UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PROTOCOLO CIENTE DE CORRELAÇÃO ESPACIAL PARA REDES DE SENSORES SEM FIO

GILMAR FAVARIN

ORIENTADORA: PROFA. DRA. REGINA BORGES DE ARAUJO
CO-ORIENTADOR: PROF. DR. CESAR A. MARCONDES

São Carlos – SP

Maio/2011

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PROTOCOLO CIENTE DE CORRELAÇÃO ESPACIAL PARA REDES DE SENSORES SEM FIO

GILMAR FAVARIN

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Processamento de Imagens e Sinais
orientadora: Profa. Dra. Regina Borges de Araujo

São Carlos – SP

Maio/2011

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

F272pc

Favarin, Gilmar.
Protocolo ciente de correlação espacial para redes de sensores sem fio / Gilmar Favarin. -- São Carlos : UFSCar, 2012.
91 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2011.

1. Redes de computação - protocolos. 2. Protocolo de roteamento. 3. Redes de sensores sem fio. 4. Correlação espacial. 5. Roteamento plano. I. Título.

CDD: 004.62 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Protocolo Ciente de Correlação Espacial para Redes de Sensores Sem Fio"

GILMAR FAVARIN

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Profa. Dra. Regina Borges de Araujo
(Orientadora - DC/UFSCar)

Prof. Dr. Jander Moreira
(DC/UFSCar)

Prof. Dr. Cesar Augusto Cavalheiro Marcondes
(DC/UFSCar)

Prof. Dr. Eleri Cardozo
(DCA/FEEC/UNICAMP)

São Carlos
Junho/2011

Dedico este trabalho:

Aos meus queridos pais, Isabel e Osvaldo, por terem me dado educação, amor e carinho, por me ensinarem a ter fé em Deus, e sempre acreditar nos meus sonhos.

À minha querida irmã, Karina, amor da minha vida, por sua paciência, incentivo, companheirismo, dedicação e amor.

Ao meu querido irmão, Luiz Henrique, por sempre estar ao meu lado em todas as horas, principalmente nas mais difíceis.

AGRADECIMENTOS

A Deus, por ter me dado a vida, saúde e proteção para poder lutar pelos meus objetivos.

Aos meus pais Isabel e Osvaldo, irmãos Karina e Luiz Henrique, e familiares por serem sempre meu alicerce, apoiando-me e incentivando a conquistar meus sonhos.

À prof^a. Dr^a. Regina Borges de Araújo, pela oportunidade de poder crescer profissionalmente. Pelas palavras de motivação, encorajamento, amizade e discussões que contribuíram para a realização deste mestrado.

Aos meus amigos Du Kobayashi, Guto, Rafa, Baiano, Hiro, Josué, Mateus, Juciara e Maísa, pelo companheirismo, amizade e apoio nas horas difíceis de estudo incessante, e me ouvindo sempre para aliviar o estresse.

Aos professores do Departamento de Computação da UFSCar, principalmente aos professores Cesar Marcondes, Sandra, Trevelin, Ricardo, Helena, Mascarenhas, Estevam e à Cristina que tive a oportunidade de conhecer melhor durante as reuniões do conselho e no decorrer da pós-graduação.

Aos membros da banca examinadora pela atenção nas sugestões e julgamento do trabalho.

A alegria está na luta, na tentativa, no sofrimento envolvido e não na vitória propriamente dita.

Mahatma Gandhi

RESUMO

Redes de Sensores Sem Fio (RSSFs) estão sendo cada vez mais utilizadas na vida diária das pessoas em aplicações que incluem desde monitoramento de gasto de energia em residências e prédios em geral, até monitoramento de sinais vitais para medicina assistida, monitoramento de infraestruturas físicas, vazamentos de produtos químicos ou biológicos em indústrias, vigilância para melhoria de segurança, monitoramento ambiental, dentre inúmeras outras.

RSSFs podem ser implantadas em diferentes densidades podendo chegar a milhares de nós. No entanto, o desenvolvimento de soluções baseadas em RSSFs é limitado, principalmente, por recursos restritos dos nós sensores, em especial recursos energéticos. O grande desafio de soluções para RSSFs é aumentar a longevidade da rede e, ao mesmo tempo, garantir a entrega, confiabilidade e precisão dos dados coletados diante de um ambiente propício a falhas de diferentes tipos. A maior fonte de consumo de energia é a transmissão de mensagens. Assim, soluções de RSSF têm que evitar comunicação intensa, mantendo o balanceamento do consumo de energia e, assim, a longevidade da rede.

Em aplicações onde é necessária alta densidade de nós sensores, o processo de sensoriamento pode produzir grande quantidade de dados similares ou redundantes devido à proximidade espacial entre esses nós. Esta proximidade espacial pode ser explorada em soluções de roteamento para reduzir a quantidade de mensagens transmitidas pela rede.

Este trabalho apresenta o algoritmo de roteamento SCARP (*Spatial Correlation Aware Routing Protocol*), que faz uso da correlação espacial para reduzir o número de transmissões pela rede. Com o SCARP, a RSSF é configurada em células e nós de cada célula são escolhidos, de maneira alternada, para transmitir dados similares ou redundantes, reduzindo assim o número de mensagens transmitidas. Essa redução de tráfego resulta em menor consumo de energia e maior longevidade da rede. Resultados de avaliação de desempenho mostram que SCARP supera soluções semelhantes descritas na literatura como o DAARP, que utiliza clusterização e agregação de dados, e mantém o desempenho positivo mesmo em situações de grande densidade de nós.

Palavras-chave: Algoritmo de Roteamento, Rede de Sensores Sem Fio, Correlação Espacial

ABSTRACT

The usage of wireless sensor network is increasingly being applied to people's everyday lives everywhere: from energy consumption in households and buildings in general, to vital signs in assistive medicine, infrastructure monitoring, chemical or biological product leaking detection in industries, better surveillance, environmental monitoring, among many others.

WSN can be deployed in different densities next to several thousands of nodes. However, the development of WSN solutions are limited mainly by energy resource restriction. The great challenge to WSN solutions is to increase the network longevity while guaranteeing data delivery, reliability and accuracy in an environment prone to different types of failures. The largest source of energy consumption is data transmission. Thus, solutions to WSN needs to avoid intense communication keeping energy consumption balance and so the network longevity.

In applications in which high density of nodes is necessary, sensing process can produce a large amount of data which are similar or redundant, due to the special proximity among the nodes. This spatial proximity can be explored in routing solutions to reduce the amount of messages transmitted throughout the network.

This work presents the *Spatial Correlation Aware Routing Protocol - SCARP* , which makes use of spatial correlation to reduce the number of network transmissions. With SCARP, the WSN is configured in cells and nodes of each cell are selected, in an alternated way, to transmit similar or redundant data, and so reducing the number of transmitted messages. This traffic reduction results in less energy consumption and longer network longevity. Evaluation results show that SCARP outperforms similar solutions described in the literature, such as DAARP, which uses clustering and aggregation. SCARP has a positive performance even for large node density scenarios.

Keywords: Routing algorithm, Wireless sensor network, spatial correlation, flat routing

LISTA DE FIGURAS

2.1 Componentes do nó sensor sem fio. 20

2.2 Consumo de energia dos componentes em um nó sensor. 23

2.3 Modelos de posicionamento em RSSF 27

3.1 Visualização das áreas de correlação dentro do evento. 45

3.2 Consumo de energia na coleta de dados. 46

4.1 Construção da árvore de saltos. 50

4.2 Seleção dos candidatos a Nós de borda. 52

4.3 Nós de borda eleitos. 53

4.4 Rede dividida em células. 54

4.5 Média aritmética do consumo de energia de todos os nós da rede. 56

4.6 Total de mensagens de controle transmitidas pela rede. 57

4.7 Total de pacotes de dados transmitidos pela rede. 58

4.8 Total de pacotes de dados recebidos pelo *sink*. 59

LISTA DE TABELAS

2.1 Comparação Redes de Sensores Sem Fio x Redes *AdHoc* Sem Fio 19

2.2 Caracterização das RSSFs quanto à configuração 30

2.3 Caracterização das RSSFs quanto ao sensoriamento 31

2.4 Caracterização das RSSFs quanto à comunicação 31

4.1 Parâmetros da simulação 55

LISTA DE ALGORITMOS

1 Construção da Árvore de Saltos 50
2 Descoberta dos Nós do Último Salto 51

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO **13**

1.1 Motivação 14

1.2 Objetivos 15

1.3 Organização do Trabalho 16

CAPÍTULO 2 – REDES DE SENSORES SEM FIO **17**

2.1 Componentes do nó sensor 20

2.2 Características de RSSFs 21

2.2.1 Consumo de Energia 22

2.2.2 Consumo de Energia com o Sensoriamento 22

2.2.3 Consumo de Energia com a Comunicação 22

2.2.4 Consumo de Energia com o Processamento 23

2.2.5 Auto-gerenciamento em RSSFs 24

2.2.6 Tolerância a Falhas 24

2.2.7 Cobertura do Ambiente Monitorado 25

2.2.8 Posicionamento dos Nós Sensores 25

2.2.9 Área de Cobertura 27

2.2.10 Segurança em RSSF 27

2.2.11 Escalabilidade 29

2.2.12 Classificação das RSSFs 29

2.2.13 Topologia da Rede 32

2.3 Arquitetura de Comunicação em RSSFs 33

2.3.1 Camada Física 33
2.3.2 Camada Enlace 34
2.3.3 Camada de Rede 35
2.3.4 Camada de Transporte 36
2.3.5 Camada de Aplicação 37
2.3.6 Cross-Layer 37
2.4 Considerações Finais 38

CAPÍTULO 3 – PROTOCOLOS DE ROTEAMENTO PARA RSSFS **39**

3.1 Protocolos Centrados em Dados (*data-centric*) 41
3.2 Roteamento Plano 41
3.3 Roteamento Hierárquico 41
3.4 Roteamento Baseado na Localização 42
3.5 Agregação de Dados 43
3.6 Correlação de dados 44
3.7 Considerações Finais 46

CAPÍTULO 4 – O PROTOCOLO SCARP **47**

4.1 Descrição do protocolo SCARP (*Spatial Correlation Aware Routing Protocol*) . 47
4.1.1 SCARP Fase 1 - Construção da árvore de saltos 49
4.1.2 SCARP Fase 2 - Eleição dos nós de borda 51
4.1.3 SCARP Fase 3 - Divisão da rede em células 53
4.2 Avaliação de Desempenho 53
4.2.1 Cenário da Simulação 54
4.2.2 Resultados da Simulação 55
4.2.3 Considerações Finais 59

CAPÍTULO 5 – CONCLUSÕES **61**

5.1 Trabalhos Futuros 62

REFERÊNCIAS BIBLIOGRÁFICAS **63**

GLOSSÁRIO **70**

ANEXO A – CÓDIGO DO SCARP NO SINALGO **71**

ANEXO B – CÓDIGO DAS MENSAGENS NO SINALGO **83**

ANEXO C – CÓDIGO DOS TIMERS NO SINALGO **89**

Capítulo 1

INTRODUÇÃO

Com a crescente inovação tecnológica dos sistemas microeletromecânicos ("*Micro-Electro-Mechanical Systems*", MEMS) e da comunicação sem fio, os nós sensores estão menores e mais baratos que os sensores comuns (YICK; MUKHERJEE; GHOSAL, 2008), tornando sua relação custo-benefício mais vantajosa.

Os nós sensores sem fio, conhecidos como "sensores inteligentes", são compostos por módulos de sensoriamento, processamento e de comunicação sem fio (AKYILDIZ et al., 2002) (YICK; MUKHERJEE; GHOSAL, 2008) (ESTRIN et al., 1999). Em sua estrutura estão acoplados um rádio de comunicação e vários tipos de sensores como, por exemplo, térmico, sísmico, acústico ou magnético, entre outros (YICK; MUKHERJEE; GHOSAL, 2008).

Tais sensores estão ligados a processos físicos, químicos e biológicos. Com essa composição, os nós sensores permitem um monitoramento remoto do ambiente mais preciso do que técnicas de instrumentação tradicionais (PINTO, 2004). Afinal, eles são capazes de sentir diferentes fenômenos no ambiente, como temperatura, presença, movimento, umidade, pressão, ruídos, entre outros. E promover a disseminação dos dados coletados de diferentes sensores através de uma rede sem fio.

Dessa maneira, ao serem depositados em uma área, são capazes de perceber o ambiente através de seus sensores e se comunicarem seguindo um propósito definido pela aplicação. Com o propósito de monitoramento do ambiente em uma determinada área, o agrupamento de nós sensores caracteriza uma Rede de Sensores Sem Fio (RSSF), projetada para utilizar o máximo possível dos recursos disponíveis (energia, raio de comunicação, largura da banda, armazenamento e processamento), pois eles são restritos.

Para atingir esse propósito, os projetos de RSSF são baseados no ambiente monitorado, o que é fundamental para se determinar o tamanho e a topologia da rede e o modo de deposição (YICK; MUKHERJEE; GHOSAL, 2008). O monitoramento confiável do ambiente é essencial no desenvolvimento do projeto de uma aplicação para RSSF, especialmente quando há risco de

ocorrerem situações de emergência (AKYILDIZ et al., 2002).

Visando que o monitoramento do ambiente físico usando RSSF seja feito com precisão, protocolos adequados devem ser utilizados, mantendo a segurança, confiabilidade e eficiência das informações transmitidas, além de garantir maior vida útil para esses nós sensores. Portanto, ao desenvolver uma solução para RSSF, o consumo de energia deve ser eficiente para garantir que o tempo de vida da rede seja adequado às necessidades da aplicação.

Para monitorar a região de interesse de pequena ou grande dimensão, pode ser necessária uma alta densidade de nós. Em alguns casos, a densidade pode chegar em torno de 20 nós por metro quadrado (SHIH et al., 2001). Independente do grau de densidade dentro da área de rede, a proximidade entre os nós resulta na probabilidade de ocorrer a captura de informações redundantes, análogas ou semelhantes. Esta coleta redundante ocorre em um mesmo domínio espacial, ou seja, são espacialmente relacionadas. Referindo ao espaço, o grau de correlação aumenta de acordo com a diminuição da distância entre os nós sensores (AKYILDIZ; VURAN; AKAN, 2004).

Neste trabalho, para a disseminação dos dados ocorridos dentro do ambiente monitorado por uma RSSF até o nó sorvedouro (*sink*)[1], foi usada a correlação espacial como a abordagem que pode superar técnicas tipicamente utilizadas de agregação/fusão de dados.

1.1 Motivação

RSSFs apresentam limitações em termos de energia, capacidade de processamento e comunicação. Sendo assim, o projeto de soluções que otimizem os recursos de RSSFs, garantindo a entrega confiável dos dados apresenta grandes desafios. A correlação espacial vem sendo utilizada como uma técnica de redução de tráfego na disseminação de dados na rede, garantindo a confiabilidade dos dados entregues ao *sink* (AKYILDIZ; VURAN; AKAN, 2004) (VURAN; AKAN; AKYILDIZ, 2004).

O uso de correlação espacial motivou a especificação de um algoritmo que permitisse o consumo eficiente de energia nas RSSF.

Pesquisas sobre correlação espacial têm apresentado resultados significativos com relação à diminuição de tráfego e garantia de entrega em RSSFs, principalmente quando há alto grau de densidade. É possível eleger nós representativos dentro das áreas onde ocorrem os eventos, e fazer com que apenas esses nós, transmitam para o *sink*, sem a necessidade de grande troca de informações dentro da área do evento. Essa abordagem diminui a transmissão excessiva pelos sensores dentro da área do evento. Tal economia de comunicação torna menor o consumo de energia. É justamente essa perspectiva de aumento de longevidade de RSSFs, com o uso de

[1] O nó *sink* é robusto, não tem as restrições que os nós sensores têm.

correlação espacial, a grande motivação para este trabalho.

1.2 Objetivos

O objetivo principal deste trabalho é desenvolver uma solução de roteamento e disseminação de dados, que utilize árvore de saltos e a abordagem de correlação espacial para garantir longevidade à RSSF, bem como avaliar este trabalho com relação às seguintes métricas:

- escalabilidade: capacidade de expandir a rede sem perder desempenho. A rede deve manter-se em pleno funcionamento mesmo em altas densidades;
- taxa de entrega de pacotes de dados: é a quantidade de pacotes que chegam ao nó *sink*, indicando a qualidade na disseminação dos dados e do roteamento até o *sink*. A quantidade de pacotes entregues ao *sink* deverá ser maior com o consumo de energia menor;
- *overhead*: o *overhead* é a quantidade de mensagens de controle usadas para qualquer tipo de configuração como criação de rotas, definição de *cluster-heads* e nós representativos;
- eficiência: no consumo de energia.

Um segundo objetivo deste trabalho é comparar a solução desenvolvida com o DAARP (*Data-Aggregation Aware Routing Protocol*), uma solução implementada por um ex-aluno de mestrado do laboratório WINDIS (*WIreless Networking and Distributed Interactive SImulation Laboratory*), que hoje está sendo estendida como parte de trabalho de doutorado do mesmo aluno no DCC da UFMG. O DAARP é de interesse porque implementa roteamento e agregação de forma ótima, utilizando árvore de Steiner (IVANOV, 1994) e foi extensamente avaliado tanto em termos teóricos como em simulação (VILLAS et al., 2009).

O interesse pelo DAARP é demonstrar as vantagens no uso de correlação espacial sobre a agregação de dados, e sem a necessidade de um roteamento complexo para garantir um mínimo de agregação.

Outro objetivo é mostrar que, diferente das soluções existentes na literatura, que utilizam a correlação espacial como abordagem na disseminação dos dados e sempre utilizam roteamento hierárquico que necessita de mais mensagens de controle na configuração da rede, optei pelo roteamento plano, pois é uma solução com menor complexidade de processamento nos nós sensores e de fácil implementação.

1.3 Organização do Trabalho

Este documento está organizado da seguinte maneira: o Capítulo 2 exibe com detalhes as capacidades das Redes de Sensores Sem Fio e como essas são capazes de realizar o monitoramento do ambiente e fornecer as informações a um sorvedouro (*sink*).

No Capítulo 3 será detalhado como funciona o roteamento e disseminação de dados em RSSFs. O Capítulo 4 é responsável por delinear como o projeto foi desenvolvido, a metodologia utilizada nas pesquisas, a forma de avaliação do projeto e, por fim, os resultados obtidos. O Capítulo 5 apresenta conclusões deste trabalho e propostas futuras.

Capítulo 2

REDES DE SENSORES SEM FIO

Com a tecnologia dos sistemas microeletromecânicos, o mercado cada vez mais volta sua atenção para as Redes de Sensores Sem Fio, menor e mais barata do que a de sensores comuns e, portanto, com melhor custo-benefício (YICK; MUKHERJEE; GHOSAL, 2008).

Contudo, apesar desses benefícios, as RSSFs têm capacidades energética, de processamento e de armazenamento de dados limitadas, cabendo aos projetistas superar tais desafios por meio do projeto de protocolos e algoritmos, o que garante que os dados do ambiente sob monitoramento sejam coletados e entregues de forma confiável até o(s) *sink*(s).

Segundo Akyildiz et al. (2002) a rede de sensores é, normalmente, composta por um grande número de nós sensores colocados de forma determinística ou não no ambiente que se deseja monitorar. A forma não determinística permite o monitoramento de ambientes inacessíveis, como operações de resgate ou florestas com mata fechada. Por outro lado, a forma determinística pode ser usada em construções concretas, para se monitorar locais com risco de acidentes, para prever emergências e perceber mudanças nesse ambiente.

Portanto, os protocolos e algoritmos de rede de sensores devem possuir a capacidade de auto-organização, uma vez que, ao serem depositados de forma não determinística no ambiente, podem existir áreas mais densas (com sensores mais próximos entre si) e outras com sensores mais afastados.

O protocolo ou algoritmo da RSSF não deve prejudicar o consumo de energia com muitas comunicações nas áreas de maior densidade e, ao mesmo tempo, deve conseguir manter a conectividade com os sensores mais afastados, criando rotas alternativas. O esforço cooperativo é uma característica dos nós sensores.

Cada nó possui um processador, mas com baixa capacidade de processamento, primeiro pela essencial necessidade de consumir o mínimo de sua bateria e também por causa de seu tamanho, normalmente minúsculo. O resultado é que não podem processar muita informação. Portanto, de acordo com o protocolo ou algoritmo desenvolvido para aquele tipo de aplicação,

os nós fazem um processamento simples das informações sentidas pelos sensores ou recebidas pela rede.

Em se tratando de rede de sensores, é possível criar inúmeros tipos de aplicações em diferentes campos (AKYILDIZ et al., 2002), como, medicina, meio-ambiente, aplicações estruturais, área militar e em residências ou empresas (LOUREIRO et al., 2003).

Alguns exemplos de aplicações são:

- controle: controlam linhas de montagem em um processo de fabricação, sendo embutidos sensores nas "peças" ou equipamentos, tornando possível coletar informações durante todo o processo (*Ibidem*);
- meio-ambiente (AKYILDIZ et al., 2002): monitoram variáveis ambientais (MAINWARING et al., 2002) tanto em locais internos, como indústrias e residências (YICK; MUKHERJEE; GHOSAL, 2008), quanto em ambientes externos (florestas, oceanos, vulcões) (AGRE; CLARE, 2000). Nesses locais, podem ser monitorados os movimentos de pequenos animais como pássaros, informações de fatores que possam afetar colheitas e plantios (previsão de combate à geada e a pragas, detecção de componentes químicos ou biológicos, controle de irrigação) (CERPA et al., 2001), mapeamento da bio-complexidade ambiental (como o estudo da poluição de rios, de cidades) (KAHN; KATZ; PISTER, 1999);
- medicina (BULUSU et al., 2001): monitoram órgãos do corpo humano podendo detectar substâncias nocivas ou de indicadores do estado de saúde; criam uma interface para deficientes; monitoram pacientes e médicos dentro de um hospital (SCHURGERS; SRIVASTAVA, 2001);
- tráfego: monitoram o tráfego de veículos dentro de uma cidade ou numa rodovia, ou de pessoas dentro de um shopping ou uma residência (LOUREIRO et al., 2003);
- aplicações militares (AKYILDIZ et al., 2002): monitoramento de tropas; monitoramento de disparos com armas de fogo; reconhecimento de terreno; detecção de alvos e da presença de material perigoso como gás venenoso ou radioativo (YICK; MUKHERJEE; GHOSAL, 2008);
- monitoramento de estrutura/equipamentos (XU et al., 2004): monitoram e identificam falhas em grandes estruturas (pontes e prédios); monitoram desgastes ou defeitos de máquinas e equipamentos (KIM, 2011), como motores e dutos de gás (KARP; KUNG, 2000);
- aplicações comerciais (WARNEKE et al., 2001): automatizam o processo de vendas e de fabricação; realizam a manutenção de inventário, monitoram qualidade de produtos; vigiam veículos e estabelecimentos (ESTRIN et al., 1999).

Para o funcionamento dessas aplicações, uma rede de sensores deve ser sem fio, mas os protocolos e algoritmos desenvolvidos para redes *AdHoc* sem fio não são adequados para aplicações em uma RSSF (AKYILDIZ et al., 2002), como demonstra a Tabela 2.1 (ALBUQUERQUE, 2007).

Tabela 2.1: Comparação Redes de Sensores Sem Fio x Redes *AdHoc* Sem Fio

Características	RSSF	*AdHoc* Sem Fio
Quantidade de nós	Centenas a milhares de nós sensores.	Dezenas até centenas de nós na rede.
Densidade da rede	Suporta alta densidade na rede.	Suporta alta densidade na rede.
Tolerância a falhas	Permite perder nós sensores durante o funcionamento da rede e continuar funcionando sem danos graves.	Permite perder nós da rede durante o seu funcionamento e continuar funcionando sem danos graves.
Topologia	Dinâmica, muda com frequência.	Dinâmica, muda com frequência.
Tipo de comunicação	Por *broadcast*.	Ponto a ponto.
Energia disponível	Limitada, restrita, de difícil reposição.	Limitada, depende do dispositivo usado, possibilidade de recarga.
Capacidade de processamento e armazenamento	Restrita.	Geralmente, são usados *laptops* que têm bom poder computacional e de armazenamento.
Roteamento	Centrado em dados.	Centrado em endereços.
Tipos de protocolos de roteamento	Planos, hierárquicos e geográficos.	Reativos, pró-ativos e híbridos.

Redes *AdHoc* sem fio não dependem de nenhum tipo de infraestrutura e podem ter, ou não, nós móveis. Toda a comunicação é feita ponto-a-ponto, ou seja, diretamente entre nós. Dadas as suas peculiaridades, este tipo de rede requer o desenvolvimento de protocolos de roteamento específicos (*Ibidem*).

A RSSF, por outro lado, deve possuir um ou mais nós de escoamento de dados, chamados de sorvedouros (*sinks*) ou estações-base, de grande poder computacional e sem restrições energéticas. É por meio dele que a rede se comunica com outras redes ou com um ou mais observadores (RUIZ; NOGUEIRA; LOUREIRO, 2004).

Isso significa que o sorvedouro faz a interface entre a aplicação e a rede, servindo de ponto de entrada para a submissão dos interesses da aplicação e de concentrador das informações coletadas e enviadas pelos nós sensores.

2.1 Componentes do nó sensor

Em Akyildiz et al. (2002), o sensor é descrito como sendo composto por quatro componentes básicos: unidade de sensoriamento, unidade de processamento, unidade de transmissão e unidade de energia. Dependendo da necessidade, ele pode ter também um sistema de localização, um gerador de energia externa ou uma unidade de movimentação, como mostra a Figura 2.1.

Todos esses elementos devem caber no nó. Para isso, além do tamanho exíguo, devem trabalhar consumindo o mínimo de energia. Também devem operar autonomamente e se adaptarem facilmente a diferentes ambientes.

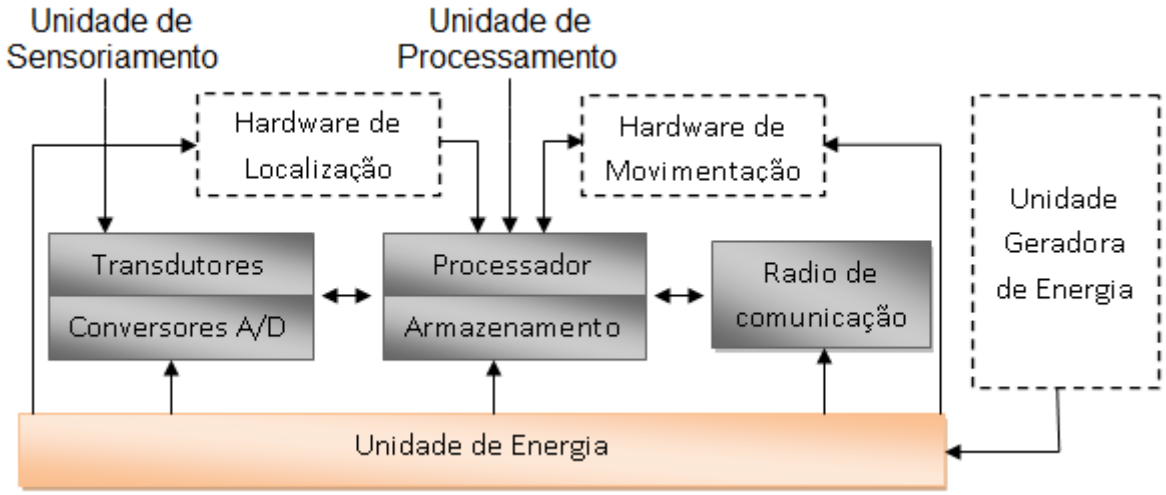


Figura 2.1: Componentes do nó sensor sem fio.

Os transdutores e os conversores A/D são as subunidades da unidade de sensoriamento. Os dados analógicos sentidos no ambiente são convertidos em informações digitais e enviados à unidade de processamento e armazenamento.

A unidade de processamento, além de processar toda informação, possui uma pequena unidade de armazenamento. A unidade de armazenamento é responsável pelo armazenamento das informações sentidas pela unidade de sensoriamento, recebidas pelo rádio de comunicação ou processadas pelo processador.

A unidade de energia é composta por células de energia e, em alguns casos, pode haver também uma unidade de geração de energia acoplada.

O rádio de comunicação, é uma unidade de transmissão e recepção responsável pela comunicação na camada física da rede.

Outras subunidades podem ser incorporadas ao nó sensor dependendo da aplicação e do interesse, como por exemplo, um hardware de localização, ou hardware para a movimentação do nó. O hardware de localização é de grande utilidade e, por isso, de ampla utilização, já que uma parte das técnicas de roteamento conhecidas requer informação de localização. Mas existem outras técnicas de localização e uma delas é o uso da intensidade do sinal de rádio.

Normalmente, todas essas unidades devem formar um dispositivo de tamanho reduzido, que podem chegar próximos a um centímetro cúbico. Outros requisitos para os nós sensores são: mínimo consumo de energia, baixo custo de produção, autonomia e adaptação ao ambiente.

Apesar de os nós sensores possuírem individualmente pouca capacidade computacional, de armazenamento e de energia, um esforço colaborativo entre eles permite a realização de uma tarefa maior (RUIZ; NOGUEIRA, 2003).

Em alguns casos, uma RSSF também pode ser composta de dispositivos atuadores que permitem ao sistema controlar parâmetros do ambiente monitorado.

2.2 Características de RSSFs

As características de RSSFs derivam das limitações tecnológicas dos dispositivos (AKYILDIZ et al., 2002). As restrições das aplicações também fazem com que essas redes compartilhem algumas características. Neste tópico, serão apresentadas as características e requisitos que têm impacto direto na arquitetura e nas decisões de projeto de uma RSSF (RUIZ et al., 2004).

Redes de sensores sem fio são diferentes de redes tradicionais pelo fato de que cada aplicação tem seu próprio projeto. Para desenvolvê-lo, levam-se em conta várias características cruciais para o bom funcionamento da rede:

- deposição dos nós sensores;
- topologia da rede;
- capacidades de armazenamento, processamento e comunicação (KARL; WILLIG, 2007);
- algoritmos de roteamento, feitos sob medida para sua aplicação[1];

[1]Com isso, retira-se aquela rígida abstração existente entre as camadas de rede tradicionais, enquanto a maioria dos protocolos de redes tradicionais usa o modelo de camadas OSI, em RSSFs é usado um conjunto de camadas "otimizado", o que será explicado com mais detalhes na seção 2.3.

- alta escalabilidade, de modo que o custo de cada nó sensor é importante para viabilizar uma rede de sensores[2].

2.2.1 Consumo de Energia

O tamanho reduzido de um nó sensor impede que tenha uma bateria grande. Por isso, ele dispõe de pouca energia durante sua vida útil. Em Akyildiz et al. (2002) e Estrin, Sayeed e Srivastava (2002), são descritos os três padrões de consumo que os sensores seguem:

- **sensoriamento:** ao fazer o sensoriamento, o nó sensor está sentindo o ambiente contido em sua área de cobertura;
- **processamento:** processando qualquer informação;
- **comunicação:** ao realizar a comunicação com outros nós, essa comunicação engloba transmissão e recebimento de qualquer informação.

Em vista dessas características, protocolos e algoritmos para RSSFs têm que ser sensíveis ao consumo de energia (AKYILDIZ et al., 2002). Nos tópicos seguintes, os padrões de consumo ora apresentados serão analisados com mais detalhes.

2.2.2 Consumo de Energia com o Sensoriamento

O sensoriamento de um nó é a unidade responsável por medir alguma característica analógica existente no ambiente e converter em sinais digitais. O consumo de energia de uma unidade de sensoriamento varia de acordo com a natureza da aplicação (AKYILDIZ et al., 2002).

2.2.3 Consumo de Energia com a Comunicação

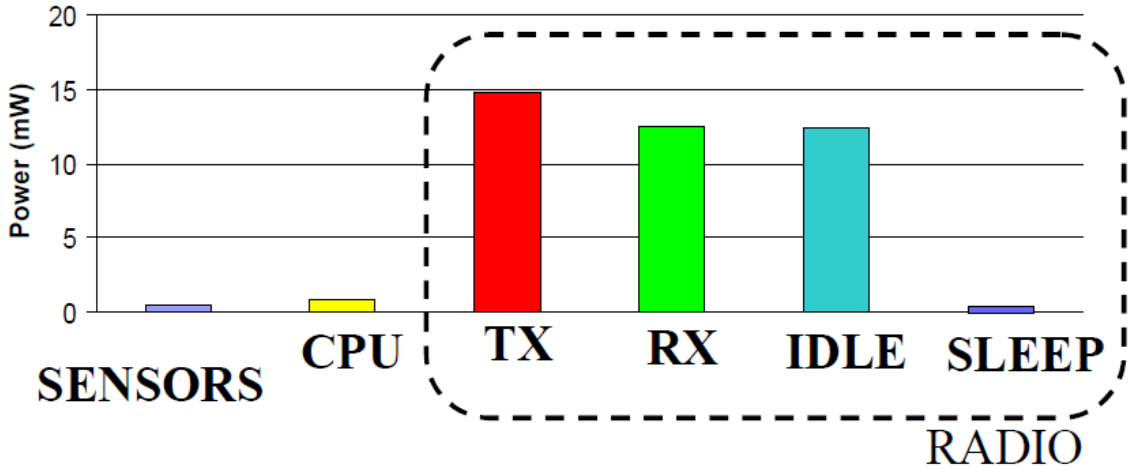
Dentre os três padrões citados anteriormente, a comunicação dos nós sensores para transmitir as informações é o responsável pela maior parte do consumo da energia em um nó sensor (BOUKERCHE; CHENG; LINUS, 2003). Para comunicações em pequenas distâncias, a quantidade de energia que se gasta para transmissão e recepção é quase a mesma, aparecendo aí o ponto principal que deve ser atacado para se desenvolver uma solução com consumo eficiente de energia.

Todos os componentes do circuito do *transceiver* consomem boa parte da energia do sistema, em especial o PLL (*Phase Locked Loop*). Quando se menciona gasto médio de energia,

[2] O custo de um nó sensor não pode ser maior que o custo de um sensor tradicional para que a implantação de uma rede dessas possa ser viável (AKYILDIZ et al., 2002).

normalmente não se leva em conta o gasto no início da transmissão em que o PLL está ativo e o consumo de energia nesta fase é maior.

Como em uma RSSF o tamanho dos pacotes de dados é tipicamente pequeno (ESTRIN; SAYEED; SRIVASTAVA, 2002), esse consumo inicial passa a dominar sobre o consumo total do sistema de transmissão (SOHRABI et al., 2000). Assim, esse gasto deve ser também levado em consideração, que é quando o PLL está ativo, mas sem envio de informação (mostrado como ocioso no gráfico), como mostrado na Figura 4.5 (BOUKERCHE; CHENG; LINUS, 2003).



$$E_{TX} \approx E_{RX} \approx E_{IDLE} >> E_{SLEEP}$$

Figura 2.2: Consumo de energia dos componentes em um nó sensor.
Fonte (ESTRIN; SAYEED; SRIVASTAVA, 2002)

2.2.4 Consumo de Energia com o Processamento

Já no processamento dos dados, mesmo diante do contínuo surgimento de novos processadores cada vez mais poderosos, tem-se o fator consumo de energia como fundamental (AKYILDIZ et al., 2002).

A otimização do consumo de energia em RSSFs é complexa, pois não se limita a diminuir o consumo de um nó sensor, e, sim, de toda rede, de forma a prolongar o tempo de vida útil da RSSF como um todo (QI; WANG; IYENGAR, 2001). Para atingir esse objetivo, é necessário que o controle de consumo de energia seja incorporado a todos os estágios de seu projeto e operação (RAGHUNATHAN et al., 2002).

Devido à escassez das fontes de energia e à dificuldade de serem substituídas, métodos de economia de energia devem ser utilizados em todas as oportunidades possíveis. É justamente por isso que muitas pesquisas têm sido feitas para melhorar os algoritmos responsáveis pela transmissão e escoamento de dados na rede, além da criação de novos algoritmos de roteamento

mais eficientes quanto à utilização de energia e até o desligamento dos módulos de comunicação dos nós em certos momentos.

2.2.5 Auto-gerenciamento em RSSFs

Após a deposição dos nós sensores dentro da área de rede, eles iniciam seu funcionamento, o qual deve ocorrer de maneira autônoma, por ser essa uma das características das RSSFs. Esse comportamento pode causar mudanças na topologia da rede devido aos seguintes fatores:

- falhas em alguns nós, em decorrência de defeitos ocorridos depois de sua deposição;
- falta de energia depois de um tempo de funcionamento;
- nó em estado de "*sleep*", como resultado de algoritmo de otimização de nós sobressalentes (em maior número que o mínimo necessário para garantia de cobertura) que são colocados em modo "*sleep*" para economia de energia;
- falhas na comunicação, e outras.

Essa característica de auto-organização tem um papel importante nos nós sensores, pois faz com que o nó tome suas próprias decisões para que a rede continue em funcionamento. Se isso não ocorresse, poderia ser perdida uma região inteira da área de rede.

Algumas decisões que podem ser tomadas autonomamente pelo sensor incluem:

- cortar totalmente a comunicação;
- não responder apenas a alguns tipos de requisição.

Definir a medida ideal dentre essas duas depende de como a aplicação funciona e quais são seus interesses.

2.2.6 Tolerância a Falhas

Tolerância a falhas é a capacidade de manter todas as funcionalidades da rede mesmo quando alguns nós falham (HOBLOS; STAROSWIECKI; AITOUCHE, 2000), mantendo a confiabilidade da rede (SHEN; SRISATHAPORNPHAT; JAIKAEO, 2001). As falhas estão sempre presentes nas RSSFs (AKYILDIZ et al., 2002) e acontecem com determinada frequência (ASSUNÇÃO, 2007) .

De acordo com os autores supracitados, o nó sensor pode apresentar falhas por diversos motivos, tais como falta de energia, defeitos físicos no hardware, interferência no sinal dificultando o recebimento ou envio de qualquer informação.

Isso demonstra que a falha ocorrida em um nó sensor dentro da área de rede pode não afetar o funcionamento da rede como um todo. Por isso, a falha tem que ser detectada rapidamente e os nós vizinhos devem resolver o problema autonomamente. As severas restrições de consumo de energia tornam inadequadas as técnicas tradicionais baseadas na redundância dos componentes dos nós sensores (FENG; KOUSHANFAR; POTKONJAK, 2002).

Nos sistemas de monitoramento de ambientes físicos é importante a confiabilidade dos dados observados. O emprego de uma RSSF em ambientes potencialmente hostis, como uma aplicação crítica, eleva as chances de falhas em algum ponto da observação.

Diferentes níveis de tolerância a falhas possibilitam a existência de diferentes algoritmos e protocolos de controle da rede, cada um mais adequado para uma situação, ou seja, se uma rede de sensores for utilizada em um ambiente no qual há pouca interferência, o protocolo pode ser um pouco mais simples, como uma rede de sensores implantada em um ambiente doméstico.

Dessa maneira, algumas prioridades particulares deste tipo de aplicação não terão um alto risco. Além disso, o protocolo desse tipo de ambiente deve ter um nível de tolerância à falha bem menor do que uma rede de sensores que será utilizada dentro da entrada de um vulcão prestes a entrar em erupção.

2.2.7 Cobertura do Ambiente Monitorado

A área de cobertura de uma RSSF corresponde à região coberta pelos sensores e o cálculo está relacionado ao raio de alcance dos nós que se encontram ativos na rede, podendo ou não considerar obstáculos. De acordo com (AKYILDIZ et al., 2002), os nós sensores podem ser colocados com altas densidades aleatoriamente no ambiente, podem ser fixados próximos ou diretamente dentro do fenômeno a ser observado pela rede de sensores.

Na verdade, devido à sua característica de trabalhar autonomamente, esses sensores podem estar geograficamente em áreas mais remotas fazendo tais observações. Não há uma regra estreita para a deposição de sensores, mas o objetivo é que eles estejam no lugar mais estratégico possível para poder sentir perfeitamente o fenômeno desejado.

2.2.8 Posicionamento dos Nós Sensores

Em Yick, Mukherjee e Ghosal (2008), fica bem clara a existência de dois tipos de posicionamento de nós sensores, que são a forma determinística e a forma não determinística. A

forma determinística de posicionar os nós sensores pode ser usada em ambientes internos (monitoramento de estruturas de prédio e grandes construções como pontes, de áreas de segurança ou com risco de acidentes em fábricas), mas também em ambientes externos, em que é feito um pré-planejamento do local exato onde vai ser instalado cada nó sensor em função de um sensoriamento e entrega de dados eficiente.

A forma não determinística é feita espalhando os nós sensores aleatoriamente dentro da área de sensoriamento (florestas, campos de batalha), estes nós sensores podem ser jogados de um avião ou helicóptero, lançados por armas em campos de batalha aleatoriamente na área de interesse a ser monitorada.

A forma aleatória aumenta a dificuldade em relação ao funcionamento dos protocolos e aplicação, definição de rotas, padronização da comunicação, eficiência de energia, latência, e outros detalhes.

Da forma análoga, a determinística faz a conectividade da rede se tornar mais simples. A área de cobertura do sensoriamento e de cobertura da comunicação podem ser calculadas previamente e configuradas de acordo com o projeto da rede, facilitando o trabalho de coleta e entrega dos dados, com mais segurança, eficácia, garantia de entrega, dentre outros benefícios.

Deve-se considerar outra forma de posicionamento de nós sensores, maneira esta que pode ser considerada inicialmente determinística ou não determinística, e que logo após, tornam-se dinâmicos, movendo-se pela área de rede em caminhos aleatórios ou pré-determinados.

Neste caso, em vez de os sensores estarem no chão em cima da terra, debaixo da terra, no fundo do mar, em estruturas de grandes construções, no alto de prédios ou pontes, eles vão estar acoplados em robôs terrestres, submarinos, helicópteros, aviões, carros, todos veículos não tripulados. Como a topologia da rede torna-se altamente dinâmica, essa forma de posicionamento dos nós sensores torna complexa a funcionalidade da rede em alguns aspectos.

A manutenção da área de cobertura está intimamente ligada à maneira como os nós sensores estão dispostos na área e deve levar em conta situações quando um sensor é retirado de um local ou um novo é adicionado à área. Tal influência da disposição dos nós na cobertura é representada na figura 2.3 (NAKAMURA, 2004):

O modelo determinístico regular considera a distribuição dos nós como regular e os posiciona em uma grade conforme ilustrado na Figura 2.3(a). O não-determinístico possui duas abordagens: na primeira o objetivo é lançar os nós visando à formação de uma grade similar ao outro modelo, porém, como no lançamento.

Há influência de fatores (como vento, obstáculos, velocidade do avião que o lançou). A formação da grade dar-se-á de maneira irregular como ilustrado na Figura 2.3(b).

A segunda abordagem do modelo não-determinístico é considerada que os nós estão dis-

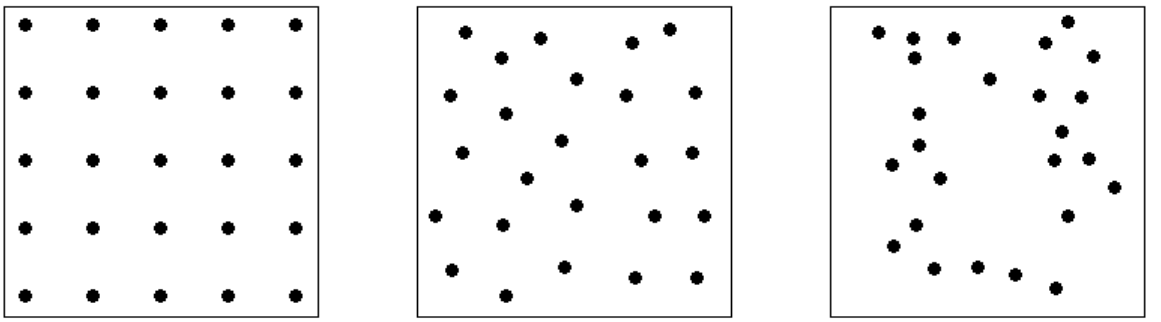


Figura 2.3: Modelos de posicionamento em RSSF

tribuídos na área de maneira aleatória com distribuição uniforme como mostra a Figura 2.3(c). Nesse caso, os valores das coordenadas dos nós estão limitados entre zero e a dimensão da área.

2.2.9 Área de Cobertura

A área de cobertura de uma RSSF é determinada pela integração das áreas de sensoriamento de cada nó sensor, ou seja, cada ponto do ambiente tem um nó sensor responsável por essa área, e a falha na cobertura é representada pela porcentagem da área de monitoramento que não está coberta por nenhum nó sensor.

2.2.10 Segurança em RSSF

Segurança em RSSF é uma das características vitais na maioria das aplicações em que alguém pode ter o interesse de invadir, roubando informação ou, simplesmente desejando causar panes na rede, fazendo com que a aplicação não funcione corretamente ou tendo acesso não autorizado às informações.

Os requisitos de segurança variam de acordo com a aplicação desenvolvida, entre eles confiabilidade, pontualidade, autenticação, integridade, controle de acesso, confidencialidade e disponibilidade. As restrições computacionais presentes nas Redes de Sensores Sem Fio são grandes limitadores das soluções de segurança que podem ser executadas nesse ambiente, especialmente a distribuição de chaves (YICK; MUKHERJEE; GHOSAL, 2008).

A segurança é um fator crítico em Redes de Sensores Sem Fio. Os problemas podem estar presentes na forma de acesso, em falhas nos protocolos de comunicação, software, hardware dos dispositivos de coleta e envio de dados ou na adulteração de nós sensores (WANG; ATTEBURY; RAMAMURTHY, 2006).

Nas RSSF, a limitação imposta pelo tamanho minúsculo de cada dispositivo, utilização

de bateria limitada, baixo custo fazem com que, ainda que um nó possa ser descartado, haja restrições quanto à utilização de protocolos e algoritmos de segurança conhecidos. Por esse motivo, torna-se imprescindível que novas tecnologias sejam elaboradas e avaliadas.

O uso de mecanismos de segurança visa garantir o uso da rede sem anomalias, da maneira original como foi implantada, eliminando a ação do inimigo. Os principais mecanismos de segurança são: métodos criptográficos, protocolos de gerência de chaves criptográficas, detecção de intrusão e, por último, filtro de pacote e de conteúdo das mensagens.

Nas RSSFs, a utilização de filtros de pacotes e conteúdos das mensagens é limitada devido às poucas camadas implementadas no modelo de rede. Portanto, a utilização desse método limita-se a impedir que pacotes com destino inválido trafeguem pela rede. Os outros mecanismos de segurança são aplicáveis às RSSFs, com algumas limitações de funcionalidades em comparação às redes convencionais.

Os requisitos de segurança para uma rede de sensores sem fio dependem do quão crítica é uma aplicação. Desse modo, serão ou não necessárias à segurança de uma RSSF (DJENOURI; KHELLADI; BADACHE, 2005) (WANG; ATTEBURY; RAMAMURTHY, 2006) (CHEN et al., 2009):

- autenticidade ponto a ponto - garante a origem das informações em comunicação;
- autenticidade em modo de difusão - garante a origem das informações em modo de difusão;
- autorização - garante que somente sensores autorizados possam prover informação para os serviços de rede;
- confidencialidade - oculta as informações do inimigo dependendo do domínio;
- dados atualizados (*freshness*) - garante que os dados replicados sejam recentes, e não replique dados antigos;
- disponibilidade - garante que as aplicações consigam utilizar os recursos de rede;
- integridade - garante que não ocorram alterações à mensagem original;
- não-repúdio - garante que o nó não possa proibir o envio de uma mensagem que originou dele próprio;
- pontualidade - possibilita o descarte de mensagens antigas repetidas que podem ser inseridas pelo invasor.

Garantir a segurança nas RSSFs não é tarefa fácil. Todas as limitações e adversidades, como o fato dos sensores de serem implantados em ambiente aberto e hostil, da comunicação ser sem

fio, dos recursos computacionais dos dispositivos serem limitados, da topologia ser variável e da localização prévia de cada sensor ser desconhecida, devem ser considerados durante o desenvolvimento do projeto de segurança dessa rede.

2.2.11 Escalabilidade

Karl e Willig (2007) afirmam que a quantidade de nós sensores dentro de uma rede de sensores pode ser de centenas, milhares ou até milhões de sensores, dependendo da aplicação que a está utilizando.

Os referidos autores consideram a escalabilidade um requisito indispensável ao bom funcionamento de toda a rede, partindo-se da entrega das informações e o roteamento, e abrangendo todas as outras características que uma rede de sensores tem que seguir normalmente independendo da quantidade de nós sensores.

Portanto, podem ser adicionados ou excluídos nós sensores da rede em grande quantidade sem que a rede perca desempenho por isso. E com todos esses sensores, dentro da área de rede pode haver alta densidade de sensores em alguns pontos e isso deve ser previsto pelo protocolo ou algoritmo.

2.2.12 Classificação das RSSFs

Todas as características citadas anteriormente são influenciadas pelos requisitos da aplicação, ou seja, uma RSSF é um tipo de sistema dependente da aplicação. Os parâmetros de configuração e manutenção variam de acordo com os objetivos da aplicação. Da mesma forma, a classificação das RSSFs também depende de seu objetivo e de sua área de aplicação (RUIZ et al., 2004).

Segue abaixo um resumo da configuração de cada característica das RSSFs. A tabela 2.2 foi adaptada de (RUIZ et al., 2004) e mostra o modelo de classificação baseado na configuração da rede.

Tabela 2.2: Caracterização das RSSFs quanto à configuração

Configuração		
Composição	Homogênea	Rede composta de nós de mesma capacidade de hardware.
	Heterogênea	Rede composta de nós com diferentes capacidades de hardware.
Organização	Hierárquica	Rede em que os nós são agrupados em grupos (*clusters*), onde cada grupo contém um líder (*cluster-head*). Em redes que possuem esta organização, os grupos formam hierarquias entre si, de forma que quando um dado necessita ser remetido à estação base, ele é remetido primeiro ao líder que irá enviar diretamente, ou através de outros líderes, à estação base. As fases de seleção dos nós líderes são necessárias. Esta organização tem primordialmente a intenção de reduzir o tráfego da rede com o objetivo de economizar energia.
	Plana	Rede em que os nós não são agrupados em grupos.
Mobilidade	Estacionária	Rede em que os nós sensores são fixos durante toda a vida da rede.
	Móvel	Rede em que os nós sensores podem ter sua posição espacial variável durante o tempo de vida de rede. Isto é, esses nós sensores podem ter a posição inicial, onde foram depositados, modificada por sua capacidade de locomoção, por forças da natureza e também pela entidade na qual estão acoplados.
Densidade	Balanceada	Quando a rede apresenta uma alta concentração de nós sensores por unidade de área considerada ideal.
	Densa	Quando a rede apresenta uma alta concentração de nós sensores por unidade de área.
	Esparsa	Quando a rede apresenta uma baixa concentração de nós sensores por unidade de área.
Distribuição	Irregular	Rede que apresenta uma distribuição não uniforme na área monitorada.
	Regular	Rede que apresenta uma distribuição uniforme na área monitorada.

RSSFs também são classificadas quanto ao sensoriamento e coleta de dados, e quanto à sua comunicação. Podem ser classificados, conforme mostrado nas tabelas 2.3 e 2.4 - segundo Ruiz

em (RUIZ et al., 2004), da seguinte forma:

Tabela 2.3: Caracterização das RSSFs quanto ao sensoriamento

Sensoriamento		
Coleta	Periódica	Os dados são coletados pelos nós sensores em intervalos regulares.
	Contínua	Os dados são coletados continuamente pelos nós sensores.
	Reativa	Os dados são coletados pelos nós sensores quando sentem algo de interesse para a aplicação, ou quando lhes são solicitados a coleta.
	Tempo Real	Os dados são coletados pelos nós sensores na maior quantidade possível no menor espaço de tempo.

Tabela 2.4: Caracterização das RSSFs quanto à comunicação

Comunicação		
Disseminação	Programada	Os nós sensores enviam os dados em intervalos programados.
	Contínua	Os nós sensores enviam os dados continuamente.
	Sob demanda	Os nós sensores enviam os dados quando há uma solicitação ou quando ocorre algo de interesse da aplicação.
Tipo conexão	Simétrica	O raio de comunicação entre todos os nós sensores são iguais, com exceção do nó *sink*.
	Assimétrica	O raio de comunicação entre os nós é diferente.
Transmissão	Simplex	Os nós sensores possuem transceptor que permite apenas a transmissão de informação.
	Half-duplex	Os nós sensores possuem transceptor que permite transmissão ou recebimento de informação em um determinado instante.
	Full-duplex	Os nós sensores possuem transceptor que permite transmissão ou recebimento de informação ao mesmo tempo.

Projetos e soluções propostas para a utilização de RSSFs devem considerar as características e limitações dos nós sensores, assim como as características do ambiente a ser monitorado.

2.2.13 Topologia da Rede

Segundo Yick, Mukherjee e Ghosal (2008), o ambiente tem um papel fundamental na determinação da dimensão da rede, do esquema de deposição dos nós sensores, e na topologia da rede. Portanto, o ambiente de interesse da aplicação em que vão ser espalhados ou colocados os nós sensores tem um papel importante na forma usada para a deposição dos nós sensores e principalmente em como vai ser a topologia da rede.

Em algumas aplicações é necessário que os nós sensores se movam dentro da área de rede também, e, para que isso seja possível, usualmente esses nós sensores são implantados em robôs que vão disponibilizar essa mobilidade para a rede.

No caso dos sensores móveis, a topologia da rede muda o tempo todo e, com isso, as rotas entre nós que detectaram algo e precisam que essa informação chegue até o *sink*, mudam frequentemente, dando um grau de complexidade aos algoritmos de roteamento. Entretanto, mudanças na topologia da rede também podem ser motivadas por outros fatores:

- quando há uma alta densidade em uma região, alguns sensores podem ser desligados ou religados quando necessário, para manter um bom funcionamento da rede, ter um backup para aquela região da rede ou evitar comunicações desnecessárias. Isso ocorre mais frequentemente quando os nós sensores são depositados de maneira aleatória, sendo jogados de um avião ou helicóptero na área de interesse;
- quando nós sensores falham devido à falta de energia, interferência na comunicação ou defeitos estruturais;
- quando não há a necessidade de sentir o ambiente com frequência, e enviar os dados em espaços de tempo longos, o nó sensor pode entrar em modo *sleep*, economizando energia e, assim, aumentando a vida útil do sensor.

Numa RSSF, mesmo nos casos em que os nós sensores ficam estáticos, a topologia é bem dinâmica, por haver sensores que podem parar de funcionar pelos motivos já mencionados acima. Toda essa heterogeneidade na topologia da rede deve-se às mudanças que ocorrem durante a vida da rede e pela sua alta capacidade de escalar que é uma característica que deve existir em todas as aplicações, podemos definir duas principais topologias: estrela e *mesh* (YICK; MUKHERJEE; GHOSAL, 2008).

A mais básica e menos usada é chamada de estrela. Nela, os nós sensores se comunicam diretamente com o *sink*, de maneira que a rede fica limitada ao tamanho da área de comunicação dos sensores.

Na topologia *mesh*, os nós podem ser escalados à vontade, ou seja, sem limitação da área de rede, e são criadas rotas entre os nós sensores e o *sink*. Essas rotas são usadas para trafegar

os dados de maneira que, se houver a perda de um nó sensor intermediário, podem ser criadas outras rotas para que a informação continue chegando até o *sink*.

Quando há uma grande densidade de nós sensores dentro da área de rede é requerido um manuseio cuidadoso da topologia da rede, a manutenção da topologia da rede quanto à ocorrência de alterações posteriores divide-se em três fases: pré-implantação, pós-implantação e reimplantação ou adição de nós (AKYILDIZ et al., 2002).

2.3 Arquitetura de Comunicação em RSSFs

Diferente do que estamos acostumados a trabalhar em redes tradicionais, nas RSSFs tem cinco camadas: física, enlace, rede, transporte e a da aplicação. Para que uma aplicação de RSSFs seja eficiente, tenha vida prolongada e seja bem-sucedida no envio das informações para o *sink*, em alguns casos é necessário fazer um *cross-layer* (cruzamento das funções entre as camadas), tirando um pouco da característica total de abstração entre as camadas da rede (YICK; MUKHERJEE; GHOSAL, 2008).

Nós sensores são espalhados em uma área de rede e cada um pode ter acoplado um ou vários tipos sensores diferentes em um nó, dependendo da necessidade da aplicação. Uma vez no ambiente de rede, os nós começam a sentir o entorno e, de acordo com a aplicação e seu algoritmo de funcionamento, dados são coletados e enviados para o *sink* usando uma arquitetura multi-saltos.

Alguns fatores influenciam no desenvolvimento de uma aplicação usando rede de sensores. Os principais: tolerância a falhas, escalabilidade da rede, custo de cada nó sensor (sabendo que normalmente é grande o número de nós sensores), topologia da rede, recursos computacionais restritos e consumo de energia (AKYILDIZ et al., 2002).

2.3.1 Camada Física

A camada física trata das necessidades simples, mas robustas, de modulação, transmissão e técnicas de recepção, é responsável pela seleção de frequência, geração de frequência de portadora, detecção de sinais e encriptação de dados (AKYILDIZ; VURAN, 2010).

Esta função principal de fazer a modulação e demodulação do sinal digital consiste em transformar as ondas de rádio recebidas em dados digitais e vice-versa. Todo esse trabalho é feito pelos *transceivers* ao receber e ao enviar qualquer informação. A confiabilidade da comunicação depende, também, das propriedades de hardware dos nós, como a sensibilidade da antena e circuito do transceptor. A maioria das vantagens exclusivas de RSSFs são fornecidos através de comunicação sem fio. Facilidade de implantação, infraestrutura de rede livre, e de

comunicação de transmissão são algumas dessas vantagens. No entanto, comunicação sem fio também traz diversos desafios em termos de alcance de comunicação limitado, erros frequentes, e interferências (AKYILDIZ; VURAN, 2010).

É bem conhecido que a longa distância de comunicação sem fio pode ser cara, tanto em termos de consumo de energia e complexidade de implementação. Ao projetar a camada física para redes de sensores, a minimização de energia assume uma importância significativa, para além da decadência, dispersão, sombreamento, reflexo, difração e efeitos de esmaecimento do sinal (AKYILDIZ; VURAN, 2010).

Essas ondas têm uma frequência pré-determinada para poder se comunicar, e nas redes de sensores sem fio é usada a frequência não-licenciada de 2.4GHz, usada nos padrões IEEE 802.11, Bluetooth, e IEEE 802.15.4.

Um problema de utilizar essa frequência é que, por não ser licenciada, ela é a mais usada entre os diferentes aparelhos e dispositivos sem fio. Então, dependendo do ambiente, pode haver interferência de outros dispositivos, atrapalhando a comunicação. E, como o seu próprio nome diz, essa camada está relacionada com a parte física do nó sensor, diretamente com o hardware (YICK; MUKHERJEE; GHOSAL, 2008).

O meio sem fio usado em RSSFs é um dos fatores mais importantes, uma vez que as propriedades únicas dos diferentes meios de comunicação colocam várias restrições sobre os recursos da camada física. Em geral, as ligações sem fios podem ser formadas por Rádio Frequência (RF), ópticas, acústicas, ou técnicas de indução magnética. Enquanto a comunicação de RF é geralmente adotada, outras técnicas de comunicação também têm cenários de aplicação específica (AKYILDIZ; VURAN, 2010).

2.3.2 Camada Enlace

A camada de enlace é responsável pela multiplexação de fluxos de dados, a detecção de *frames* de dados, acesso ao meio e controle de erro. Ela garante confiança ponto a ponto e conexões ponto a multiponto em uma rede de comunicação (AKYILDIZ; VURAN, 2010).

Esta camada tem tarefas como o controle de erros, a formação e manutenção dos *links* criados entre seus nós sensores vizinhos e garantia de uma confiável e eficiente transferência de informações entre esses *links*. Mecanismos da camada de enlace devem alcançar isso visando o baixo consumo de energia, dos dados recebidos e enviados e a baixa taxa de erros (KARL; WILLIG, 2007).

Os protocolos MAC (Controle de Acesso ao Meio) em uma rede de sensores sem fio multissalto autônoma deve alcançar dois objetivos. O primeiro objetivo é a criação da infraestrutura de rede. Uma vez que milhares de nós sensores podem ser densamente espalhados em uma área

de rede, o sistema MAC deve estabelecer *links* de comunicação para transferência de dados. Isto constitui a infra-estrutura básica necessária para o *hop-by-hop* de comunicação sem fio e oferece a capacidade de auto-organização. O segundo objectivo é compartilhar de forma justa e eficiente os recursos de comunicação entre os nós sensores. Esses recursos incluem tempo, energia e frequência. Vários protocolos MAC têm sido desenvolvidos para RSSFs para atender a esses requisitos durante a última década (AKYILDIZ; VURAN, 2010).

Controle de erros deve alcançar um bom nível de confiança na transmissão dos pacotes, com o mínimo de energia consumido. São usados vários métodos e mecanismos para a detecção e controle de erros pois a probabilidade de ocorrerem erros de transmissão é maior quando esse processo é feito pelo, em vez de por meio de cabo. Exemplos desse mecanismo é o *checksum* nos cabeçalhos dos pacotes para que, quando o pacote chegar ao outro lado ou chegar um de volta, seja detectado se houve alguma interferência no meio de transmissão e isso alterou algum bit (KARL; WILLIG, 2007).

Esta camada é responsável também por dividir ou reagrupar pacotes que eram maiores e foram segmentados para poderem ser enviados. É importante observar que, entretanto, em rede de sensores normalmente não há pacotes grandes.

O controle de fluxo também é uma das tarefas importantes da camada de enlace e principalmente em rede de sensores que têm capacidade de armazenamento restrita. Nesse tipo de rede pode-se ficar horas, dias, ou até anos sem comunicação, à espera de ocorrer alguma alteração no que está sendo sentido pelo nó sensor no fenômeno monitorado.

Contudo, quando o nó sensor percebe alguma ocorrência, ele pode começar a se comunicar e a receber comunicações de vizinhos. É nesse momento que melhor se vislumbra o papel do controle de fluxo, que cuida para que só recebam algum pacote da rede quando o nó sensor tem a capacidade de armazená-lo ou de processá-lo. Além disso, seu *buffer* pode ser preenchido rapidamente em momentos de pico.

2.3.3 Camada de Rede

A camada de rede é a que tem maior influência no sucesso das RSSFs, pois é a responsável pelo roteamento dos dados de toda a rede, desde a origem até seu destino. Afinal, devido ao fato de sensores serem espalhados densamente dentro da área de interesse ou próximos a ela, é necessário haver um protocolo de roteamento entre nós sensores e *sink* (KARL; WILLIG, 2007).

Por ser tão crucial ao bom funcionamento das RSSFs, o protocolo de roteamento deve ser desenvolvido em consonância com alguns princípios básicos (YICK; MUKHERJEE; GHOSAL, 2008):

- eficiência no consumo de energia;
- confiabilidade e segurança da informação que chega ao *sink*, uma vez que, na maioria das vezes, a rede de sensores é centrada em dados;
- criação de rotas baseada na quantidade de energia disponível em cada sensor, usando os nós têm mais energia armazenada;
- capacidade de funcionar normalmente diante de alta escalabilidade, uma vez que o número de nós sensores podem aumentar e o protocolo de roteamento tem que gerenciar a comunicação mantendo o consumo mínimo de energia, o nível de tolerância a falhas, as prerrogativas de segurança.

Protocolos de roteamento em RSSF diferem dos protocolos de redes tradicionais de diversas maneiras. Um caso evidente disso é que nós sensores não têm endereço de rede, por exemplo endereço IP (*Internet Protocol*), de forma que protocolos de roteamento baseados nessa qualidade não funcionam em RSSFs, por serem centradas em dados. Para conseguir seguir esses requisitos, uma das soluções é a agregação de dados (KARL; WILLIG, 2007).

2.3.4 Camada de Transporte

A camada de transporte é responsável por garantir que as informações saiam da origem e cheguem ao destino confiáveis e seguras. Dessa forma, ao desenvolver um protocolo para a camada de transporte deve-se ter o cuidado de abstrair ao máximo para que ele funcione estavelmente com qualquer aplicação (YICK; MUKHERJEE; GHOSAL, 2008).

É por isso que a camada de transporte em RSSF deve suportar múltiplas aplicações, conseguir recuperar casos de perda de pacotes, ter confiabilidade variável e controlar o congestionamento. A tolerância à perda de pacotes é especialmente necessária devido ao tipo de falhas propícias a acontecerem na comunicação sem fio, como congestionamentos, colisão de pacotes e falhas inerentes aos nós sensores (que podem, por exemplo, "sumirem" da rede).

Perder qualquer pacote vai ser um prejuízo enorme para a vida útil da rede, pelo desperdício de energia e pelo comprometimento da QoS (Qualidade de Serviço). A detecção e a recuperação correta de um pacote perdido vão melhorar o desempenho da rede e otimizar o consumo de energia.

É possível que a recuperação seja feita em cada salto que o pacote faz ao caminhar pela rede, através da detecção nos nós intermediários. Tal possibilidade de agir de imediato é a ideal, por ser energeticamente eficiente e evitar o retrabalho, uma vez que não será necessário fazer o caminho inverso do pacote por inteiro para buscá-lo na origem da comunicação (KARL; WILLIG, 2007).

O controle de congestionamento monitora e detecta o congestionamento, causado, principalmente, por colisão de pacotes. Logo que uma ocorrência dessa natureza é registrada pelo mecanismo de controle, comanda os nós sensores responsáveis pelo congestionamento que diminuam sua taxa de transferência para evitar danos maiores.

Este controle ajuda a não causar preenchimento total do *buffer*, o que também causaria perda de pacotes.

Outra possibilidade de realizá-lo é a detecção a cada salto dado na rede. A seleção de uma dessas alternativas sempre vai depender dos interesses da aplicação.

2.3.5 Camada de Aplicação

A camada de aplicação contém, usualmente, uma série de protocolos de alto nível, comumente necessária por fazer a interface entre o protocolo de comunicação e o aplicativo que pediu ou receberá a informação através da rede.

Existem diversas aplicações definidas e propostas para RSSFs, feitas de acordo com as necessidades advindas do objetivo do projeto. Na literatura, vários trabalhos (TILAK; Abu-Ghazaleh; HEINZELMAN, 2002) destacam a importância da participação da aplicação no processo de comunicação em RSSFs.

Afinal, otimizações específicas da aplicação podem reduzir o número de transmissões, diminuindo o consumo total de energia na rede. No entanto, mesmo nesse cenário, protocolos para camada de aplicação de RSSFs ainda constituem uma região inexplorada.

Embora existam diversas propostas de aplicações, os protocolos para camada de aplicação ainda são pouco explorados. São sugeridos três possíveis protocolos de aplicações de apoio: o de gerenciamento de sensores (SMP); o de aviso de dados e designação de tarefas (TADAP); e o de consultas e disseminação de dados (SQDDP).

2.3.6 Cross-Layer

O *cross-layer* é a interação entre as camadas, função que descaracteriza parcialmente a abstração absoluta entre os protocolos no modelo de camadas. Essa técnica traz enormes benefícios para RSSF, interagindo com os protocolos como se não houvesse camadas individuais.

Quando um pacote, saindo da camada de aplicação, passa pelas outras camadas, vai sendo encapsulado e é adicionada informação em seu cabeçalho. Ao chegar ao destino com todo esse cabeçalho, cada camada vai desencapsulando, interpretando e retirando esses cabeçalhos. Todos esses encapsulamentos causam *overhead*, o que é prejudicial para RSSF.

Portanto, uma das justificativas para a proposta de, na interação entre as camadas, deixar certas "formalidades" de lado é economizar processamento, banda de comunicação, armazenamento e, principalmente, energia (pois não há o consumo de energia relativo ao envio e recebimento de cabeçalhos grandes e desnecessários) (YICK; MUKHERJEE; GHOSAL, 2008).

2.4 Considerações Finais

O presente capítulo apresentou as principais características de Redes de Sensores Sem Fio, além dos requisitos e necessidades que devem ser observados para o desenvolvimento de aplicações e protocolos para esse tipo de rede.

Foi mostrado que, em se tratando de RSSFs é imprescindível a otimização do consumo de energia, e que o tempo de vida da rede varia de acordo com os protocolos, algoritmos e medidas adotadas na solução do projeto como um todo.

Sendo assim, diferentes soluções de protocolos e técnicas promissoras de agregação e de correlação entre os dados têm sido pesquisadas e aplicadas em RSSFs, como serão apresentadas no capitulo 3.

Capítulo 3

PROTOCOLOS DE ROTEAMENTO PARA RSSFS

Determinar continuamente a topologia da rede para que o escoamento dos dados seja feito com o mínimo possível de dissipação de energia, latência reduzida, confiabilidade e garantia na entrega dos dados, garantindo o sucesso das ações executadas, é um grande desafio, podendo ser ainda maior dependendo da deposição, densidade e mobilidade da rede.

Os algoritmos de redes de sensores sem fio em que os nós são fixos conseguem definir a topologia da rede mais facilmente, pois não precisa haver confirmação da posição de cada nó sensor a cada espaço de tempo. Já os algoritmos para RSSF com nós e *sinks* móveis têm a árdua tarefa de atualizar a topologia da rede com o mínimo de comunicação, usando a mobilidade dos nós a favor para o escoamento mais eficiente possível dos dados na rede.

Roteamento é um dos desafios das RSSFs devido às diversas características que as distingue das redes convencionais e das redes *AdHoc* sem fio (AKKAYA; YOUNIS, 2005; PERKINS, 2000) a saber:

1. impossibilidade de construção de esquema de endereçamento global para a deposição de uma grande quantidade de nós - as soluções clássicas baseadas em IP não podem ser aplicadas em RSSFs;
2. ao contrário das redes convencionais de comunicação, as aplicações de RSSF requerem que dados coletados de uma ou mais regiões sejam encaminhados para um nó sorvedouro;
3. dados gerados possuem significativa redundância já que múltiplos nós podem produzir os mesmos dados próximos à vizinhança de um fenômeno - esta redundância deve ser explorada, por exemplo, por meio de correlação espacial, pelos algoritmos de roteamento para redução de consumo de energia e utilização de largura de banda;
4. as limitações de recursos dos nós sensores exigem gerenciamento cuidadoso dos recursos.

Devido a essas características, vários protocolos foram desenvolvidos para tentar solucio-

nar o problema de roteamento nas RSSFs (SCHURGERS; SRIVASTAVA, 2001) (HEINZELMAN; KULIK; BALAKRISHNAN, 1999) (INTANAGONWIWAT; GOVINDAN; ESTRIN, 2000) (INTANAGONWIWAT et al., 2003) (BRAGINSKY; ESTRIN, 2002) (CHU et al., 2002) (MANJESHWAR; AGRAWAL, 2001) (YAO; GEHRKE, 2002) (SHAH; RABAEY, 2002) (SADAGOPAN; KRISHNAMACHARI; HELMY, 2003). Os protocolos apresentados são classificados em *data-centric* e hierárquico, e, em nenhuma das soluções citadas acima, é utilizado o conceito de correlação espacial que é apresentado na solução deste trabalho.

Existem algumas soluções que utilizam correlação espacial (YOON; SHAHABI, 2005) (VURAN; AKYILDIZ, 2006) (YOON; SHAHABI, 2007) (DAI; AKYILDIZ, 2009) (YUAN; CHEN, 2009), que se mostram eficientes na camada MAC e na camada de rede. Essas soluções estão sempre atreladas a protocolos de roteamento hierárquicos, utilizando clusterização e, em alguns casos, também a agregação.

Além do roteamento, como visto, há a necessidade de se preocupar com a disseminação dos dados coletados pelos nós sensores. A preocupação é garantir que o mínimo de dados seja roteado pela rede, e garantir a confiabilidade dos dados roteados. Para isso, duas das principais soluções incluem: agregação de dados, apresentada na seção 3.5; e correlação espacial, na seção 3.6.

Três classificações baseadas na topologia de protocolos de roteamento para RSSF são encontradas na literatura:

- pró-ativos: o roteamento é configurado previamente, e, a qualquer momento. Os nós sensores têm a capacidade de enviar a informação para o *sink*, sem a necessidade de envio de mensagens de controle a fim de descobrir a rota (AL-KARAKI; KAMAL, 2004) (LAMBROU; PANAYIOTOU, 2009).

- reativos: este tipo define as configurações de rota quando há uma demanda para isso, não haverá a necessidade de nenhuma rota para os destinos até que precisem enviar alguma informação (*Ibidem*).

- híbridos: como o próprio nome diz, usa das duas abordagens acima, mantendo a configuração pré-estabelecida em alguns nós da rede, mas outros nós fazem a atualização da rota periodicamente (*Ibidem*).

Outra classe de protocolos de roteamento é chamada de cooperativa. No roteamento cooperativo, os dados enviados ao nó *sink* podem ser agregados e sujeitos a transformações, portanto, reduzindo o custo em termos de percurso (AL-KARAKI; KAMAL, 2004).

Em geral, o roteamento em RSSFs pode ser dividido em roteamento plano, hierárquico, e baseado na localização. No roteamento plano, todos os nós tem papéis ou funcionalidades

iguais. No hierárquico, os nós irão desempenhar diferentes papéis na rede. E no roteamento baseado na localização, as posições dos nós sensores são exploradas para encaminhar os dados na rede (*Ibidem*).

3.1 Protocolos Centrados em Dados (*data-centric*)

Na abordagem centrada em dados, busca-se agregar dados redundantes advindos de múltiplas fontes ao longo da rota até o *sink*, reduzindo o número de transmissões e assim economizando energia (KRISHNAMACHARI; ESTRIN; WICKER, 2002; AL-KARAKI; KAMAL, 2004).

O SPIN (HEINZELMAN; KULIK; BALAKRISHNAN, 1999) foi o primeiro protocolo proposto que tem como característica a negociação de dados entre os nós para eliminar a redundância e assim economizar energia. O *Directed Diffusion* (DD) (INTANAGONWIWAT; GOVINDAN; ESTRIN, 2000) foi desenvolvido posteriormente tornando-se referência nesse tipo de protocolo baseado em dados. Alguns protocolos foram baseados no DD e outros seguem conceitos similares (SCHURGERS; SRIVASTAVA, 2001) (BRAGINSKY; ESTRIN, 2002).

3.2 Roteamento Plano

O roteamento plano é feito através de multi saltos, em que todos os nós têm uma identificação de quantos saltos ele está do destino, e cada salto tem o tamanho do raio de comunicação dos nós sensores. Protocolos que utilizam roteamento plano são usados em uma topologia também chamada de plana em que os nós da rede são homogêneos, ou seja, têm as mesmas características com relação a suas capacidades. Tipicamente cada nó tem o mesmo papel e os nós colaboram para executar a tarefa de coletar os dados no ambiente monitorado (AL-KARAKI; KAMAL, 2004).

3.3 Roteamento Hierárquico

A abordagem de roteamento hierárquico prega que uma rede com apenas um nível hierárquico pode sobrecarregar o *sink*, caso ocorra um aumento de nós na rede. Essa sobrecarga pode afetar o funcionamento por completo, gerando atrasos de comunicação, rotas inadequadas para os pacotes e, assim, aumentar o tráfego na rede.

Para permitir que o sistema suporte uma carga adicional e ainda seja capaz de cobrir uma vasta área de interesse sem comprometer a qualidade do serviço, mecanismos de *clustering* em redes têm sido introduzidos em algumas soluções para protocolo de roteamento.

Os protocolos baseados em hierarquia prezam pelo consumo homogêneo de energia entre os nós, de modo que os sensores agrupem-se em *clusters* e troquem informações com o coordenador de seu grupo, facilitando a fusão dos dados para reduzir o número de mensagens e informações trafegadas na rede.

No entanto, existem outras arquiteturas - a maioria das arquiteturas de disseminação de dados em RSSFs são hierárquicas, que podem ser classificadas como (LE-TRUNG et al., 2009):

- baseada em *cluster*: como apresentado acima, um conjunto de nós que trabalham cooperativamente como colaboradores e elegem um nó coordenador para gerenciar ou nós colaboradores do *cluster*;
- baseada em cadeias: a idéia principal aqui é transmitir apenas os dados de seus vizinhos mais próximos;
- baseada em grade (*grid*): um conjunto de nós são considerados concentradores de dados de uma região (grade) fixa;
- baseada em árvore: os nós são organizados em árvores, em que os dados transmitidos para o *sink* vão sendo processados pelos nós intermediários.

Arquiteturas baseadas em *clusters* são de particular interesse neste trabalho, pois uma delas será utilizada para a avaliação dos resultados deste trabalho de mestrado.

A formação de *clusters* é tipicamente baseada na reserva de energia dos sensores e na proximidade dos mesmos com o *cluster-head* (LIN; GERLA, 1997) (SRIKANTH; BABU, 2009). O primeiro protocolo do tipo hierárquico é o LEACH (*Low Energy Adaptive Clustering Hierarchy*) (INTANAGONWIWAT; GOVINDAN; ESTRIN, 2000) que prega a ideia de formar *clusters* baseados na força do sinal recebido e no uso de *cluster-heads* como roteadores para o *sink*.

A comunicação com o *sink* é feita apenas pelos *cluster-heads*, em vez de todos os sensores, economizando energia. O conceito do LEACH é inspiração para outros modelos de roteamento hierárquico (MANJESHWAR; AGRAWAL, 2001) (GUPTA; RIORDAN; SAMPALLI, 2005) (LINDSEY; RAGHAVENDRA, 2002) (NING; CASSANDRAS, 2007).

3.4 Roteamento Baseado na Localização

Neste caso, nós sensores estão interessados em sua localização geográfica. A distância entre os vizinhos pode ser estimada com base no sinal de radio, ou pode ser obtida simplesmente através de um hardware de localização como o GPS (AL-KARAKI; KAMAL, 2004).

A maioria dos protocolos de roteamento para redes de sensores necessita de informação de localização para os nós sensores. Em grande parte dos casos, a informação de localização é necessária para calcular a distância entre dois nós para poder estimar o consumo de energia.

Como não há um esquema de endereçamento para redes de sensores como os endereços IP e os nós estão espalhados em uma região, a informação de localização pode ser utilizada no roteamento de dados de uma forma mais eficiente em termos de energia.

Por exemplo, se a região a ser monitorada é conhecida, usando a localização dos sensores, as consultas podem ser difundidas apenas para uma região particular, reduzindo o número de transmissões significativamente. Alguns protocolos baseados em localização foram desenvolvidos primeiramente para redes *AdHoc* móveis (XU; HEIDEMANN; ESTRIN, 2001) (RODOPLU; MENG, 1999) (LI; HALPERN, 2001). Entretanto, eles podem ser aplicados em redes de sensores em que há pouca ou nenhuma mobilidade dos nós.

Porém, GPS apresenta desafios para uso em RSSF já que funciona apenas em ambientes externos e sem obstrução. São também dispositivos de alto custo não sendo adequados para uso com nós sensores pequenos e de baixo custo. Outras técnicas de localização baseadas em trilateração ou multilateração não oferecem precisão suficiente para uso em RSSF (KRISHNAMACHARI; ESTRIN; WICKER, 2002).

Técnicas de localização de nós, que requerem apenas dois pontos de referência, sem o uso de GPS, foram propostas que reduzem o erro de localização (OLIVEIRA et al., 2009; BOUKERCHE et al., 2007). Outras soluções existentes são baseadas em arranjos adaptativos de antenas (KUCUK et al., 2008).

Outros protocolos baseados em localização para redes *AdHoc* sem fio incluem *Cartesian* e *Trajectory-Based Routing* (NATH; NICULESCU, 2003). Porém, muitos desses protocolos não são aplicados em redes de sensores, pois eles não foram desenvolvidos considerando a economia de energia.

3.5 Agregação de Dados

Agregação de dados é uma técnica importante implementada nos protocolos de roteamento para RSSF, que funciona da seguinte maneira: o *sink* pode requisitar informações sobre o fenômeno observado a qualquer momento, e, como resposta à requisição, vários sensores enviarão a informação requisitada, que passará por vários nós até chegar ao *sink*.

Dependendo da aplicação, um nó pode enviar qualquer informação a qualquer momento, se for detectada alguma anormalidade no ambiente monitorado. Vários nós podem enviar informações a nós intermediários, os quais, por sua vez, podem aplicar técnicas de agregação a esses dados de modo a reduzir a quantidade de dados enviados, otimizando a escolha de rotas e

aumentando a longevidade da rede em termos de conservação de energia.

Alguns exemplos de técnicas de agregação incluem a aplicação de média, moda, valor máximo, sobre dados resultantes da agregação. O tipo de técnica a ser adotado depende do tipo da aplicação (NAKAMURA; LOUREIRO; FRERY, 2007).

A agregação pode ser feita dentro dos *clusters* ou durante o roteamento de pacotes contendo dados coletados. Portanto, esse processamento dos dados coletados é chamado de *in-network*, pelo fato de que a agregação ocorre entre o nó de origem e o nó destino (*sink*) (FASOLO et al., 2007).

Duas abordagens podem ser distinguidas:

- agregação com redução de tamanho: é o processo de agregação que combina e comprime os dados de diferentes origens a fim de reduzir a quantidade de informação enviada pela rede. Isso é possível quando são recebidos dois ou mais valores do mesmo fenômeno como, por exemplo, de duas ou mais informações de temperatura pode ser feita uma média, ou valor mínimo, ou máximo (FASOLO et al., 2007);
- agregação sem redução de tamanho: nesse processo é feita a concatenação dos dados, a consequência é que não vai ser diminuído o tamanho dos dados contidos no pacote roteado, mas vai economizar pelo menos em *overhead* de pacotes. Um exemplo é que um nó agregador pode receber uma informação de temperatura e outra de presença de fumaça ou de umidade (FASOLO et al., 2007).

3.6 Correlação de dados

Nas RSSFs, é normal que as áreas de rede fiquem com alta densidade de nós sensores para garantir boa cobertura. Com isso, alguns nós sensores coletarão informação sobre o mesmo evento. Devido a essa proximidade física dos nós sensores, as informações coletadas estarão altamente correlacionadas e com isso pode ser definido o grau de correlação entre os dados (VURAN; AKAN; AKYILDIZ, 2004). Podem ser enumerados alguns itens que impactam no grau de correlação:

- distância entre os nós - quanto menor a distância entre os nós, maior é o grau de correlação (VURAN; AKAN; AKYILDIZ, 2004);
- características do fenômeno observado - cada sensor "sente" o ambiente de uma maneira, dependendo do fenômeno. Se o sensor é de temperatura, um modelo de correlação pode ser desenvolvido e usado para calcular esse grau de correlação, que vai ser diferente do sensor de fumaça ou de pressão, ou seja, cada um tem suas próprias características,

como podemos ver nas soluções (FARUQUE; HELMY, 2004) (VURAN; AKYILDIZ, 2006) (PATTEM; KRISHNAMACHARI; GOVINDAN, 2004) que utilizam modelos de correlação para atribuir um limite de distorção entre a informação real e o dado que chega ao *sink*, mas que dependem de variáveis que mudam de acordo com as características do fenômeno observado;

- requisitos da aplicação - as RSSFs coletam informações do ambiente e concentram esses dados coletados em um ou mais sorvedouros de acordo com a aplicação. As aplicações em RSSFs podem ser para áreas distintas e com objetivos distintos como foi apresentado no capítulo 2. Desta maneira, os requisitos do sistema influenciam no grau de correlação como, por exemplo, uma aplicação para monitoramento de infraestruturas críticas necessitam de confiabilidade e precisão diferentes de uma aplicação que monitora um rebanho de animais.

Como foi apresentado, há um desafio com relação ao desenvolvimento de um modelo de correlação que satisfaça a aplicação em RSSFs. Uma vez definido o grau de correlação dos nós sensores dentro da área do evento (área do evento cobre todos os nós sensores que detectarem o mesmo evento), serão calculadas as áreas de correlação contidas dentro daquela área do evento, como mostra a figura 3.1. Finalmente, é eleito um nó sensor representativo dentro de cada área de correlação.

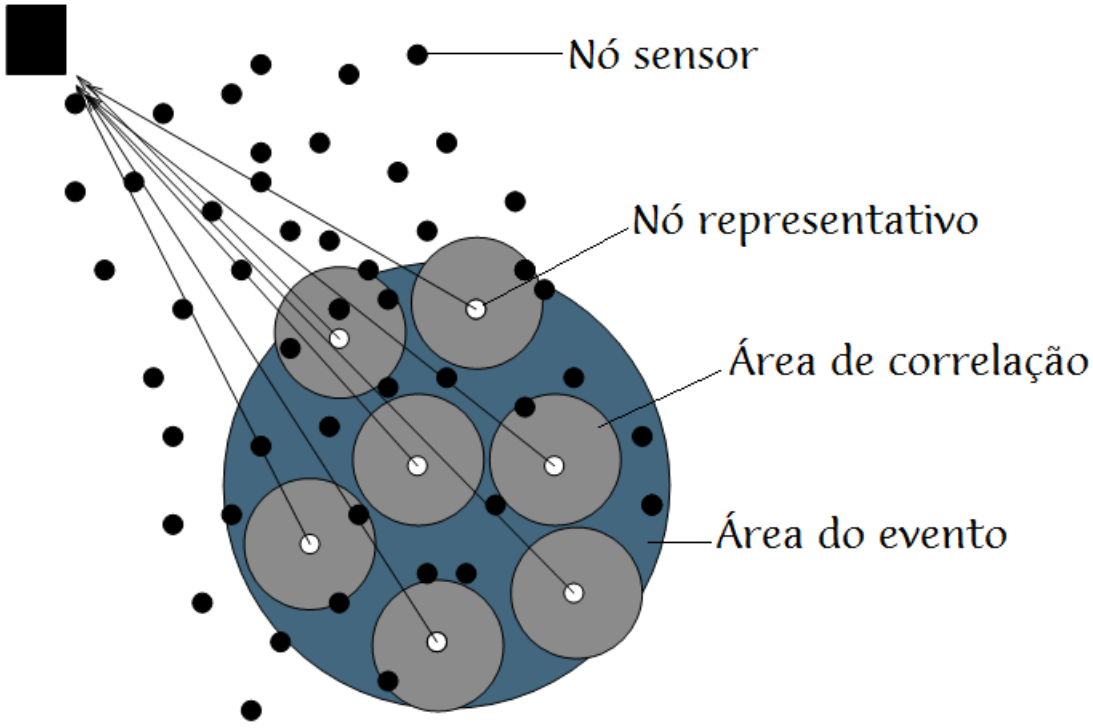
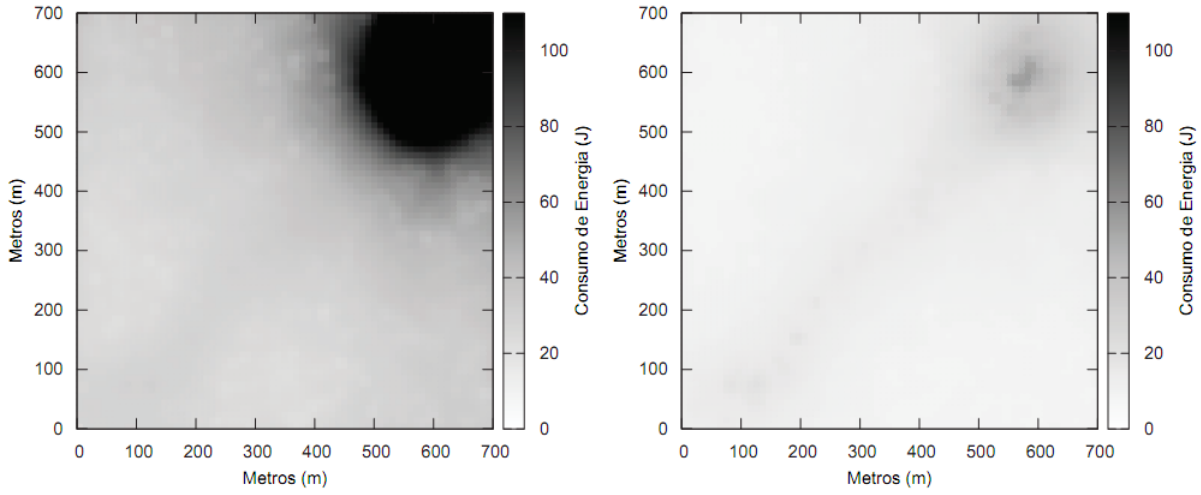


Figura 3.1: Visualização das áreas de correlação dentro do evento.
Adaptado de (AKYILDIZ; VURAN; AKAN, 2004)

Se comparada a exploração da correlação espacial com uma abordagem clássica como, por exemplo, clusterização, o consumo de energia dos nós sensores na abordagem clássica que estão dentro da área do evento é maior que dos outros nós da rede, como mostra a figura 3.2(a). Isso é devido ao fato de que todos os nós dentro da área do evento permanecerão em constante comunicação durante a ocorrência do evento.

As informações que estão dentro de uma mesma área de correlação são consideradas espacialmente correlacionadas e, em consequência, serão redundantes ou semelhantes. Com isso, é nomeado apenas um nó que representará os outros nós dentro de uma mesma área de correlação, enquanto os outros não transmitem seus dados coletados, consumindo menos energia dentro da área do evento, como mostra a figura 3.2(b).



(a) Abordagem clássica (b) Explora a correlação espacial

Figura 3.2: Consumo de energia na coleta de dados.
Fonte (VILLAS et al., 2011)

A figura 3.2 compara uma abordagem clássica de uma que explora a correlação espacial, e demonstra a vantagem do uso da correlação espacial obtendo menor consumo de energia principalmente dentro da área onde está ocorrendo o evento.

3.7 Considerações Finais

O presente capítulo mostrou diferentes soluções de protocolos e técnicas promissoras de agregação e de correlação entre os dados.

Foram apresentados os conceitos sobre topologias, arquiteturas e protocolos que foram considerados ao desenvolver o projeto de um protocolo para roteamento e disseminação de dados e que vai ser explicado com detalhes no capítulo 4.

Capítulo 4

O PROTOCOLO SCARP

Este capítulo descreve o SCARP, que é um protocolo de roteamento e disseminação de dados para RSSFs que utiliza a abordagem da correlação espacial com o propósito de alcançar consumo eficiente de energia.

Essa solução foi projetada com a preocupação de se obter o melhor resultado, por intermédio de uma abordagem simples e eficaz. Para isso, foi usada baixa complexidade nos algoritmos processados nos nós sensores.

O SCARP oferece consumo de energia eficiente e balanceado, independente da densidade na deposição dos nós sensores.

Este capítulo está organizado da seguinte forma: a seção 4.1 descreve as fases do algoritmo do SCARP e a seção 4.2 apresenta cenário, métricas e resultados da simulação.

4.1 Descrição do protocolo SCARP (*Spatial Correlation Aware Routing Protocol*)

O SCARP é um protocolo para roteamento de dados em RSSFs que visa garantir que:

- sejam entregues os dados coletados do ambiente monitorado;
- os dados entregues sejam confiáveis, representando a realidade coletada do evento;
- seja transmitido o mínimo de pacotes pela rede, com relação a pacotes de dados e também de *overhead*.

Para que os dados coletados sejam entregues, foi utilizada árvore de saltos conhecida como roteamento *multihop*, com a abordagem de roteamento e topologia plana, em que todos os nós podem ser iguais sem necessidade de uma infraestrutura diferenciada. No entanto, não é

necessário que a rede seja homogênea, o protocolo funciona em redes heterogêneas também, com a consequência de que o roteamento é plano e isso não irá influenciar no resultado.

O roteamento utilizando árvore de saltos é simples, que viabiliza seu uso em redes com nós sensores com restrições e baixo custo. Esse roteamento também possibilitou que o protocolo projetado fosse pró-ativo. A configuração da árvore de saltos é feita inicialmente e, em todo o funcionamento da rede, esse roteamento é mantido, salvo quando acontecem falhas em alguns nós, e no momento em que um desses nós for solicitado e não responder, rapidamente será substituído pelo mais próximo do *sink*.

O propósito de fazer uma solução pró-ativa é de que não haja troca de informações desnecessárias durante o funcionamento da rede caso haja uma grande quantidade de eventos ocorrendo. Desta maneira, o consumo de energia com mensagens de controle fica próximo de zero durante essa fase, porque todos os nós já sabem exatamente qual o menor caminho até o *sink*. Isso garante a minimização de mensagens de controle transmitidas.

Aproveitando as características *AdHoc* das RSSFs a fim de garantir a escalabilidade, não é necessário armazenar tabelas de roteamento fim a fim, a informação de roteamento se limita a um salto apenas.

O uso da correlação espacial garante o mínimo de troca de mensagens de controle e de dados dentro da área do evento, mantendo a confiabilidade dos dados, conforme apresentado na seção 3.6, criando áreas de correlação e elegendo nós que representem cada área de correlação.

No SCARP, a área de rede é dividida em células e, dentro da área de ocorrência do evento, cada célula elege seu nó representativo que irá enviar para o *sink* o dado coletado dentro dessa área. No entanto, para que o protocolo garanta à aplicação a confiabilidade dos dados, a aplicação tem a possibilidade de definir mais de um nó representativo dentro de cada célula. Isso vai depender dos requisitos da aplicação e das características do grau de correlação de cada fenômeno observado. Por exemplo, o controle de umidade de um ambiente não necessitaria de mais de um nó de cada célula, enquanto a temperatura de um ambiente onde há produtos inflamáveis pode necessitar de mais de um nó em cada célula.

O protocolo não elege um nó representativo fixo, além de cada área de correlação (célula) ter um nó representativo, esse nó representativo não é fixo, o protocolo define uma sequência de nós representativos dentro das células que estão dentro de um evento. Com isso, a cada envio de dados é usado um nó representativo e há a probabilidade de que cada nó use uma rota diferente, por isso é possível obter o balanceamento do consumo dentro das células e fora também usando diferentes rotas a cada envio de dados, e dependendo da posição de cada nó e de cada célula dentro do evento e da rede. Portanto, além de garantir a confiabilidade dos dados recebidos pelo *sink*, a consequência disso é um "balanceamento aleatório" no consumo de energia dos nós, conforme apresentado na figura 3.2 da seção 3.6. Outra característica importante do SCARP é

que não é utilizado *hardware* de localização que consumiria energia e adicionaria custo aos nós.

Finalmente, a solução usa um algoritmo de baixa complexidade de processamento. O maior objetivo nesse projeto foi desenvolver uma solução que considerasse as limitações das RSSFs de processamento, comunicação e consumo de energia. O protocolo é descrito a seguir.

O protocolo SCARP é dividido em três fases:

- **fase 1:** configuração da árvore de saltos com relação ao *sink* - esse processo é iniciado pelo *sink* e auxiliará cada nó ao encaminhar seus pacotes até o *sink*;
- **fase 2:** eleição dos nós de borda - dois nós localizados no último salto definido pela fase 1 são eleitos; estes nós possuem a maior distância entre si e são usados pela fase 3 na divisão da rede;
- **fase 3:** divisão da rede em células - coordenadas (A, B) são atribuídas às células através de divisão por salto, usando os dois nós de borda eleitos na fase 2.

As próximas seções apresentam com detalhes o algoritmo de funcionamento do protocolo SCARP.

4.1.1 SCARP Fase 1 - Construção da árvore de saltos

Como apresentado na seção 3.2, todos os nós sensores que são capazes de enviar uma mensagem diretamente para o *sink* são considerados como a um salto de distância do *sink*.

A configuração da árvore de saltos é iniciada com uma mensagem enviada pelo nó raiz da árvore (*sink*). Um nó, ao receber uma mensagem com um valor de salto, armazena este número, incrementa este número de salto da mensagem e a reencaminha. A cada retransmissão, esta operação é repetida pelos nós.

Nesta fase, o nó *sink* inicia o processo enviando a mensagem de configuração MsgConf, para seus nós vizinhos. MsgConf contém dois dados: [ID, HOP]. O valor do campo HOP contém o número de salto e esse valor é iniciado com 1. No campo ID, é armazenada a identificação do nó que enviou a mensagem. Cada nó, depois de receber as mensagens MsgConf dos nós vizinhos, irá executar o algoritmo 1, descrito abaixo e representado na figura 4.1.

Algoritmo 1: Construção da Árvore de Saltos

O *sink* envia uma mensagem MsgConf com HOP = 1, para todos seus vizinhos.

para todo *mensagem recebida* **faça**
  se $HOP_{(nó)} > HOP_{(msg)}$ **então**
    $\text{NextHop}_{(nó)} \longleftarrow \text{ID}_{(msg)}$
    $\text{HOP}_{(nó)} \longleftarrow \text{HOP}_{(msg)}$
    mensagem MsgConf[$\text{ID}_{(nó)}$, HOP + 1] é enviada aos nós vizinhos
  senão
    Descarta a mensagem
  fim
fim

Este processo de configuração é equivalente a um *flooding* controlado[1].

Como resultado, a mensagem MsgConf com os valores [ID, HOP + 1] é enviada para seus nós vizinhos pelos nós que acabaram de receber a MsgConf que, por sua vez, seguem o mesmo procedimento até o último nó da rede. Esses últimos nós da rede conterão o maior valor de salto dentro desta rede, sendo considerados assim, nós do último salto. O atributo NextHop, que está contido no nó, garante o roteamento pelo menor caminho até o *sink*, porque ele foi definido durante a configuração da árvore de saltos e com isso ele vai armazenar o ID do nó mais próximo do *sink*.

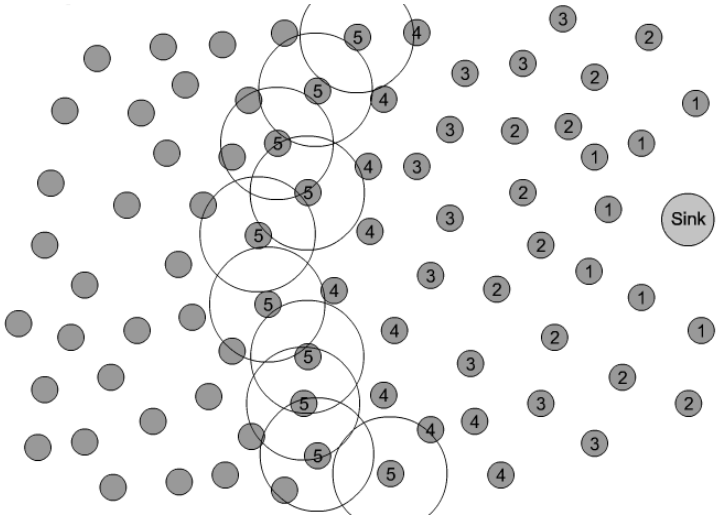


Figura 4.1: Construção da árvore de saltos.

Na figura 4.1 é representada a construção da árvore de saltos, sendo que os círculos maiores representam o momento em que está ocorrendo uma transmissão entre os nós. O número dentro

[1] *Flooding* ou inundação funciona assim: a cada pacote recebido, se esse nó não for o destino, o nó encaminhará esse pacote para todos os seus nós vizinhos menos para o que lhe enviou esse pacote. E a versão controlada é quando os pacotes recebidos e que já foram encaminhados são descartados.

do nó é o valor do atributo HOP do nó, onde é armazenado o valor do salto, os nós que não tem número são os que ainda não foram configurados. Ao final quando a árvore de saltos estiver totalmente configurada, todos os nós terão seu atributo HOP com o valor do salto.

4.1.2 SCARP Fase 2 - Eleição dos nós de borda

A Fase 2 é dividida nas seguintes etapas:

- descoberta de quais são os nós do último salto;
- seleção de quais desses nós de último salto serão candidatos a nós de borda;
- eleição dos dois nós de borda que irão configurar a rede em células na fase 3.

O primeiro passo da fase 2 é descobrir quais nós fazem parte do último salto. Essa informação é obtida naturalmente sem custo adicional com envio de mensagens de controle. É preciso apenas verificar se existe algum nó com valor de salto maior que o do próprio nó. É possível obter essa informação da seguinte maneira: cada nó tem um atributo chamado LastHop. Quando o nó é inicializado, esse atributo é inicializado também, e seu valor padrão é "verdadeiro". Após ser executado o algoritmo 1 para todas as mensagens, é executado o algoritmo 2 para a mensagem que for descartada pelo algoritmo 1.

Algoritmo 2: Descoberta dos Nós do Último Salto

```
para todo mensagem descartada no algoritmo 1 faça
    se HOP_(nó) < HOP_(msg) então
        LastHop ⟵ "falso"
    fim
fim
```

No momento em que o nó recebe a primeira MsgConf, é iniciado um temporizador[2]. O segundo passo é iniciado ao temporizador disparar. Neste momento se o atributo LastHop do nó continuar "verdadeiro", esse nó é considerado pelo protocolo como nó de último salto.

Todos os nós de último salto enviam uma mensagem MsgDiscoveryNeighbor que é formada apenas pelo seu ID. Como resultado, todos os nós do último salto irão armazenar uma lista com os IDs dos seus nós vizinhos. Uma mensagem MsgDiscoveryNeighborList com essa lista de vizinhos é enviada para seus nós vizinhos[3].

[2]O temporizador é necessário para garantir que não haverá mais mensagens MsgConf que possam mudar o valor do atributo LastHop. A existência do temporizador no código não implica que a rede seja sincronizada para esta solução, isso teria um custo energético e não se faz necessário a esse protocolo.

[3]Nós vizinhos são aqueles nós que estão dentro do raio de comunicação de um nó, e que, com apenas uma transmissão, são alcançados e podem receber a mensagem enviada.

Para toda mensagem MsgDiscoveryNeighborList recebida, o nó verifica se todos os nós da sua lista de vizinhos estão contidos nessas mensagens recebidas, se todos os nós contidos nas mensagens estiverem em sua lista de vizinhos, então, esse nó é um nó candidato a nó de borda. Caso contrário essa execução se encerra neste ponto. O resultado desse algoritmo está representado na figura 4.2.

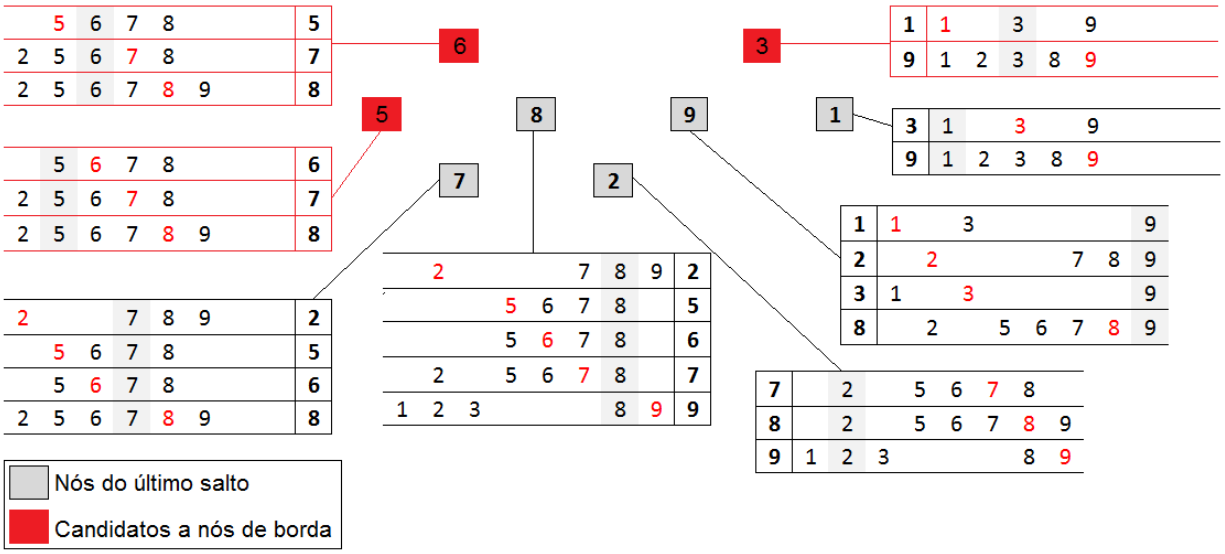


Figura 4.2: Seleção dos candidatos a Nós de borda.

A figura 4.2 mostra apenas o último salto de uma rede e demonstra a lista de mensagens MsgDiscoveryNeighborList recebidas por cada nó, e destacados em vermelho estão os nós que foram escolhidos como candidatos a nó de borda.

Com os candidatos selecionados, o próximo passo é eleger os dois nós de borda para a fase 3. Assim que um nó se intitula candidato, envia uma mensagem MsgBoard apenas com seu ID ao nó *sink*, informando a ele seu papel de candidato a nó de borda.

Após todas as mensagens MsgBoard chegarem ao *sink*, por não existir nenhuma informação geográfica, o critério para seleção é aleatório; neste caso na simulação descrita na seção 4.2 que foi executada elegendo um nó com o menor ID, que é chamado de nó A, e outro nó com o maior ID, chamado de nó B.

Ao final dessa fase, os dois nós de borda que dividirão a rede em células estarão eleitos e o resultado é representado na figura 4.3.

Vê-se que a fase 2 tem a tarefa de eleger os dois nós de borda. Portanto, é possível, dependendo da aplicação e da deposição, que a esses dois nós de borda sejam atribuídos seus papéis de nós de borda intencionalmente antes da rede começar a ser configurada.

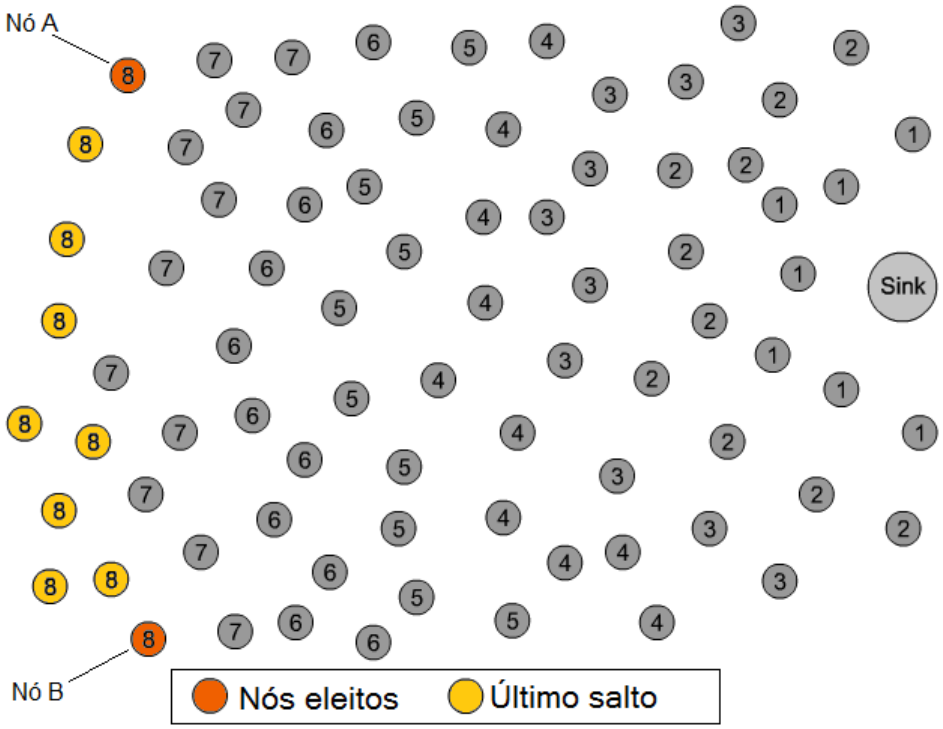


Figura 4.3: Nós de borda eleitos.

4.1.3 SCARP Fase 3 - Divisão da rede em células

Nesta última fase de configuração, a rede é dividida em células. Cada célula é considerada uma área de correlação, como foi explicado na seção 3.6.

Depois de eleitos os nós A e B, o processo de divisão da rede em células começa. Os dois nós escolhidos na fase 2 enviarão uma mensagem MsgConf da mesma forma que o nó *sink* fez na fase 1, criando a árvore de saltos com relação ao nó A e outra árvore com relação ao nó B. Esse procedimento irá atribuir aos nós as chamadas coordenadas (HopA, HopB) em cada nó.

A figura 4.4 mostra a rede divida em células, dentro de cada nó estão representadas as coordenadas (HOP, HopA, HopB), sendo HOP o valor do salto com relação ao *sink*, HopA e HopB são as coordenadas da célula, todo nó com coordenadas iguais fazem parte de uma mesma célula.

Assim, todos os nós que têm a mesma coordenada (HopA, HopB) fazem parte da mesma célula.

4.2 Avaliação de Desempenho

A proposta deste trabalho foi comparada ao protocolo DAARP (*Data Aggregation Aware Routing Protocol*) (VILLAS et al., 2009), que utiliza clusterização e agregação de dados. Sua

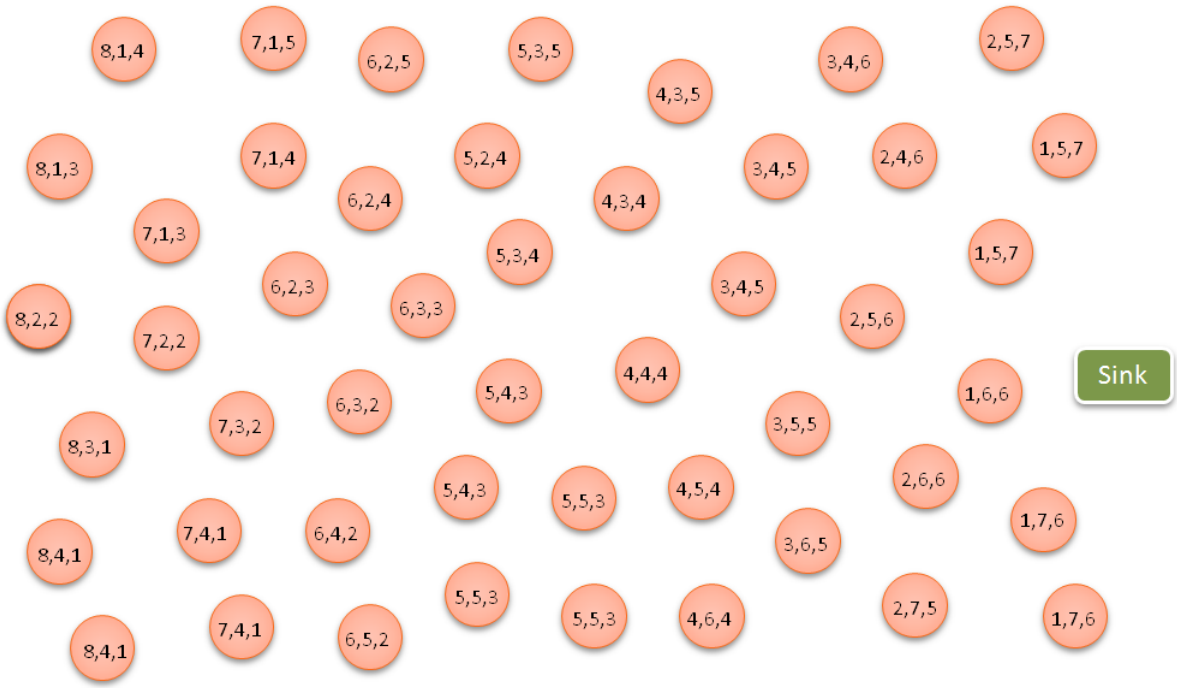


Figura 4.4: Rede dividida em células.

solução de roteamento utiliza árvore de Steiner (IVANOV, 1994) para obter o menor custo de transmissão entre o ponto de ocorrência dos eventos e o *sink*. A agregação é aplicada dentro dos *clusters* e fora deles nos pontos onde as rotas convergem-se através da solução de Steiner.

A avaliação foi realizada através de simulações utilizando o simulador SiNAlgo (*Simulator for Network Algorithms*) - versão 0.75.3 (SINALGO, 2007). SiNAlgo é um *framework* simulador escrito em Java para testar e validar algoritmos de redes, que permite a simulação de redes sem fio, abstraindo as camadas subjacentes da pilha de protocolos. As principais vantagens do SiNAlgo são:

- ter alto desempenho - as simulações podem ser executadas com muitos nós em tempo aceitável;
- ter independência de plataforma;
- ser *software* livre com fonte aberto (licença BSD).

4.2.1 Cenário da Simulação

A deposição dos nós no cenário de simulação foi não-determinística, com eventos ocorrendo aleatoriamente na área da rede. Considera-se que que a conectividade da rede de sensores sem fio é garantida, isto é, que não há pontos isolados sem conexão com o resto da rede.

No cenário simulado, existe apenas um *sink* e dez eventos bem distribuídos dentro da área de rede, com um raio de 80 metros. A densidade dos nós varia de acordo com o número de nós de cada simulação. Foram feitas em 5 simulações diferentes com 200, 400, 600, 800, 1000 nós. Portanto será maior a densidade em cada simulação, porque o tamanho da área da rede não mudará. A área da rede foi configurada em todas as simulações com o tamanho de 700x700m.

Durante a simulação, a troca de mensagens e disparo de eventos foram feitos assincronamente. Os parâmetros de simulação são descritos na tabela 4.1.

Tabela 4.1: Parâmetros da simulação

Parâmetros	Valores
Nó Sink	1
Nós sensores	(200, 400, 600, 800, 1000)
Raio de comunicação (m)	80
Qtde de eventos	10
Raio dos eventos (m)	80
Duração dos eventos (h)	8
Duração da simulação	10
Intervalo de envio de dados (seg)	60
Área de rede (m^2)	700 x 700

4.2.2 Resultados da Simulação

A simulação foi realizada em condições similares para ambos os protocolos SCARP e DA-ARP. As seguintes métricas foram utilizadas para confeccionar os gráficos desta seção:

- consumo de energia: é a média aritmética do consumo de energia de toda a rede;
- *overhead*: na transmissão de dados, *overhead* representa mensagens de controle, por exemplo, as mensagens usadas para a construção da árvore de saltos e divisão da rede em células;
- pacotes de dados entregues: é o total de pacotes de dados recebidos pelo *sink*;
- pacotes de dados transmitidos: é o total de pacotes de dados transmitidos pela rede inteira.

A escalabilidade é considerada uma métrica na sessão 1.2 pelo fato de a rede comporta-se de maneira estável desde baixas densidades até alta densidade.

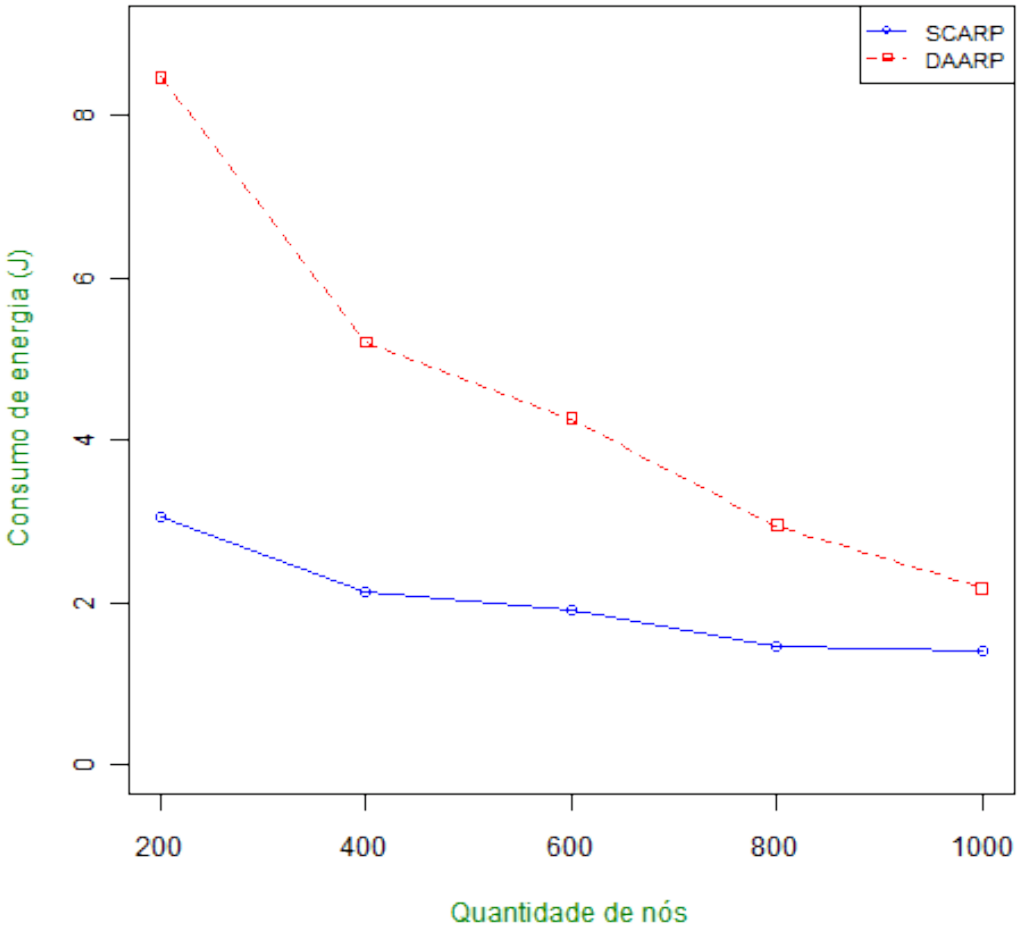


Figura 4.5: Média aritmética do consumo de energia de todos os nós da rede.

O número de simulações foi de 10 vezes a cada cenário[4]. Foi plotada, em todos os gráficos, a média aritmética das simulações em cada ponto. No gráfico da figura 4.5, SCARP mostrou estabilidade e menor consumo de energia independente da densidade dos nós.

A média de consumo de energia pelos nós sensores tende a diminuir quando o número de nós aumenta. O DAARP consome mais que o dobro de energia com relação ao SCARP em uma rede com 200 nós, mas continua consumindo uma quantidade considerável de energia a mais que o SCARP mesmo em redes com 1000 nós. Isso ocorre devido ao maior número de mensagens de controle que o DAARP transmite. No SCARP o uso de mensagens de controle foi otimizado para alcançar esse resultado.

[4]o sentido de cenário aqui é a cada mudança de densidade, simulando 10 vezes com 200 nós, 10 vezes com 400 nós e assim por diante

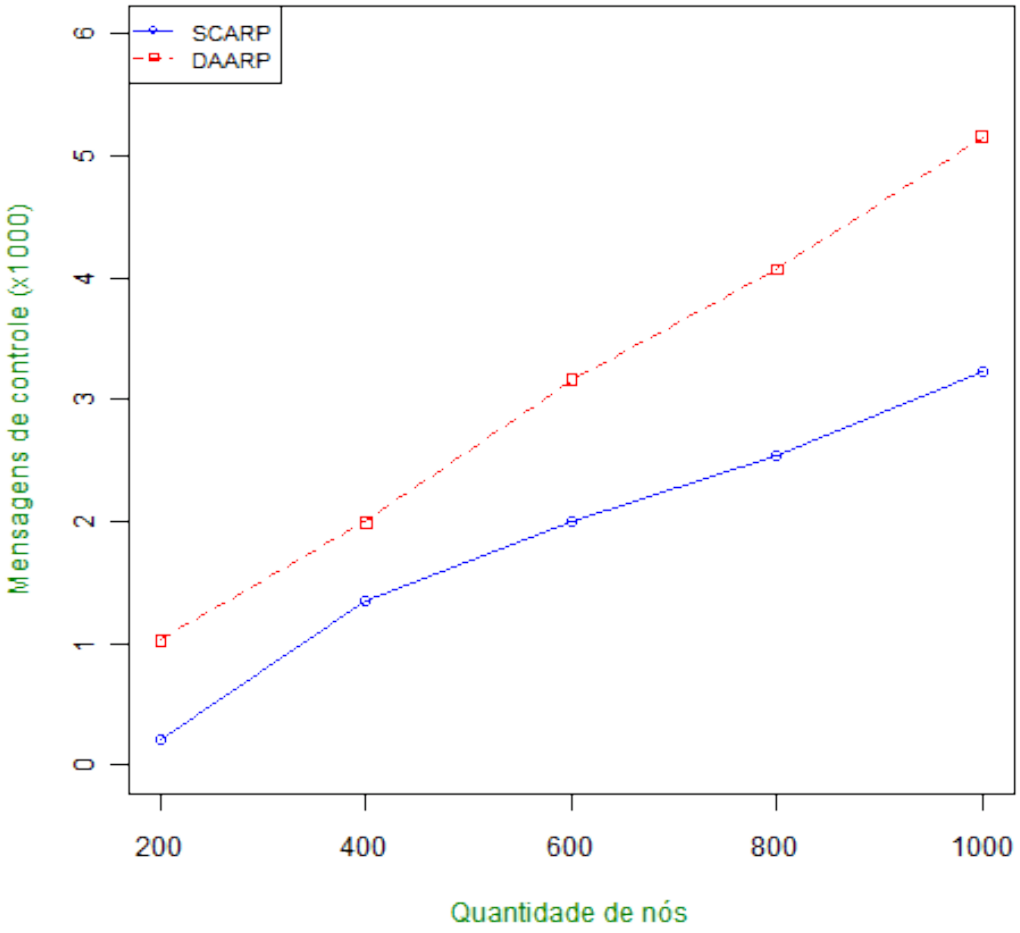


Figura 4.6: Total de mensagens de controle transmitidas pela rede.

O *overhead* para ambos protocolos aumentou com o aumento do número de nós. Entretanto o SCARP usa menos mensagens de controle que o DAARP, como é mostrado no gráfico da figura 4.6. Isso acontece porque o SCARP cria um grande número de mensagens de controle somente quando a rede executa sua configuração inicial, já no DAARP é utilizado grande número de mensagens de controle a cada novo evento para garantir a agregação a criação dos pontos de agregação entre a área do evento e o *sink*.

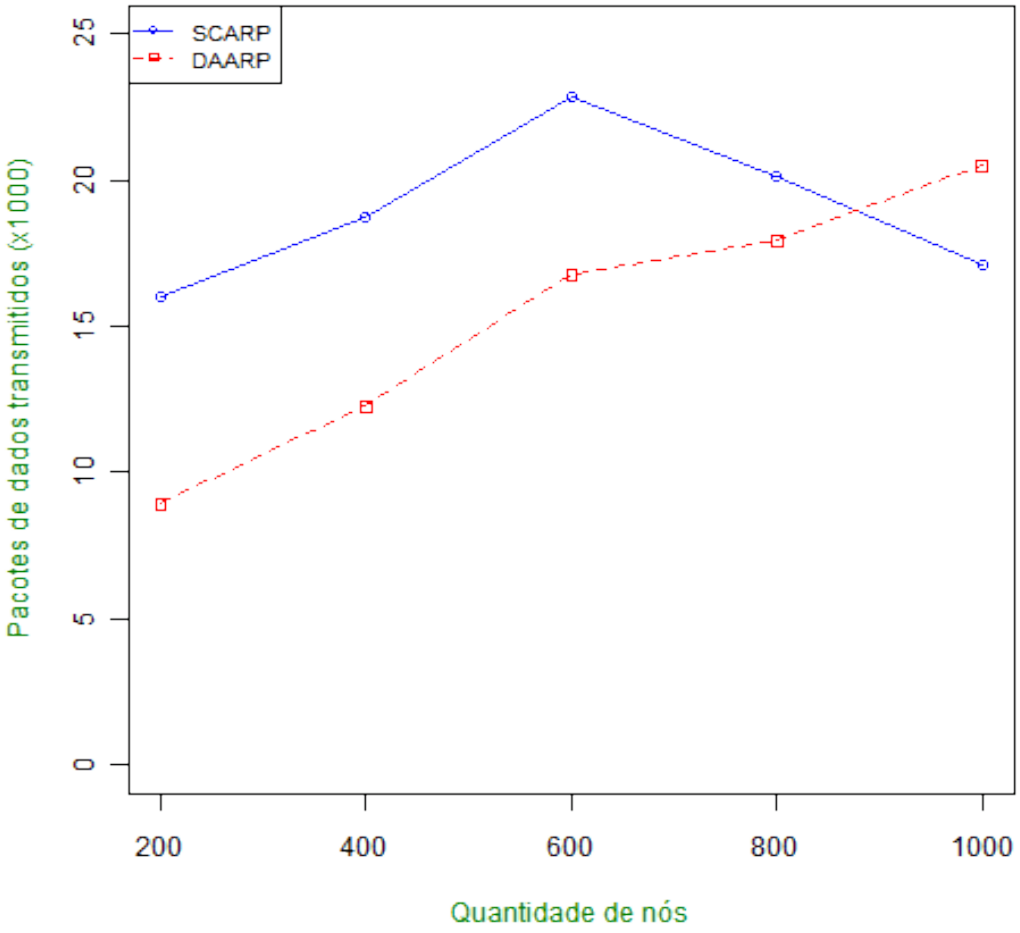


Figura 4.7: Total de pacotes de dados transmitidos pela rede.

Com o SCARP mais pacotes de dados irão trafegar pela rede, como mostra o gráfico da figura 4.7. É interessante observar que, a partir dos 600 nós até 1000 nós, momento em que a densidade começou a ficar alta, a quantidade de pacotes de dados que trafegaram pela rede diminuiu.

Isso acontece como consequência do uso de árvore de saltos no roteamento e da correlação espacial dentro da área do evento. Quando a densidade da rede aumenta, a distância entre cada salto vai ficando maior, e quanto maiores os saltos, menor é a quantidade de saltos entre a origem e o destino dos pacotes. Portanto, menos saltos significa menos transmissões até o *sink*. E o uso da correlação espacial, dentro da área do evento, evita maior transmissão de pacotes de dados dentro dela, porque apenas um nó de dentro de cada área de correlação vai transmitir a cada envio de dados entre área de correlação e *sink*.

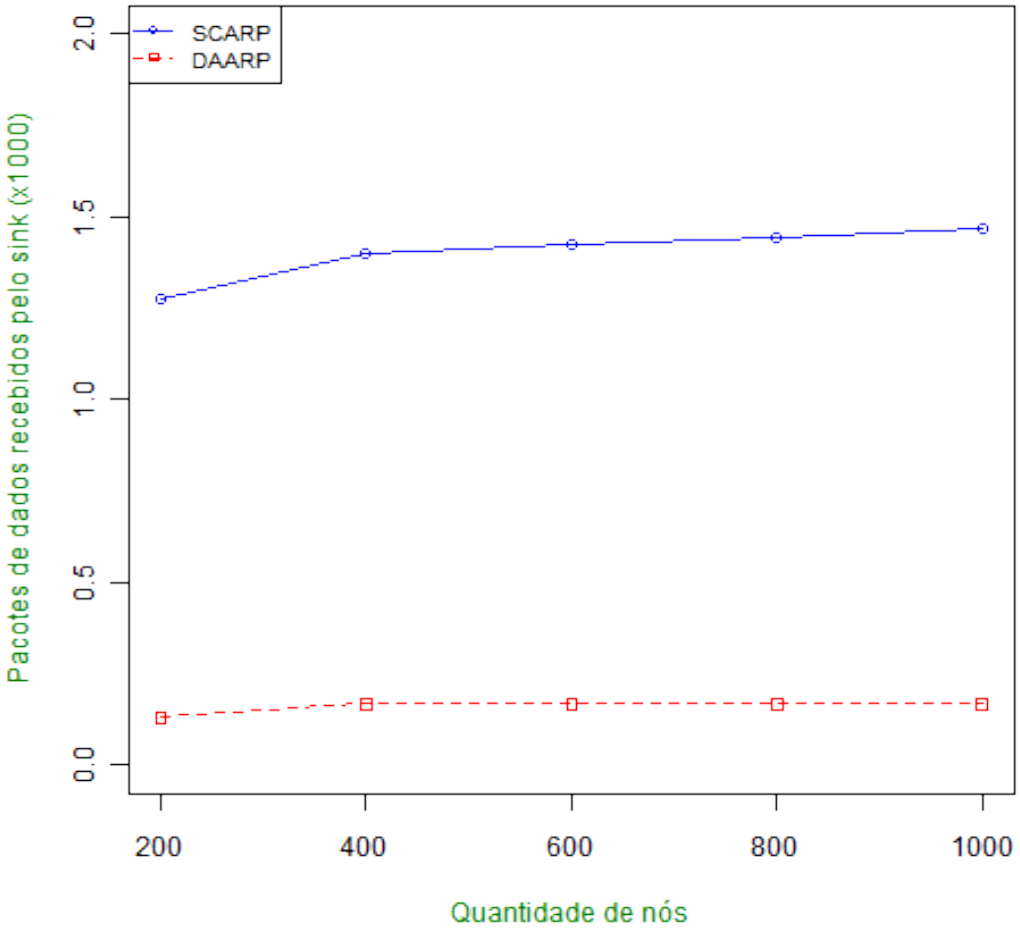


Figura 4.8: Total de pacotes de dados recebidos pelo *sink*.

No gráfico da figura 4.8, é mostrado que, com o SCARP, uma quantidade considerável de pacotes de dados chegam até o *sink*. Isso significa maior precisão e confiabilidade da informação que o *sink* recebe. Devido ao fato de não haver agregação fora da área de rede (porque não são criados pontos de agregação entre a área do evento e o *sink*), o SCARP transmite mais pacotes de dados durante o roteamento de pacotes, porém, em compensação, não tem transmissão intensa de pacotes de dados e mensagens de controle dentro da área do evento.

4.2.3 Considerações Finais

Este trabalho apresentou o protocolo SCARP, um protocolo para roteamento de dados com correlação espacial, com a principal meta de reduzir o número de transmissões entre os nós sensores, utilizando o conceito de correlação espacial e aumentar o tempo de vida da rede através do menor consumo de sua energia residual.

A avaliação dos resultados mostra que o SCARP superou o protocolo DAARP, que utiliza clusterização e agregação, mostrando que correlação espacial é uma importante e eficiente abordagem para se obter a redução do consumo de energia de uma RSSF e aumentar a longevidade da rede.

Capítulo 5

CONCLUSÕES

Este trabalho apresentou o protocolo SCARP (*Spatial Correlation Aware Routing Protocol*) para redes homogêneas de sensores sem fio, ciente de correlação espacial e com roteamento plano; O protocolo SCARP visa reduzir redundância de dados transmitidos pela rede até o nó sorvedouro, economizando energia e assim aumentando a longevidade da rede.

O protocolo proposto é capaz de trabalhar de forma autônoma nas RSSFs por meio do seu roteamento plano e correlação espacial dos dados.

A solução foi avaliada com relação a escalabilidade, taxa de entrega de pacotes, *overhead* e eficiência no consumo de energia e comparada com o protocolo DAARP, que utiliza agregação de dados, em vez de correlação espacial, para reduzir o número de transmissão de mensagens

Os estudos realizados e o desenvolvimento do trabalho apresentaram as seguintes conclusões:

- em relação a escalabilidade, o SCARP demonstrou estabilidade em redes com baixa e altíssima densidade;
- em relação a taxa de entrega, entregou quantidade suficiente de pacotes de dados dando confiança e credibilidade a eles;
- em relação ao *overhead*, o SCARP diminuiu o *overhead* principalmente dentro da área do evento, utilizando a correlação espacial como abordagem para disseminação de dados;
- em relação ao consumo de energia, o SCARP obteve consumo eficiente e balanceado, e possibilitou maior longevidade da rede de sensores.

As principais contribuições desta dissertação são:

- proposta e implementação de protocolo ciente de correlação espacial que pode ser usado

como base para futuras extensões e ou adaptações gerando novos protocolos ainda mais eficientes em termos de energia;

- avaliação comparativa de dois protocolos com os mesmos objetivos, porém utilizando técnicas diferentes (correlação espacial x agregação/fusão de dado).

O resultado deste trabalho de mestrado foi publicado no seguinte evento nacional:

FAVARIN, G.; VILLAS, L. A.; VILELA, M. A.; BOSSONARO, A. A.; MARCONDES, C. A. C.; ARAUJO, R. B. "Exploring Spatial Correlation through Virtual Cells in Wireless Sensor Networks", CBSEC'2011 - 1th Conferência Brasileira de Sistemas Embarcados Críticos. São Carlos, São Paulo, Brasil, Maio 11-13, 2011.

5.1 Trabalhos Futuros

O protocolo SCARP foi avaliado por meio de simulação, implementada no simulador SiNAlgo. O protocolo poderá ser implementado no ambiente experimental (*testbed*) WISEBED (*Wlreless SEnsor networks testBED*), em que será possível testar seu funcionamento em um ambiente real e comparado ao resultado obtido com a simulação.

Uma extensão do algoritmo para trabalhar com nós sensores móveis poderia ser feita. Nesse caso, o desafio é maior, mas seria interessante aproveitar o conceito do uso de células em uma rede móvel, garantindo a mesma eficiência no consumo de energia.

Outro trabalho é adicionar a correlação temporal nesse protocolo. Nessa solução, o intervalo de envio de dados é fixo e, se adicionada a correlação temporal, esse intervalo pode ser calculado e usado para manter a precisão nos dados do *sink* e melhorar o já eficiente consumo de energia.

REFERÊNCIAS BIBLIOGRÁFICAS

AGRE, J.; CLARE, L. An integrated architecture for cooperative sensing networks. *Computer*, v. 33, n. 5, p. 106–108, may 2000. ISSN 0018-9162. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=841788>.

AKKAYA, K.; YOUNIS, M. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, v. 3, p. 325–349, 2005.

AKYILDIZ, I. et al. A survey on sensor networks. *Communications Magazine, IEEE*, IEEE, v. 40, n. 8, p. 102–114, 2002.

AKYILDIZ, I.; VURAN, M.; AKAN, O. *On exploiting spatial and temporal correlation in wireless sensor networks*. [S.l.], 2004. 71–80 p.

AKYILDIZ, I. F.; VURAN, M. C. *Wireless sensor networks*. Chichester West Sussex U.K.: John Wiley & Sons, 2010. (Advanced Texts in Communications and Networking). ISBN 9780470515198.

AL-KARAKI, J.; KAMAL, A. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, v. 11, n. 6, p. 6–28, 2004. ISSN 1536-1284.

ALBUQUERQUE, E. C. R. de Oliveira e Célio V. N. de. Avaliação de protocolos de roteamento para redes ad hoc e rssf aplicados à tv digital interativa e cidades digitais. 2007. Disponível em: <http://www.ic.uff.br/ celio/papers/clei07-etienne.pdf>.

ASSUNÇÃO, H. P. *Gerenciamento de Serviços em Redes de Sensores Sem Fio Autonomicas*. Tese (Doutorado) — UFMG, Belo Horizonte, 2007.

BOUKERCHE, A.; CHENG, X.; LINUS, J. Energy-aware data-centric routing in microsensor networks. In: *Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*. New York, NY, USA: ACM, 2003. (MSWIM '03), p. 42–49. ISBN 1-58113-766-4. ACM ID: 941000.

BOUKERCHE, A. et al. Localization systems for wireless sensor networks. *Wireless Communications, IEEE*, v. 14, n. 6, p. 6 –12, december 2007. ISSN 1536-1284.

BRAGINSKY, D.; ESTRIN, D. Rumor routing algorthim for sensor networks. In: *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. New York, NY, USA: ACM, 2002. (WSNA '02), p. 22–31. ISBN 1-58113-589-0. ACM ID: 570742.

BULUSU, N. et al. Scalable coordination for wireless sensor networks: Self-configuring localization systems. In: *Proceedings of the Sixth International Symposium on*

Communication Theory and Applications (ISCTA '01). [s.n.], 2001. Disponível em: <http://lecs.cs.ucla.edu/ bulusu/papers/Bulusu01c.html>.

CERPA, A. et al. Habitat monitoring: application driver for wireless communications technology. In: *Workshop on Data communication in Latin America and the Caribbean*. New York, NY, USA: ACM, 2001. (SIGCOMM LA '01), p. 20–41. ISBN 1-58113-354-5. ACM ID: 371720.

CHEN, X. et al. Sensor network security: a survey. *Communications Surveys e Tutorials, IEEE*, v. 11, n. 2, p. 52–73, 2009. ISSN 1553-877X.

CHU, M. et al. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *INTERNATIONAL JOURNAL OF HIGH PERFORMANCE COMPUTING APPLICATIONS*, v. 16, ago. 2002. Disponível em: <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.120.6943>.

DAI, R.; AKYILDIZ, I. F. A spatial correlation model for visual information in wireless multimedia sensor networks. *Multimedia, IEEE Transactions on*, v. 11, n. 6, p. 1148–1159, 2009. ISSN 1520-9210.

DJENOURI, D.; KHELLADI, L.; BADACHE, A. A survey of security issues in mobile ad hoc and sensor networks. *Communications Surveys & Tutorials, IEEE*, v. 7, n. 4, p. 2–28, 2005. ISSN 1553-877X.

ESTRIN, D. et al. Next century challenges: scalable coordination in sensor networks. In: *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1999. (MobiCom '99), p. 263–270. ISBN 1-58113-142-9. ACM ID: 313556.

ESTRIN, D.; SAYEED, A.; SRIVASTAVA, M. *Tutorial "Wireless Sensor Networks"*. 2002. Http://nesl.ee.ucla.edu/tutorials/mobicom02. Disponível em: <http://nesl.ee.ucla.edu/tutorials/mobicom02>.

FARUQUE, J.; HELMY, A. RUGGED: RoUting on finGerprint gradients in sEnsor networks. In: *Proceedings of the The IEEE/ACS International Conference on Pervasive Services*. Washington, DC, USA: IEEE Computer Society, 2004. (ICPS '04), p. 179–188. ISBN 0-7695-2535-0. ACM ID: 1033557. Disponível em: <http://dx.doi.org/10.1109/ICPS.2004.27>.

FASOLO, E. et al. In-network aggregation techniques for wireless sensor networks: a survey. *Wireless Communications, IEEE*, v. 14, n. 2, p. 70–87, 2007. ISSN 1536-1284.

FENG, J.; KOUSHANFAR, F.; POTKONJAK, M. System-architectures for sensor networks issues, alternatives, and directions. In: *Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002 IEEE International Conference on*. [S.l.: s.n.], 2002. p. 226–231. ISBN 1063-6404.

GUPTA, I.; RIORDAN, D.; SAMPALLI, S. Cluster-head election using fuzzy logic for wireless sensor networks. In: *Communication Networks and Services Research Conference, 2005. Proceedings of the 3rd Annual*. Los Alamitos, CA, USA: IEEE Computer Society, 2005. p. 255–260. ISBN 0-7695-2333-1.

HEINZELMAN, W. R.; KULIK, J.; BALAKRISHNAN, H. Adaptive protocols for information dissemination in wireless sensor networks. In: *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1999. (MobiCom '99), p. 174–185. ISBN 1-58113-142-9. ACM ID: 313529.

HOBLOS, G.; STAROSWIECKI, M.; AITOUCHE, A. Optimal design of fault tolerant sensor networks. p. 467–472, 2000. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=897468>.

INTANAGONWIWAT, C.; GOVINDAN, R.; ESTRIN, D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: *Proceedings of the 6th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2000. (MobiCom '00), p. 56–67. ISBN 1-58113-197-6. ACM ID: 345920.

INTANAGONWIWAT, C. et al. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking (TON)*, v. 11, p. 2–16, fev. 2003. ISSN 1063-6692. ACM ID: 638335. Disponível em: <http://dx.doi.org/10.1109/TNET.2002.808417>.

IVANOV, A. *Minimal networks : the Steiner problem and its generalizations*. [S.l.]: CRC Press, 1994. ISBN 9780849386428.

KAHN, J. M.; KATZ, R. H.; PISTER, K. S. J. Next century challenges: mobile networking for smart dust. In: *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1999. (MobiCom '99), p. 271–278. ISBN 1-58113-142-9. ACM ID: 313558.

KARL, H.; WILLIG. *Protocols and architectures for wireless sensor networks*. 2007 john wiley & sons ltd. pbk. ed.. ed. Chichester West Sussex England ;;Hoboken NJ: John Wiley & Sons, 2007. ISBN 9780470519233.

KARP, B.; KUNG, H. T. GPSR: greedy perimeter stateless routing for wireless networks. In: *Proceedings of the 6th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2000. (MobiCom '00), p. 243–254. ISBN 1-58113-197-6. ACM ID: 345953.

KIM, S. *Structural health monitoring of the golden gate bridge*. fev. 2011. Http://www.eecs.berkeley.edu/~binetude/ggb/. Disponível em: <http://www.eecs.berkeley.edu/ binetude/ggb/>.

KRISHNAMACHARI, B.; ESTRIN, D.; WICKER, S. Modelling data-centric routing in wireless sensor networks. In: *In Proceedings of IEEE INFOCOM*. [S.l.: s.n.], 2002. v. 2, p. 1–11.

KUCUK, K. et al. A new wireless sensor networks localization scheme with antenna arrays. In: *Wireless Conference, 2008. EW 2008. 14th European*. [S.l.: s.n.], 2008. p. 1–6.

LAMBROU, T.; PANAYIOTOU, C. A survey on routing techniques supporting mobility in sensor networks. In: *Mobile Ad-hoc and Sensor Networks, 2009. MSN '09. 5th International Conference on*. [S.l.: s.n.], 2009. p. 78–85.

LE-TRUNG, Q. et al. Information storage, reduction and dissemination in sensor networks: A survey. In: *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*. [S.l.: s.n.], 2009. p. 1–6.

LI, L.; HALPERN, J. Y. Minimum-energy mobile wireless networks revisited. v. 1, p. 278–283 vol.1, jun. 2001. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=936317>.

LIN, C. R.; GERLA, M. Adaptive clustering for mobile wireless networks. *Selected Areas in Communications, IEEE Journal on*, v. 15, n. 7, p. 1265–1275, set. 1997. ISSN 0733-8716. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=622910>.

LINDSEY, S.; RAGHAVENDRA, C. PEGASIS: power-efficient gathering in sensor information systems. In: *Proceedings, IEEE Aerospace Conference*. Big Sky, MT, USA: [s.n.], 2002. v. 3, p. 1125–1130. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1035242>.

LOUREIRO, A. A. et al. Redes de sensores sem fio. In: *texto de mini-cursos do XXI Simpósio Brasileiro de Redes de Computadores (SBRC)*. [S.l.: s.n.], 2003. p. 179–226. ISBN 85-88442-48-5.

MAINWARING, A. et al. Wireless sensor networks for habitat monitoring. In: *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. New York, NY, USA: ACM, 2002. (WSNA '02), p. 88–97. ISBN 1-58113-589-0. ACM ID: 570751.

MANJESHWAR, A.; AGRAWAL, D. TEEN: a routing protocol for enhanced efficiency in wireless sensor networks. In: *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*. San Francisco, CA, USA: [s.n.], 2001. p. 2009–2015. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=925197>.

NAKAMURA, E. F. Planejamento dinâmico para controle de cobertura e conectividade em redes de sensores sem fio. p. 182–191, 2004.

NAKAMURA, E. F.; LOUREIRO, A. A. F.; FRERY, A. C. Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Computing Surveys (CSUR)*, v. 39, set. 2007. ISSN 0360-0300. ACM ID: 1267073.

NATH, B.; NICULESCU, D. Routing on a curve. *ACM SIGCOMM Computer Communication Review*, v. 33, p. 155–160, jan. 2003. ISSN 0146-4833. ACM ID: 774788.

NING, X.; CASSANDRAS, C. G. Optimal cluster-head deployment in wireless sensor networks with redundant link requirements. In: *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007. (ValueTools '07). ISBN 978-963-9799-00-4. ACM ID: 1345294. Disponível em: <http://portal.acm.org/citation.cfm?id=1345263.1345294>.

OLIVEIRA, H. de et al. An efficient directed localization recursion protocol for wireless sensor networks. *Computers, IEEE Transactions on*, v. 58, n. 5, p. 677–691, may 2009. ISSN 0018-9340.

PATTEM, S.; KRISHNAMACHARI, B.; GOVINDAN, R. The impact of spatial correlation on routing with compression in wireless sensor networks. p. 28– 35, abr. 2004. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1307320>.

PERKINS, C. *Ad hoc networking*. Boston MA: Addison-Wesley, 2000. ISBN 9780201309768.

PINTO, A. J. G. *Mecanismo de Agregação de Dados Empregando Técnicas Paramétricas em Redes de Sensores*. Tese (Doutorado) — UFRJ, Rio de Janeiro, jun. 2004. Disponível em: <http://www.gta.ufrj.br/ftp/gta/TechReports/Antonio04/Antonio04.pdf>.

QI, H.; WANG, X.; IYENGAR, S. S. Multisensor data fusion in distributed sensor networks using mobile agents. *IN PROCEEDINGS OF 5TH INTERNATIONAL CONFERENCE ON INFORMATION FUSION*, p. 11—16, 2001. Disponível em: <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.7251>.

RAGHUNATHAN, V. et al. Energy-aware wireless microsensor networks. *Signal Processing Magazine, IEEE*, v. 19, n. 2, p. 40–50, mar. 2002. ISSN 1053-5888. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=985679>.

RODOPLU, V.; MENG, T. H. Minimum energy mobile wireless networks. *Selected Areas in Communications, IEEE Journal on*, v. 17, n. 8, p. 1333–1344, ago. 1999. ISSN 0733-8716. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=779917>.

RUIZ, L. B. et al. Arquitetura para rede de sensores sem fio. In: . Gramado, Rio Grande do Sul: [s.n.], 2004.

RUIZ, L. B.; NOGUEIRA, J. M. S. *MANNA: Uma Arquitetura para Gerenciamento de Redes de Sensores Sem Fio*. Tese (Doutorado) — UFMG, Belo Horizonte, dez. 2003. Disponível em: <http://hdl.handle.net/1843/SLBS-5WAJRV>.

RUIZ, L. B.; NOGUEIRA, J. M. S.; LOUREIRO, A. A. F. capitulo III: sensor network management. In: *Handbook of Sensor Network: Compact Wireless and Wired Sensing Systems*. [S.l.]: CRC Press, 2004. v. 1.

SADAGOPAN, N.; KRISHNAMACHARI, B.; HELMY, A. The ACQUIRE mechanism for efficient querying in sensor networks. p. 149– 155, maio 2003. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1203365>.

SCHURGERS, C.; SRIVASTAVA, M. B. Energy efficient routing in wireless sensor networks. v. 1, p. 357– 361 vol.1, 2001. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=985819>.

SHAH, R. C.; RABAEY, J. M. Energy aware routing for low energy ad hoc sensor networks. v. 1, p. 350– 355 vol.1, mar. 2002. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=993520>.

SHEN, C.; SRISATHAPORNPHAT, C.; JAIKAEO, C. Sensor information networking architecture and applications. *Personal Communications, IEEE*, v. 8, n. 4, p. 52–59, ago. 2001. ISSN 1070-9916. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=944004>.

SHIH, E. et al. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In: *Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2001. (MobiCom '01), p. 272–287. ISBN 1-58113-422-3. ACM ID: 381703.

SINALGO, E. Z. *SiNAlgo - Algorithms for Network Simulator*. Switzerland, disco.ethz.ch: DCG - Distributed Computing Group, 2007. Disponível em: <http://dcg.ethz.ch/projects/sinalgo/>.

SOHRABI, K. et al. Protocols for self-organization of a wireless sensor network. *Personal Communications, IEEE*, v. 7, n. 5, p. 16–27, out. 2000. ISSN 1070-9916. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=878532>.

SRIKANTH, V.; BABU, I. R. Cluster head selection for wireless sensor networks: A survey. In: . [S.l.: s.n.], 2009. v. 5, p. 44–53.

TILAK, S.; Abu-Ghazaleh, N. B.; HEINZELMAN, W. A taxonomy of wireless micro-sensor network models. *ACM SIGMOBILE Mobile Computing and Communications Review*, v. 6, p. 28–36, abr. 2002. ISSN 1559-1662. ACM ID: 565708.

VILLAS, L. et al. Explorando a correlação espacial na coleta de dados em redes de sensores sem fio. *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2011.

VILLAS, L. A. et al. A reliable and data aggregation aware routing protocol for wireless sensor networks. In: *Proceedings of the 12th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*. New York, NY, USA: ACM, 2009. (MSWiM '09), p. 245–252. ISBN 978-1-60558-616-8. ACM ID: 1641846.

VURAN, M. C.; AKAN, O. B.; AKYILDIZ, I. F. Spatio-temporal correlation: theory and applications for wireless sensor networks. *Computer Networks*, v. 45, n. 3, p. 245–259, jun. 2004. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128604000519>.

VURAN, M. C.; AKYILDIZ, I. Spatial correlation-based collaborative medium access control in wireless sensor networks. *Networking, IEEE/ACM Transactions on*, v. 14, n. 2, p. 316–329, 2006. ISSN 1063-6692.

WANG, Y.; ATTEBURY, G.; RAMAMURTHY, B. A survey of security issues in wireless sensor networks. *Communications Surveys & Tutorials, IEEE*, v. 8, n. 2, p. 2–23, 2006. ISSN 1553-877X.

WARNEKE, B. et al. Smart dust: communicating with a cubic-millimeter computer. *Computer*, v. 34, n. 1, p. 44–51, jan. 2001. ISSN 0018-9162. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=895117>.

XU, N. et al. A wireless sensor network for structural monitoring. In: *Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004. (SenSys '04), p. 13–24. ISBN 1-58113-879-2. ACM ID: 1031498.

XU, Y.; HEIDEMANN, J.; ESTRIN, D. Geography-informed energy conservation for ad hoc routing. In: *Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2001. (MobiCom '01), p. 70–84. ISBN 1-58113-422-3. ACM ID: 381685.

YAO, Y.; GEHRKE, J. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, v. 31, p. 9–18, set. 2002. ISSN 0163-5808. ACM ID: 601861.

YICK, J.; MUKHERJEE, B.; GHOSAL, D. Wireless sensor network survey. *Computer Networks*, v. 52, n. 12, p. 2292–2330, ago. 2008. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/B6VRG-4S8TBBT-1/2/b242d2fd1f6d2cf5c6fce0a24c4cb029>.

YOON, S.; SHAHABI, C. Exploiting spatial correlation towards an energy efficient clustered aggregation technique (CAG) [wireless sensor network applications]. In: *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*. [S.l.: s.n.], 2005. v. 5, p. 3307–3313.

YOON, S.; SHAHABI, C. The clustered AGgregation (CAG) technique leveraging spatial and temporal correlations in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, v. 3, mar. 2007. ISSN 1550-4859. ACM ID: 1210672.

YUAN, J.; CHEN, H. The optimized clustering technique based on spatial-correlation in wireless sensor networks. In: *Information, Computing and Telecommunication, 2009. YC-ICT '09. IEEE Youth Conference on*. [S.l.: s.n.], 2009. p. 411–414.

GLOSSÁRIO

BSD (licença) – *Berkeley Software Distribution*

DAARP – *Data-Aggregation Aware Routing Protocol*

DD – *Direct Diffusion*

GPS – *Global Positioning System*

IP – *Internet Protocol*

LEACH – *Low Energy Adaptive Clustering Hierarchy*

MENS – *Micro-Electro-Mechanical Systems*

PLL – *Phase Locked Loop*

QoS – *Qualidade de Serviço*

RF – *Rásio Frequência*

RSSF – *Rede de Sensores Sem Fio*

SCARP – *Spatial Correlation Aware Routing Protocol*

SMP – *Sensor Management Protocol*

SQDDP – *Sensor Query and Data Dissemination Protocol*

SiNAlgo – *Simulator for Network Algorithms*

TADAP – *Task Assignment and Data Advertisement Protocol*

WINDIS – *WIreless Networking and Distributed Interactive SImulation Laboratory*

WISEBED – *WIreless SEnsor networks testBED*

WSN – *Wireless Sensor Networks*

Anexo A

CÓDIGO DO SCARP NO SINALGO

Listagem A.1: SCARPNode.java

```
1  package projects.SCARP.nodes.nodeImplementations;
2
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import java.util.ArrayList;
6  import java.util.List;
7
8  import projects.SCARP.nodes.messages.MsgBoard;
9  import projects.SCARP.nodes.messages.MsgData;
10 import projects.SCARP.nodes.messages.MsgDiscoveryNeighbor;
11 import projects.SCARP.nodes.messages.MsgDiscoveryNeighborList;
12 import projects.SCARP.nodes.messages.MsgHop;
13 import projects.SCARP.nodes.messages.MsgHopCellA;
14 import projects.SCARP.nodes.messages.MsgHopCellB;
15 import projects.SCARP.nodes.messages.MsgRepresentative;
16 import projects.SCARP.nodes.messages.MsgTreeCost;
17 import projects.SCARP.nodes.timers.DirectMessageTimer;
18 import projects.SCARP.nodes.timers.EventTimer;
19 import projects.SCARP.nodes.timers.MsgTimer;
20 import projects.SCARP.nodes.timers.SCARPTimer;
21 import projects.energy.EnergyMode;
22 import projects.energy.simple.SimpleEnergy;
23 import sinalgo.configuration.CorruptConfigurationEntryException;
24 import sinalgo.configuration.WrongConfigurationException;
25 import sinalgo.gui.transformation.PositionTransformation;
26 import sinalgo.nodes.Node;
27 import sinalgo.nodes.messages.Inbox;
28 import sinalgo.nodes.messages.Message;
29 import sinalgo.runtime.Global;
30 import sinalgo.tools.Tools;
31 import sinalgo.tools.logging.Logging;
32
```

```
33 public class SCARPNode extends Node {
34
35     private int hop = 19999;
36     private int hopA = 19999;
37     private int hopB = 19999;
38     private static int IDA;
39     private static int IDB;
40     private boolean lastHop = true;
41     private boolean neighborsTimed = false;
42     private boolean neighborsSended = false;
43     private boolean msgBoardTimed = false;
44     private boolean firstSendData = true;
45     private boolean representative = false;
46     private Node nextNode;
47     public SimpleEnergy bateria = new SimpleEnergy();
48     private List<Integer> election = new ArrayList<Integer>();
49     private List<Integer> neighbor = new ArrayList<Integer>();
50     private List<List<Integer>> neighbors = new ArrayList<List<Integer
           >>();
51     private List<Integer> cells = new ArrayList<Integer>();
52     public enum TNO {TNO_SENDMSGHOP, TNO_SENDMSGHOPCELLA,
           TNO_SENDMSGHOPCELLB, TNO_BOARDELECTED,
           TNO_SENDDISCOVERYNEIGHBOR, TNO_SENDDISCOVERYNEIGHBORLIST,
           TNO_SENDMSGBOARD, TNO_MONITORING};
53     private SCARPTimer timerSENDMSGHOP, timerSENDMSGHOPCELLA,
           timerSENDMSGHOPCELLB, timerBOARDELECTED,
           timerSENDMSGDISCOVERYNEIGHBOR,
           timerSENDMSGDISCOVERYNEIGHBORLIST, timerSENDMSGBOARD,
           timerMONITORING;
54     private int eventID = 0;
55     private ArrayList<Integer> sonsInTree = new ArrayList<Integer>();
56     public static Logging scarp = Logging.getLogger("scarpLog.txt",
           true);
57     public static List<SCARPNode> logNode = new ArrayList<SCARPNode>()
           ;
58
59     //** Parametros de Simulacao **//
60
61     public static int probSend;
62     public static int eventsAmount;
63     public static int eventsTimes;
64     public static int density;
65     private double endEventTimer;
66     public static double dataRate = 60;
67     public static int eventSize;
68     public static int communicationRadius;
69     public static double dropRate;
70     public static double erro;
71
72     //** Variaveis utilizadas para armazenar valores de logs **//
73
```

```
74      public static int overheads = 0;
75      public static int detects = 0;
76      public static int dataPackets = 0;
77      public static int notifications = 0;
78      public static int receivers = 0;
79      public static int edges = 0;
80      public static int edgesWorstCase = 0;
81      public static double simulationTime = 0;
82      public static int[] eventIDCost;
83      private boolean[] eventIDTreeAux;
84      private boolean[] eventIDTree;
85      private int treeCostRecv = 0;
86      private int treeCost = 0;
87
88      public boolean insideEvent(sinalgo.nodes.Position p, int eventID){
89          double xc = 0;
90          double yc = 0;
91          double r = 0;
92
93          try {
94              xc = sinalgo.configuration.Configuration.getDoubleParameter(
                    "Event/Xposition" + eventID);
95              yc = sinalgo.configuration.Configuration.getDoubleParameter(
                    "Event/Yposition" + eventID);
96              r = sinalgo.configuration.Configuration.getDoubleParameter("
                    Event/EventSize");
97          } catch (CorruptConfigurationEntryException e) {
98              e.printStackTrace();
99              System.exit(0);
100         }
101         if (Math.pow((xc - p.xCoord),2) + Math.pow((yc -p.yCoord),2) <
                Math.pow(r/2,2)){
102             this.setEventID(eventID);
103             return true;
104         } else {
105             return false;
106         }
107     }
108
109     public void startDetection(){
110         this.setColor(Color.blue);
111         timerMONITORING = new SCARPTimer(this, TNO.TNO_MONITORING);
112         timerMONITORING.tnoStartRelative(0.00001, this, TNO.
                TNO_MONITORING);
113     }
114
115     public void timeout(TNO tno){
116         switch(tno){
117             case TNO_SENDMSGHOP:
118                 sendMsgHop(this, this.hop + 1, this.getEventID());
119                 break;
```

```
120
121             case TNO_SENDMSGHOPCELLA:
122                 sendMsgHopCellA(this.hopA + 1);
123                 break;
124
125             case TNO_SENDMSGHOPCELLB:
126                 sendMsgHopCellB(this.hopB + 1);
127                 break;
128
129             case TNO_SENDDISCOVERYNEIGHBOR:
130                 if (this.lastHop) {
131                     sendMsgDiscoveryNeighbor(this.ID);
132                 }
133                 break;
134
135             case TNO_SENDDISCOVERYNEIGHBORLIST:
136                 if ((!neighborsSended) && (lastHop)) {
137                     sendMsgDiscoveryNeighborList(this.neighbor);
138                     neighborsSended = true;
139                 } else if (this.isBoard()) {
140                     this.setColor(Color.ORANGE);
141                     sendMsgBoard(this.ID);
142                     timerBOARDELECTED = new SCARPTimer(this, TNO.
                            TNO_BOARDELECTED);
143                     timerBOARDELECTED.tnoStartRelative(3, this, TNO.
                            TNO_BOARDELECTED);
144                 }
145                 break;
146
147             case TNO_SENDMSGBOARD:
148                 IDA = min(election);
149                 IDB = max(election);
150                 break;
151
152             case TNO_MONITORING:
153                 sendData();
154                 break;
155
156             case TNO_BOARDELECTED:
157                 tratarMsgBoardElected();
158         }
159     }
160
161     @Override
162     public void draw(Graphics g, PositionTransformation pt, boolean
            highlight) {
163         if (this.ID == 1) {
164             drawNodeAsSquareWithText(g, pt, highlight, "Sink", 8, Color.
                    WHITE);
165             setColor(Color.LIGHT_GRAY);
166         } else if ((hopA == 19999) && (hopB == 19999)) {
```

```
167             drawNodeAsDiskWithText(g, pt, highlight, "" + ID + ",A,B",
                    7, Color.WHITE);
168         } else if ((hopA != 19999) && (hopB == 19999)) {
169             drawNodeAsDiskWithText(g, pt, highlight, "" + ID + "," +
                    hopA + ",B", 7, Color.WHITE);
170         } else if ((hopA == 19999) && (hopB != 19999)) {
171             drawNodeAsDiskWithText(g, pt, highlight, "" + ID + ",A," +
                    hopB, 7, Color.WHITE);
172         } else {
173             drawNodeAsDiskWithText(g, pt, highlight, ID + "," + hopA + "
                    ," + hopB, 7, Color.WHITE);
174         }
175     }
176
177     @Override
178     public void init() {
179         try {
180
181             simulationTime = sinalgo.configuration.Configuration.
                    getDoubleParameter("SimTime");
182             probSend = sinalgo.configuration.Configuration.
                    getIntegerParameter("ProbSend");
183             eventsAmount = sinalgo.configuration.Configuration.
                    getIntegerParameter("Event/NumEvents");
184             eventsTimes = sinalgo.configuration.Configuration.
                    getIntegerParameter("Event/Time");
185             density = sinalgo.configuration.Configuration.
                    getIntegerParameter("Density");
186
187             eventIDCost = new int[eventsAmount + 1];
188             eventIDTree = new boolean[eventsAmount + 1];
189             eventIDTreeAux = new boolean[eventsAmount + 1];
190
191             //** Schedule events to occur during the simulation **//
192             for (int i = 1; i <= eventsAmount; i++) {
193                 EventTimer t = new EventTimer(i);
194                 t.startEventAbsolute(sinalgo.configuration.Configuration.
                        getDoubleParameter("Event/EventStart"+i), this, i);
195             }
196
197             endEventTimer = sinalgo.configuration.Configuration.
                    getDoubleParameter("Event/EventEnd");
198             dataRate = sinalgo.configuration.Configuration.
                    getDoubleParameter("Event/DataRate");
199             eventSize = sinalgo.configuration.Configuration.
                    getIntegerParameter("Event/EventSize");
200             communicationRadius = sinalgo.configuration.Configuration.
                    getIntegerParameter("UDG/rMax");
201             dropRate = sinalgo.configuration.Configuration.
                    getIntegerParameter("TaxadePerda/dropRate");
```

```
202             erro = sinalgo.configuration.Configuration.
                    getDoubleParameter("TaxadePerda/dropRate");
203
204         } catch (CorruptConfigurationEntryException e) {
205             // TODO: handle exception
206             e.printStackTrace();
207             System.exit(0);
208         }
209
210         if (this.ID == 1){
211             this.hop = 1;
212             this.lastHop = false;
213             sendMsgHop(this, this.hop, 0);
214         }
215         //O primeiro ID da lista neighbor é sempre o proprio ID
216         neighbor.add(this.ID);
217     }
218
219     @Override
220     public void handleMessages(Inbox inbox) {
221         while (inbox.hasNext()) {
222             Message msg = inbox.next();
223
224             if (msg instanceof MsgHop) {
225                 tratarMsgHop((MsgHop) msg);
226             } else if (msg instanceof MsgHopCellA) {
227                 tratarMsgHopCellA((MsgHopCellA) msg);
228             } else if (msg instanceof MsgHopCellB) {
229                 tratarMsgHopCellB((MsgHopCellB) msg);
230             } else if ((msg instanceof MsgDiscoveryNeighbor) && (this.
                    lastHop)) {
231                 tratarMsgDiscoveryNeighbor((MsgDiscoveryNeighbor) msg);
232             } else if ((msg instanceof MsgDiscoveryNeighborList) && (
                    this.lastHop)) {
233                 tratarMsgDiscoveryNeighborList((MsgDiscoveryNeighborList)
                        msg);
234             } else if (msg instanceof MsgBoard) {
235                 tratarMsgBoard((MsgBoard) msg);
236             } else if (msg instanceof MsgData) {
237                 tratarMsgData((MsgData) msg);
238             } else if (msg instanceof MsgRepresentative) {
239                 tratarMsgRepresentative();
240             }
241             bateria.spend(EnergyMode.RECEIVE);
242         }
243     }
244
245     private void tratarMsgHop(MsgHop msg) {
246         if (this.hop > msg.getHop()) {
247             if (this.nextNode == null) {
```

```
248                 timerSENDMSGDISCOVERYNEIGHBOR = new SCARPTimer(this, TNO.
                        TNO_SENDDISCOVERYNEIGHBOR);
249                 timerSENDMSGDISCOVERYNEIGHBOR.tnoStartRelative(1, this,
                        TNO.TNO_SENDDISCOVERYNEIGHBOR);
250             }
251             this.nextNode = msg.getNode();
252             this.hop = msg.getHop();
253             this.setEventID(msg.getEventID());
254             // Inicializa um tempo que vai executar o metodo fire() ao
                    termino desse tempo
255             timerSENDMSGHOP = new SCARPTimer(this, TNO.TNO_SENDMSGHOP);
256             timerSENDMSGHOP.tnoStartRelative(0.1, this, TNO.
                    TNO_SENDMSGHOP);
257         } else if (this.hop < msg.getHop() - 1) {
258             this.lastHop = false;
259             this.setColor(Color.GRAY);
260         }
261     }
262
263     private void tratarMsgHopCellA(MsgHopCellA msg) {
264         if (this.hopA > msg.getHop()) {
265             this.hopA = msg.getHop();
266             // Inicializa um tempo que vai executar o metodo fire() ao
                    termino desse tempo
267             timerSENDMSGHOPCELLA = new SCARPTimer(this, TNO.
                    TNO_SENDMSGHOPCELLA);
268             timerSENDMSGHOPCELLA.tnoStartRelative(0.1, this, TNO.
                    TNO_SENDMSGHOPCELLA);
269         }
270     }
271
272     private void tratarMsgHopCellB(MsgHopCellB msg) {
273         if (this.hopB > msg.getHop()) {
274             this.hopB = msg.getHop();
275             // Inicializa um tempo que vai executar o metodo fire() ao
                    termino desse tempo
276             timerSENDMSGHOPCELLB = new SCARPTimer(this, TNO.
                    TNO_SENDMSGHOPCELLB);
277             timerSENDMSGHOPCELLB.tnoStartRelative(0.1, this, TNO.
                    TNO_SENDMSGHOPCELLB);
278         }
279     }
280
281     private void tratarMsgDiscoveryNeighbor(MsgDiscoveryNeighbor msg)
            {
282         neighbor.add(msg.getId());
283         if (!neighborsTimed) {
284             neighborsTimed = true;
285             // Inicializa um tempo que vai executar o metodo fire() ao
                    termino desse tempo
```

```
286             timerSENDMSGDISCOVERYNEIGHBORLIST = new SCARPTimer(this, TNO
                    .TNO_SENDDISCOVERYNEIGHBORLIST);
287             timerSENDMSGDISCOVERYNEIGHBORLIST.tnoStartRelative(1, this,
                    TNO.TNO_SENDDISCOVERYNEIGHBORLIST);
288         }
289     }
290
291     private void tratarMsgDiscoveryNeighborList(
            MsgDiscoveryNeighborList msg) {
292         neighbors.add(msg.getNeighbor());
293         // Inicializa um tempo que vai executar o metodo fire() ao
                termino desse tempo
294         timerSENDMSGDISCOVERYNEIGHBORLIST = new SCARPTimer(this, TNO.
                TNO_SENDDISCOVERYNEIGHBORLIST);
295         timerSENDMSGDISCOVERYNEIGHBORLIST.tnoStartRelative(1, this, TNO
                .TNO_SENDDISCOVERYNEIGHBORLIST);
296     }
297
298     private void tratarMsgBoard(MsgBoard msg) {
299         if (this.ID != 1) {
300             sendMsgBoard(msg.getId());
301         } else {
302             election.add(msg.getId());
303             if (!msgBoardTimed) {
304                 msgBoardTimed = true;
305                 // Inicializa um tempo que vai executar o metodo fire()
                        ao termino desse tempo
306                 timerSENDMSGBOARD = new SCARPTimer(this, TNO.
                        TNO_SENDMSGBOARD);
307                 timerSENDMSGBOARD.tnoStartRelative(2, this, TNO.
                        TNO_SENDMSGBOARD);
308             }
309         }
310     }
311
312     private void tratarMsgBoardElected() {
313         if (IDA == this.ID) {
314             this.setColor(Color.GREEN);
315             this.hopA = 1;
316             sendMsgHopCellA(this.hopA);
317         } else if (IDB == this.ID) {
318             this.setColor(Color.GREEN);
319             this.hopB = 1;
320             sendMsgHopCellB(this.hopB);
321         }
322     }
323
324     private void tratarMsgData(MsgData msg) {
325         if (this.ID != 1) {
326             MsgTimer timer = new MsgTimer(msg, this.nextNode);
327             timer.startRelative(0.0001, this);
```

```
328             bateria.spend(EnergyMode.SEND);
329             dataPackets++;
330         } else {
331             if (!cells.contains(msg.getEventID())) {
332                 cells.add(msg.getEventID());
333                 sendMsgRepresentative(msg.getSender());
334                 Tools.appendToOutput(msg.getSender() + " rep " + msg.
                        getEventID() + "\n");
335             }
336             receivers++;
337         }
338     }
339
340     private void tratarMsgRepresentative() {
341         this.representative = true;
342     }
343
344     private void sendMsgHop(Node node, int hop, int eventID) {
345         MsgTimer timer = new MsgTimer(new MsgHop(this, hop, eventID));
346         timer.startRelative(0.00001, this);
347         bateria.spend(EnergyMode.SEND);
348         overheads++;
349     }
350
351     private void sendMsgDiscoveryNeighbor(int id) {
352         MsgTimer timer = new MsgTimer(new MsgDiscoveryNeighbor(id));
353         timer.startRelative(0.00001, this);
354         bateria.spend(EnergyMode.SEND);
355         overheads++;
356     }
357
358     private void sendMsgDiscoveryNeighborList(List<Integer> neighbor)
            {
359         MsgTimer timer = new MsgTimer(new MsgDiscoveryNeighborList(
                neighbor));
360         timer.startRelative(0.00001, this);
361         bateria.spend(EnergyMode.SEND);
362         overheads++;
363     }
364
365     private void sendMsgBoard(int id) {
366         MsgTimer timer = new MsgTimer(new MsgBoard(id), this.nextNode);
367         timer.startRelative(0.00001, this);
368         bateria.spend(EnergyMode.SEND);
369     }
370
371     private void sendMsgHopCellA(int hopA) {
372         MsgTimer timer = new MsgTimer(new MsgHopCellA(hopA));
373         timer.startRelative(0.00001, this);
374         bateria.spend(EnergyMode.SEND);
375         overheads++;
```

```
376     }
377
378     private void sendMsgHopCellB(int hopB) {
379         MsgTimer timer = new MsgTimer(new MsgHopCellB(hopB));
380         timer.startRelative(0.00001, this);
381         bateria.spend(EnergyMode.SEND);
382         overheads++;
383     }
384
385     private void sendData() {
386         if (Global.currentTime > endEventTimer) {
387             if (this.sonsInTree.size() == 0) {
388                 TreeCalculate();
389             }
390             return;
391         }
392         if ((representative) || (firstSendData)) {
393             MsgTimer timer = new MsgTimer(new MsgData(this, "DADO␣
                    COLETADO␣PELO␣SENSOR"), this.nextNode);
394             timer.startRelative(0.0001, this);
395             bateria.spend(EnergyMode.SEND);
396             dataPackets++;
397             notifications++;
398             firstSendData = false;
399         }
400         timerMONITORING = new SCARPTimer(this, TNO.TNO_MONITORING);
401         timerMONITORING.tnoStartRelative(dataRate, this, TNO.
                TNO_MONITORING);
402     }
403
404     private void sendMsgRepresentative(Node n) {
405         if (this.ID == 1) {
406             DirectMessageTimer timer = new DirectMessageTimer(new
                    MsgRepresentative(), n);
407             timer.startRelative(0.00001, this);
408         }
409     }
410
411     private boolean isBoard() {
412         boolean isBoard = false;
413         for (Integer i : neighbor) {
414             for (List<Integer> a : neighbors) {
415                 if(a.contains(i)) {
416                     isBoard = true;
417                 } else {
418                     isBoard = false;
419                     break;
420                 }
421             }
422         }
423         return isBoard;
```

```
424     }
425
426     private Integer min(List<Integer> lista) {
427        Integer min = lista.get(0);
428        for (Integer i : lista) {
429           min = ((i < min) ? i : min);
430        }
431        return min;
432     }
433
434     private Integer max(List<Integer> lista) {
435        Integer max = lista.get(0);
436        for (Integer i : lista) {
437           max = ((i > max) ? i : max);
438        }
439        return max;
440     }
441
442     public int maiorvalor() {
443        Tools.appendToOutput(ID + "␣maiorvalor" + "\n");
444          int maior = eventIDCost[1];
445          for ( int cont=1; cont<eventIDCost.length; cont++)
446             if (eventIDCost[cont]>maior)
447                maior = eventIDCost[cont];
448             return maior;
449     }
450
451     public void TreeCalculate(){
452        Tools.appendToOutput(ID + "␣TreeCalculate" + "\n");
453        eventIDTree[this.getEventID()] = true;
454        eventIDCost[this.getEventID()] = 1;
455     }
456
457     public SCARPNode() {
458        logNode.add(this);
459     }
460
461     @Override
462     public void preStep() {}
463
464     @Override
465     public void neighborhoodChange() {}
466
467     @Override
468     public void postStep() {}
469
470     @Override
471     public void checkRequirements() throws WrongConfigurationException
           {}
472
473     public void setEventID(int eventID) {
```

```
474         this.eventID = eventID;
475     }
476
477     public int getEventID() {
478         return eventID;
479     }
480 }
```

Anexo B

CÓDIGO DAS MENSAGENS NO SINALGO

Listagem B.1: MsgBoard.java

```
1  package projects.SCARP.nodes.messages;
2
3  import sinalgo.nodes.messages.Message;
4
5  public class MsgBoard extends Message {
6
7      private final int id;
8
9      public MsgBoard(int id) {
10         this.id = id;
11     }
12
13     public int getId() {
14         return id;
15     }
16
17     @Override
18     public Message clone() {
19         // TODO Auto-generated method stub
20         return this;
21     }
22
23 }
```

Listagem B.2: MsgBoardElected.java

```
1  package projects.SCARP.nodes.messages;
2
3  import sinalgo.nodes.messages.Message;
4
5  public class MsgBoardElected extends Message {
6
```

```
7       private final int idA;
8       private final int idB;
9
10      public MsgBoardElected(int idA, int idB) {
11          this.idA = idA;
12          this.idB = idB;
13      }
14
15      public int getIdA() {
16          return idA;
17      }
18
19      public int getIdB() {
20          return idB;
21      }
22
23      @Override
24      public Message clone() {
25          // TODO Auto-generated method stub
26          return this;
27      }
28
29  }
```

Listagem B.3: MsgData.java

```
1   package projects.SCARP.nodes.messages;
2
3   import projects.SCARP.nodes.nodeImplementations.SCARPNode;
4   import sinalgo.nodes.messages.Message;
5
6   public final class MsgData extends Message {
7       private SCARPNode s;
8       private final String payload;
9
10      public MsgData(SCARPNode s, String payload) {
11          this.s = s;
12          this.payload = payload;
13      }
14
15      public SCARPNode getSender() {
16          return s;
17      }
18
19      public int getEventID() {
20          return s.getEventID();
21      }
22
23      public String getPayload() {
24          return payload;
25      }
```

```
26
27      @Override
28      public Message clone() {
29          // TODO Auto-generated method stub
30          return this;
31      }
32
33  }
```

Listagem B.4: MsgDiscoveryNeighbor.java

```
1   package projects.SCARP.nodes.messages;
2
3   import sinalgo.nodes.messages.Message;
4
5   public final class MsgDiscoveryNeighbor extends Message {
6       private final int id;
7
8       public MsgDiscoveryNeighbor(int id) {
9           this.id = id;
10      }
11
12      public int getId() {
13          return id;
14      }
15
16      @Override
17      public Message clone() {
18          // TODO Auto-generated method stub
19          return this;
20      }
21
22  }
```

Listagem B.5: MsgDiscoveryNeighborList.java

```
1   package projects.SCARP.nodes.messages;
2
3   import java.util.ArrayList;
4   import java.util.List;
5
6   import sinalgo.nodes.messages.Message;
7
8   public final class MsgDiscoveryNeighborList extends Message {
9       private List<Integer> neighbor = new ArrayList<Integer>();
10
11      public MsgDiscoveryNeighborList(List<Integer> neighbor) {
12          this.neighbor = neighbor;
13      }
14
15      public List<Integer> getNeighbor() {
```

```
16          return this.neighbor;
17      }
18
19      @Override
20      public Message clone() {
21          // TODO Auto-generated method stub
22          return this;
23      }
24
25  }
```

Listagem B.6: MsgHop.java

```
1   package projects.SCARP.nodes.messages;
2
3   import sinalgo.nodes.Node;
4   import sinalgo.nodes.messages.Message;
5
6   public final class MsgHop extends Message {
7
8       private final Node node;
9       private final int hop;
10      private final int eventID;
11
12      public MsgHop(Node node, int hop, int eventID) {
13          this.node = node;
14          this.hop = hop;
15          this.eventID = eventID;
16      }
17
18      public int getHop() {
19          return hop;
20      }
21
22      public Node getNode() {
23          return node;
24      }
25
26      public int getEventID() {
27          return eventID;
28      }
29
30      @Override
31      public Message clone() {
32          // TODO Auto-generated method stub
33          return this;
34      }
35
36  }
```

Listagem B.7: MsgHopCellA.java

```
1  package projects.SCARP.nodes.messages;
2
3  import sinalgo.nodes.messages.Message;
4
5  public class MsgHopCellA extends Message {
6
7      private final int hop;
8
9      public MsgHopCellA(int hop) {
10         this.hop = hop;
11     }
12
13     public int getHop() {
14         return hop;
15     }
16
17     @Override
18     public Message clone() {
19         // TODO Auto-generated method stub
20         return this;
21     }
22
23 }
```

Listagem B.8: MsgHOpCellB.java

```
1  package projects.SCARP.nodes.messages;
2
3  import sinalgo.nodes.messages.Message;
4
5  public class MsgHopCellB extends Message {
6
7      private final int hop;
8
9      public MsgHopCellB(int hop) {
10         this.hop = hop;
11     }
12
13     public int getHop() {
14         return hop;
15     }
16
17     @Override
18     public Message clone() {
19         // TODO Auto-generated method stub
20         return this;
21     }
22
23 }
```

Listagem B.9: MsgRepresentative.java

```
1  package projects.SCARP.nodes.messages;
2
3  import sinalgo.nodes.messages.Message;
4
5  public class MsgRepresentative extends Message {
6
7      public MsgRepresentative () {
8
9      }
10
11     @Override
12     public Message clone() {
13         // TODO Auto-generated method stub
14         return this;
15     }
16
17 }
```

Anexo C

CÓDIGO DOS TIMERS NO SINALGO

Listagem C.1: MsgTimer.java

```
1  package projects.SCARP.nodes.timers;
2
3  import sinalgo.nodes.Node;
4  import sinalgo.nodes.messages.Message;
5  import sinalgo.nodes.timers.Timer;
6
7  public class MsgTimer extends Timer {
8      private Message msg;
9      private boolean isBroadcat;
10     private Node destino;
11
12     public MsgTimer(Message msg) {
13         this.msg = msg;
14         this.isBroadcat = true;
15     }
16
17     public MsgTimer(Message msg, Node destino) {
18         this.msg = msg;
19         this.destino = destino;
20         this.isBroadcat = false;
21     }
22
23     @Override
24     public void fire() {
25         if (isBroadcat) {
26             node.broadcast(msg);
27         } else {
28             node.send(msg, destino);
29         }
30     }
31 }
```

Listagem C.2: SCARPTimer.java

```
1  package projects.SCARP.nodes.timers;
2
3  import projects.SCARP.nodes.nodeImplementations.SCARPNode;
4  import sinalgo.nodes.Node;
5  import sinalgo.nodes.timers.Timer;
6
7  public class SCARPTimer extends Timer{
8
9      private SCARPNode.TNO tno;
10     SCARPNode n;
11
12     public SCARPTimer(SCARPNode n, SCARPNode.TNO tno){
13         this.tno = tno;
14         this.n = n;
15     }
16
17     @Override
18     public void fire() {
19         n.timeout(tno);
20     }
21
22     public void tnoStartRelative(double time, Node n, SCARPNode.TNO
           tno){
23         this.tno = tno;
24         super.startRelative(time, n);
25     }
26     public void tnoStartGlobalRelative(double time, SCARPNode.TNO tno)
           {
27         this.tno = tno;
28         super.startGlobalTimer(time);
29     }
30     public void tnoStartAbsolute(double time, Node n, SCARPNode.TNO
           tno){
31         this.tno = tno;
32         super.startAbsolute(time, n);
33     }
34
35 }
```

Listagem C.3: EventTimer.java

```
1  package projects.SCARP.nodes.timers;
2
3  import projects.SCARP.nodes.nodeImplementations.SCARPNode;
4  import sinalgo.nodes.Node;
5  import sinalgo.nodes.timers.Timer;
6
7  public class EventTimer extends Timer {
8
9      int eventID = -1;
```

```
10
11      public EventTimer(int eventID){
12          this.eventID = eventID;
13      }
14
15      @Override
16      public void fire() {
17          // TODO Auto-generated method stub
18          if(((SCARPNode)getTargetNode()).insideEvent(getTargetNode().
                getPosition(),this.eventID))
19              ((SCARPNode)getTargetNode()).startDetection();
20      }
21      public void startEventAbsolute(double absoluteTime, Node n, int
            eventID){
22          this.eventID = eventID;
23          startAbsolute(absoluteTime, n);
24      }
25
26  }
```