuamente exclusivas, ou seja, existem algoritmos que utilizam ambas as políticas. Algoritmos como, por exemplo, InFRA [Nakamura et al., 2009] e DAARP [Villas et al., 2009] criam *clusters* e procuram criar árvores que favoreçam a existência de pontos de agregação. As próximas três seções descrevem algumas propostas para cada uma das classificações descritas.

2.2.2 Algoritmos hierárquicos

De modo geral, os algoritmos hierárquicos, ou baseados em *cluster*, definem um papel especial para alguns nós, conhecidos como *cluster heads*. Esses nós serão encubidos de realizar tarefas como, por exemplo, envio e agregação das leituras de vários nós sensores para o *sink*

Em [Heinzelman et al., 2000] é proposto o *Low-Energy Adaptive Clustering Hierarchy* (LEACH), este protocolo define que os *cluster heads* farão a agregação dos pacotes e enviarão o resultado para o *sink* diretamente, utilizando alto poder de transmissão. Os *cluster-heads* são escolhidos aleatoriamente em cada rodada, pois, desta forma, o gasto de energia é distribuído entre todos os nós. Variações do LEACH foram propostas, em [Heinzelman et al., 2002] é proposto o LEACH-C, no qual o *sink* define quem serão os *cluster heads*, já em [Zhao et al., 2004] é proposto o protocolo OSE, o qual não utiliza o *sink* para decidir quem serão os *cluster heads* e, além disto, uma forma mais igualitária é utilizada para definir quem serão os *cluster heads* a cada rodada. De qualquer modo, ao assumir que o *sink* é alcançável utilizando-se apenas uma transmissão, o LEACH e suas variações, que mantêm esta mesma política, restringem muito os cenários em que eles podem ser aplicados, principalmente com relação ao tamanho do campo de sensoriamento. De qualquer modo, este é um problema facilmente contornável e, existem versões mais recentes deste protocolo como, por exemplo, o MR-LEACH [Farooq et al., 2010] que fazem comunicação *multi-hop* entre *clusters* com o intuito de economizar energia.

Em [Lindsey and Raghavendra, 2002] é proposto o *Power-Efficient GAthering in Sensor Information Systems* (PEGASIS), protocolo baseado no LEACH que organiza uma cadeia de nós em que, cada nó, ao receber um pacote, efetua a agregação da sua leitura ao pacote recebido e o retransmite. Apenas um nó aleatoriamente escolhido nesta cadeia será o responsável por enviar o pacote ao *sink*. Ao assumir que os nós têm conhecimento global da rede para construir a cadeia, este protocolo sofre um problema de escalabilidade em relação ao número de sensores.

Em [Culpepper et al., 2004] é proposto o protocolo *Hybrid Indirect Transmissions* (HIT), baseado no LEACH e no PEGASIS. Este protocolo usa a formação de *clusters* do primeiro e a possibilidade de comunicação entre *clusters* do segundo e, além disto, alia a possibilidade de comunicação entre os nós que são e não são *cluster heads*.

Em [Nakamura et al., 2009] é proposto o InFRA, este algoritmo organiza *clusters* utilizando os nós que detectaram eventos. Para cada evento, o *cluster head* escolhido será o nó mais próximo do *sink*, utilizando-se para isto o número de saltos. Este protocolo visa construir a árvore que maximize a fusão de dados e, para tanto, uma vez que os *clusters* são formados, os *cluster heads* escolhem os caminhos mais curtos até o *sink* que maximizem a agregação usando a distância agregada dos coordenadores. A desvantagem deste protocolo está na necessidade de se fazer um *flooding* a cada novo evento ocorrido no campo sensoriado, pois, desta forma, os nós que não detectaram o evento são informados e a distância agregada aos coordenadores é atualizada. Este procedimento pode limitar a escalabilidade do InFRA quanto ao número de eventos que ocorre na rede. Em [Villas et al., 2009], é proposto o DAARP, este protocolo que, assim como o InFRA, constrói *clusters* e elege seus líderes a cada evento. Na ocorrência do primeiro evento, o *cluster head*, responsável por este evento, envia os pacotes pelo menor caminho, em número de saltos, até o *sink*. Nos eventos subsequentes, rotas são criadas até o nó mais próximo contido na estrutura já existente, sendo este nó selecionado como ponto de agregação. De maneira geral, o DAARP utiliza menos transmissões que InFRA para realizar a construção da árvore e, além disso, possui rotas mais propícias para a agregação de dados.

2.2.3 Algoritmos não-estruturados

Em [Fan et al., 2006] é proposto um protocolo *anycast* para agregação de dados sem a necessidade de construção de uma estrutura de roteamento. A agregação leva em consideração aspectos temporais como, por exemplo, um nó pode esperar um intervalo de tempo aleatório para que outros pacotes cheguem e possam ser agregados. Além disto, este protocolo considera correlação espacial, os pacotes agregados devem estar,

no máximo, a um salto de distância do nó agregador. Por estas considerações, este protocolo não garante agregação de todos os pacotes e, portanto, num cenário em que a rede é grande, transmitir um pacote não agregado pode ser custoso.

2.2.4 Algoritmos baseados em árvore

O modo mais fácil de agregar os dados que trafegam, dos nós sensores ao *sink*, é selecionar alguns nós especiais, que funcionem como pontos de agregação, e definir a direção que os pacotes terão no momento da retransmissão. Nesta abordagem, inicialmente os protocolos constroem a estrutura em árvore e, posteriormente, a usam para trafegar os dados da coleta, ou para responder a consultas geradas pelo *sink*. A agregação acontece sempre que dois, ou mais pacotes, chegam a um mesmo nó da árvore, que posteriormente encaminhará apenas um pacote com os dados agregados.

Estes algoritmos são adequados para projeto de funções ótimas de agregação e são eficientes quanto ao consumo de energia. *Shortest Path Tree* (SPT) [Krishnamachari et al., 2002] é um dos mais simples algoritmos distribuídos para a construção de uma estrutura de árvore de roteamento. Quando um evento é detectado, os nós enviam as mensagens através de um caminho mais curto em direção ao *sink*. Oportunisticamente, o algoritmo pode mesclar as rotas mais curtas de cada origem, sempre que ocorre sobreposição das mesmas. A principal vantagem deste protocolo está na baixa quantidade de mensagens que precisam ser trocadas para a construção da árvore, uma transmissão por nó. Por outro lado, a qualidade das rotas tende a ser pequena. Outro algoritmo bem conhecido é o *Center at Nearest Source* (CNS) [Krishnamachari et al., 2002] em que o nó que detecta um evento, e está mais próximo do *sink*, será eleito como o agregador. Consequentemente, as rotas de cada fonte serão, a cada evento, reconstruídas em direção ao novo agregador, que é responsável por enviar os dados agregados para o *sink*. A principal desvantagem deste protocolo está na alta quantidade de mensagens trocadas para a construção da árvore, pois existe a necessidade de se fazer uma inundação na rede a cada novo evento.

Em [Intanagonwiwat et al., 2003] é proposto o *Direct Diffusion*, um dos primeiros protocolos a fazer o roteamento baseado em atributos, ao invés do uso dos endereços dos nós. Este protocolo agrega os dados de maneira oportunística quando eles se encontram em algum nó intermediário. Tendo como base o *Direct Diffusion*, em [Intanagonwiwat et al., 2002] é proposto *Greedy Incremental Tree* (GIT). Neste protocolo, a primeira rota é construída utilizando-se o caminho mais curto até o *sink* e, as demais rotas, são construídas conectando-se cada fonte de evento na estrutura já existente. Embora este trabalho defenda que a árvore construída de maneira gulosa promove a eficiência

em energia, este algoritmo mostra-se inadequado para RSSFs [Nakamura et al., 2006]. Em [Harris et al., 2007] é proposta uma função para avaliar a eficiência dos pontos de agregação da árvore. Ainda neste trabalho é proposto o *Oceanus*, algoritmo de agregação em árvore no qual o *sink* é responsável por definir quem serão os pontos de agregação maximizando a função de agregação proposta.

Para cenários em que os eventos não são necessariamente correlatos em uma região, é proposto o *MVSink* [Fernandes and Murphy, 2009]. Este protocolo visa minimizar o tamanho da árvore de agregação em número de arestas movimentando, de maneira incremental, um *sink* virtual a partir de sua localização inicial para um local próximo às fontes. Se necessário, o *sink* virtual pode ser dividido em várias partes, criando alguns *sinks* que continuam a caminhar mais profundamente na árvore de agregação, mais perto da fonte, deixando para trás um *sink* virtual para combinar os dados de outras árvores.

Outros fatores podem ser utilizados para a proposição de algoritmos de roteamento em árvore como, por exemplo, o uso de características topológicas. Em [Souihli et al., 2009], é proposta uma forma de trafegar dados na rede de maneira balanceada utilizando-se a centralidade dos nós como parâmetro de decisão. Da mesma forma, pretende-se para o trabalho aqui proposto o uso de uma característica topológica específica, a centralidade, no algoritmo de roteamento com o intuito de maximizar a fusão de dados nas RSSFs.

Ideas are the root of creation.

Ernest Dimnet

Capítulo 3

Proposta

Em cenários de RSSFs, a comunicação de dados é, tipicamente, feita entre os nós sensores e o nó *sink*, e vice-versa. Além disto, muitas aplicações em RSSFs são dirigidas por evento, i.e., a rede é mantida em modo de economia de energia até que um evento seja detectado. Tipicamente, durante esta fase, os nós fazem apenas funções básicas com um ciclo de trabalho muito baixo e não mantêm qualquer infraestrutura de rede. E, utilizando-se de uma abordagem reativa, assim que um evento é detectado, os nós constroem a estrutura de roteamento, geralmente uma árvore, para entregar o tráfego gerado por este eventos, e param de manter esta estrutura assim que os eventos se encerram. Esta é uma estrutura de roteamento desejável que consome pouca energia e entrega os dados gerados pelos eventos assim que os mesmos ocorrem, mas que pode implicar em maior latência devido a necessidade da criação desta estrutura.

Tendo em mente o cenário mencionado, esta seção discute as métricas de centralidade propostas, *Sink Stress* e *Sink Betweenness*, seus algoritmos distribuídos, e os algoritmos de roteamento em árvore que mostram o potencial das mesmas.

3.1 Sink Stress

Nesta seção será apresentada uma métrica de centralidade proposta para o contexto das RSSFs, o *Sink Stress*, que, por sua vez, se baseia na métrica *Stress Centrality* (2.1).

3.1.1 A Métrica

Considere um grafo $G = (V, E)$ que representa uma RSSF, onde V é o conjunto de vértices (nós) e E é o conjunto de arestas (*links*), o *Sink Stress* de um nó qualquer t é

definido como

$$\text{SStress(t)} = \sum_{s \neq t \neq v \in V} \sigma_{sv}(t) \tag{3.1}$$

onde s é o *sink* e $\sigma_{sv}(t)$ é o número de caminhos mínimos do *sink* até v que passam pelo vértice t.

3.1.2 O Algoritmo Distribuído - Cálculo do *Sink Stress*

O algoritmo 1 apresenta o algoritmo distribuído proposto para o cálculo do *Sink Stress* de cada nó. As variáveis utilizadas pelo algoritmo estão descritas na tabela 3.1. Apenas por caráter de implementação deste algoritmo, o *sink* é arbitrariamente escolhido como o nó de *id* 1, linha 7, mas esta não é uma restrição da métrica. O algoritmo possui apenas uma fase, um *flooding* do pacote *Hello* (descrito na tabela 3.1), iniciado pelo *sink*, linha 11. Ao receber este pacote, linha 14, cada nó, precisa verificar duas possibilidades para contabilizar os caminhos mais curtos em direção ao *sink*. A primeira é verificar se o pacote veio por um caminho mais curto do que aquele que já era conhecido por ele, linha 15. Neste caso, este nó recomeça a contabilização do número de caminhos mínimos, linha 16, pois até o momento da chegada deste pacote, ele tinha uma visão de que estava mais longe do *sink*. Caso o nó receba um pacote que veio por um caminho tão longo quanto já era conhecido por ele, é necessário apenas incrementar o contador do número de caminhos mínimos, linha 21. Como descrito em 3.1.1, o *Sink Stress* de um nó qualquer é a quantidade de caminhos mínimos que partem do *sink* e chegam a este nó. Portanto, atribui-se a quantidade de caminhos mínimos calculada à variável alocada para esta função, $SStress_t$, linha 27. É importante ressaltar que cada nó utiliza apenas uma transmissão neste algoritmo e, para tanto, é necessário algum sincronismo no encaminhamento das mensagens de forma que os nós recebam, de seus vizinhos, a informação de distância atualizada. Para tanto, cada nó espera um tempo proporcional à sua distância em relação ao *sink* antes de retransmitir seu pacote *Hello*, linha 30. Portanto, a complexidade de tempo para este algoritmo é O(1) para o tempo e O(n) para o número de mensagens, pois cada nó transmite apenas uma vez, ou seja, O(n). Além disto, uma vez que cada nó espera um tempo proporcional em relação à sua distância ao *sink* para retransmitir o pacote, o tempo de execução está atrelado ao diâmetro da rede.

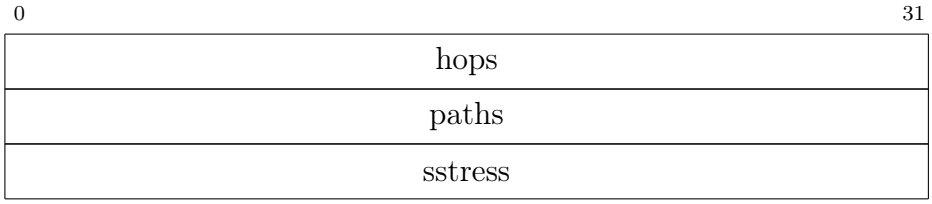

Figura 3.1. Estrutura do pacote *Hello* usado para o cálculo do *Sink Stress*. Os campos foram armazenados como valores inteiros de 32 bits.

Tabela 3.1. Descrição das variáveis no algoritmo para o cálculo do *Sink Stress*

$paths_t$	número de caminhos mínimos do *sink* ao nó t
$hops_t$	distância, em saltos, do *sink* ao nó t
$SStress_t$	o valor *Sink Stress* do nó t

Algoritmo 1 Algoritmo Distribuído para o Sink Stress

```
1: procedure Initalizes Node t
2:     paths_t ← 1
3:     hops_t ← 0
4:     SStress_t ← 0
5:     if i_id = 1 then                                              ▷ Sink
6:         Initializes P as Hello
7:         P.hops ← hops_t
8:         P.paths ← paths_t
9:         broadcast P
10:    end if
11: end procedure
12: procedure Receives Packet P
13:    if P.hops_t + 1 < hops_t then          ▷ Um caminho mais curto até t foi descoberto
14:        hops_t ← P.hops + 1   ▷ Atualiza, em t, a distância pela distância do anterior + 1
15:        P.hops ← hops_t                  ▷ Atualiza, no pacote, a distância para o sink
16:        paths_t ← P.paths                      ▷ Atualiza, em t, o número de caminhos
17:    end if
18:    if P.hops + 1 = hops_t then      ▷ Recebeu de um nó tão distante do sink quanto t
19:        paths_t ← paths_t + P.paths                ▷ Une as perspectivas sobre a distância
20:    end if
21:    SStress_t ← paths_t
22:    schedule broadcast P
23: end procedure
```

Tabela 3.2. Descrição das variáveis adicionais ao algoritmo do cálculo do *Sink Stress* para a criação da estrutura em árvore *Stress Tree*

$neighborMaxSStress_t$	o maior valor de *Sink Stress* recebido por t
$nextHop_t$	identificador do nó selecionado por t como seu próximo salto ao *sink*

3.1.3 O Algoritmo de Roteamento em Árvore: *Stress Tree*

O algoritmo de roteamento, chamado *Stress Tree* (ST), usa a métrica *Sink Stress* para construir a estrutura de roteamento. O principal objetivo deste algoritmo é a construção de caminhos mais curtos a partir de fontes de eventos para o *sink* de maneira a privilegiar a sobreposição de rotas. O algoritmo para construção da árvore pode ser conseguido pela adição das variáveis descritas na tabela 3.2 no algoritmo que faz o cálculo do *Sink Stress*. Para tanto, são adicionados ao algoritmo 1 os trechos de código nos quais a numeração das linhas está em negrito gerando-se com isto o algoritmo de roteamento em árvore para o *Stress Tree*, descrito no algoritmo 2.

A variáveis necessárias são inicializadas no método *Initalizes Node* depois da linha 4, estas são usadas para manter, respectivamente, o vizinho que possui o menor valor de *Sink Stress* e o próximo salto do nó t. No geral, quanto mais distante do *sink*, mais ramificada é a árvore de roteamento e, consequentemente, maiores os valores do *Sink Stress*. Desta forma, um nó t ao usar o ST escolhe o próximo salto através do menores valores de *Sink Stress*, linha 23, dentre seus vizinhos. Antes de retransmitir um pacote *Hello*, o nó atualiza os dados do pacote com o seu identificador e seu *Sink Stress*.

3.2 Sink Betweenness

Nesta seção será apresentada uma métrica de centralidade proposta para o contexto das RSSFs, o *Sink Betweenness*, que, por sua vez, se baseia na métrica *Betweenness* (2.2).

3.2.1 A Métrica

Considere um grafo $G = (V, E)$ que representa uma RSSF, onde V é o conjunto de vértices (nós) e E é o conjunto de arestas (*links*), o *Sink Betweenness* é de um nó

Algoritmo 2 O Algoritmo de Roteamento em Árvore: Stress Tree

```
1: procedure Initalizes Node t
2:     paths_t ← 1
3:     hops_t ← 0
4:     SStress_t ← 0
5:     neighborMaxSStress_t ← −1
6:     nextHop_t ← −1
7:     if i_id = 1 then                                              ▷ Sink
8:         Initializes P as Hello
9:         P.hops ← hops_t
10:        P.paths ← paths_t
11:        broadcast P
12:    end if
13: end procedure
14: procedure Receives Packet P
15:    if P.hops_t + 1 < hops_t then          ▷ Um caminho mais curto até t foi descoberto
16:        hops_t ← P.hops + 1   ▷ Atualiza, em t, a distância pela distância do anterior + 1
17:        P.hops ← hops_t                    ▷ Atualiza, no pacote, a distância para o sink
18:        paths_t ← P.paths                  ▷ Atualiza, em t, o número de caminhos
19:    end if
20:    if P.hops + 1 = hops_t then      ▷ Recebeu de um nó tão distante do sink quanto t
21:        paths_t ← paths_t + P.paths        ▷ Une as perspectivas sobre a distância
22:    end if
23:    if P.SStress < neighborMaxSStress_t then
24:        neighborMaxSStress_t ← P.sstress
25:        nextHop_t ← P.senderID
26:    end if
27:    SStress_t ← paths_t
28:    P.senderID ← t_id
29:    P.sstress ← SStress_t
30:    schedule broadcast P
31: end procedure
```

qualquer t é definido como

$$\mathrm{SBet(t)} = \sum_{i \in \psi_t} \frac{\sigma_{ts}}{\sigma_{is}}, \tag{3.2}$$

onde s é *sink*, σ_{ts} é o número de caminhos de t para o *sink*, σ_{is} é o número de caminhos mínimos de i para o *sink*, onde $\psi_t = \{i \in V | t \in SP_{i \to s}\}$, $SP_{i \to s}$ é o conjunto de todos os caminhos mínimos de um nó qualquer i para o *sink*. Intuitivamente, ψ_t é o conjunto de nós que contêm t em pelo menos um de seus caminhos em direção ao *sink*.

Tabela 3.3. Descrição das variáveis no algoritmo para o cálculo do *Sink Betweenness*

$role_t$	papel atual de t
$sonsPaths_t$	array com número e freqüência de $SP_{t \to sink}$ em ψ_t
$paths_t$	número de caminhos mínimos do *sink* ao nó t
$hops_t$	distância, em saltos, do *sink* ao nó t
$sBet_t$	o valor do *Sink Betweenness* do nó t

3.2.2 O Algoritmo Distribuído - Cálculo do *Sink Betweenness*

O algoritmo 3.2.2 apresenta o algoritmo distribuído proposto para o cálculo do *Sink Betweenness* de cada nó. As variáveis utilizadas pelo algoritmo estão descritas na tabela 3.3. O algoritmo possui duas fases, um *flooding* do pacote *Hello* iniciado pelo *sink*, linha 23 e outro *flooding* refletido pelos nós de borda, onde nós de borda são aqueles que não participam como intermediários em qualquer caminho mínimo ao *sink*. De modo geral, o primeiro *flooding* serve para calcular o numerador do *Sink Betweenness* de cada nó e utiliza o pacote *Hello*, figura 3.2, enquanto o segundo provê o denominador e utiliza o pacote *"Border"*, figura 3.3. Uma vez que cada nó transmite duas vezes, a complexidade deste algoritmo, em número de mensagens, é de O(n). O algoritmo é descrito em detalhes a seguir. Além disto, uma vez que cada nó espera um tempo proporcional em relação à sua distância ao *sink* para retransmitir o pacote, o tempo de execução está atrelado ao diâmetro da rede.

Inicialização dos nós Da linha 1 até 11 as variáveis são inicializadas e o nó *sink* inicializa o cálculo enviando uma mensagem através de um *flooding* do pacote *Hello*, linha 9, que será retransmitido para toda a rede.

Tratando o pacote *Hello* Nesta etapa, compreendida das linhas 13 a 24, o objetivo é calcular, para cada nó, sua distância ao *sink* e o número de caminhos mínimos, que representa o numerador na definição do *Sink Betweenness*. Estes cálculos são feitos utilizando-se o pacote *Hello* da mesma maneira como no algoritmo para o cálculo do *Sink Stress*. A estrutura do pacote *Hello* é composta por duas variáveis inteiras, distância até o *sink* do nó que o recebeu e o número de caminhos mínimos, ambas medidas em número de saltos, figura 3.2. Além destes cálculos, é feita definição de papel ($role$) para cada um dos nós, *Sink*, *Relay* ou *Border*. Assim como na implementação do *Sink Stress*, o papel de *Sink* é arbitrariamente escolhido como o nó de id 1, linha 6, mas é importante ressaltar que tal escolha existe apenas na construção do código, não

sendo uma restrição da métrica. Os nós *Relay* serão aqueles que estão em algum ponto intermediário da rede. Um nó se identificará como *Relay* quando receber um pacote de algum nó mais distante em relação ao *sink*, linha 21. Caso o nó não receba pelo menos um pacote de um nó que esteja mais distante em relação ao *sink*, seu o papel inicial não será modificado, que para todos os nós começa como *Border*, linha 2. A figura 3.4 ilustra uma rede em que os nós em cinza tem o papel de *Border*, os azuis são *Relay* e o papel de *Sink* está em vermelho.

Enviando o pacote *Border* Para os nós que não tiveram o papel modificado para *Sink*, ou *Relay*, a definição inicial, i.e., *Border* foi mantida. Nós no papel de *Border* são responsáveis por iniciar a transmissão do pacote *Border*, linha 50, depois da recepção do pacote *Hello*.

Tratando o pacote *Border* e calculando o Sink Betweenness A estrutura do pacote *Border* contém o mesmo campo *hops* mostrado no pacote *Hello* e, além dele, mais dois outros campos, *sbet*, e $sonsPaths$.

Para um nó t qualquer, o campo $sonsPaths$ é um array no qual cada chave representa, isoladamente, o número de caminhos mínimos com origem nos nós apresentados em ψ_t, ou seja, uma posição para cada variável *path* diferente sendo, no total, do tamanho do conjunto ψ_t. O valor definido para cada chave de $sonsPaths$ é o número de ocorrências da chave. A tabela 3.4 mostra o conteúdo do array $sonsPaths$ para cada um dos nós contidos numa rede ilustrativa mostrada na figura 3.4. Para cada pacote recebido de um nó contido em ψ_t, t atualiza seu $sonsPaths$, linha 28, e também atualiza o array $sonsPaths$ contido no pacote *Border* com sua própria tabela, linha 29, e calcula o *Sink Betweenness*, linha 32. No pior caso, o $sonsPaths$ possuirá $n-1$ elementos, ou seja, o número de elementos igual ao número de nós da rede menos menos o *sink*, a complexidade de tempo deste algoritmo é de O(n), pois será necessário percorrer todo este array para calcular o SBet.

Por exemplo, o processo de calcular o *Sink Betweenness* para o no c da figura 3.4 é feito da seguinte forma. O primeiro *flooding* calcula como 1 o número de caminhos mínimos do nó sink ao nó c ($paths_c$). O segundo *flooding* trás até c um pacote *Border* que indica sua importância no *controle* dos caminhos até o *sink*. Este pacote contém o array $sonsPaths$ com dois elementos $[(2,3),(3,1)]$, ou seja, existem três nós (e, g, i) com dois caminhos a partir do *sink* e um nó (h) com três caminhos a partir do *sink*. Estes quatro nós abrangem o conjunto ψ_c. É importante ressaltar que o conjunto ψ é meramente uma terminologia conceitual, e o nó não precisa, nem mantém qualquer estrutura para armazená-lo, ou mesmo conhece seu conjunto ψ, uma vez que

Tabela 3.4. O conteúdo do array *sonsPaths* no pacote *Border*, e o conjunto ψ para cada nó da rede ilustrada na figura 3.4. Cada posição do array é formatada como (chave, valor).

Nó	$sonsPaths$	ψ_n
a	[∅]	[∅]
b	[(1,3), (2,3), (3,1)]	[c, d, e, f, g, h, i]
c	[(2,3), (3,1)]	[e, g, h, i]
d	[(1,1), (2,3), (3,1)]	[e, f, g, h, i]
e	[(2,2), (3,1)]	[g, h, i]
f	[(3,1)]	[h]
g	[(2,1)]	[i]
h	[∅]	[∅]
i	[∅]	[∅]

o algoritmo distribuído aqui proposto não precisa considerar este aspecto para calcular o *Sink Betweenness*. Agora que c tem os dados necessários para calcular seu *Sink Betweenness*, aplicando a definição desta métrica leva-nos à seguinte expressão:

$\mathrm{SBet_c} = \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{3}$.

O campo *sbet* apresentado em na estrutura do pacote *Border* tem o intuito de informar a vizinhança do nó sobre seu próprio valor de *Sink Betweenness*, que será necessário para construir uma das árvores de roteamento proposta.

As transmissões do pacote *Hello* e *Border* mostradas, respectivamente, nas linhas 23 e 24, são atrasadas para permitir que cada nó receba os pacotes da toda sua vizinhança. Esta característica permite aos nós transmitir apenas um pacote de cada tipo (*Hello* e *Border*) e assegurar a execução de dois *floodings*, ou seja, *overhead* de custo $O(n)$ mensagens. Nos experimentos, antes da retransmissão, o nó aguarda um tempo proporcional à sua distância para o *sink*, que é guardada na variável $hops_t$. Os melhores resultados foram conseguidos escolhendo um atraso diretamente proporcional quando transmitindo o pacote *Hello* e inversamente proporcional quando transmitindo o pacote *Border*.

3.2.3 O Algoritmo de Roteamento em Árvore: *Centrality Tree*

O algoritmo de roteamento, chamado *Centrality Tree* (CT), usa a métrica *Sink Betweenness* para construir a estrutura de roteamento. O principal objetivo deste algoritmo é a construção de caminhos mais curtos a partir de fontes de eventos para o *sink* de maneira a privilegiar a sobreposição de rotas. A intuição por trás dessa abordagem é

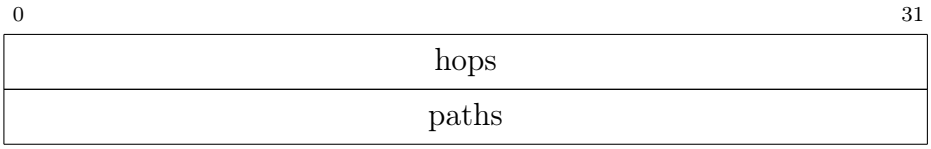

Figura 3.2. Estrutura do pacote *Hello* usado para o cálculo do *Sink Betweenness*. Os campos foram armazenados como valores inteiros de 32 bits.

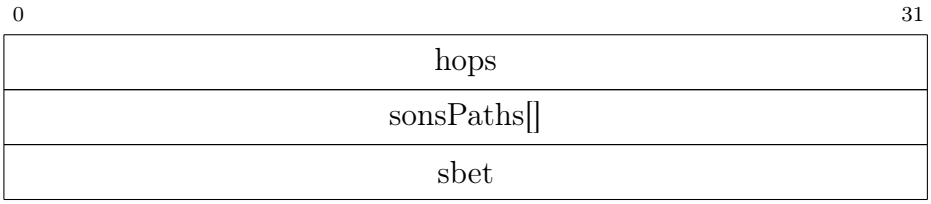

Figura 3.3. Estrutura do pacote *Border* usado para o cálculo do *Sink Betweenness*. Os campos foram armazenados como valores inteiros de 32 bits.

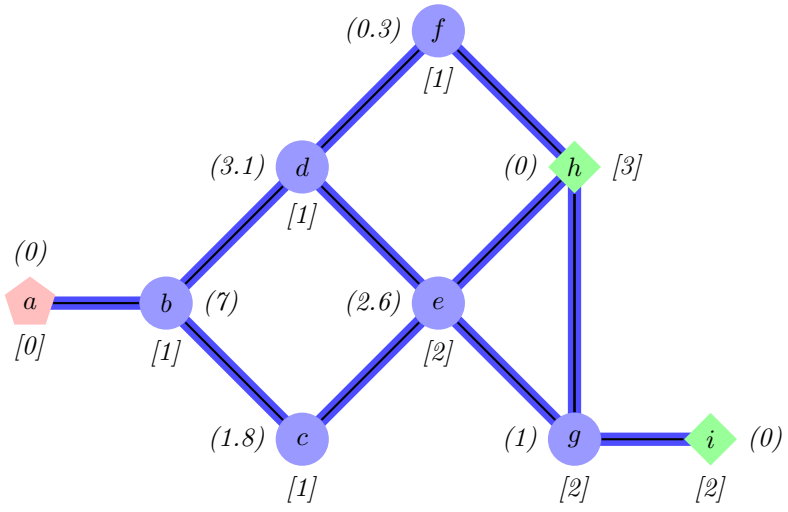

Figura 3.4. Uma rede ilustrativa com o *sink* (pentágono) e nós sensores (círculos e losangos). Para cada nó, o valor do *Sink Betweenness* é mostrado entre parênteses, e o número de caminhos mínimos, com origem no *sink*, entre colchetes. Os nós sensores em formato circular tem o papel de *Relay* e os losangos são *Border*

Algoritmo 3 Algoritmo Distribuído para o Sink Betweenness

```
1: procedure INITALIZES NODE T
2:     role_t ← Border
3:     sonsPaths_t{∅}
4:     paths_t ← 1
5:     hops_t ← 0
6:     sBet_t ← 0
7:     if i_id = 1 then
8:         Initializes P as Hello
9:         role_t ← Sink
10:        P.hops ← hops_t
11:        P.paths ← paths_t
12:        broadcast P
13:    end if
14: end procedure
15: procedure RECEIVES PACKET P
16:    if P.type = Hello then
17:        if P.hops_t + 1 < hops_t then
18:            hops_t ← P.hops + 1
19:            P.hops ← hops_t
20:            paths_t ← P.paths
21:        end if
22:        if P.hops + 1 = hops_t then      ▷ Recebeu de um nó tão distante do sink quanto t
23:            paths_t ← paths_t + P.paths      ▷ Une as perspectivas sobre a distância
24:        end if
25:        if P.hops > hops_t then
26:            role_t ← Relay
27:        end if
28:        schedule broadcast P
29:        schedule Send Border
30:    else                                                    ▷ P.type = Border
31:        if hops_t < P.hops then                          ▷ From some descendant
32:            for j ← 1..length(P.sonsPaths) do
33:                sonsPaths_t[j] ← sonsPaths_t[j] + P.sonsPaths[j]
34:                P.sonsPaths[j] ← sonsPaths_t[j]
35:            end for
36:            for k ← 1..length(sonsPaths) do
37:                sBet ← sBet + sonsPaths_t[k] ∗ (paths_t/k)
38:            end for
39:            broadcast P
40:        end if
41:    end if
42: end procedure
43: procedure SEND BORDER
44:    Initializes P as Border
45:    if role_t = Border then
46:        P.hops ← hops_t
47:        P.paths ← paths_t
48:        P.sonsPaths ← (paths_t, 1)
49:        broadcast P
50:    end if
51: end procedure
```

Tabela 3.5. Descrição das variáveis adicionais ao algoritmo do cálculo do *Sink Betweenness* para a criação da estrutura em árvore *Centrality Tree*

$neighborMaxSBet_t$	o maior valor de *Sink Betweenness* recebido por t
$nextHop_t$	identificador do nó selecionado por t como seu próximo salto ao *sink*

que quanto mais centrais são os nós, mais provavelmente este estará numa rota que sobrepõe vários caminhos em direção ao *sink*.

A infraestrutura de roteamento utiliza a mesma estrutura algorítmica e de mensagens usadas no cálculo do *Sink Betweenness*, sendo necessário o adicionar as variáveis descritas na tabela 3.5 e os trechos de código que serão explicados a seguir. A idéia principal é que o nó vai escolher o próximo salto dentre os nós que estão mais perto do *sink*, em número de saltos e que apresentarem o maior valor de *Sink Betweenness*. Para tanto, são adicionados os trechos de código nos quais a numeração das linhas está em negrito. Para tanto, são adicionados ao algoritmo 3.2.2 os trechos de código nos quais a numeração das linhas está em negrito gerando-se com isto o algoritmo de roteamento em árvore para o *Stress Tree*, descrito no algoritmo 3.2.3.

O trecho de código mostrado na linha 5 é necessário para fazer a inicialização das variáveis $neighborMaxSBet$ e $nextHop_t$, usadas para manter, respectivamente, o vizinho que possui o maior valor de *Sink Betweenness* e o próximo salto do nó t. O trecho de código representado pelo desvio condicional na linha 43 é usado para escolher o nó que apresenta o maior valor de *Sink Betweenness* entre os vizinhos do nó de t e para definir esse nó como seu próximo salto ($nexthop$). Finalmente, o trecho de código mostrado na linha 49 é responsável por definir o *Sink Betweenness* e o $senderID$ do nó t no pacote que será enviado.

Uma vez que a construção da árvore e, consequentemente, o processo de roteamento é baseado no *Sink Betweenness* de cada nó, o algoritmo distribuído deve ser executado, primeiramente, para definir o valor desta métrica para cada nó t e, posteriormente, para permitir que os vizinhos de t também conheçam seu valor. Assim que cada nó calculou o seu *Sink Betweenness* e de seus vizinhos, a escolha do próximo salto na árvore acontece pela seleção do vizinho que possui o maior *Sink Betweenness*, ou seja, o vizinho mais central.

Algoritmo 4 O Algoritmo de Roteamento em Árvore: Centrality Tree

```
1: procedure INITALIZES NODE T
2:     role_t ← Border
3:     sonsPaths_t{∅}
4:     paths_t ← 1, hops_t ← 0, sBet_t ← 0
5:     neighborMaxSBet_t ← −1, nextHop_t ← −1
6:     if i_id = 1 then
7:         Initializes P as Hello
8:         role_t ← Sink, P.hops ← hops_t, P.paths ← paths_t
9:         broadcast P
10:    end if
11: end procedure
12: procedure RECEIVES PACKET P
13:    if P.type = Hello then
14:        if P.hops_t + 1 < hops_t then
15:            hops_t ← P.hops + 1, P.hops ← hops_t, paths_t ← P.paths
16:        end if
17:        if P.hops + 1 = hops_t then      ▷ Recebeu de um nó tão distante do sink quanto t
18:            paths_t ← paths_t + P.paths          ▷ Une as perspectivas sobre a distância
19:        end if
20:        if P.hops > hops_t then
21:            role_t ← Relay
22:        end if
23:        schedule broadcast P
24:        schedule Send Border
25:    else                                                    ▷ P.type = Border
26:        if hops_t < P.hops then                          ▷ From some descendant
27:            for j ← 1..length(P.sonsPaths) do
28:                sonsPaths_t[j] ← sonsPaths_t[j] + P.sonsPaths[j]
29:                P.sonsPaths[j] ← sonsPaths_t[j]
30:            end for
31:            for k ← 1..length(sonsPaths) do
32:                sBet ← sBet + sonsPaths_t[k] ∗ (paths_t/k)
33:            end for
34:            broadcast P
35:        end if
36:    end if
37: end procedure
38: procedure SEND BORDER
39:    Initializes P as Border
40:    if role_t = Border then
41:        P.hops ← hops_t, P.paths ← paths_t
42:        P.sonsPaths ← (paths_t, 1)
43:        if hops_t > P.hops then
44:            if P.sbet > neighborMaxSBet_t then
45:                neighborMaxSBet_t ← P.sbet
46:                nextHop_t ← P.senderID
47:            end if
48:        end if
49:        P.senderID ← t_id, P.sbet ← SBet_t
50:        broadcast P
51:    end if
52: end procedure
```

"Change is the end result of all true learning."

Leo Buscaglia

Capítulo 4

Resultados

A tabela 4.1 descreve os principais parâmetros de simulação utilizados. Para todas as avaliações, foi empregado o método de Monte Carlo considerando os seguintes fatores, número de nós uniformemente distribuídos de maneira aleatória no campo de sensoriamento (n) e densidade (grau médio). Para as comparações feitas entre os algoritmos de roteamento em árvore, o fator quantidade de eventos (ev) é uma porcentagem relativo ao n. O nó *sink* foi posicionado no canto superior esquerdo de rede com topologia plana e o alcance da comunicação dos nós foi fixado em 10 metros. Para todos os cenários avaliados, os enlaces entre os nós são bidirecionais, ou seja, se o alcance de comunicação do nó i chega a um nó j implica que o alcance deste último também chega no primeiro. A rede tem formato quadrado e o tamanho do lado do campo de sensoriamento é calculado para atingir a densidade desejada. A dimensão do lado é calculada utilizando-se a equação $\left\lfloor \sqrt{r^2 * n * \pi / densidade} \right\rfloor$ Nakamura et al. [2009], onde r é o raio de comunicação do nó. Nas simulações foi considerado o canal ideal, ou seja, o alcance de comunicação é um círculo perfeito e as perdas não são considerados nem por interferências nem modulação de rádio. O algoritmo proposto foi avaliado utilizando o simulador de eventos discretos Sinalgo [Group, 2008] a simulação dura até que todos os eventos gerados sejam processados. Neste capítulo, serão feitas duas avaliações: a primeira, seção 4.1, avalia os algoritmos distribuídos no cálculo das métricas e a segunda, seção 4.2, compara os algoritmos de roteamento em árvore com soluções bem conhecidas na literatura. A fim de avaliar os algoritmos distribuídos propostos, as métricas de centralidade foram calculadas tanto de forma distribuída e de forma centralizada, utilizando o pacote `igraph` da ferramenta estatística `R` [Team, 2009]. Os seguintes resultados correspondem à média aritmética das m[1]repetições, onde m é definida de acordo com o intervalo de confiança desejado [Jain, 1991]. Neste trabalho, todos os

Parâmetro	Valor
Número de nós n	128, 256, 512 e 1024
Densidade	10, 15 e 20
Quantidade de eventos ev	10%, 15% e 20%
Duração dos eventos	10 minutos
Raio de comunicação	10 metros
Posição do *sink*	Canto superior esquerdo
Topologia	Plana
Nós (*hardware*)	Homogêneos

Tabela 4.1. Parâmetros e valores utilizados nas simulações

resultados são mostrados para o intervalo de confiança com nível de significância de 95%.

4.1 Algoritmos Distribuídos

A figura 4.1 mostra o erro absoluto no cálculo do *Sink Betweenness*, quando seu respectivo algoritmo distribuído é usado verificando-se os valores calculados em diversos níveis de roteamento, ou seja, a distância em saltos para o *sink*. Para cada um dos cenários simulados composto por n nós, há uma curva que indica a diferença absoluta entre o valor exato calculado usando uma versão centralizada das métricas em comparação com os valores calculados pelos algoritmos distribuídos propostos neste trabalho. No caso do *Sink Stress* o algoritmo distribuído calcula de maneira exata a métrica para os nós e, portanto, o gráfico do erro foi omitido. Já para o *Sink Betweenness* o erro é insignificante, devido à diferença entre a precisão aritmética de linguagem Java, no qual o simulador foi escrito, e da linguagem R, em que a versão centralizada e exata do *Sink Betweenness* foi implementado para ser um referencial.

Embora o algoritmo distribuído para *Sink Betweenness* funcione adequadamente para os cenários simulados, há algumas situações em que o resultado pode estar impreciso, como quando enfrenta a perda de pacotes, sem um mecanismo de recuperação. Uma vez que a implementação não contempla cenários com perda de pacotes devido as questões do meio, a única fonte de erro é um atraso desajustado para o envio do pacote *Border*. O erro ocorre quando um pacote chega após a expiração do temporizador, assim, o nó calcula e transmite o seu valor de *Sink Betweenness* sem as informações

[1]Seja $m = (100zs/r\bar{x})^2$, onde z é a variação normal do nível de confiança desejado, s é o desvio padrão da amostra, r é a precisão desejada, $\bar{x}$ é a amostra média da amostra inicial. O tamanho da amostra inicial pode ser definido arbitrariamente.

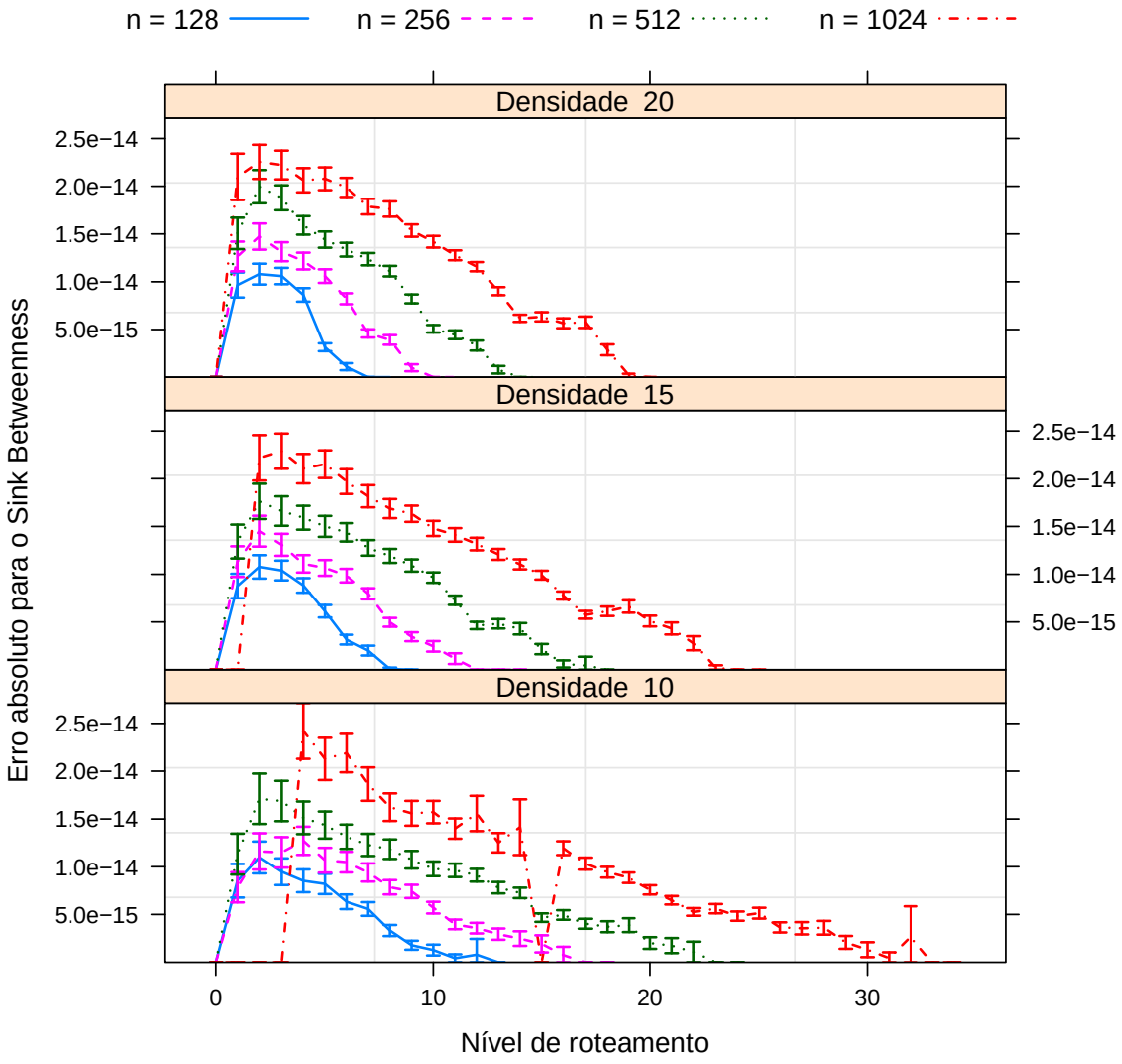

Figura 4.1. Erro absoluto, por nível de roteamento, do algoritmo distribuído para o cálculo do *Sink Betweenness*

completas de seus vizinhos. Conforme descrito na seção 3.2.2, um atraso inversamente proporcional à distância até o *sink* foi, nas simulações, a melhor maneira de contornar este problema, resultando quase sem erros.

Para um nó t, o intervalo de tempo utilizado no processo de agendamento, representado na linha 24 do algoritmo 3.2.2, foi de $1/e^{hops_t}$ segundos. O atraso exponencial aqui utilizado foi suficiente para a chegada de todos os pacotes *Border* dos nós apresentados em ψ_t ao nó t antes do temporizador expirar. Na implantação do mundo real uma constante multiplicativa deve ser usada e ajustada a fim de compensar a latência do canal.

Outro aspecto que deve ser avaliado é o tamanho do vetor $sonsPaths$ do pacote de *Border* uma vez que, em ambientes de RSSFs, o tamanho da carga útil é tipicamente pequeno, por exemplo, é 114 bytes para um sensor *TelosB*. Assim, deve-se considerar essa restrição na utilização do algoritmo proposto.

A figura 4.2 mostra que tamanho do vetor $sonsPaths$ cresce à medida que se aproxima do *sink*, em termos de número de inteiros a serem armazenados. Nós mais próximos do *sink* são susceptíveis a ter um conjunto ψ maior e, consequentemente, mais elementos no vetor $sonsPaths$. De maneira geral, quanto maior o número de nós e a densidade, maior é o pacote, principalmente para nós perto do *sink*.

Observe que os nós não precisam armazenar informações sobre todos os nós em seu ψ, na verdade, o vetor $sonsPaths$ armazena apenas um histograma do número de caminhos mais curtos. Em outras palavras, ele armazena a frequência de nós que apresentam $1, 2, \ldots$ caminhos mais curtos para o *sink*. Uma solução para este problema é representar esses valores de forma eficiente uma vez que os inteiros armazenados, de maneira geral, não são grandes, mas, no máximo, será um valor igual ao número de nós. Memória RAM para alocar esse vetor não é um desafio real uma vez que a memória dos sensores é muito maior do que o tamanho da carga útil do pacote. Uma solução inicial e complementar à representação eficiente dos valores armazenados, é o aumento do tamanho do *bin* do histograma, sempre que o tamanho $sonsPaths$ chegar a um limite máximo. Uma abordagem desta solução foi proposta neste trabalho, o estudo e os resultados do impacto da imposição de um limite e, consequentemente, de erro no cálculo do *Sink Betweenness* são apresentados no apêndice A.

4.2 Algoritmos de Roteamento

O modelo de tráfego definido utiliza uma aplicação baseada em eventos. O número de eventos gerado na rede foi baseado na percentagem de nós depositados no campo

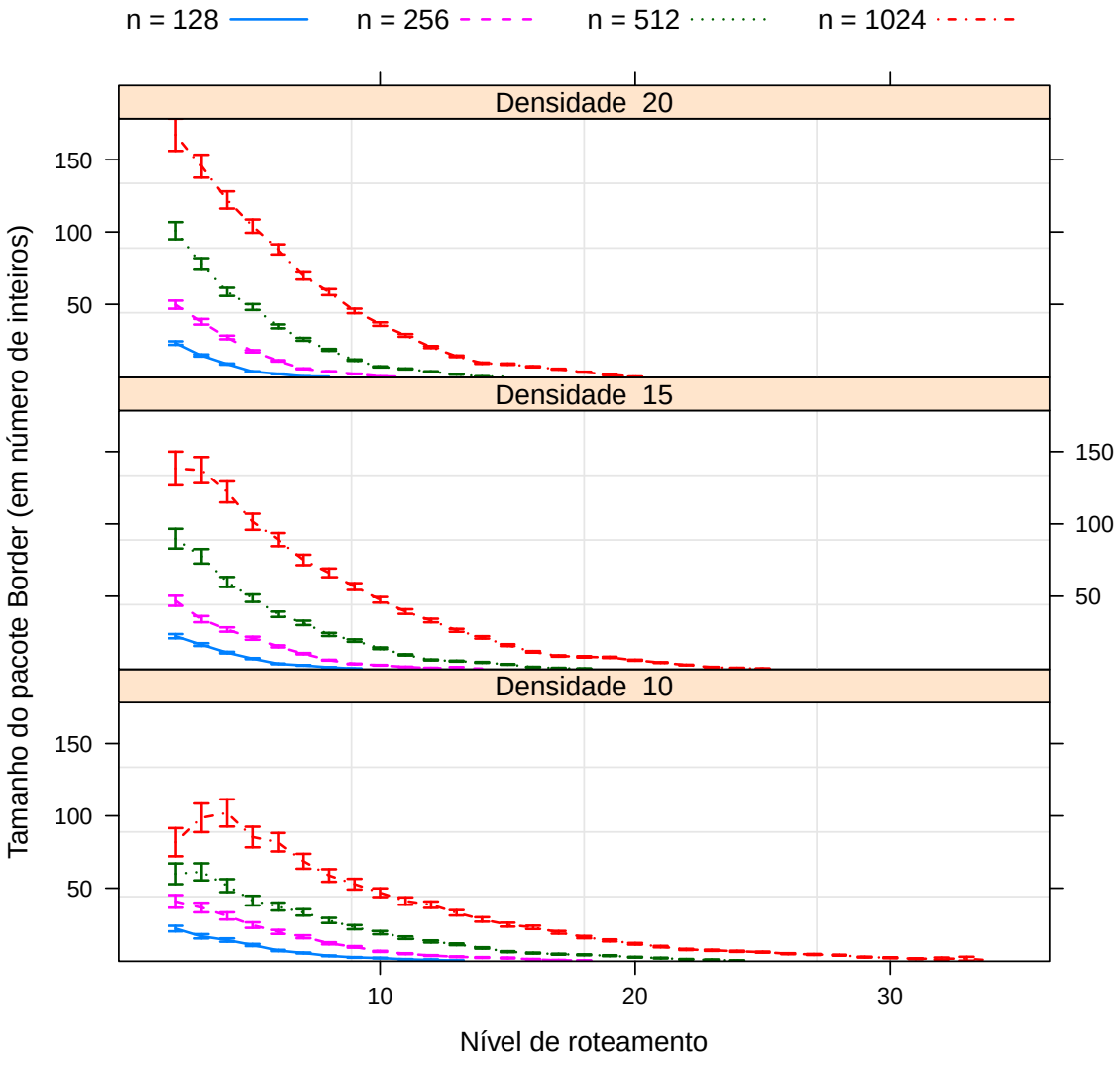

Figura 4.2. Tamanho do pacote *Border* em diversos níveis de roteamento variando-se a densidade da rede

de sensoriamento, $ev \in \{10, 15, 20\}$. Portanto, em cada simulação, $ev\%$ dos nós são escolhidos de maneira aleatória uniforme para reportar eventos, delegando o roteamento aos algoritmos em árvore propostos.

Observando que, em cenários de coleta de dados a estrutura ótima de roteamento para fusão de dados é alcançada quando a informação é encaminhada através da árvore de Steiner ótima [Krishnamachari et al., 2002], foram consideradas as seguintes métricas para avaliar os algoritmos de roteamento propostos:

- *Overhead*: o número de pacotes necessários para a construção da estrutura de roteamento em árvore.
- Número de nós *Steiner*: refere-se ao número de nós que participam da árvore de roteamento, mas que não são fontes de evento. Num contexto de fusão de dados, podem medir a qualidade da árvore, pois a árvore será tão boa quanto menor o número de nós *Steiner* ela tiver.
- Taxa de agregação: É calculado como a razão entre o número de pacotes que chega até o *sink*, pelo número de pacotes enviados pelos nós de origem de eventos. Observe que nem todos os pacotes detectados deverão ser entregues até o *sink*, porque alguns deles são susceptíveis de serem agregados. Em um cenário sem perdas, essa métrica indica o quão eficiente é a aplicação de agregação. Como a mesma função de agregação, que transforma p pacotes em apenas um, é usada em todos os protocolos, essa métrica avalia o favorecimento da agregação de dados conseguido pela estrutura de roteamento.
- Pacotes por dado reportado: esta métrica captura a eficiência do algoritmo de roteamento. É calculada como a relação de todos os pacotes transmitidos na simulação, incluindo os pacotes de *overhead*, pelos pacotes enviados pelos nós de origem dos eventos. Esta métrica também avalia a estrutura de roteamento em termos de agregação de dados, mas inclui o custo para construir a árvore.

Avaliou-se os algoritmos propostos, ST e CT, comparando-os com alguns protocolos bem conhecidos da literatura, SPT, CNS, InFRA e DAARP. O algoritmo CT utilizado nas comparações a seguir não utiliza a limitação do array $sonsPaths$ proposta no apêndice A.

A figura 4.3 mostra os resultados referentes à quantidade de pacotes necessários para se construir a árvore de roteamento em consequência da variação do número de nós e dos percentuais de nós gerando eventos. Com o intuito de promover uma comparação justa entre algoritmos, não foram consideradas as transmissões utilizadas

para a construção dos *clusters*, necessárias por InFRA e DAARP. Uma vez que SPT e ST necessitam de uma transmissão por nó e o CT requer duas transmissões para estabelecer suas rotas, seus custos com *overhead* são constantes. Por outro lado, os resultados referentes ao InFRA foram omitidos, pois seu alto *overhead* prejudicava a visualização do gráfico. Isto acontece, uma vez que o InFRA faz um *flooding* na rede a cada novo evento para calcular a distância agregada dos coordenadores, apresentando um custo linear que cresce de acordo com a quantidade de eventos *ev* e nós na rede. Por outro lado, DAARP e CNS se mostraram similares e razoavelmente sensíveis ao aumento do *ev* e do número de nós da rede. Isto pode ser explicado, pois o DAARP usa *floodings* limitados a cada novo evento, enquanto o CNS utiliza *flooding* em toda a rede apenas se o evento mais recente aconteceu mais próximo do *sink* em relação ao evento anterior. Assim, quando se avalia os extremos da variação do parâmetro *ev*, o InFRA apresenta o maior *overhead* entre os algoritmos avaliados sendo 77% a 93% maior que o DAARP, 78% a 90% maior que o CNS, 79% a 86% maior que o CT e 94% a 97% maior que o SPT e ST.

A figura 4.4 mostra o número de nós *Steiner* ao longo da árvore que conectam os nós de origem com o *sink*, variando o número de nós e o percentual de nós gerando eventos, *ev*. Para facilitar a descrição dos cenários durante avaliação dos resultados, o cenário menos denso e com menor quantidade de nós agindo como fontes eventos será chamado de (a), e o cenário mais denso e com maior quantidade de fontes de eventos, (b). Por se tratar de um protocolo que não visa formação de rotas com o intuito de maximizar a fusão de dados, o SPT apresentou a maior quantidade de nós de *Steiner* em todas as situações. No outro extremo, por se tratar de um protocolo elaborado com o intuito de construir rotas com pouca quantidade de nós *Steiner*, o protocolo DAARP, de maneira geral, estabeleceu as menores árvores. Os protocolos baseados em centralidade obtiveram desempenhos próximos aos melhores resultados podendo-se ressaltar o cenário com maior densidade e quantidade de eventos, no qual estes algoritmos apresentaram os melhores resultados. Para o cenário (a), o CT e o ST geraram, respectivamente, árvores com 37% a 45% e 43% a 49% mais nós *Steiner* que o DAARP. Por outro lado, quando se avalia o cenário (b), CT e ST geram, respectivamente, árvores de 3% a 19% e 3% a 6% melhores que o DAARP. Ao se comparar com o InFRA, percebe-se que o aumento na densidade beneficia ainda mais os algoritmos baseados em centralidade, pois para o cenário (a), o CT e ST, respectivamente, geram árvores com 23% a 29% e 30% a 34% mais nós que o InFRA, mas por outro lado, no cenário (b), CT e ST geram, respectivamente, árvores com 15% a 24% e 2% a 18% menos nós *Steiner* que o InFRA. Esse comportamento é devido ao crescimento do número de nós elegíveis para o próximo salto, com o aumento da densidade. Com mais opções,

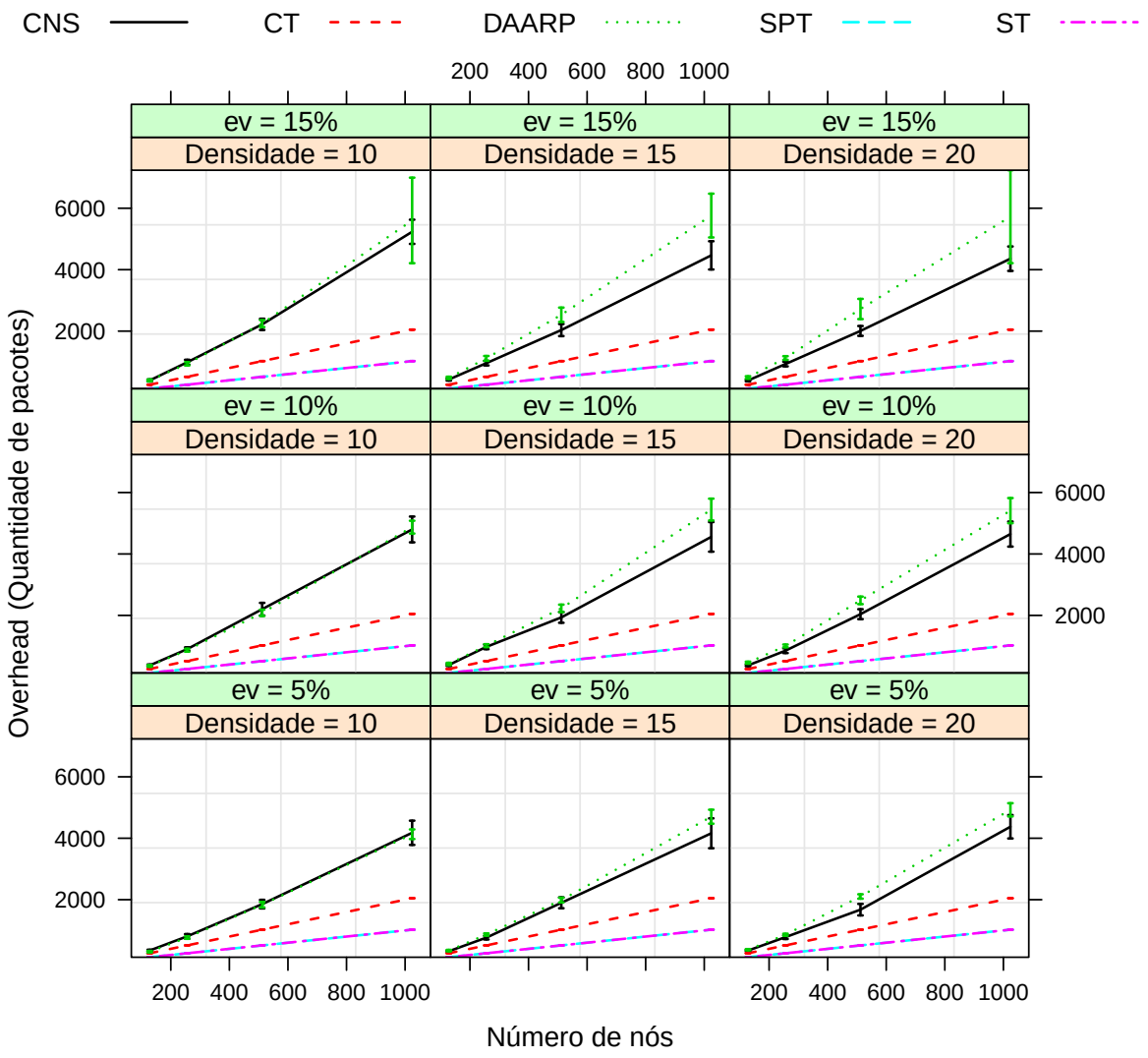

Figura 4.3. Quantidade de pacotes de *overhead* em consequência da variação do número de nós gerando eventos e da densidade

mais nós centrais estarão acessíveis a mais nós e são susceptíveis a serem escolhidos e, consequentemente, melhores rotas são formadas. Outro aspecto interessante é a diferença entre os protocolos baseados em centralidade. O CT, em todos os cenários, gera árvores com menor quantidade de nós *Steiner* que o ST. Este comportamento se deve à métrica usada no CT, o *Sink Betweenness*, uma métrica que descreve melhor a centralidade que o *Sink Stress*, desta forma, as árvores geradas pelo CT foram estabelecidas utilizando-se, em média, 8% menos nós *Steiner* que as geradas pelo ST. Na avaliação do protocolo CNS, percebe-se que o mesmo possui, em todos os cenários, resultados melhores apenas que o SPT. Isto se dá porque o CNS força a existência de um ponto de agregação na árvore, enquanto o SPT apenas possuirá pontos de agregação em cenários em que as rotas ocasionalmente se sobreponham, sem qualquer esforço para que isto aconteça. Quando comparados aos protocolos propostos neste trabalho, verifica-se que o CNS, para o cenário (a), constrói rotas de 14% a 22% maiores que o CT e de 6% a 16% maiores no cenário que o ST e, para o cenário (b), rotas de 35% a 46% e 25% a 44% maiores em relação ao CT e ST, respectivamente. Ao se comparar o SPT com os protocolos propostos, verifica-se que, para o cenário (a), o mesmo constrói rotas que são de 34% a 37% maiores que as do CT e 29% a 31% maiores que o ST e, para o cenário (b), de 61% a 64% e 58% a 59% maiores em comparação ao CT e ST, respectivamente. Estes resultados reforçam a importância do uso da centralidade como fator de decisão na construção da árvore de roteamento, pois, por exemplo, o ST e SPT possuem complexidade computacional igual, mas ao considerar a centralidade, o primeiro possui, para os cenários avaliados, resultados significativamente melhores.

A figura 4.5 apresenta os resultados referentes aos percentuais de agregação. Para fornecer um limite inferior para as transmissões de dados, foi usada uma função de agregação que recebe m pacotes de dados e envia somente um pacote agregado de tamanho fixo. Esta função é desempenhada pelos nós de agregação a cada vez que eles enviam um pacote. De maneira geral, o aumento da densidade e da quantidade de eventos sendo gerados propicia um aumento da taxa de dados agregados. Este comportamento é devido a maior quantidade de tráfego passando pelos nós agregadores. Por se tratar de um protocolo que não faz esforço para construir a estrutura de roteamento visando agregação de dados, apresentando agregação apenas quando as rotas se sobrepõe (agregação oportunística), o SPT apresentou as menores taxas de agregação para todos os cenários. Por outro lado, o protocolo CNS impõe a presença de um nó agregador e, portanto, este apresentou taxa de agregação similar aos melhores resultados. Os algoritmos InFRA e DAARP apresentam um crescimento rápido da taxa de agregação quando se aumenta a quantidade de nós depositados na rede. Este comportamento se dá, pois mais nós estarão disponíveis a serem escolhidos como pontos de agregação

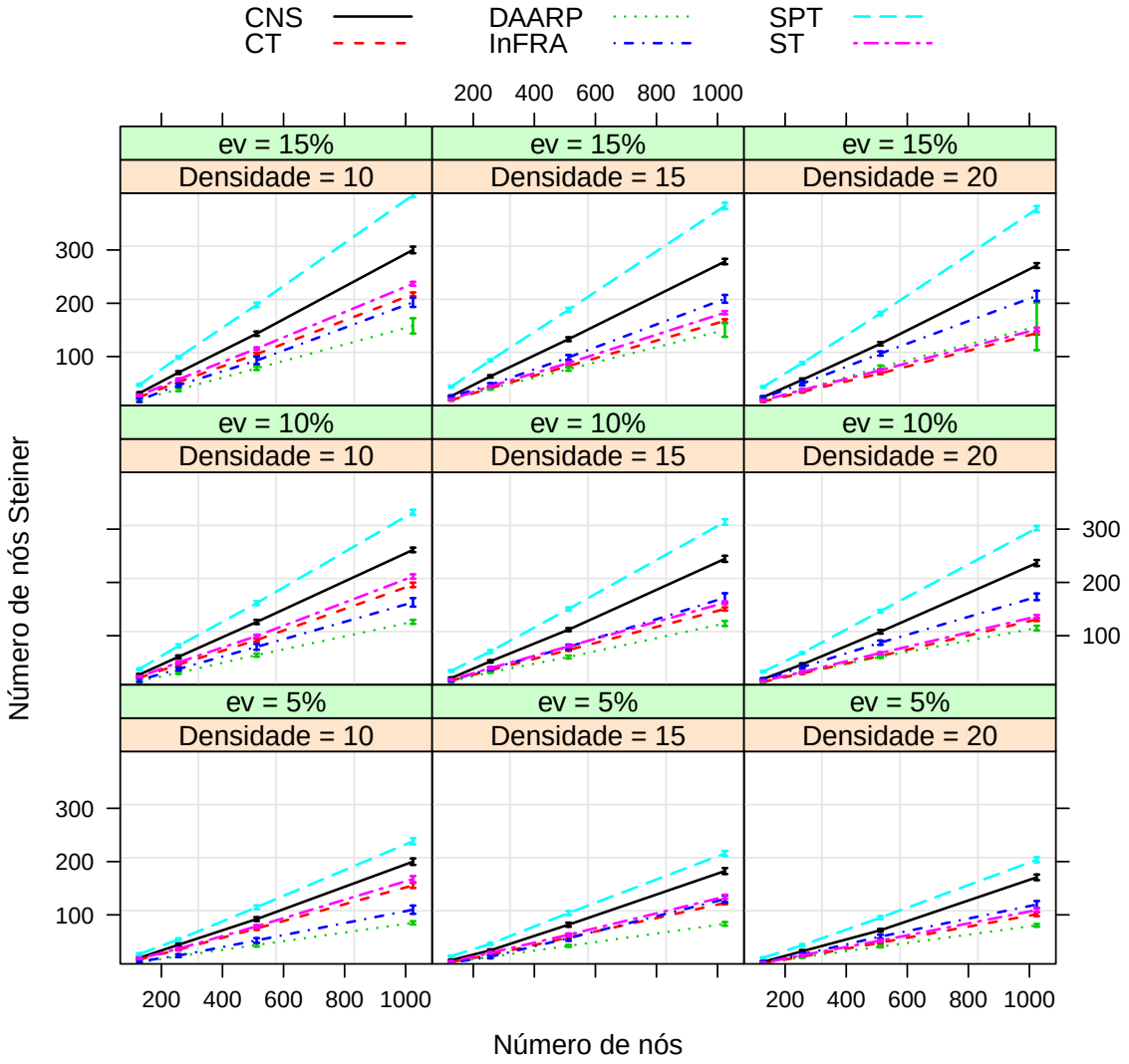

Figura 4.4. Número de nós *Steiner* em consequência da variação do número de nós gerando eventos e da densidade

nas estruturas já estabelecidas, possibilitando melhores escolhas. Para os algoritmos baseados em centralidade, a pouca densidade e pouca quantidade de nós gera as taxas mais baixas de agregação, mas mesmo neste casos estes foram superiores aos demais algoritmos, característica que mostra a eficiência na escolha da centralidade para construção das rotas. Para as situações em que tanto a densidade e a quantidade de eventos é alta, todos os algoritmos, com exceção do SPT, obtiveram taxas de agregação iguais para o intervalo de confiança utilizado, pois seus intervalos de confiança se sobrepõe pela média.

A figura 4.6 ilustra os resultados acerca da métrica pacotes por dado reportado. Esta métrica tende a mostrar os resultados tanto sobre a perspectiva da qualidade da árvore gerada, mas também do esforço necessário para construí-la, uma vez que leva em consideração o *overhead*. Esta relação custo-benefício faz com que os menores valores indiquem os algoritmos mais eficientes. De maneira geral, é importante ressaltar a variação dos resultados para esta métrica com a variação da densidade, pois a variação na quantidade de eventos teve pouco impacto nos resultados. Portanto, nesta avaliação serão utilizados os termos cenário (a) para fazer referência aos cenários com a menor densidade, (b) com densidade intermediária e (c) com a maior densidade.

Nesta figura, os resultados referentes ao protocolo CNS foram omitidos para evitar a perda dos detalhes na escala referente aos demais protocolos comparados. Por se tratar de um algoritmo que necessita de muitas transmissões para o estabelecimento das rotas e por ter construído, nos cenários simulados, rotas melhores apenas que o SPT, o CNS obteve os piores resultados. Em média, com o aumento da densidade, este algoritmo necessitou para os cenários (a), (b) e (c), respectivamente, 69%, 65% e 64%, mais pacotes por dado reportado que o CT. Comparando-se o CNS com o ST para os mesmos cenários, tem-se que o primeiro necessita de 69%, 66% e 64% mais pacotes para reportar a mesma quantidade de dados que o segundo. Com o aumento da densidade, há uma ligeira diminuição da distância entre os desempenhos de CT e ST comparados ao CNS, pois a rede torna-se menor e, consequentemente, eventos próximos ao *sink* ocorrem mais facilmente, dando melhores pontos de agregação ao CNS. De qualquer modo, o ganho dos algoritmos propostos frente ao CNS é reflexo de uma árvore de melhor qualidade, como visto na avaliação da quantidade de nós *Steiner* aliado ao menor *overhead*.

DAARP e SPT apresentaram resultados similares para cenários com baixa densidade. De fato, o SPT não produz rotas tão curtas como o DAARP, mas por outro lado o primeiro tem um *overhead* linear, enquanto o segundo possui *overhead* proporcional ao número de eventos. Com o aumento da densidade, o DAARP se mostra significativamente melhor que o SPT pois, com maior densidade o protocolo tem mais

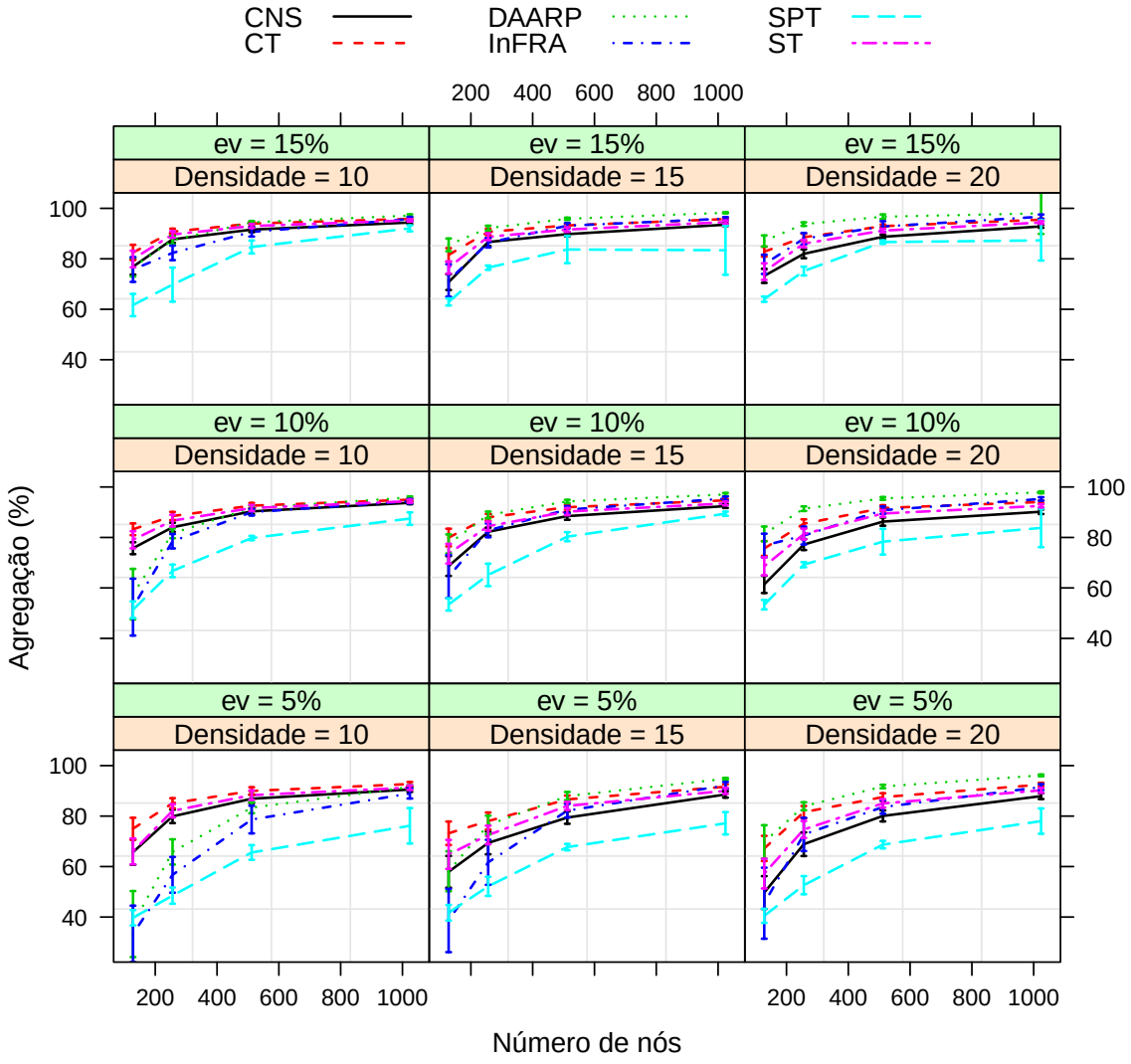

Figura 4.5. Taxa de agregação em consequência da variação do número de nós gerando eventos e da densidade

possibilidades de encontrar melhores pontos de agregação. Para pouca quantidade de nós, o SPT se mostra mais eficiente, mas este comportamento vai se modificando lentamente, pois quando se trata de cenários com maior quantidade de nós, densidade e eventos, o DAARP se mostra melhor. Analisando-se a quantidade de pacotes por dado reportado para o cenário (a) com as menores quantidades de nós, 128 e 256, fica evidente que o DAARP não é tão adequado quanto o SPT, pois o segundo é, respectivamente, 13% e 11% mais eficiente que o primeiro. Isto se dá pois, para cenários menores, as árvores de melhor qualidade do DAARP não pagam o custo de sua construção, bastando árvores mais simples e que requerem menor *overhead*, abordagem utilizada pelo SPT. Por outro lado, analisando-se esta métrica no cenário (a) com maior quantidade de nós, 512 e 1024, vemos que o DAARP é, respectivamente, 10% e 15% mais eficiente que o SPT. Este comportamento é justificado pelo fato de que, em cenários maiores, árvores de melhor qualidade têm muito impacto na quantidade de mensagens enviadas, diferentemente de cenários menores, portanto, o DAARP consegue tirar proveito da qualidade de sua árvore frente ao SPT. Para os cenários (b) e (c), o DAARP possui resultados melhores que o SPT, pois o aumento da densidade disponibiliza melhores pontos de agregação à escolha do primeiro.

Ao se comparar o DAARP com os algoritmos baseados em centralidade pode-se ressaltar um comportamento interessante, os algoritmos baseados em centralidade são mais eficientes para os cenários menos densos, igualmente eficientes para o cenário com densidade intermediária e menos eficientes para o cenário mais denso. Para o cenário (a), tanto ST quanto CT foram mais eficientes que o DAARP, respectivamente, 23% e 24%. No cenário (b), a mudança de fase começa ocorrer e, para as situações com 128 e 256 nós, ST é 11% e 10% mais eficiente enquanto CT é 11% e 8%. Para o cenário (b) com 512 e 1024 nós, o desempenho de ST, CT e DAARP foi similar. Para o cenário (c), o DAARP se mostra mais eficiente que os algoritmos baseados em centralidade, sendo 9% mais eficiente que o CT e 7% quando comparado com o ST. Estes resultados mostram que o aumento da densidade favorece o algoritmo DAARP, pois o aumento do número de vizinhos propicia melhores escolhas de pontos de agregação quando comparadas as escolhas baseadas em centralidade. Nestes cenários, os valores de centralidade ficam "*diluídos*"e, portanto, não há tantos nós com centralidades que se destaquem frente aos demais. Por outro lado, cenários com menor densidade fazem com que os valores de centralidade estejam concentrados em poucos nós, propiciando uma melhor estrutura em árvore que terá seus pontos de agregação mais centrais. A duração dos eventos também tem influência nos resultados da quantidade de pacotes por dado reportado, pois eventos de curta duração tendem a requerer a reconstrução dar rotas com maior frequência e, portanto, algoritmos mais simples e menos custosos

são mais adequados. Por outro lado, algoritmos como DAARP, que tem custo mais elevado para construção das rotas, são mais adequados para eventos de maior duração. Durante as simulações, diversas durações de eventos foram avaliadas, mas o resultado mostrado aqui refere-se a eventos com duração de 10 minutos. Para valores muito menores que este, CT e ST possuem resultados melhores que DAARP, mas para valores maiores, DAARP apresenta melhores desempenhos segundo a métrica de pacotes por dado reportado.

4.3 Conclusões

Neste capítulo foram feitas avaliações dos algoritmos distribuídos propostos para o cálculo das métricas desenvolvidas, com o intuito de avaliar sua precisão. Quando existiram, os erros se restringiram a precisão numérica e sem implicações práticas, portanto, estes algoritmos se mostraram adequados e de baixo custo computacional. Além disto, a avaliação dos algoritmos de roteamento propostos mostrou bons resultados, principalmente com relação ao custo-benefício, pois árvores razoavelmente curtas foram conseguidas com pouco *overhead*.

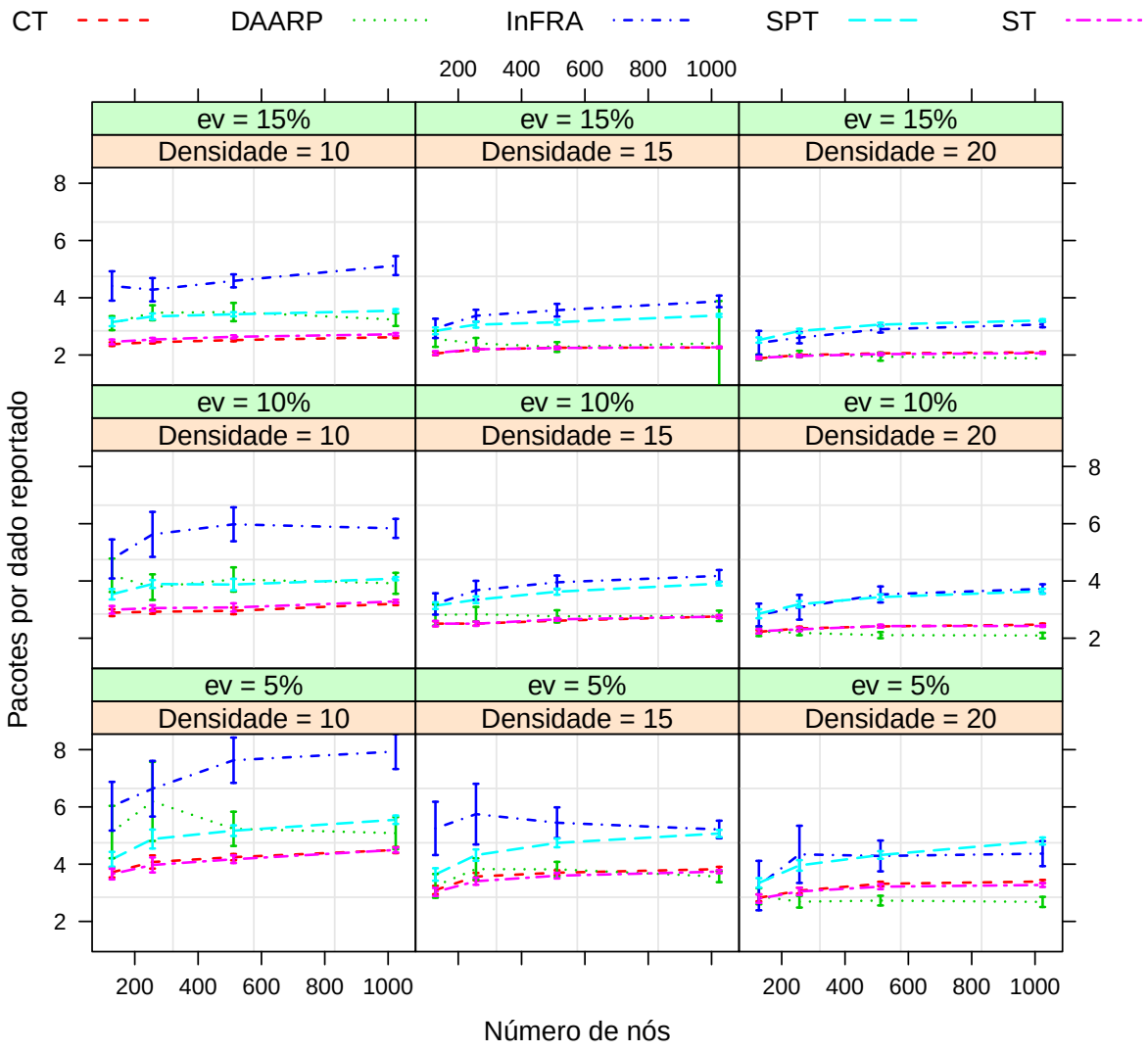

Figura 4.6. Quantidade de pacotes por dado reportado em consequência da variação do número de nós gerando eventos e da densidade

Capítulo 5

Conclusões

Neste trabalho, foram propostas duas novas métricas de centralidade adequadas para as RSSFs, *Sink Stress* e *Sink Betweenness*, algoritmos distribuídos para o cálculo das mesmas, e uma possível abordagem para utilizá-las no roteamento com agregação de dados. Os resultados abrem um campo para a utilização da centralidade no projeto de algoritmos de roteamento para RSSFs com a proposição das árvores *Stress Tree* e *Centrality Tree*. Os resultados de simulação mostram que a abordagem baseada em centralidade possui baixo custo para a construção de rotas aliada a criação de rotas relativamente curtas, com resultados significativamente melhores em cenários de baixa densidade. Apesar do fato de que nós centrais são mais propensos a gastar mais energia, o comportamento aleatório e de curta duração esperado dos eventos, faz com que isso não seja um grande problema, pois a árvore muda com muita freqüência.

Capítulo 6

Trabalhos Futuros

Por se tratar de um campo ainda pouco explorado, o uso das características topológicas e, principalmente, da centralidade em RSSFs propicia várias oportunidades de estudos futuros. Dentre estas, pode-se destacar:

- Estudar o crescimento da frequência do número de caminhos mínimos que passam pelos nós. E, a partir disto, propor soluções para o problema do tamanho do pacote no algoritmo distribuído para o cálculo do *Sink Betweenness*
- Projetar outras funções de rede baseadas na centralidade dos nós, e.g., sincronização de relógios baseada nos nós mais centrais, ou menos centrais
- Relação entre mobilidade da rede e centralidade, e.g., como lidar com a mobilidade da rede e a necessidade de se manter os valores de centralidade atualizados. Entender o que a centralidade representa num cenário de redes móveis.

Referências Bibliográficas

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

J. N. Al-Karaki, R. Ul-Mustafa, and A. E. Kamal. Data aggregation in wireless sensor networks - exact and approximate algorithms. In *Proceedings of the Workshop on High Performance Switching and Routing*, pages 241–245, Phoenix, AZ, 2004.

Jamal N. Al-karaki and Ahmed E. Kamal. Routing techniques in wireless sensor networks: A survey. *IEEE Wireless Communications*, 11:6–28, 2004.

G. Anastasi, M. Conti, M. Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, May 2009. ISSN 15708705. doi: 10.1016/j.adhoc.2008.06.003.

Jacob M. Anthonisse. The rush in a directed graph. Technical Report BN 9/71, Stichting Mathematisch Centrum, Amsterdam, The Netherlands, October 1971.

A. Bavelas. A mathematical model for group structure. *Human Organizations*, 7:16–30, 1948.

Murray A. Beauchamp. An improved index of centrality. *Behavioral Science*, 10: 161–163, 1965. doi: http://dx.doi.org/10.1002/bs.3830100205.

Philip Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, 2(1):113–120, 1972.

Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

Ulrik Brandes and Christian Pich. Centrality estimation in large networks. In *International Journal of Bifurcation and Chaos, Special Issue on Complex Networks Structure and Dynamics*, volume 17, pages 2303–2318, 2007.

Sergey Brin. The anatomy of a large-scale hypertextual web search engine. In *Computer Networks and ISDN Systems*, volume 30, pages 107–117, 1998.

Chee-Yee Chong and S. P. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, 2003. doi: 10.1109/JPROC.2003.814918.

Lluís Coromina, Jaume Guia, Germà Coenders, and Anuska Ferligoj. Duocentered networks. *Social Networks*, 30(1):49 – 59, 2008. ISSN 0378-8733. doi: DOI:10.1016/j.socnet.2007.07.001.

Paolo Crucitti, Vito Latora, and Sergio Porta. Centrality in networks of urban streets. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 16(1):015113, 2006. doi: 10.1063/1.2150162.

Benjamin J. Culpepper, Lan Dung, and Melody Moh. Design and analysis of hybrid indirect transmissions (hit) for data gathering in wireless micro sensor networks. *SIGMOBILE Mobile Computing and Communications Review*, 8:61–83, January 2004. ISSN 1559-1662. doi: http://doi.acm.org/10.1145/980159.980169.

Luciano da F. Costa, Francisco A. Rodrigues, Gonzalo Travieso, and Villas P. R. Boas. Characterization of complex networks: A survey of measurements. *Advances in Physics*, 56(0001-8732):167–242, 2007. URL `http://www.informaworld.com/10.1080/00018730601170527`.

C. Dangalchev. Residual closeness in networks. *Physica A Statistical Mechanics and its Applications*, 365:556–564, June 2006. doi: 10.1016/j.physa.2005.12.020.

Kai-wei Fan, Sha Liu, and Prasun Sinha. Structure-free data aggregation in sensor networks. In *IEEE Transactions on Mobile Computing*, page 2007, 2006.

Muhammad Omer Farooq, Abdul Basit Dogar, and Ghalib Asadullah Shah. Mr-leach: Multi-hop routing with low energy adaptive clustering hierarchy. In *Proceedings of the 2010 Fourth International Conference on Sensor Technologies and Applications*, SENSORCOMM '10, pages 262–268, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4096-2. doi: http://dx.doi.org/10.1109/SENSORCOMM.2010.48. URL `http://dx.doi.org/10.1109/SENSORCOMM.2010.48`.

Leonardo L. Fernandes and Amy L. Murphy. Mvsink: Incrementally building in-network aggregation trees. In *Proceedings of the 6th European Conference on Wireless Sensor Networks*, EWSN '09, pages 216–231, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-00223-6. doi: 10.1007/978-3-642-00224-3_14.

L. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1(3):215–239, 1979. ISSN 03788733. doi: 10.1016/0378-8733(78)90021-7.

Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, March 1977.

Linton C. Freeman, Stephen P. Borgatti, and Douglas R. White. Centrality in valued graphs: A measure of betweenness based on network flow. *Social Networks*, 13(2): 141–154, June 1991. doi: 10.1016/0378-8733(91)90017-N.

Tianjun Fu and Hsinchun Chen. Analysis of cyberactivism: A case study of online free tibet activities. In *IEEE International Conference on Intelligence and Security Informatics, 2008.*, ISI 2008, pages 1–6, June 2008. doi: 10.1109/ISI.2008.4565020.

Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck. Reach for a *:efficient point-to-point shortest path algorithms. In *Workshop on Algorithm Engineering & Experiments*, pages 129–143, 2006.

ETHZ Distributed Computing Group. Sinalgo – simulator for network algorithms. `http://dcg.ethz.ch/projects/sinalgo/`, 2008. URL `http://dcg.ethz.ch/projects/sinalgo/`.

Ronald J. Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *Proceedings 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 100–111. SIAM, 2004.

Per Hage and Frank Harary. Eccentricity and centrality in networks. *Social Networks*, 17(1):57–63, 1995. ISSN 0378-8733. doi: DOI:10.1016/0378-8733(94)00248-9.

Albert F. Harris, III, Robin Kravets, and Indranil Gupta. Building trees based on aggregation efficiency in sensor networks. *Ad Hoc Networks*, 5:1317–1328, November 2007. ISSN 1570-8705. doi: 10.1016/j.adhoc.2007.02.021.

W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002. doi: 10.1109/TWC.2002.804190.

Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, volume 8 of *HICSS '00*, pages 8020–, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0493-0.

Stefan Hougardy and Hans Jürgen Prömel. A 1.598 approximation algorithm for the steiner problem in graphs. In *Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 448–453, 1999.

Wen-Wen Huang, Ya-Li Peng, Jian Wen, and Min Yu. Energy-efficient multi-hop hierarchical routing protocol for wireless sensor networks. *Wireless Communications and Trusted Computing, International Conference on Networks Security*, 2:469–472, 2009. doi: 10.1109/NSWCTC.2009.352.

Chalermek Intanagonwiwat, Deborah Estrin, Ramesh Govindan, and John Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, ICDCS '02, pages 457–, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1585-1.

Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11:2–16, February 2003. ISSN 1063-6692. doi: 10.1109/TNET.2002.808417.

Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling.* John Wiley, 1991.

Yongyu Jia, Lian Zhao, and Bobby Ma. A hierarchical clustering-based routing protocol for wireless sensor networks supporting multiple data aggregation qualities. *International Journal of Sensor Networks*, 4(1/2):79–91, 2008. ISSN 1748-1279. doi: http://dx.doi.org/10.1504/IJSNET.2008.019254.

Ferenc Jordan. Keystone species and food webs. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1524):1733–1741, June 2009. doi: 10.1098/rstb.2008.0335.

Ferenc Jordan, Zsofia Benedek, and Janos Podani. Quantifying positional importance in food webs: A comparison of centrality indices. *Ecological Modelling*, 205(1-2): 270–275, 2007. ISSN 0304-3800. doi: DOI:10.1016/j.ecolmodel.2007.02.032.

Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999. ISSN 0004-5411. doi: 10.1145/324133.324140.

Dirk Koschützki, Katharina A. Lehmann, Leon Peeters, Stefan Richter, Dagmar T. Podehl, and Oliver Zlotowski. *Centrality Indices*, volume 3418/2005 of *Lecture Notes in Computer Science: Network Analysis*, chapter Part I: Elements, pages 16–61. Springer, 2005. doi: content/9ylyvje90wdev5ql.

Wolfram Krause, Jan Scholz, and Martin Greiner. Optimized network structure and routing metric in wireless multihop ad hoc communication. *Physica A: Statistical Mechanics and its Applications*, 361(2):707–723, 2006. ISSN 0378-4371. doi: DOI: 10.1016/j.physa.2005.06.085.

H. Kretschmer and T. Kretschmer. Application of a new centrality measure for social network analysis to bibliometric and webometric data. In *1st International Conference on Digital Information Management*, pages 199–204, Dec. 2007. doi: 10.1109/ICDIM.2007.369353.

Bhaskar Krishnamachari, Deborah Estrin, and Stephen B. Wicker. The impact of data aggregation in wireless sensor networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 575–578, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1588-6.

Harold. J. Leavitt. Some effects of communication patterns on group performance. *Journal of Abnormal and Social Psychology*, 46, 1951.

S. Lindsey and C.S. Raghavendra. Pegasis: Power-efficient gathering in sensor information systems. In *IEEE Aerospace Conference*, volume 3, pages 1125–1130, 2002. doi: 10.1109/AERO.2002.1035242.

Antonio A.F. Loureiro, José Marcos S. Nogueira, Linnyer Beatrys Ruiz, Raquel Aparecida de Freitas Mini, and Carlos Maurício Sérgio Figueiredo Eduardo Freire Nakamura. Redes de sensores sem fio, 2003. XXI Simpósio Brasileiro de Redes de Computadores, p. 179–226, Natal, RN, Brasil. Tutorial.

Itziar Marin, Eduardo Arceredillo, Aitzol Zuloaga, and Jagoba Arias. Wireless sensor networks: A survey on ultra-low power-aware design. In *Proceedings of World Academy of Science, Engineering and Technology Volume*, volume 8, 2005. ISBN 1307-6884.

O. Mason and M. Verwoerd. Graph theory and networks in biology. *Systems Biology, IET*, 1(2):89–119, March 2007. ISSN 1751-8849. doi: 10.1049/iet-syb:20060038.

Eduardo F. Nakamura, Horacio A. B. F. de Oliveira, Luciana F. Pontello, and Antonio A. F. Loureiro. On demand role assignment for event-detection in sensor networks. In *Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 941–947, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2588-1. doi: 10.1109/ISCC.2006.110.

Eduardo F. Nakamura, Heitor S. Ramos, Leandro A. Villas, Horacio A. B. F. de Oliveira, Andre L. L. de Aquino, and Antonio A. F. Loureiro. A reactive role assignment for data routing in event-based wireless sensor networks. *Computer Networks*, 53(12): 1980–1996, 2009. doi: http://dx.doi.org/10.1016/j.comnet.2009.03.009.

M. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1):39–54, January 2005. ISSN 03788733. doi: 10.1016/j.socnet.2004.11.009.

S. Porta, P. Crucitti, and V. Latora. Multiple centrality assessment in parma campus: a network analysis of paths and open spaces. *Urban Design International*, 13:41–50, 2008.

G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/332833.332838.

Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 770–779, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. ISBN 0-89871-453-2.

Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, December 1966. doi: 10.1007/BF02289527.

A. Shimbel. Structural parameters of communication networks. *Bulletin of Mathematical Biophysics*, 1953.

Oussama Souihli, Mounir Frikha, and Mahmoud Ben Hamouda. Load-balancing in manet shortest-path routing protocols. *Ad Hoc Networks*, 7(2):431–442, 2009. ISSN 1570-8705. doi: http://dx.doi.org/10.1016/j.adhoc.2008.04.007.

K. Stephenson and M. Zelen. Rethinking centrality: Methods and examples. *Social Networks*, 11(1):1–37, 1989.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. URL `http://www.r-project.org`. ISBN 3-900051-07-0.

Thomas W. Valente and Robert K. Foreman. Integration and radiality: Measuring the extent of an individual's connectedness and reachability in a network. *Social Networks*, 20(1):89–105, 1998. ISSN 0378-8733. doi: DOI:10.1016/S0378-8733(97)00007-5.

René van den Brink and Robert P. Gilles. Measuring domination in directed networks. *Social Networks*, 22(2):141–157, 2000. ISSN 0378-8733. doi: DOI:10.1016/S0378-8733(00)00019-8.

A. Leandro Villas, A. Boukerche, B. R. Araujo, and A. F Loureiro. A reliable and data aggregation aware routing protocol for wireless sensor networks. In *Proceedings of the 12th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems*, pages 1–9, Tenerife, Canary Islands, Spain, 2009. ACM.

Jianwei Wang, Lili Rong, and Tianzhu Guo. A new measure of node importance in complex networks with tunable parameters. In *4th International Conference on Wireless Communications, Networking and Mobile Computing.*, WiCOM 2008, pages 1–4, Oct. 2008. doi: 10.1109/WiCom.2008.1170.

S. Wasserman and K. Faust. *Social Network Analysis: methods and applications*. Cambridge University Press, 1994.

Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2008.04.002.

Fang Yiwei, Cui Wentian, and Yan Huahai. Identification of important actors in edge-weight social networks. In *International Conference on Service Systems and Service Management*, volume 2, pages 1643–1647, Oct. 2006. doi: 10.1109/ICSSSM.2006.320792.

Y. Yu, B. Danila, J. A. Marsh, and K. E. Bassler. Optimal transport on wireless networks. *Europhysics Letters*, 79(4):48004 (5pp), 2007. URL `http://stacks.iop.org/0295-5075/79/48004`.

Liang Zhao, Xiang Hong, and Qilian Liang. Energy-efficient self-organization for wireless sensor networks: a fully distributed approach. In *IEEE Global Telecommunications Conference (GLOBECOM)*, volume 5, pages 2728–2732, November 2004. doi: 10.1109/GLOCOM.2004.1378851.

Apêndice A

Uma solução para o problema do crescimento do pacote *Border*

Neste capítulo, é avaliada uma solução inicial proposta para o problema do crescimento do array *sonsPaths* contido no pacote *Border* e, consequentemente, do mesmo.

Como forma de limitar o tamanho do pacote, durante o cálculo do *Sink Betweenness*, caso o tamanho do array *sonsPaths*, contido no pacote, ultrapassar o valor limite, será feita a média entre duas posições consecutivas. As figuras A.1 e A.2 mostram, respectivamente, os resultados para esta solução proposta utilizando-se *sonsPaths* de tamanho máximo em 5 e 30 elementos, sendo este último a quantidade máxima em um sensor *TelosB*. Na figura, cada uma das curvas é referente ao erro relativo percentual, com relação ao valor exato. A curva relativa à topologia com 1024 nós foi omitida, pois seu erro é, em média, 102% e 110% maior do que os valores apresentados para a topologia com 512 nós, comparando-se, respectivamente, os cenários limitados em 5 e 30 elementos. É possível notar que o erro começa a crescer a partir do momento que os dados precisam ser sumarizados pela média, ou seja, o tamanho do *sonsPaths* já atingiu o limite imposto.

A solução proposta limita-se apenas a restringir o tamanho do array *sonsPaths* sem analisar o comportamento dos dados contidos no mesmo. A figura A.3 mostra a quantidade de nós *Steiner* necessária para construir a árvore *Centrality Tree* quando se limita o array *sonsPaths* em 5 (CT-5) e 30 (CT-30) elementos comparando-os com a construção que não possui restrição, CT. É possível perceber que a árvore construída com menor restrição, 30, possui praticamente a mesma quantidade de nós *Steiner* que aquela que não utilizou restrição. Para a construção de uma árvore de roteamento baseada nos dados da centralidade, como é o caso da proposta neste trabalho, erros

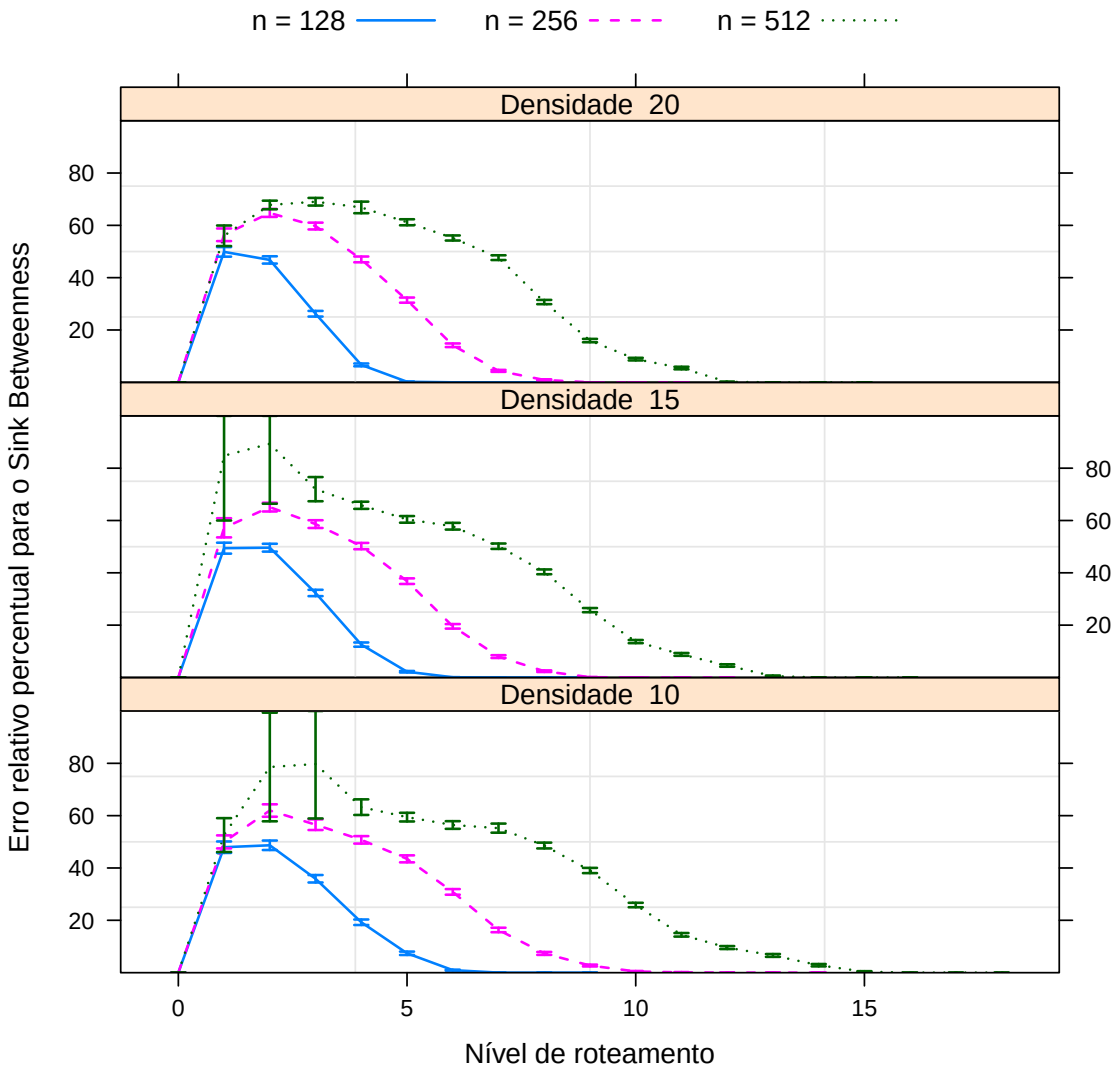

Figura A.1. Erro percentual relativo quando se limita em 5 elementos o tamanho do array *sonsPaths* contido no pacote *Border*

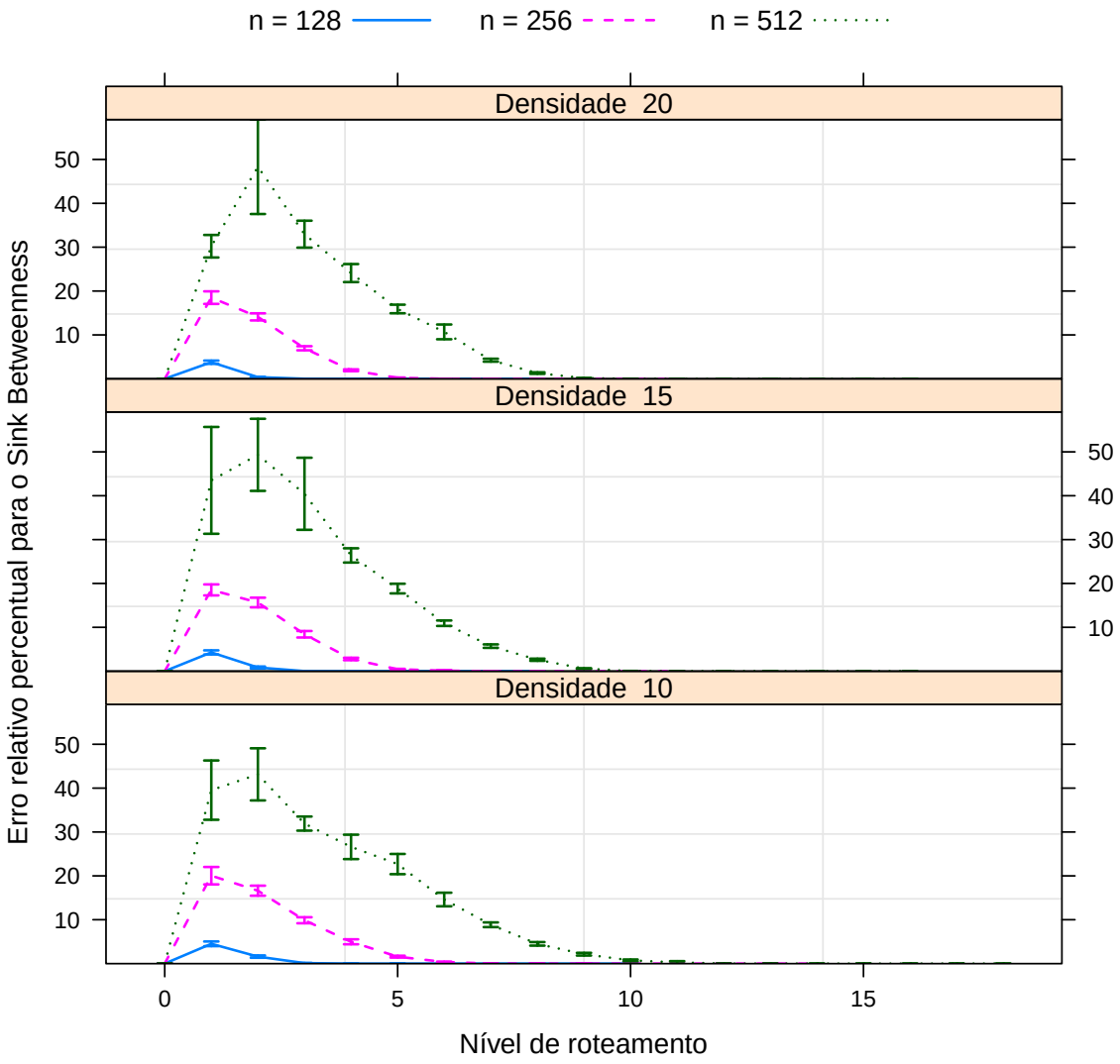

Figura A.2. Erro percentual relativo quando se limita em 30 elementos o tamanho do array *sonsPaths* contido no pacote *Border*

em nós mais distantes do *sink* são piores para a qualidade da árvore do que aqueles encontrados em nós próximos ao *sink*. Uma vez que cada nós faz a escolha do próximo salto baseado nas centralidades dos vizinhos e, caso estes valores não estejam precisos, uma escolha não apropriada de próximo salto tende tem maior possibilidade de se propagar quanto maior for a distância para o *sink*. Uma vez que este problema é tratado como trabalho futuro, soluções que avaliem, por exemplo, o comportamento do crescimento da frequência dos caminhos mínimos podem trazer melhores resultados.

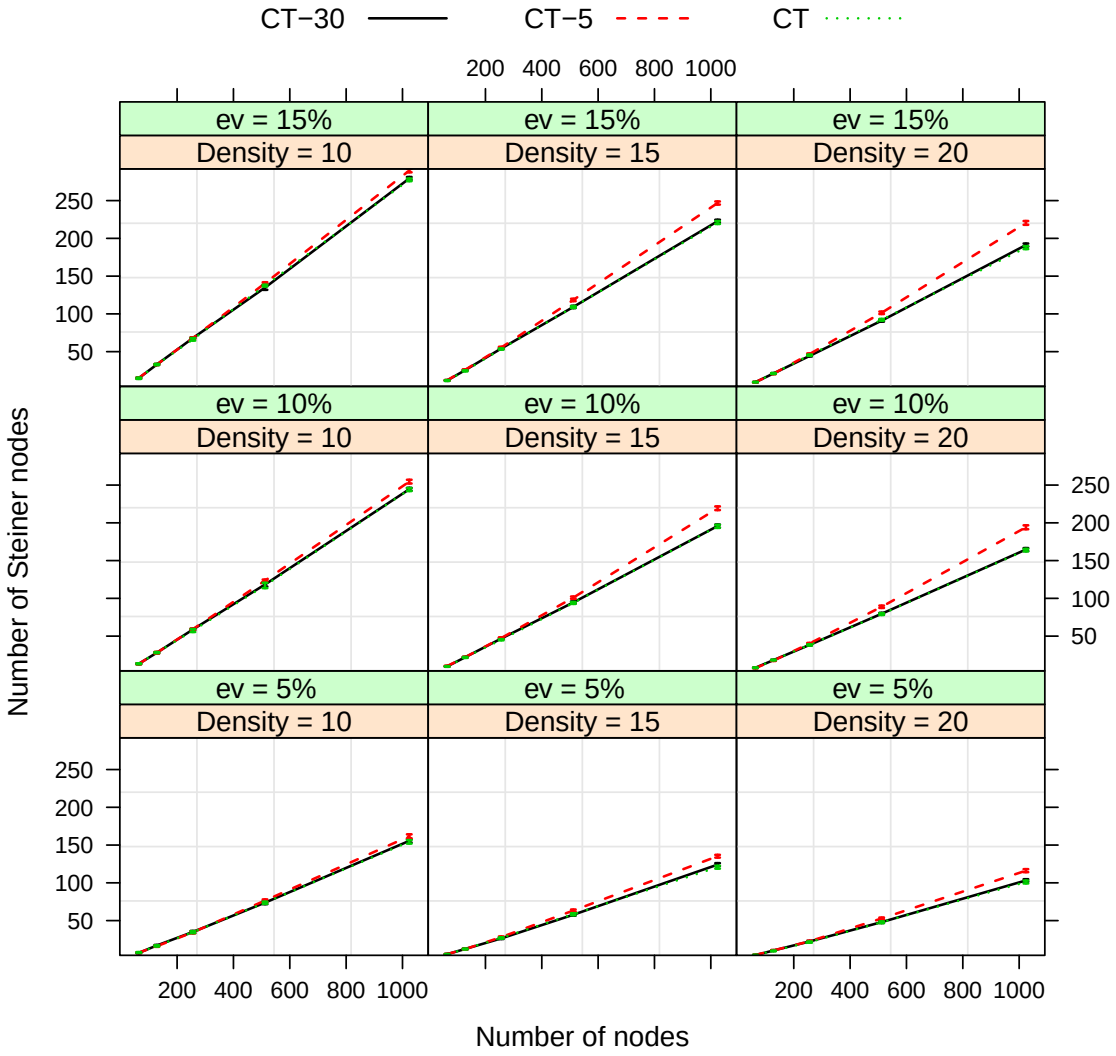

Figura A.3. Número de nós *Steiner* presentes na árvore criada pelo CT com limites impostos ao array *sonsPaths* contido no pacote *Border* em 5 e 30 elementos