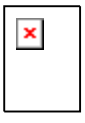
Pós-Graduação em Ciência da Computação

IDEA: Um Protocolo para o Isolamento e Desvio de Anomalias em Redes de Sensores

Por

Hermano Pontual Brandão

Dissertação de Mestrado



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife, Agosto de 2009

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Hermano Pontual Brandão

IDEA: Um Protocolo para o Isolamento e Desvio de Anomalias em Redes de Sensores

Orientador: Prof. Dr. Paulo André da Silva Guimarães

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Recife, Agosto de 2009

Agradecimentos

Gostaria de agradecer primeiramente ao professor Paulo Gonçalves por ter me acolhido, pela sua dedicação e incentivos na produção deste trabalho.

À banca de avaliadores, pelas críticas e sugestões que contribuíram na melhoria desse trabalho em vários aspectos.

Aos meus pais, que sempre me deram todo suporte e incentivos necessários para o meu crescimento acadêmico, profissional e pessoal.

Ao meu amigo Jayro, pelas aulas de C++ e posteriores consultorias, que desde o início me fez acreditar que eu conseguiria concluir esse trabalho, que sempre se preocupou e acompanhou a minha evolução, que sempre escutou meus desabafos e que contribuiu decisivamente com idéias relevantes para o desfecho dessa dissertação.

Aos meus amigos do trabalho que sempre foram compreensivos e colaboraram com a realização dos meus experimentos. Particularmente a Iuri, por sempre ter me substituído nos momentos em que eu tive que me ausentar.

Aos meus amigos do grupo do mestrado, que escutaram pacientemente minhas apresentações e forneceram idéias e sugestões para a melhoria desse trabalho. Especificamente a Bruno, que me apresentou Rafael para ajudar a compreender melhor algumas questões matemáticas desse trabalho.

Por fim, e em especial, à minha namorada, Carol. Primeiramente por sempre ser coerente, fiel à verdade e me aconselhar sobre o que era correto, mesmo que isso me incomodasse e eu preferisse escutar o que era mais conveniente. Por me ouvir com atenção e paciência e nunca menosprezar as minhas angústias e preocupações, mesmo que essas tenham sido exageradas em alguns momentos. Por contribuir diretamente na redação e nas idéias dessa dissertação. E por certamente ter sido a pessoa que mais me deu ânimo, incentivos e forças para que nos momentos finais e mais cansativos eu conseguisse seguir em frente e concluir esse trabalho.

Resumo

O número de aplicações que utilizam redes de sensores sem fio cresceu rapidamente. Entretanto, essas redes estão sujeitas a falhas e ataques que podem atrapalhar ou impedir a propagação de informações importantes. Em muitos cenários, é inviável simplesmente remover os nós responsáveis por essas anomalias. A alternativa encontrada para solucionar esse problema foi desenvolver um protocolo capaz de isolar a região de anomalia e construir desvios em torno da mesma: o IDEA. Para construir esses desvios, o IDEA define múltiplas trajetória de referência, as quais são as representações gráficas de equações no plano cartesiano.

A fim de avaliar essa proposta, foi realizada uma implementação do IDEA no Network Simulator (ns-2). As simulações mostraram, dentre outros resultados, que esse protocolo possui uma boa taxa de sucesso na construção dos desvios e que a utilização de múltiplos caminhos prolonga a energia dos nós envolvidos.

Abstract

The number of applications that relies on wireless sensor networks has grown fast. However those networks are vulnerable to fails and attacks that may disturb or even stop completely important information diffusion through it. In most scenarios, removing or disabling the bad nodes causing those anomalies is simply not possible. The alternative we found to solve this problem was the development of a protocol able to isolate such anomaly regions and build detours around them: the IDEA. To build those detours, the IDEA protocol defines multiple reference path lines that are graphical representations of equations in Cartesian plane.

In order to evaluate this proposal of us, we made an implementation of IDEA on ns-2. The simulations we did showed, among other results, that this protocol has a good success rate on building the detours, and that the use of multipath extends the energy of the nodes involved on the detours.

Sumário

Índice de figuras

Lista de acrônimos

DD	Directed Difusion
DoS	Denial of Service
DSR	Dynamic Source Routing
EEDD	Energy-Efficient Data Dissemination
EFR	Eletric-Field-Based Routing
EHDS	Efficient Hole Detour Scheme
EI	Energia Inicial
GEAR	Geographical and Energy Aware Routing
GG	Gabriel Graph
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing
HA	Hole Announcement Message
HAIR	Hole Avoinding In Advance Routing Protocol
HBD	Hole Boundary Detection
IDEA	Isolamento e Desvio de Anomalias
LA	Limite de Afastamento
LEACH	Low Energy Adaptive Clustering Hierarchy
MEMS	Sistemas Micro-Eletro-Mecânicos
PIF	Ponto Ideal Final (ajeitar isso no documento)
PIN	Ponto Inicial
REQD	Mensagem de Requisição de Desvio
RNG	Relative Neighborhood Graph
RSSF	Redes de sensores Sem Fio
TDMA	Time Division Multiple Access

CAPÍTULO

1

Introdução

Nesse capítulo são apresentadas as motivações que levaram ao desenvolvimento desta dissertação. Além disso, serão mostrados os objetivos gerais desse trabalho e, para alcançá-lo, os objetivos específicos necessários. Em seguida, será dada uma visão geral sobre a proposta desenvolvida e, por fim, a estrutura em que os demais capítulos deste documento estão organizados.

1.1 Motivação

Os avanços nos sistemas micro-eletro-mecânicos (MEMS), a miniaturização de componentes, as melhorias no poder de processamento e o surgimento de fontes de energia de maior duração impulsionaram fortemente o desenvolvimento das redes de sensores sem fio (RSSF). Os sensores possuem características e limitações próprias, como baixo poder de processamento e restrições de comunicação e energia [1], que os diferenciam dos tradicionais dispositivos computacionais.

1.1.1 Redes de sensores

As redes de sensores surgiram a partir da necessidade de se monitorar e coletar constantemente informações sobre algum fenômeno ou elemento de uma região e comunicá-lo a um observador. Entretanto, muitas vezes a região monitorada não dispõe de uma infra-estrutura mínima para a instalação de cabeamento ou equipamentos de comunicação tradicionais. Além disso, algumas dessas regiões também podem ser extensas, inacessíveis e inóspitas (como uma floresta, um deserto ou um vulcão). Isso impulsionou o desenvolvimento de sensores cada vez mais baratos para que centenas ou milhares deles pudessem ser simplesmente lançados em alguma região e talvez nunca mais recuperados.

Contudo, a utilização de muitos sensores para o preenchimento de grandes áreas demanda outra característica importante desses dispositivos: o tamanho reduzido. O

transporte e implantação de uma grande quantidade de sensores são bem mais fáceis quando são manipulados dispositivos com poucos centímetros.

Dessa forma, a demanda por dispositivos pequenos, capazes de se comunicarem sem fio, e ainda assim, com um baixo custo de produção obrigaram o desenvolvimento de equipamentos com grandes restrições de processamento, memória, comunicação e energia.

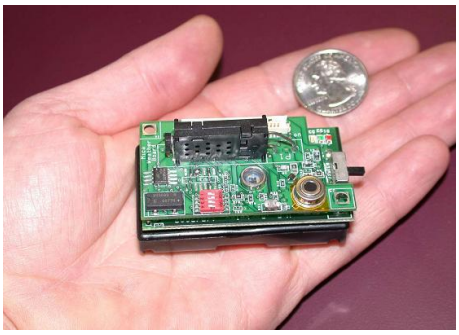


Figura 1 - Um sensor [2]

Inicialmente, quando as redes de sensores começaram a ser pesquisadas, não era possível atender a todos esses requisitos simultaneamente. Assim, as primeiras pesquisas focavam apenas em melhorar a eficiência dos dispositivos de transdução e desenvolver redes cabeadas com protocolos de comunicação específicos [3]. Contudo, a evolução tecnológica proporcionou o desenvolvimento de sensores cada vez menores, mais baratos, com maior poder de processamento, memória e comunicação, sem que isso impactasse significativamente nos custos de produção.

Nas novas gerações de sensores, esses não são mais vistos como apenas um dispositivo que captura as características do ambiente e a transmitem para uma estação base. Nas RSSF modernas, os sensores realizam a transdução de algum fator do ambiente, analisam através de processamento local o dado coletado e se comunicam colaborativamente para produzir um resultado global que reflita as condições do ambiente [4].

As redes de sensores podem ser estáticas ou dinâmicas. Quando os sensores, os observadores e os fenômenos observados não se movem a rede é considerada estática. O escopo desse trabalho se limita a esse tipo de rede.

As RSSF podem ser vistas como um tipo particular de rede ad-hoc onde o protocolo de comunicação deve levar em consideração limitações de hardware ainda mais restritas. As redes ad-hoc preocupam-se principalmente com a comunicação e os seus nós possuem objetivos individuais. Já nas redes de sensores, esses colaboram entre si para executar uma tarefa de interesse comum designada por um observador. Além

disso, as redes de sensores podem possuir milhares de nós sem identificadores, enquanto as redes ad-hoc normalmente contêm menos nós e esses possuem identificadores únicos.

1.1.2 Anomalias em redes de sensores

À medida que os sensores se tornam mais baratos, menores e mais poderosos, o número de aplicações que utilizam redes de sensores cresce rapidamente. Em [1] são apresentadas várias aplicações na área de segurança, monitoração do meio ambiente, casas e habitações, indústrias, controle de tráfego, saúde e comércio.

Porém, quais seriam as conseqüências se alguns dos sensores de uma rede falhassem, por exemplo, no processo de transdução ou de comunicação com os demais sensores? Ou pior ainda, quais seriam as conseqüências se algum sensor malicioso fosse introduzido na rede e adulterasse intencionalmente os dados trafegados através dele?

Em se tratando de redes de sensores, é fácil vislumbrar várias situações em que isso pode ocorrer. Considere, por exemplo, uma rede de sensores para monitorar incêndios em uma floresta densa e de difícil acesso. Nessa, os sensores estariam incumbidos em monitorar a temperatura da região na qual se encontram. Se essa temperatura atingisse um valor pré-definido, o sensor emitiria um alarme, o qual seria propagado através da rede até alcançar uma estação base. Dessa forma, seria possível informar as autoridades sobre a ocorrência e localização de focos de incêndio, enquanto esses ainda fossem facilmente controláveis. Contudo, a exposição dos sensores a intempéries climáticas, como chuva e sol forte, poderia, com o passar do tempo, provocar uma falha no componente responsável pela detecção dos incêndios. Outra possível anomalia poderia ocorrer se um sensor intermediário não encaminhasse o alarme para outros nós em direção à estação base. Em ambos os casos, as autoridades provavelmente só tomariam conhecimento do incêndio quando esse já estivesse fora de controle.

Outra forma de anomalia que pode ocorrer nas redes de sensores é aquela provocada por algum tipo de ataque. A partir da introdução ilegal de nós maliciosos na rede é possível, por exemplo, adulterar o conteúdo dos dados que passem por esses nós ou até mesmo encaminhar apenas uma parte das mensagens recebidas. Em ambos os casos, o nó interessado nas mensagens de dados receberia informações erradas e/ou incompletas, o que poderia provocar tomadas de decisões equivocadas por parte do observador.

Quando situações como essas ocorrem em uma rede de sensores, é possível dizer que há uma anomalia na rede. E os nós responsáveis por provocar essas anomalias são denominados de nós anômalos. De uma forma mais precisa, uma anomalia ocorre em

uma rede de sensores quando um ou mais nós apresentam uma falha ou promovem um ataque direto à rede.

Em alguns casos é necessário identificar não só os nós anômalos, mas também a região afetada por eles. Por exemplo, considere que algum nó malicioso foi inserido na rede apenas para capturar de forma promiscua a comunicação entre os seus vizinhos. Assim, a região afetada pelos nós anômalos é chamada de **região de anomalia**. Essa, por sua vez, pode ser decomposta em duas partes: o núcleo da anomalia e a faixa de isolamento (Figura 2).

- **Núcleo da anomalia**: a menor circunferência a qual compreende todos os sensores anômalos. Nessa região também pode haver nós legítimos.
- **Faixa de isolamento**: a menor coroa circular na qual não há nós anômalos, mas as transmissões dos sensores que estão nessa faixa podem ser capturadas por sensores que estão no núcleo da anomalia. Assim, também é possível definir a faixa de isolamento como a coroa circular cuja largura é igual ao alcance de transmissão dos nós da rede.

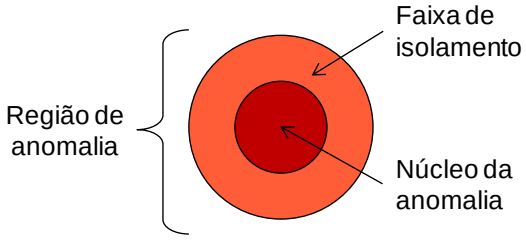


Figura 2 - Detalhamento de uma região de anomalia

Na maioria dos casos em que ocorre uma anomalia, é necessário que os nós relacionados sejam anulados. A forma mais intuitiva de se fazer isso é através da remoção dos nós anômalos do conjunto de nós que compõem a rede. Entretanto, em situações como a da floresta apresentada anteriormente, o acesso aos sensores pode ser caro e/ou difícil. Além disso, a demora em se remover sensores anômalos, principalmente quando a rede está sob ataque, pode provocar grandes prejuízos (como a propagação do incêndio no exemplo da floresta) ou até mesmo comprometer vidas humanas (caso, por exemplo, a rede seja utilizada em ambientes hospitalares).

1.2 Objetivos

A partir do que já foi exposto, é possível perceber que as anomalias em redes de sensores devem ser combatidas. Entretanto, a remoção dos nós maliciosos pode ser inviável.

Dessa forma, esse trabalho visa o desenvolvimento de um protocolo capaz de impedir de forma eficaz o encaminhamento ou captura de mensagens pelos nós anômalos. Em outras palavras, promover o isolamento da região de anomalia.

Entretanto, ao promover esse isolamento, uma parte da rede ficará inacessível e, conseqüentemente, não será permitira a utilização de rotas que passarem por essa região. Por isso, além de isolar a região de anomalia, o protocolo deverá ser capaz de construir desvios em torno dessa região.

Porém, a utilização de um único caminho para desviar as mensagens de dados pode provocar um consumo excessivo dos (já limitados) recursos dos sensores. Conseqüentemente, é necessário também que o protocolo seja capaz de construir múltiplos desvios em torno da região de anomalia a fim de promover um balanceamento de carga.

Assim, o objetivo desse trabalho pode ser sintetizado da seguinte forma: ***desenvolver um protocolo para redes de sensores capaz de isolar as regiões de anomalias e construir múltiplos desvios em torno das mesmas: o IDEA***. Sigla formada pelas iniciais "I" de isolamento, "DE" de desvio e "A" de anomalia.

A fim de alcançar esse objetivo geral, os seguintes objetivos específicos são estabelecidos:

1. Identificar se já existem técnicas para se detectar regiões de anomalia e se essas técnicas podem ser assumidas nesse trabalho;
2. Desenvolver um algoritmo para o isolamento de regiões de anomalia;
3. Avaliar os pontos fortes e os pontos fracos de técnicas já existentes para a construção de desvios em redes de sensores;
4. Considerar esses pontos fortes e fracos no desenvolvimento de um protocolo simples e eficiente capaz de construir desvios em torno de regiões de anomalias;
5. Realizar simulações a fim de avaliar a eficiência, o desempenho e o impacto da utilização do IDEA nas redes de sensores;

1.3 O IDEA

O IDEA é um protocolo para isolamento e múltiplos desvios de anomalias em redes de sensores, o qual funciona integrado com algum protocolo de roteamento tradicional para esse tipo de rede. A parte do isolamento é a mais simples: uma vez detectada a anomalia, quase todas as mensagens para a região de anomalia, ou vindas dessa região, são descartadas. Já a parte da construção do desvio é o ponto forte do desse trabalho. Primeiramente será assumido que através de algum sistema de localização (como um dos apresentados em [5]) cada nó conhece suas coordenadas e a

dos seus vizinhos diretos. A partir de então, o IDEA constrói múltiplos caminhos com base em uma curva continua determinada por uma equação matemática. Esses caminhos serão utilizados para desviar da região de anomalia as mensagens trocadas entre a origem e o destino.

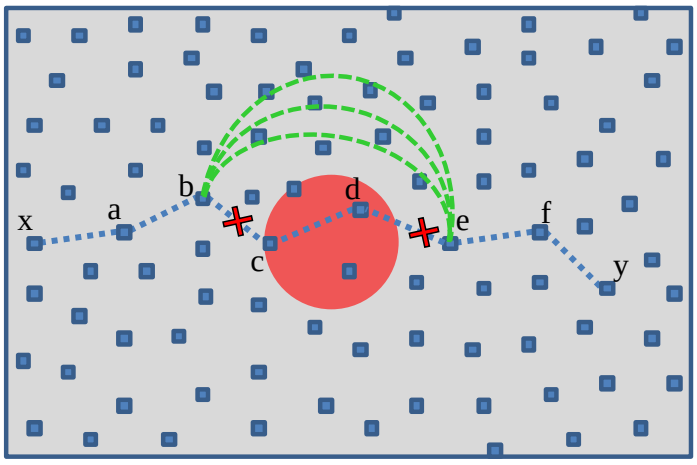


Figura 3 - Trajetórias de referência determinadas após a detecção de uma região de anomalia

Isso é ilustrado através da Figura 3. Nessa um nó *x* (também chamado de *sink*) receberá informações sobre o nó *y* (conhecido como fonte). Através de um protocolo de roteamento, a rota que passa pelos nós *a*, *b*, *c*, *d*, *e* e *f* é estabelecida e os dados da fonte podem ser enviados para o *sink*. Contudo, em um dado momento, a região de anomalia (representada pelo círculo vermelho) é detectada. Caberá então ao IDEA duas tarefas: a primeira é impedir a troca de mensagens entre os nós externos e internos a essa região, e a segunda é construir caminhos alternativos tomando como referência as trajetórias destacadas em verde. A essas trajetórias é dado o nome de **trajetórias de referência**. A Figura 4 mostra em amarelo um dos possíveis desvios criados a partir de uma das trajetórias de referência.

Um mecanismo de encaminhamento de mensagens seguindo uma trajetória base já foi anteriormente apresentado em [6] e, nesse trabalho, os autores revelam os benefícios desse tipo de abordagem. O IDEA aproveita alguns desses benefícios ao adotar esse mesmo conceito para encaminhar as mensagens através de desvios.

O IDEA é um protocolo que integra paradigmas do roteamento geográfico com os de multicaminhos. Isso permite uma melhor distribuição das mensagens de dados na parte da rede por onde são realizados os desvios e, conseqüentemente, prolonga a energia dos nós envolvidos. Além disso, todas as interações ocorrem de forma localizada, o que garante a escalabilidade do protocolo. Outro benefício do IDEA é a sua flexibilidade. O IDEA pode ser facilmente configurado e parametrizado para se

comportar de diferentes formas e utilizar diferentes tipos de equações para as trajetórias de referência.

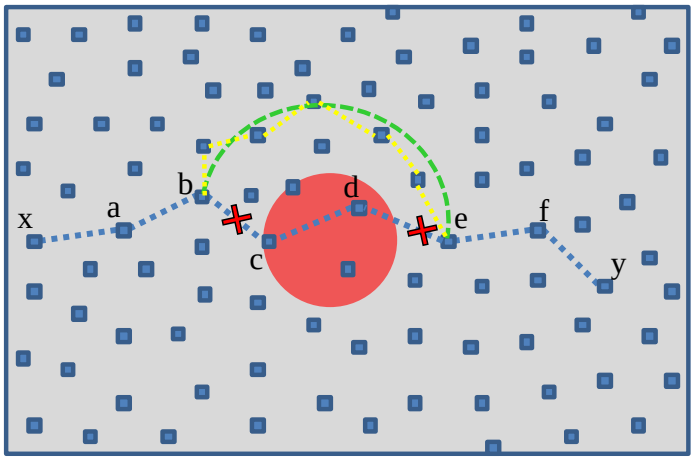


Figura 4 - Desvio construído a partir de uma das trajetórias de referência

A fim de se avaliar a potencialidade do IDEA, foi realizada a implementação do mesmo no ns-2 integrado com o Directed Diffusion [7] e esse com o GEAR (*Geographical and Energy Aware Routing*) [8]. Nessa implementação foi adotada a equação que descreve as linhas de força de um campo elétrico para determinar as trajetórias de referência para os desvios. Essa decisão foi motivada inicialmente pelos resultados de alguns trabalhos que se inspiraram na eletrostática para realizar o roteamento em redes de sensores e por algumas propriedades das linhas de força (as quais serão apresentadas na seção **Erro! Fonte de referência não encontrada.**). orém, fora a isso, foi desenvolvido um estudo comparativo dos gráficos de algumas funções polinomiais com os das linhas de força. Esse estudo forneceu indícios de que as linhas de força possuem uma distribuição espacial mais adequada para o IDEA do que as demais curvas avaliadas. Isso foi decisivo para que, nesse trabalho, a implementação do IDEA fosse realizada utilizando as linhas de força eletrostáticas.

As simulações realizadas no ns-2 mostraram que o IDEA possui uma boa taxa de sucesso na construção dos desvios e que a utilização de múltiplos caminhos é capaz de economizar 30% de energia nas transmissões das mensagens. Por fim, as simulações também mostraram que o IDEA consegue contornar a região de anomalia mesmo quando a rede não é densa.

1.4 Organização da dissertação

No Capítulo 2 serão apresentados os trabalhos relacionados ao IDEA. Mais especificamente, serão mostradas algumas técnicas existentes para detecção de

anomalias e analisadas algumas soluções já desenvolvidas para a construção de desvios em redes de sensores.

No Capítulo 3, será apresentado detalhadamente o IDEA. Inicialmente serão apresentados alguns pré-requisitos para o funcionamento do protocolo. Em seguida, será mostrado o algoritmo responsável por isolar a região de anomalia. Na seqüência, serão revelados todos os detalhes sobre o processo de construção dos desvios.

No Capítulo 4 serão apresentadas as simulações e os seus resultados. Nesse capítulo, será apresentado o estudo comparativo entre os gráficos de algumas equações para as trajetórias de referência e porque as linhas de força foram escolhidas para as simulações. A apresentação dos resultados das simulações será realizada através de vários gráficos. Porém, além disso, cada gráfico será analisado textualmente para que o leitor perceba os benefícios do IDEA, bem como os impactos dessa proposta no funcionamento da rede.

No capítulo 8 serão apresentadas as conclusões sobre esse trabalho, as contribuições que o mesmo proporciona para a área acadêmica e quais trabalhos futuros poderão ser realizados.

No Apêndice A e B será apresentado de uma forma sucinta o Directed Diffusion e o GEAR respectivamente. Através desses dois apêndices será possível conhecer os conceitos básicos sobre esses dois protocolos, o que facilitará a compreensão do apêndice seguinte.

No Apêndice C será possível consultar a implementação detalhada do IDEA no ns-2 e como ele se integra com o Directed Diffusion e o GEAR.

Por fim, no apêndice D serão apresentados os diagramas de classes do ns-2 relacionados com a implementação do IDEA no ns-2.

CAPÍTULO

2

Trabalhos relacionados

Nesse capítulo serão apresentados os trabalhos já desenvolvidos para a detecção de anomalias em RSSF. Essa apresentação será feita em duas partes: uma primeira para a detecção de falhas e uma segunda voltada para detecção de ataques. Será mostrado também que a necessidade de se construir desvios em RSSF não é uma novidade. Assim, serão citadas propostas de desvios desenvolvidas por outros autores e quais os seus pontos fracos.

2.1 Detecção de anomalias em redes de sensores

Os sensores são dispositivos que possuem restrições em vários aspectos. Porém, eles possuem propriedades específicas, como ausência de mobilidade e padrões de tráfego, as quais permitem a detecção de anomalias no comportamento da rede.

A detecção de anomalias visa identificar significantes desvios no funcionamento normal de um sistema [9]. Entretanto, em uma rede de sensores, ela pode se manifestar de duas formas distintas: através de uma falha provocada pro problemas de hardware ou software, ou através de uma intrusão, um ataque intencional e direcionado contra a rede.

2.1.1 Detecção de falhas

A análise e modelagem de falhas é uma área de pesquisa já bastante avançada e seus conceitos podem ser também aplicados às falhas em RSSF [10]. Os nós das RSSF podem falhar por vários motivos: esgotamento da fonte de energia, destruição acidental ou até mesmo devido a ataques diretos. Em geral, a **falha** é o estado incorreto do hardware ou de um programa como conseqüência da **falta** de um componente [11]. Um sistema falha quando a execução de um serviço desvia do serviço oferecido. Um **erro** é a parte do sistema a qual está sujeita a desencadear uma falha. A causa de um erro é uma **falta** (um erro de um programador, um curto-circuito, perturbações eletromagnéticas, etc.). A ocorrência de uma falta cria um **erro latente**, o qual se torna

efetivo quando é ativado. Se o erro afeta a execução do serviço, ocorre então uma falha [12].

Ruiz et. al. afirmam em [13] que a detecção de falhas em redes de sensores deve ser considerada vital. Nós com mau funcionamento podem representar um grande problema de desempenho. Simulações mostram que se de 10% a 40% dos nós de uma rede apresentarem algum tipo de falha, a vazão média sofrerá uma perda de 16% a 32% [14]. Fluxos de mensagens que forem encaminhados através de um nó defeituoso experimentarão uma taxa de perda muito maior do que a média da rede. Além disso, as retransmissões necessárias para compensar as perdas de mensagens provocam um grande aumento do consumo de energia e uma conseqüente redução no tempo de vida útil da rede.

Vários tipos de faltas podem ocorrer em uma rede de sensores. Os principais tipos encontrados na literatura são: faltas por quebra total de um nó [15,10,16,17,13], faltas no roteamento de mensagens [14,18], faltas por esgotamento de energia [19,20,21,22,23], faltas na comunicação [10,24,25], faltas na consistência de dados [26] e faltas na transdução [11,27,24,28,29].

Esses tipos de faltas foram extraídos de trabalhos que possuíam como principal objetivo desenvolver um mecanismo para a detecção de um desses tipos de falhas. Uma vez que cada tipo de falha possui uma particularidade, seria muito difícil desenvolver um mecanismo genérico e eficiente.

Dentre os principais mecanismos de detecção de falhas presentes na literatura, alguns dos que mais se relacionam com este plano de trabalho foram apresentados em [14,25,26].

Marti et al. em [14] descrevem duas técnicas que, em conjunto, são capazes de detectar sensores que deveriam realizar o encaminhamento de mensagens, mas falham no momento da execução. O princípio geral do funcionamento da solução proposta está ilustrado na Figura 5. A linha tracejada indica que A está dentro do raio de transmissão de B e a outra linha indica a direção da transmissão da mensagem de B para C. Dessa forma, se um nó S envia uma mensagem para D, passando pelos nós A, B e C, quando a mensagem é transmitida de B para C, A pode "escutar" essa transmissão e verificar que B tentou transmitir para C. Entretanto, se A não escutar a transmissão de B para C, A considera que ouve uma falha no encaminhamento da mensagem.

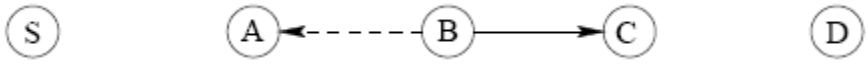


Figura 5 - Abordagem utilizada por Marti et al. para detecção de falhas.

Em [28]. Krishnamachari et al. propõem o seguinte esquema: considere que um sensor "i" é representado pela variável binária T_i. Essa variável pode então assumir o valor $T_i = 0$, se o sensor estiver verdadeiramente em uma região considerada normal, e $T_i = 1$ se o nó está verdadeiramente em uma região de evento. Em seguida, a transdução do sensor é armazenada em uma variável abstrata binária S_i. Essa variável assume o valor $S_i = 0$ se a transdução indicar um valor normal e $S_i = 1$ caso indique a transdução indique um valor atípico (que pode ser um evento).

Dessa forma há quatro possíveis cenários:

$S_i = 0$, $T_i = 0$: o sensor reporta corretamente uma leitura normal

$S_i = 0$, $T_i = 1$: o sensor reporta erradamente uma leitura normal

$S_i = 1$, $T_i = 1$: o sensor reporta corretamente uma leitura atípica

$S_i = 1$, $T_i = 0$: o sensor reporta erradamente uma leitura atípica

Enquanto cada nó estiver ciente do valor S_i, há uma grande probabilidade de uma leitura errada ter ocorrido de $S_i \neq T_i$. Dessa forma, um algoritmo Bayesiano para o reconhecimento de faltas foi desenvolvido com a capacidade de determinar uma estimativa R_i que avalia o grau de verdade da leitura feita pelo nó T_i, a partir da obtenção de informações de leituras dos vizinhos. Esse algoritmo possui três formas distintas para a decisão de reconhecimento de faltas: uma randômica, uma por comparação com um valor limite e outra também por comparação com um valor limite, mas de forma otimizada. A forma otimizada consiste basicamente nos três passos seguintes:

São obtidas as leituras S_j de todos os sensores os vizinhos N_i ao nó "i"

É determinado K_i, o número de nós j que são vizinhos de i com $S_j = S_i$

Se $K_i \geq 0.5\ N_i$, então é feito $R_i = S_i$, (indicando que o valor S_i é válido). Se não, será feito $R_i = \neg S_i$ (indicando que o valor S_i não é válido)

Já o algoritmo de detecção de falhas apresentado em [26] quando um evento é observado por um sensor, esse envia duas cópias da mensagem com os dados observados para um comando central, uma pelo caminho mais curto e outra por um caminho alternativo. Quando essas mensagens chegam ao comando central, esse compara o conteúdo das duas. Se esses forem divergentes, um terceiro caminho, disjunto aos outros dois, é criado e três cópias idênticas à primeira mensagem são enviadas pelo nó que observou o evento, uma por cada caminho. Após o recebimento dessas três mensagens, ou a ocorrência de um *timeout* em um dos três caminhos, o comando central compara o conteúdo das mensagens e defini em qual dos três caminhos há um sensor em falta.

2.1.2 Detecção de intrusão

Mais uma vez, devido às particularidades das redes de sensores, as tradicionais questões sobre segurança em redes e os tradicionais mecanismos de detecção de intrusão não podem ser diretamente aplicados.

Em um recente trabalho [30] vários dos problemas de segurança em redes de sensores foram apresentados e classificados em dois grupos: ataques de roteamento e ataques de negação de serviço (*Denial of Service*, DoS).

Dentre os ataques de roteamento mais comuns podem ser destacados o de *spoof*, o de encaminhamento seletivo e o de inundação por mensagens de HELLO. Dentre os de DoS, Wood e Stankovic em [31,32] classificaram vários dos tipos ataques de DoS baseados na camada em que eles atingem. No nível físico, os ataques de DoS são também conhecidos como *jamming* e *tampering*. *Jamming* é o processo de interferir na freqüência das ondas de rádio que são utilizadas pelos sensores. Enquanto que *tampering* tem haver com alterações físicas (roubo, deslocamento, substituição) ou até mesmo danificação dos nós.

Ao longo dos anos, vários trabalhos para detectar e reagir a esses tipos de ataques foram desenvolvidos. Karlof e Wagner em [33] fazem uma análise das ameaças ao roteamento das redes de sensores e propõem métodos para lidar com cada uma delas. Wood and Stankovic [31] estudaram ataques do tipo DoS e vários tipos de defesas contra os mesmos. JAM [34] é um serviço para redes de sensores o qual detecta regiões onde estão ocorrendo ataques de *jamming* e possui mecanismos para possibilitar que a rede continue a rotear. Cagalj et al. [35] apresentaram a utilização de *wormholes* nas redes de sensores como uma forma de defesa reativa para a prevenção de *jamming*. SPINS [36] é um conjunto de protocolos de segurança otimizados para redes de sensores. TinySec [37] é uma arquitetura de segurança que atua no nível de enlace. Newsome, et al. [38] analizam o ataque Sybil nas redes de sensores e propõem vários mecanismos de defesa. Karakehayov [39] apresenta o REWARD, um algoritmo de roteamento ajustável que pode agir contra simples *black holes* ou grupos de nós maliciosos. Já em [40,41,42] esquemas de gerenciamento de chaves e pré-distribuição de chaves são apresentadas. Du et al. [43] criaram o LEAP+ (*Localized Encryption and Authentication Protocol*), um protocolo para gerenciamento de chaves que minimiza o impacto do processamento com a segurança.

2.2 Protocolos de desvios para redes de sensores

A necessidade de se construir desvios em redes de sensores não é uma novidade. Entretanto, em trabalhos anteriores a este e que já desenvolveram alguma técnica de

desvio [44,45,46,47,48,49,50,51,52] os autores não utilizam o conceito de anomalia. Ao contrário disso, o objetivo visado é a construção de desvios ao redor de buracos nas redes de sensores.

Em se tratando de roteamento, os buracos representam regiões da rede na qual os sensores estão dispostos de tal forma que ocorre uma impossibilidade de se prosseguir com o encaminhamento geográfico tradicional em direção ao destino. Entretanto, outros tipos de buracos podem ocorrer em redes de sensores, impossibilitando o seu correto funcionamento. Ahmed et al. em [7] definem esses tipos buracos e realizam uma pesquisa sobre várias soluções para se identificar e desviar dos mesmos.

Ainda em [7], Ahmed et al. afirmam que quatro tipos de buracos podem ocorrer nas redes de sensores:

- **Buracos de cobertura (*coverage holes*)**: dado um conjunto de sensores e uma região alvo, é possível dizer que não existem buracos de cobertura se todos os pontos da área alvo estão cobertos por pelo menos k sensores, onde k é o nível de cobertura exigido por uma determinada aplicação.
- **Buracos de roteamento (r*outing holes*)**: são regiões na rede de sensores que contêm nós que não estão disponíveis ou que não podem participar do roteamento de dados. Podem acontecer também devido ao fenômeno de mínimo local, onde o envio de mensagens é baseado no destino. Nesse último caso, o nó tenta mandar a mensagem para um nó vizinho mais próximo do destino do que ele próprio, porém tal nó não existe. Então, o nó tem que mandar a mensagem para um nó mais afastado do destino do que ele, para que esse então possa continuar o envio até o destino.
- **Buracos de tráfego (*jamming holes*)**: ocorrem quando os nós não conseguem enviar mensagens devido ao alto tráfego na rede, que pode ser intencional ou não.
- **Buracos negros (*Sink/Black Holes/Worm Holes*)**: são regiões em que nós maliciosos começam a anunciar rotas bastante atrativas para o sink. Desse modo, os vizinhos a esse nó malicioso o escolhem como próximo salto. Assim, ao invés de serem direcionadas para o sink, as mensagens são atraídas para o nó malicioso o qual pode descartá-las, encaminhá-las seletivamente ou modificar o conteúdo da mensagem antes realmente enviá-la para o sink.

Dentre esses tipos de buracos, os que mais se relacionam com o IDEA são os buracos de roteamento. Dessa forma, nas sessões seguintes, serão apresentados alguns dos trabalhos já desenvolvidos e que se preocupam em resolver os problemas causados por esse tipo de buraco.

2.2.1 Greedy Perimeter Stateless Routing

O *Greedy Perimeter Stateless Routing* (GPSR) é proposto em [47]. Este protocolo assume que todos os nós conhecem sua própria posição, e que eles podem determinar a localização do nó de destino do pacote. A descoberta das posições dos seus vizinhos é realizada através de um *beacon*, que envia periodicamente a identificação e a posição de cada nó. Uma das vantagens do GPSR é que nele cada nó só precisa conhecer seus vizinhos imediatos, o que faz com que a quantidade de informações que cada nó armazena seja insignificante em relação ao tamanho da rede.

O GPSR utiliza dois métodos para direcionamento de pacotes: *greedy fowarding*, usado sempre que possível, e *perimeter forwarding*, utilizado nos casos em que o *greedy forwarding* não pode ser empregado.

No *greedy forwarding*, como os pacotes saem da origem sabendo a localização da mesma e a do seu destino, é possível que cada nó escolha o próximo salto de maneira ótima. A escolha ótima é definida como a adoção do vizinho geograficamente mais próximo do destino e dentro do alcance do nó. Esse processo é repetido por todos os nós, até que o destino seja alcançado.

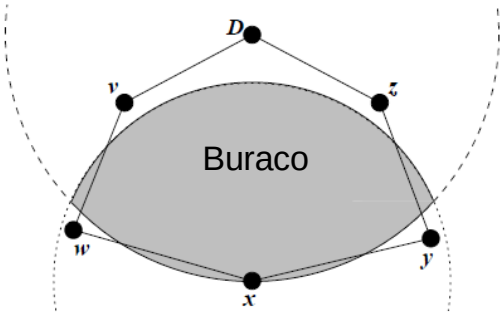


Figura 6 - O nó x é um ponto de máximo local.

Porém, certas redes podem ter topologias que forcem um pacote a ir temporariamente para um nó mais longe do destino. Na Figura 6 é possível observar que na área de cobertura de x não existem vizinhos mais próximos de D do que o próprio x. Quando isso ocorre, é dito que x é um ponto de máximo local em relação a D. E a região entre eles é dita vazia.

Quando isso ocorre, o outro mecanismo de encaminhamento é utilizado para o roteamento: o *perimeter forwarding*. Através desse, x procura em volta da área vazia um caminho para D. Para isso, é utilizada a regra da mão direita. Essa regra diz que para um nó percorrer um grafo, ao receber uma mensagem de um vizinho, ele deverá encaminhar essa mensagem para o primeiro vizinho no sentido anti-horário a partir da localização do remetente da mensagem. Aplicando essa regra para o nó x a região

poderá ser contornada da seguinte forma: $x \blacktriangleright w \blacktriangleright v \blacktriangleright D \blacktriangleright z \blacktriangleright y \blacktriangleright x$. A sequência de arestas percorridas pela regra da mão direita é chamada de perímetro.

Contudo, para que o *perimeter forwarding* possa ser aplicado, é necessária a construção e o armazenamento de um grafo planar (ou seja, um grafo sem arestas que se cruzem) para representar a rede. O *Relative Neighborhood Graph* (RNG) e o *Gabriel Graph* (GG) são dois grafos planares bastante conhecidos e estudados nesse trabalho para a construção do grafo planar. Deve-se levar em consideração que a remoção de arestas de um grafo para que ele seja reduzido ao RNG ou ao GG não pode desconectar esse grafo.

O cabeçalho do pacote GPSR inclui uma flag que indica se o pacote está no *greedy mode* ou no *perimeter mode*. Inicialmente, todos os pacotes estão em *greedy mode*. Quando um nó recebe um pacote nesse modo, ele checa se ele é um ponto de máximo local. Ou seja, se existe um vizinho geograficamente mais perto do destino do que ele. Se existir, ele envia o pacote para este nó. Se não existir, ele seta a *flag* para *perimeter mode* e tenta alcançar o extremo oposto da região vazia através da regra da mão direita.

Assim como o GPSR, outros trabalhos também propuseram técnicas de desvios de buracos que sempre utilizam o perímetro da região contorná-la [45,48,49,50,53]. Entretanto, essa abordagem força as mensagens a desviarem sempre pelos mesmos nós (aqueles que estão no perímetro do buraco). Ao fazer isso, o algoritmo provocará um maior consumo de energia dos nós da rede que estiverem envolvidos no desvio. Conseqüentemente, haverá uma tendência de que a energia dos nós localizados no perímetro do buraco se esgote rapidamente e o buraco aumente cada vez mais de tamanho.

Além disso, algumas dessas técnicas necessitam da construção inicial de um grafo planar para a posterior realização dos desvios. E isso obriga todos os nós da rede a armazenar informações sobre esses grafos. Entretanto, essa abordagem possui a desvantagem de que a maioria dos pacotes alcançam o destino através do *greedy fowarding* apenas. Por isso, manter um grafo planar em cada nó (inclusive naqueles em que não há problemas de encaminhamento), e o tempo todo (inclusive naqueles em que não há requisições de roteamento) é desnecessário.

2.2.2 Hole Avoinding in Advance Routing Protocol

O *Hole Avoinding In Advance Routing Protocol* (HAIR) é apresentado em [46], no qual um pacote evita antecipadamente o encontro com um buraco, em vez de contorná-lo. Esse protocolo foi inspirado no processo de uma fila de pessoas andando em direção ao mesmo destino. Se existir uma barreira no caminho, a primeira pessoa da

fila vai ser a primeira a encontrar a barreira. Essa pessoa então avisa à pessoa imediatamente atrás dela que encontrou uma barreira, de modo que a segunda pessoa da fila possa evitá-la antecipadamente. Sucessivamente, cada pessoa repassa a mensagem para a pessoa imediatamente atrás, o que faz com que as pessoas do fim da fila não precisem se aproximar da barreira, otimizando assim o caminho para contorná-la. Esse processo está ilustrado na figura Figura 7.

A fim de facilitar a compreensão de como esse conceito foi adaptado para as redes de sensores será utilizado o exemplo da figura Figura 8(a). Nessa, considere que cada nó está ao alcance de todos aqueles que o circundam em um primeiro nível. Em um primeiro momento, o nó E percebe que ele é um nó-buraco, pois nenhum dos seus vizinhos está mais próximo do sink do que ele mesmo. Em seguida, E informa a todos os seus vizinhos que ele é um nó-buraco.

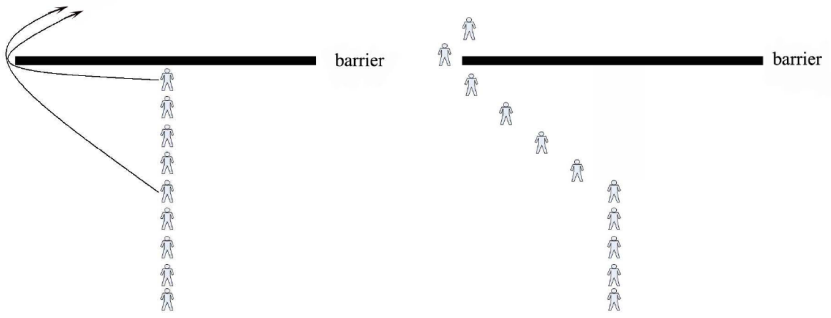


Figura 7 - Rotas com e sem informações sobre a barreira

Logo depois, quando uma mensagem chega ao nó H, o HAIR é executado em duas etapas: na primeira, o nó H verifica na sua lista de vizinhos se algum deles é um nó-buraco. Ao perceber que o nó E se enquadra nessa condição, o nó H escolhe outro nó que esteja mais próximo do sink.

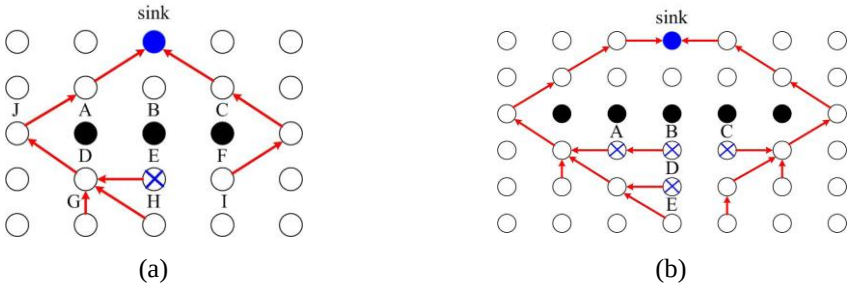


Figura 8 - Um exemplo do protocolo HAIR

Agora considere o cenário mais complexo da figura Figura 8(b). Nessa os nós A, B e C percebem que são nós-buracos por estarem no limite de um buraco. Em seguida essa informação é comunicada ao nó D. Dessa forma, esse último nó também irá se

considerar um nó-buraco, pois todos os seus vizinhos que estão mais próximos do sink são nós-buracos. Por fim, o nó D irá que ele é um nó-buraco ao nó E. Conseqüentemente, o nó é não enviará a mensagem até B para que ela seja desviada por esse nó. Antecipadamente ele irá enviar a mensagem para o nó que está mais próximo do sink e não é um nó-buraco.

A segunda etapa serve para os nós que possam não ter encontrado um caminho para o destino pela primeira etapa. A fim de solucionar esse problema, o nó que já conhece um caminho para o destino notifica seus vizinhos que ainda não o conhecem, para que eles o selecionem como o próximo nó da rota.

Ao contrário do GPSR e dos demais trabalhos que utilizam o perímetro do buraco para contorná-lo, o HAIR tenta antecipar o desvio antes que a mensagem chegue nesse perímetro. Além disso, o HAIR não necessita da construção de um grafo. Porém, uma vez iniciado o desvio, a tendência é que as mensagens sejam sempre encaminhadas pelos mesmos nós. Esse fato recai no problema do consumo excessivo de energia pelos nós participantes do desvio.

2.2.3 Energy-Efficient Data Dissemination Protocol

Em [51], Yu et al. propõem o *Efficient Hole Detour Scheme* (EHDS) para solucionar os problemas causado por buracos em redes de sensores de forma eficiente, fazendo com que os nós do perímetro do buraco não consumam energia em excesso nem fiquem congestionados, como ocorre freqüentemente nos protocolos que utilizam o roteamento por perímetro.

Para tal, um círculo virtual é criado em torno do buraco, e quando um nó no perímetro do buraco recebe um pacote, ele envia para a origem o *sink* uma mensagem contendo informações sobre o círculo virtual (ver Figura 9). Assim, a origem S saberá que existe um buraco no caminho até o destino D. Ela então calcula a posição de um nó âncora V (baseado na sua localização, na localização do destino e no círculo virtual), que recebe o pacote e o encaminha para o destino.

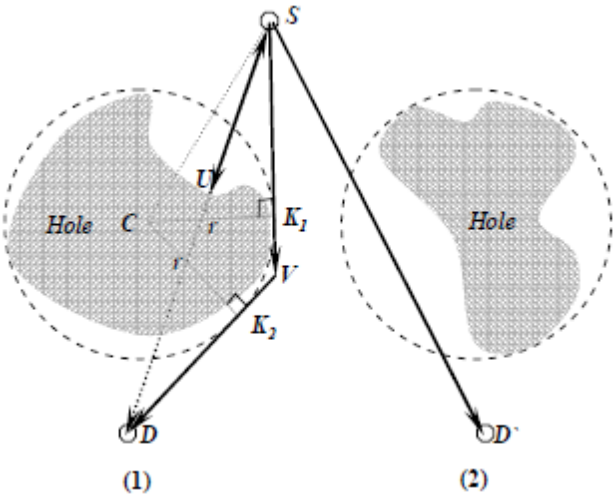


Figura 9 - Definição do nó âncora no EHDS

A Figura 9 também mostra um caso em que a mensagem passa pelo círculo virtual, mas não intercepta o buraco. Nesse caso, nenhuma ação é realizada e o encaminhamento adotado será sempre o geográfico.

Para calcular o círculo virtual, quando um nó B_0 detecta que está no limite de um buraco, ele utiliza a regra da mão direita para mandar ao redor deste um pacote chamado *Hole Boundary Detection* (HBD), que coleta informações sobre todos os nós do limite (B_1, B_2, B_3, etc.). Quando o pacote HBD retorna a B_0, esse calcula dois pontos P e Q dados por:

$$P\{(x_p,y_p)|\ x_p=max\{x_0,x_1\cdots x_n\},\ y_p=max\{y_0,y_1\cdots y_n\}\}$$
$$Q\{(x_q,y_q)|\ x_q=min\{x_0,x_1\cdots x_n\},\ y_q=min\{y_0,y_1\cdots y_n\}\}$$

Sendo C o ponto central do segmento PQ e r a maior distância entre C e todos os nós do limite, B_0 calcula o círculo virtual, tendo C como centro e r como raio. Desse modo, o círculo cobre toda a área do buraco. Novamente utilizando a regra da mão direita, B_0 envia um pacote chamado *Circle Distribution* (CD), com a coordenada do centro e o raio do círculo, para que todos os nós da borda conheçam a posição do círculo. A figura 2 ilustra esses passos.

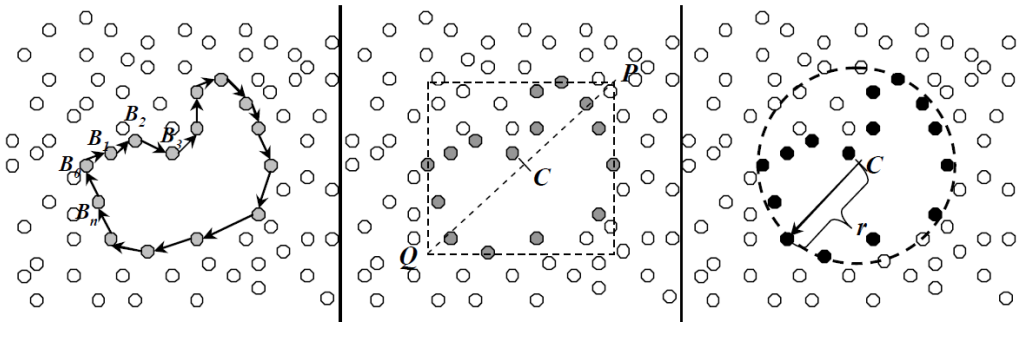


Figura 10 - Determinação do círculo virtual

Para o cálculo da âncora, quando um nó na borda de um buraco recebe um pacote, ele manda um *Hole Announcement Message* (HA) contendo informações sobre o círculo virtual para o nó inicial, informando que existe um buraco. Este nó calcula então a posição da âncora como sendo o ponto de interseção de duas retas tangentes ao círculo virtual, passando pelo sink e pela fonte. Esse processo também pode ser observado através da Figura 9.

O EHDS reduz o congestionamento nos nós da borda de um buraco, uma vez que os pacotes de dados não são mandados através dela. Além disso, ele também reduz a probabilidade do problema do mínimo local ocorrer, já que a âncora fica fora do círculo virtual, enquanto o buraco está completamente contido nele. Entretanto, esse protocolo tem uma grande limitação: a rota de desvio, que passa pela âncora, é estática, ou seja, o consumo de energia nos nós nessa rota é mais rápido do que nos nós em geral.

A fim de solucionar esse problema, em [52] é apresentado o protocolo *Energy-Efficient Data Dissemination* (EEDD). Esse protocolo é uma melhoria do EHDS e foi proposto para balancear o consumo de energia na rede, tornado a rota de desvio dinâmica. Uma das diferenças para o trabalho anterior é que nesse a mensagem não é encaminhada do *sink* para o nó âncora, mas sim de um ponto tangente ao círculo virtual, como os pontos U e U' na Figura 11.

Além disso, o algoritmo desenvolvido nesse trabalho utiliza a função Gaussiana bidimensional para variar dinamicamente a posição do nó âncora. Para isso, de forma similar ao EHDS, é calculado primeiramente o comprimento L e em seguida o posicionamento do ponto V. Porém, o ponto V no EEDD é identificado como o ponto âncora básico. Em seguida, a partir da função Gaussiana bidimensional esse ponto é dinamicamente movido para uma nova localização. Assim, as coordenadas desse novo ponto serão utilizadas como as coordenadas do nó âncora para o qual a mensagem será encaminhada.

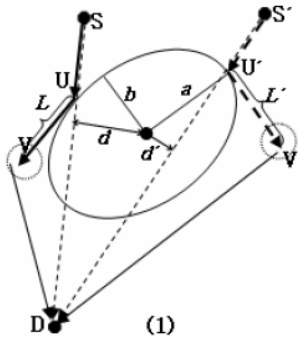


Figura 11 – Determinação do nó âncora pelo EEDD

O EEDD foi publicado durante o desenvolvimento desse trabalho e ele é o que mais se aproxima do IDEA, pois ele se preocupa em economizar a energia dos nós envolvidos nos desvios. Além disso, a abordagem utilizada pelo EEDD é similar à apresentada nesse trabalho, ou seja, construir múltiplos caminhos de desvios em torno de uma região. Entretanto, como poderá ser observado nas sessões seguintes, há grandes diferenças na forma que esses dois protocolos promovem esses desvios.

CAPÍTULO

3

O IDEA

A seguir serão detalhadas cada uma das ações promovidas pelo IDEA quando uma região de anomalia é detectada. Em uma primeira parte será apresentada a solução desenvolvida para impedir que novas mensagens sejam trocadas entre os sensores que estão dentro da região afetada e os que estão fora. Depois serão detalhados todos os passos necessários para a construção dos desvios. Através desse detalhamento será possível responder às seguintes questões: o que leva o IDEA a iniciar a construção de um desvio? Como são definidas as trajetórias de referência? Quais fatores influenciam na escolha do próximo salto? Quando e como a construção de um desvio é finalizada? Entretanto, para que isso tudo funcione, foi necessário assumir que o IDEA possui algumas habilidades. Dessa forma, a primeira seção desse capítulo será dedicada à apresentação de quais são essas habilidades.

3.1 Pré-requisitos

No capítulo 2, foram apresentados alguns dos trabalhos já desenvolvidos na área de redes de sensores. Dessa forma, será assumido que o IDEA possuirá algumas habilidades sem que necessariamente seja apresentado o desenvolvimento e a implementação das mesmas.

O IDEA foi desenvolvido com o intuito de resolver um problema que pode ocorre em muitas redes de sensores: a presença de regiões de anomalias. A abordagem adotada para solucionar esse problema possui algumas premissas, as quais são, em sua maioria, atendidas pelos paradigmas do roteamento geográfico. Dessa forma, o escopo desse trabalho se limita a redes que adotem esse tipo de protocolo de roteamento.

A partir de então, é aceitável considerar que o protocolo de roteamento seja dotado de alguma técnica para determinar a sua própria localização [5]. Além disso, cada sensor também será equipado com um hardware capaz de avaliar a sua quantidade

de energia remanescente. Contudo, nenhuma informação adicional sobre o estado da rede será conhecida.

Adicionalmente a isso, o protocolo de roteamento enviará periodicamente algum tipo de mensagem, como os *beacons* adotados em [8] ou [47], através da qual será possível cada nó conhecer a localização e a energia remanescente dos seus vizinhos diretos. Ou seja, os vizinhos que estão dentro do raio de alcance de transmissão e recepção da interface de rádio que integra o sensor.

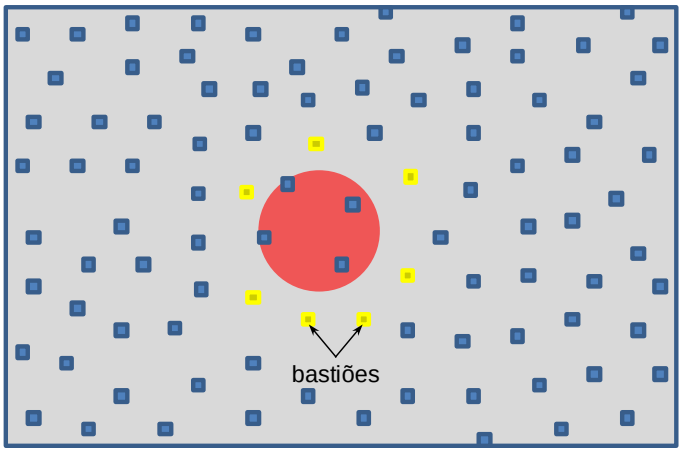


Figura 12 - Bastiões em uma rede de sensores

Será assumido que um nó de origem, quando desejar estabelecer uma comunicação com um nó de destino, enviará uma mensagem do tipo *request* endereçada para alguma coordenada geográfica. Esse tipo de mecanismo pode ser observado em protocolos de roteamento como no GPSR [47] e o Directed Diffusion [7].

Por fim, será assumido que as anomalias serão devidamente detectas por alguma das técnicas apresentadas na seção 2.1. Além disso, os nós responsáveis por essa detecção deverão ser capazes de calcular as dimensões da região de anomalia e associá-la ao menor círculo capaz de circunscrevê-la. Essas informações serão então repassadas aos nós que estiverem a um salto de distância desse círculo. A esses nós, é dado o nome de bastiões (Figura 12). Será mostrado posteriormente que os bastiões exercem um papel importantíssimo no funcionamento do IDEA. Eles são responsáveis tanto por isolar a região de anomalia, como por iniciar a construção dos desvios.

3.2 Visão geral

Para que o IDEA possa alcançar o seu objetivo, são necessários vários procedimentos. O fluxograma apresentado na Figura 13 ilustra uma visão geral dos procedimentos necessários para a construção de um único desvio. A construção dos múltiplos desvios se dá quando esse fluxograma é executado mais de uma vez.

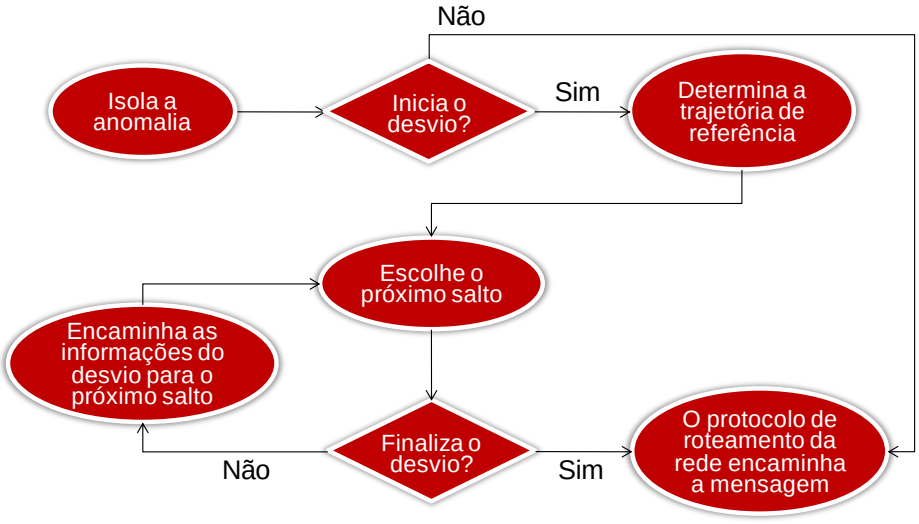


Figura 13 - Fluxograma do funcionamento do IDEA

A primeira ação realizada pelo IDEA quando uma anomalia é detectada é promover o **isolamento da anomalia**. Isso é feito através de um algoritmo de isolamento. Quando uma mensagem chega a um bastião, é feita uma análise sobre o seu tipo, a sua origem e do seu destino. O resultado dessa análise irá determinar se a mensagem deve ser descartada, encaminhada ou se um desvio deve ser iniciado.

Se a mensagem em análise for enviada por um nó a fim de encontrar um caminho para outro nó (ou seja, for uma mensagem de *request*) e ela for em direção da região de anomalia, o bastião irá **iniciar a construção de um desvio**.

O procedimento seguinte realizado pelo bastião é determinar uma trajetória que será usada apenas como referência para a construção do desvio. A essa trajetória é dado o nome de **Trajetória de Referência**. Ou seja, ela é apenas um caminho virtual através do qual a mensagem de *request* deveria ser encaminhada idealmente para se contornar a região de anomalia.

Uma vez determinada a trajetória de referência, o passo seguinte é **escolher o próximo salto**. Esse deverá ser um nó que esteja próximo da trajetória de referência e que forneça um avanço no sentido de contornar a região de anomalia. Isso fará com que o *request* seja propagado por um caminho que se assemelha com o determinado pela trajetória de referência.

Porém, para que os nós escolhidos como próximo salto possam escolher o salto seguinte é necessário **encaminhar as informações do desvio** entre os mesmos. Dessa forma, cada salto intermediário poderá conhecer a trajetória de referência definida pelo bastião e escolher o próximo salto baseado nessa mesma trajetória.

Após eleições sucessivas para a escolha do próximo salto levando em consideração a trajetória de referência, haverá um momento em que o IDEA irá

interromper esse modo de encaminhamento. Isso caracteriza o fim do desvio e, a partir de então, **o protocolo de roteamento da rede encaminha a mensagem** diretamente para o destino.

Nas próximas sessões, cada uma das etapas ilustradas no fluxograma e descritas nos parágrafos anteriores serão mais bem detalhadas.

3.3 Isolando anomalias

Uma anomalia pode ser provocada por um nó malicioso ou um nó defeituoso. O principal objetivo desse trabalho é construir desvios em torno de uma região de anomalia. Entretanto, para que fosse possível garantir a eficiência do algoritmo de desvio, foi necessário desenvolver outros dois algoritmos para impedir a troca de mensagens entre os nós internos e externos à região de anomalia. Um executado na recepção das mensagens e outro executado antes do envio.

Contudo, essa troca de mensagens não pode ser bloqueada por completo. Para que seja possível manter os bastiões atualizados sobre a existência da região de anomalia, é necessário permitir que esses recebam *beacons* vindos da faixa de isolamento. Esses *beacons* devem possuir informações sobre as coordenadas da região de anomalia, o status da mesma (ativa ou inativa) e as coordenadas geográficas do remetente. Manter as informações sobre o status da região de anomalia e a localização dos nós que estão na faixa de isolamento permitirá determinar, posteriormente, se o algoritmo de desvio deve ser iniciado ou não (como será apresentado na seção 3.4.1).

Permitir a troca de mensagens (mesmo que apenas de *beacons*) de um nó que está na faixa de isolamento com um nó externo à região de anomalia, pode, a princípio, parecer inseguro, uma vez que não foi determinando qual tipo de anomalia atuará sobre a rede. Contudo, é bom lembrar que as transmissões dos nós que estão no núcleo da anomalia, não são capazes de alcançar aqueles que estão fora da região de anomalia (ver capítulo 1). Assim, mesmo que um nó malicioso tentasse se passar por um nó o qual estivesse na faixa de isolamento, a sua interface de rádio não teria condições de atingir os sensores situados no exterior da região de anomalia. É importante ressaltar que, se for introduzido na rede um nó malicioso cujo alcance de transmissão seja maior do que o dos demais, caberá ao algoritmo de detecção de anomalias perceber isso e determinar as dimensões do núcleo da anomalia de acordo com o alcance de transmissão desse nó malicioso.

A partir do que já foi exposto nessa seção, será possível apresentar o funcionamento dos dois algoritmos desenvolvidos para isolar as regiões de anomalia. O primeiro atua na recepção das mensagens para identificar e filtrar aquelas vindas da região de anomalia: pré-algoritmo de isolamento. Já o segundo impede que as

mensagens sejam enviadas pra dentro da região de anomalia: pós-algoritmo de isolamento.

O pré-algoritmo pode ser subdividido em três partes, dentre as quais, apenas uma será executada. A definição de qual dessas partes será executada dependerá do posicionamento do nó em relação à região de anomalia. Assim, quando uma mensagem é recebida em um nó, o IDEA procede da seguinte forma:

1. Se o nó estiver no núcleo da anomalia
    a. Nenhum tipo de restrição é aplicado à recepção das mensagens. Essa abordagem é necessária, pois não é possível prever a forma como os nós anômalos irão se comportar (principalmente se esses forem maliciosos)
2. Se o nó estiver na faixa de isolamento
    a. Se o remetente da mensagem recebida também estiver na região de anomalia, essa será descartada
    b. Se a mensagem vier de fora da região de anomalia e for um *beacon* requisitando informações sobre, por exemplo, o posicionamento e a energia restante, essa mensagem é recebida. Entretando, se ela não for um *beacon*, mesmo que de fora da região de anomalia, ela será descartada.
3. Se o nó estiver fora da região de anomalia
    a. Se a mensagem vier de outro nó que está na região de anomalia e for um *beacon* de resposta, essa mensagem é recebida. Qualquer outro tipo de mensagem vinda da região de anomalia será bloqueada.
    b. Todas as mensagens vindas de fora da região de anomalia serão recebidas sem restrições. Pois, nesse caso, não há riscos de que as mesmas sejam encaminhadas ou interceptadas pela anomalia.

O pós-algoritmo de isolamento também pode ser subdividido em três partes. E nesse, da mesma forma que no pré-algoritmo, apenas uma dessas será executada dependendo do posicionamento do nó em relação à região de anomalia:

1. Se o nó estiver no núcleo da anomalia
    a. Nenhum tipo de restrição é aplicado ao envio das mensagens.
2. Se o nó estiver na faixa de isolamento
    a. Se o próximo salto estiver na região de anomalia, a mensagem será bloqueada.
    b. Se o próximo salto estiver fora da região de anomalia e a mensagem for um *beacon* de resposta, ela é enviada.
3. Se o nó estiver fora da região de anomalia

a. Se o próximo salto estiver na região de anomalia e a mensagem for um *beacon* de requisição, ela é enviada. Qualquer outro tipo de mensagem em direção à região de anomalia será bloqueada.
b. Se o próximo salto estiver fora da região de anomalia, mensagens de qualquer tipo são enviadas.

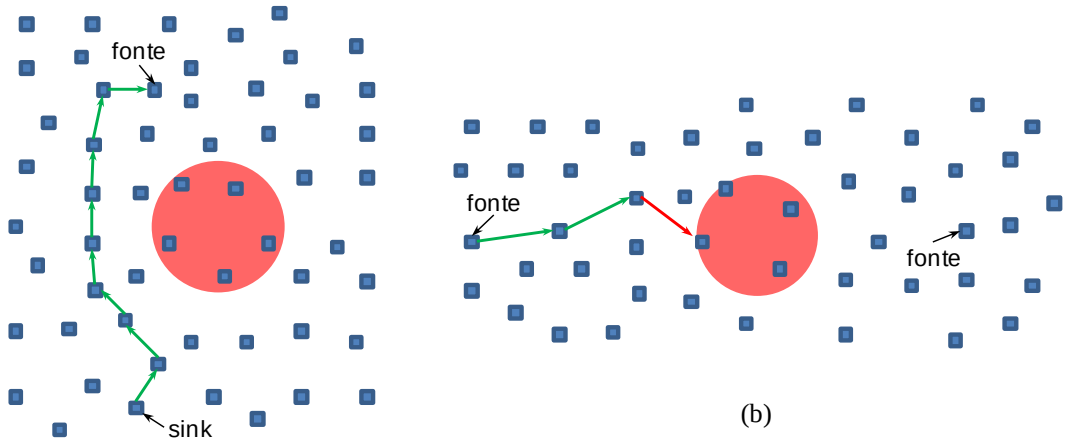
Para que esses algoritmos funcionem adequadamente, a região de anomalia deve ter um tamanho mínimo. No caso mais simples, o núcleo da anomalia é formado por um único sensor. Assim, a faixa de isolamento seria a coroa circular cuja circunferência interna coincide com as dimensões do sensor anômalo e a sua largura é igual ao raio de alcance do mesmo. Mais precisamente, o menor tamanho possível para a região de anomalia é igual à área definida pelo raio de alcance de um único sensor anômalo.

3.4 Construindo desvios

Após a fase de isolamento, o IDEA pode desempenhar o seu principal papel: a construção dos desvios em torno da região de anomalia. Essa funcionalidade pode ser dividida em quatro partes. Na primeira são verificadas as condições para que um desvio seja iniciado. Na segunda, os bastiões determinam as trajetórias que serão usadas como referência para a construção do desvio. Na terceira, cada nó intermediário do desvio avalia a energia restante dos seus vizinhos e a posição dos mesmos em relação à trajetória de referência para eleger o próximo salto. Por fim, na quarta parte, é verificado quando o desvio deve ser finalizado.

3.4.1 Iniciando a construção de um desvio

Uma vez determinados os bastiões, esses nós passam a examinar todas as mensagens de *request* para decidir se o algoritmo de construção do desvio deve ser iniciado. Nesse momento, é importante relembrar que as mensagens de *request* são aquelas utilizadas para se encontrar um caminho através do qual as mensagens de dados serão propagadas em seguida.



(a) (b)

Figura 14 - Situações que determinam o inicio da construção de um desvio

Quando uma mensagem de *request* é recebida em um bastião ela é primeiramente processada pelo protocolo de roteamento da rede. Isso irá eleger o próximo salto para o qual o *request* seria normalmente encaminhado. Após esse processamento, o IDEA verifica se o nó eleito está dentro da região de anomalia. Caso essa hipótese não se confirme, nenhuma modificação será realizada e o *request* será encaminhado normalmente para o nó determinado pelo protocolo de roteamento (Figura 14(a)). Entretanto, se o nó eleito estiver dentro da região de anomalia, o IDEA iniciará a rotina de construção do desvio (Figura 14(b)).

A localização do bastião responsável por iniciar essa rotina passará a representar o ponto inicial (PIN) da trajetória de referência.

Entretanto, ainda é necessário determinar o ponto final. Uma vez que foram adotados os paradigmas do roteamento geográfico, se não houvesse a região de anomalia, quando uma mensagem é recebida em um nó, os próximos saltos deveriam seguir, idealmente, uma reta (r) determinada por esse nó e a fonte (ver a Figura 15).

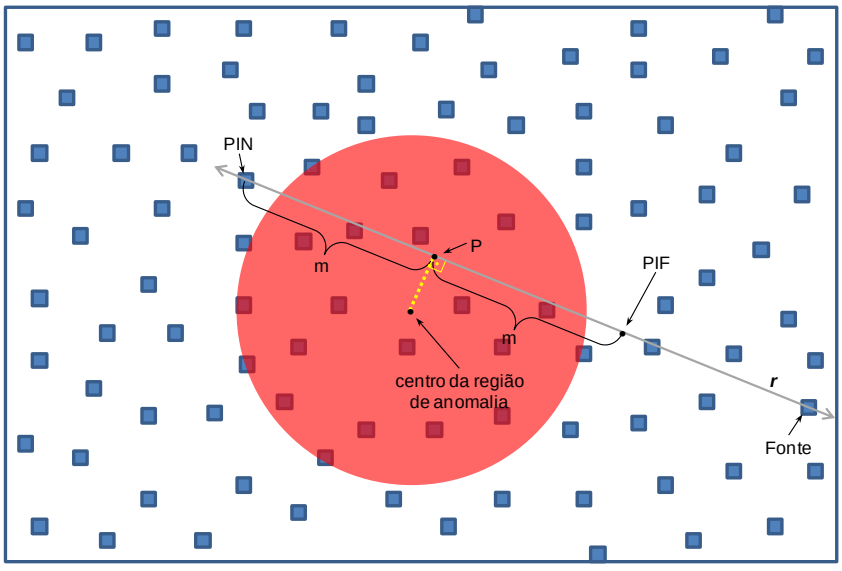


Figura 15 - Determinação do PIN e do PIF

Dessa forma, é válido supor que o ponto ideal para o fim do desvio também esteja sobre essa reta. Contudo, não é possível prever se haverá algum nó posicionado exatamente sobre a reta r. Por isso, esse ponto será chamado de Ponto Ideal Final (PIF).

Uma vez que já foi estabelecido que o PIF deve estar sobre a reta r determinada pelo PIN e pela fonte, agora é necessário definir a localização exata do PIF. Para isso, o bastião executará a seguinte rotina do IDEA:

1. A reta r é determinada a partir das coordenadas do PIN e da fonte.
2. Se a reta r não interceptar a região de anomalia (ver Figura 16), uma nova eleição para se escolher o próximo salto será realizada pelo protocolo de roteamento da rede. Contudo, nessa nova eleição, o nó anteriormente eleito não será considerado um candidato. Esse procedimento se repetirá até que um dos vizinhos que não esteja na região de anomalia seja eleito. Quando isso acontece, a mensagem de *request* é encaminhada para esse nó, nenhuma ação adicional é realizada e essa rotina é interrompida (o desvio não é construído).
    a. Se todos os vizinhos estiverem na região de anomalia, a mensagem de *request* será descartada.
3. Se a reta r interceptar a região de anomalia (ver Figura 15), o próximo passo é encontrar a projeção ortogonal P do centro dessa região sobre a reta r. Em seguida, é calculada a distância m do ponto P para o PIN. Por fim, o PIF será determinado pelo ponto da reta r o qual está mais próximo da fonte e também está a uma distância m para o ponto P. Dessa forma, o ponto P será o ponto médio entre o PIN e o PIF.

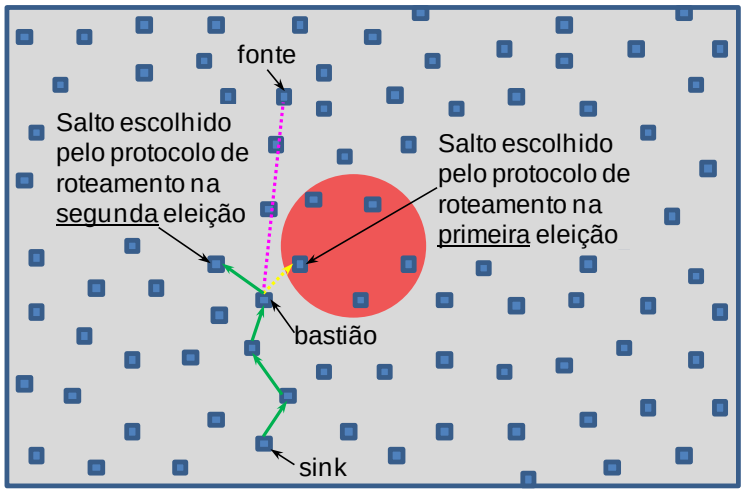


Figura 16 - Caso em que a reta (r) não intercepta a região de anomalia

Agora que o IDEA conhece a posição inicial e final do desvio, será possível determinar as trajetórias que serão utilizadas como referência para encaminhar a mensagem de *request.*

3.4.2 Definição das trajetórias de referência

Agora que a localização do PIN e do PIF foram determinadas, o objetivo passa a ser definir uma trajetória de referência a qual intercepte esses dois pontos e não passe pela região de anomalia. Uma vez que a trajetória será definida pelo bastião que iniciou a rotina de desvio, os demais nós da rede não conhecerão, a princípio, essa trajetória. Dessa forma, à medida que os nós do desvio forem eleitos, a trajetória definida pelo bastião terá que ser informada a cada um deles. Por isso, é necessário que a trajetória de referência possa ser determinada através de poucos elementos.

Muitas equações matemáticas podem ser utilizadas nesse sentido. Para isso, é preciso considerar inicialmente um plano cartesiano C determinado pela reta r, associada ao eixo das abscissas, e centrado no ponto P (assumindo a reta r e o ponto P especificados na seção 3.4.1). Em seguida, é necessário definir um tipo de equação e os seus coeficientes de forma que o gráfico dessa equação contenha o PIN, não intercepte a região de anomalia e também contenha o PIF (ver Figura 17). Nesse trabalho, os gráficos de equações com essas características serão utilizados como trajetórias de referência para construir desvios a partir do PIN e em direção ao PIF.

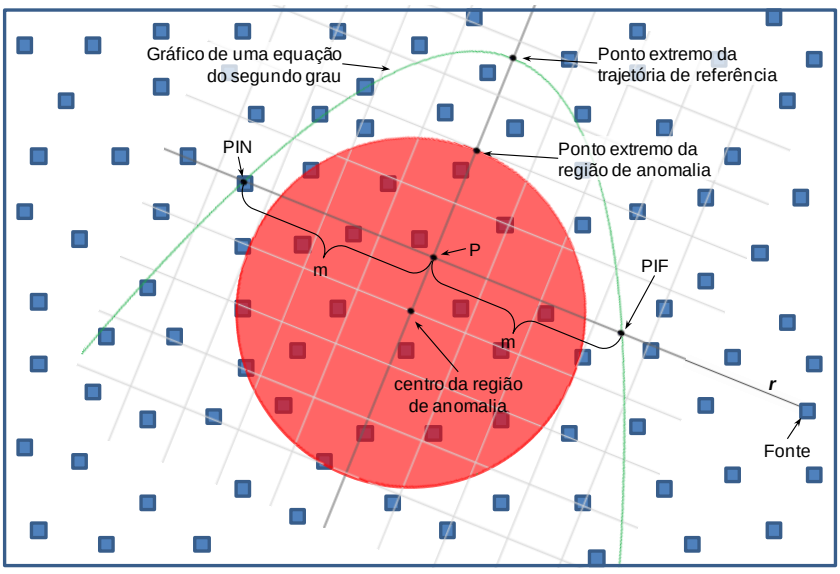


Figura 17 – Definição de uma trajetória de referência

Essa abordagem permite que a trajetória de referência seja facilmente comunicada de um nó para outro durante a construção do desvio. Para isso, é necessário que todos os nós da rede conheçam previamente o tipo de equação que será utilizada. Em seguida, deve ser obtido um conjunto de valores os quais determinem a trajetória de referência. Na maioria dos casos, os coeficientes da equação e as coordenadas do PIN e do PIF são suficientes. A esse conjunto de valores capazes de determinar a trajetória de referência é dado o nome de **determinantes da trajetória**.

Por exemplo, suponha que antes de se iniciar uma rede de sensores, cada nó seja configurado para adotar equações na forma $ax^2 + bx + cy + d = 0$. Em uma primeira etapa, o bastião coincidente com o PIN ficaria responsável por encontrar valores reais para a, b, c e d de forma que a representação gráfica dessa equação passasse pelo PIN, não interceptasse a região de anomalia e passasse pelo PIF. Em seguida, para que o próximo salto do desvio adotasse essa mesma trajetória de referência, seria suficiente repassar para ele apenas os valores de a, b, c e d (juntamente com a posição do PIN e do PIF).

Um requisito importante para o tipo de equação a ser adotada é o de que os seus gráficos possuam um **ponto extremo**. Ou seja, um ponto o qual separa o momento em que a mensagem de *request* deixa de se afastar da reta r e volta a se aproximar novamente. É recomendável que a menor distância entre o ponto extremo e a região de anomalia seja igual ao raio de alcance dos sensores. Essa recomendação visa aumentar as chances de haver nós dos dois lados da trajetória de referência. E, assim, também aumentar a probabilidade de que os sensores eleitos para o desvio estejam mais próximos dessa trajetória.

Outro requisito para as equações é o de que os seus coeficientes possam assumir mais do que um único conjunto de valores. Assim, será possível utilizar mais de uma trajetória de referência e, conseqüentemente, alcançar um dos principais objetivos do IDEA: criar múltiplos caminhos de desvio. No exemplo anterior da equação $ax^2 + bx + cy + d = 0$, se $a \neq 0$ e $c \neq 0$, pode-se variar o valor de d para afastar ou aproximar da reta r o ponto extremo da equação. Dessa forma, para cada nova mensagem de *request* recebida o bastião escolhe um conjunto de coeficientes diferentes para determinar uma trajetória de referência distinta da anterior. Conseqüentemente, os bastiões que iniciarem um desvio precisarão armazenar um histórico de quais trajetórias já foram utilizadas. Esse histórico pode ser mínimo, contendo apenas a última trajetória, ou pode ser tão grande quanto o número de diferentes coeficientes que atenderem às condições necessárias para o desvio.

Contudo, é importante ressaltar que, se forem adotadas trajetórias muito próximas umas das outras, a distribuição das mensagens pelos vários nós da rede provavelmente será mínima. Além disso, é importante observar que, quando um nó realiza uma transmissão, há um consumo de energia por parte de todos os seus vizinhos, e não apenas daquele para o qual a mensagem foi encaminhada. Em RSSF que adotam um padrão de comunicação via rádio, o nível físico é compartilhado. Isso obriga cada nó a escutar as transmissões dos demais para, só posteriormente, verificar se o pacote foi endereçado para ele. Assim, toda vez que um nó percebe o início de uma transmissão, ele sai de um estado no qual há pouco consumo de energia (*idle*) para um estado de recepção, em que o consumo é bem maior. Por isso, proporcionar um espaçamento entre as trajetórias de referências também ajuda na distribuição do consumo de energia entre os sensores. Desse modo, é recomendável que a menor distância entre as trajetórias de referência seja igual ao raio de alcance dos sensores. Pois, a única forma de preservar um sensor em seu estado *idle* é mantê-lo fora do alcance daqueles que estão realizando alguma transmissão.

Intuitivamente, algumas equações são mais adequadas do que outras para se determinar as trajetórias de referência. Por exemplo, equações cujos pontos extremos situam-se sobre o eixo das ordenadas do plano cartesiano C (ver Figura 17). Uma vez que o ponto extremo da região de anomalia sempre estará situado sobre esse mesmo eixo, é natural acreditar que gráficos com essa característica produzam um desvio mais uniforme. Contudo, o estudo sobre quais equações produzem os melhores resultados na construção dos desvios foge do escopo desse trabalho.

3.4.3 Escolha do próximo salto

À medida que os nós da rede são eleitos para participar da construção de um desvio, algumas informações têm que ser repassadas entre eles. A forma como o IDEA faz isso é através da inclusão dessas informações na mensagem de *request*. Quando isso acontece, a mensagem de *request* passa a ser chamada de **Mensagem de Requisição de Desvio (REQD)**. Essa abordagem permite que as mensagens de *request* não percam as suas características originais. Assim, da mesma forma que faz com os *request*, ao receber uma REQD, o nó poderá extrair as informações básicas do protocolo de roteamento para criar nas suas tabelas uma entrada para o salto anterior.

Entretanto, para que um *request* passe a ser considerado uma REQD, as seguintes informações que devem ser adicionadas ao mesmo:

- As coordenadas do PIN
- As coordenadas do PIF
- Os determinantes da trajetória de referência
- As dimensões da região de anomalia (ou seja, a localização do centro e o comprimento do raio)

Adicionalmente a essas informações, uma *flag* de que a mensagem é uma REQD também é adicionada para facilitar a identificação da mesma pelos demais nós da rede.

Quando uma REQD é recebida por um nó, o IDEA identifica essa mensagem e inicia o processamento da mesma. Primeiramente, a mensagem de *request* original é extraída para que o protocolo de roteamento da rede possa processá-la e adicionar (ou atualizar) um registro na sua tabela de encaminhamentos.

Em seguida, é realizada a escolha do próximo salto. A eleição para se encontrar o vizinho mais adequado para fazer parte do desvio leva em consideração três fatores: o ganho angular, a distância para a trajetória de referência e a energia remanescente.

O ganho angular α de um nó N qualquer, representa quantos radianos serão percorridos a partir do PIN, e na direção do PIF, caso o nó N seja escolhido como o próximo salto de um desvio. O ganho angular α é determinado pelo ângulo formado pelo PIN, pelo ponto P (definido na seção 3.4.1) e pela posição de N. Na Figura 18 está ilustrado o ganho angular para dois vizinhos do nó N, V_1 e V_2.

Uma vez que o PIN e o PIF são diametralmente opostos em relação ao ponto P, e que o objetivo do desvio é sair do PIN e se aproximar do PIF, α só poderá assumir algum valor entre zero e π.

Já a distância D de um nó N qualquer para a trajetória de referência é definida como a distância geométrica de N para a curva que representa a trajetória de referência. Ou seja, é a distância de N para a sua projeção ortogonal sobre a trajetória de referência.

Como será mostrado detalhadamente na seção 3.4.4, em alguns casos pode acontecer que a distribuição dos nós no plano provoque o encaminhamento da mensagem para muito longe da trajetória de referência. Dessa forma, foi definido um limite a partir do qual o desvio será interrompido se a REQD continuar a se afastar dessa trajetória. A esse limite foi dado o nome de **Limite de Afastamento (LA)** e foi atribuído ao mesmo o valor de duas vezes o alcance de transmissão de um sensor.

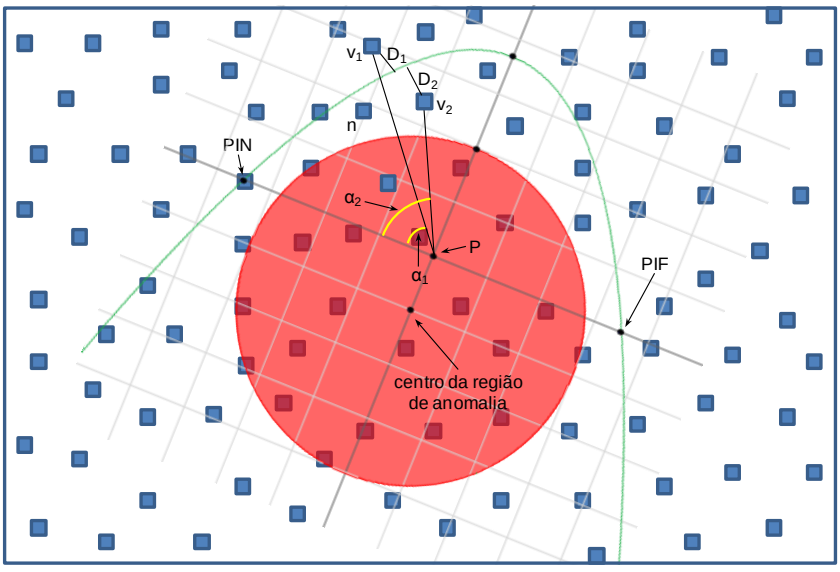


Figura 18 - Determinação do ganho angular (α_1 e α_2) e da distância para a trajetória (D_1 e D_2)

A energia remanescente E, intuitivamente, é a quantidade de energia, medida em Joules, que resta no reservatório de cada sensor. Periodicamente, cada sensor deve consultar os seus vizinhos sobre a quantidade de energia remanescente dos mesmos. As respostas obtidas devem ser armazenadas em *cache* para consultas futuras. Será considerado que, antes de se ativar a rede, todos os sensores são abastecidos e possuem uma mesma quantidade de energia, ou seja, uma **Energia Inicial (EI)**.

Exposto isso, a eleição do próximo salto obedece ao algoritmo ilustrado no Quadro 1. Porém, antes de se iniciar esse algoritmo, é necessário determinar o ponto médio P entre o PIN e o PIF a partir das suas coordenadas. Nesse ponto médio será posicionada a origem do plano cartesiano C, como foi discutido na seção 3.3.2. Dessa forma, para que um nó S possa escolher o próximo salto, deve ser considerado que o mesmo possui n vizinhos V_n, onde $n = \{1, 2, 3, \dots, n\}$. Assim, para cada um dos vizinhos de S o IDEA verifica primeiramente se V_n é um nó fonte. Se for, esse nó será automaticamente eleito como próximo salto. Caso contrário, será verificado se V_n foi o último salto ou está dentro da região de anomalia. Se alguma dessas duas hipóteses se confirmarem, esse vizinho é automaticamente descartado. Caso contrário, o IDEA irá calcular o ganho angular α_n, a distância D_n do vizinho V_n até a trajetória de referência e

irá obter a energia restante E_n do vizinho V_n armazenada na *cache* do nó S. Porém, se D_n for maior do que o LA, esse nó será automaticamente descartado. Em seguida, serão Normalizados os valores de α_n, D_n e E_n. Para isso, cada um desses valores é dividido pelo máximo que a respectiva variável pode assumir. Por exemplo, no caso do ganho angular α_n, divide-se por π. Já no caso da distância D_n, divide-se pelo LA. Por fim, no caso da energia remanescente E_n, divide-se pela EI. Assim, os valores obtidos após essa normalização são representados por α'_n, D'_n e E'_n.

```
1:  Para cada V_n faça {
2:      Se V_n é a fonte {
3:          Faça V_n o próximo salto
4:      } Se não {
5:          Se V_n está na Região de Anomalia ou V_n é o último salto {
6:              Desconsidere V_n
7:          } Se não {
8:              Calcule α_n
9:              Calcule D_n
10:             Obtenha E_n
11:             Faça α'_n = α_n / π
12:             Faça D'_n = D_n / LA
13:             Faça E'_n = E_n / EI
14:             Faça C_n = P_1 D'_n + P_2 (1 / α'_n) + P_3 (1 / E'_n)
15:             Se próximo salto ainda não foi definido {
16:                 Faça V_n o próximo salto
17:                 Faça C_m = C_n
18:             } Se não {
19:                 Se C_n < C_m {
20:                     Faça V_n o próximo salto
21:                     Faça C_m = C_n
22:                 }
23:             }
24:         }
25:     }
26: }
```

Quadro 1 - Algoritmo para a escolha do próximo salto

O passo seguinte do algoritmo é calcular o custo C_n para que o vizinho V_n seja eleito como próximo salto. Esse cálculo inicia-se invertendo os valores de α'_n e E'_n. Em seguida realiza-se uma média ponderada desses dois valores invertidos juntamente com D'_n. A operação de inversão de α'_n e E'_n é necessária, pois, ao contrário da distância para a trajetória, quanto maior o α_n e o E_n, maior prioridade esse vizinho deverá ter na escolha do próximo salto e, conseqüentemente, menor deverá ser o custo para o mesmo.

Assim, o custo C_n para que o vizinho V_n seja eleito como próximo salto seria dado pela seguinte expressão:

$$C_n = P_1 D'_n + P_2 \frac{1}{\alpha'_n} + P_3 \frac{1}{E'_n}, \quad (2)$$

onde P_1, P_2 e $P_3 \in \mathbb{R}\ [0,1]$ e $P_1 + P_2 + P_3 = 1$

Assim, o vizinho que possuir o menor custo será eleito como próximo salto. Se nenhum dos n vizinhos for elegível para próximo salto (ou seja, todos os vizinhos estão dentro da região de anomalia, ou estão além do LA, ou foi o último salto), o desvio será abortado.

Os coeficientes P_1, P_2 e P_3 da expressão (2) representam os pesos que cada um desses elementos terá ao se calcular o custo de um vizinho. Entretanto, os valores desses pesos são iguais para todos os vizinhos e eles são configuráveis pelo administrador da rede. A modificação desses pesos pode alterar drasticamente a forma como as mensagens são encaminhadas. Porém, a escolha dos pesos ideais e como eles afetam exatamente a escolha do próximo salto não estão no escopo desse trabalho (ver capítulo 0). Por isso, nas simulações será adotado o mesmo peso para todos os fatores.

3.4.4 Finalizando o desvio

Após a escolha das trajetórias de referência e do encaminhamento da REQD através de vários nós nas proximidades dessas trajetórias, é necessário estabelecer o momento em que o desvio deve ser interrompido. Intuitivamente, o momento ideal seria quando a REQD alcançasse o PIF. Contudo, dificilmente isso será possível. Primeiramente, porque é muito pequena a probabilidade de haver um nó situado exatamente sobre o PIF. Em segundo lugar, quando a REQD alcança um nó o qual está muito próximo do PIF, isso indica que a região de anomalia já não está mais entre esse nó e a fonte. Assim, na maioria dos casos, tentar levar a REQD até o nó mais próximo do PIF, ao invés de interromper o desvio um pouco antes, pode ser ineficiente e representar um grande desperdício de energia. Entretanto, devido à distribuição dos nós na rede, até mesmo encaminhar a REQD para algum nó nas redondezas do PIF pode ser um objetivo impossível de se alcançar.

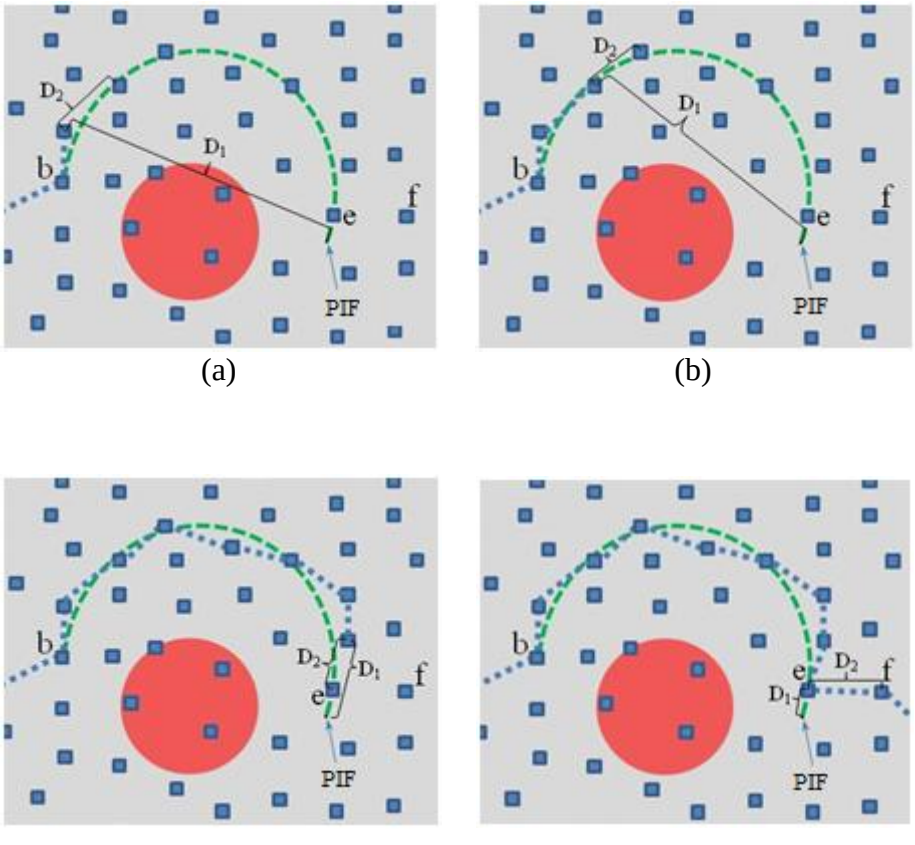
A seguir, serão apresentadas as condições sob as quais o IDEA interrompe a construção de um desvio. Em uma primeira seção será abordada a situação na qual o desvio é finalizado normalmente, ou seja, pela aproximação da REQD com o PIF. Na seção seguinte, serão discutidas as circunstâncias em que um desvio pode ser finalizado prematuramente.

Finalizando normalmente

Em uma situação ideal, cada ponto da trajetória de referência possuiria um sensor e a REQD seria encaminhada através deles até alcançar o PIF e, a partir desse ponto, o protocolo de roteamento da rede voltaria a encaminhar o *request* até a fonte. Porém, isso dificilmente acontecerá. Na realidade, o que deve ocorrer na maioria das redes de sensores é que nenhum dos nós estará posicionado exatamente sobre a trajetória de referência e muito menos algum nó coincidirá com o PIF.

Dessa forma, o IDEA utiliza uma abordagem de melhor esforço para alcançar o PIF. Isso quer dizer que a REQD será encaminhada em direção ao PIF até o momento em que seja mais vantagem remover as informações adicionadas pelo IDEA e encaminhar o *request* original em direção à fonte através do protocolo de roteamento da rede. Essa verificação do que é melhor, aproximar a REQD ainda mais do PIF ou encaminhá-la em direção à fonte, é realizada por cada nó do desvio e antes que esse encaminhe a REQD para o próximo salto. Se a distância do nó do desvio para o PIF for menor do que a distância do mesmo para o próximo salto, o desvio é finalizado (ver Figura 19).

Quando a finalização normal do desvio ocorre, as informações adicionadas pelo IDEA são removidas do *request* e o mesmo volta a ser conduzido em direção à fonte através do encaminhamento geográfico.



(c) (d)

Figura 19 - Regra do melhor esforço para finalizar o desvio

Finalizando prematuramente

A disposição dos nós da rede em relação à trajetória de referência é determinante para a forma como o desvio será finalizado: normalmente ou não. Quando a densidade da rede é baixa ou os sensores estão mal distribuídos, é provável que a mensagem seja encaminhada para um nó o qual não possua vizinhos capazes de continuar a construção do desvio.

A fim de evitar loops de roteamento, o DD e o GEAR descartam mensagens já recebidas anteriormente (ver seção **Erro! Fonte de referência não encontrada.**). Um esses casos ocorre quando uma mensagem de REQD é conduzida para um nó o qual só possui como vizinho o último salto. Quando isso acontece, o IDEA nem envia a REQD de volta para o último salto. Antecipadamente, ele a descarta. Porém, nesse caso, a conseqüência desse descarte é a interrupção prematura da construção do desvio. É importante lembrar que o IDEA leva em consideração somente os vizinhos válidos na eleição do próximo salto. Ou seja, apenas aqueles que estão fora da região de anomalia.

Além desse caso em que um sensor só possui como vizinho o último salto, há outras situações em que uma mensagem poderá efetivamente retornar a um nó e ser descartada. Por exemplo, se for dada uma prioridade muito maior aos nós que estão mais próximos da trajetória de referência. Esse caso é demonstrado através da Figura 20. Nessa o nó A encaminha primeiramente a REQD para o nó B (como esperado). Entretanto, esse último, ao invés de encaminhar a mensagem para o nó C, o qual possui um ganho angular melhor, ele a encaminhará para o nó D. Esse, por sua vez, não terá alternativa e enviará a mensagem de volta ao nó A, o qual irá subitamente descartá-la.

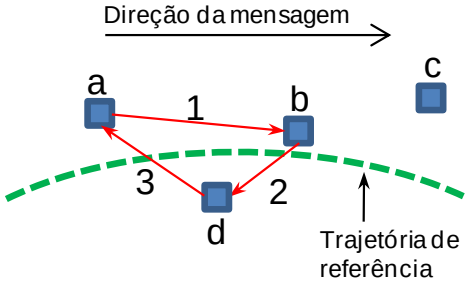
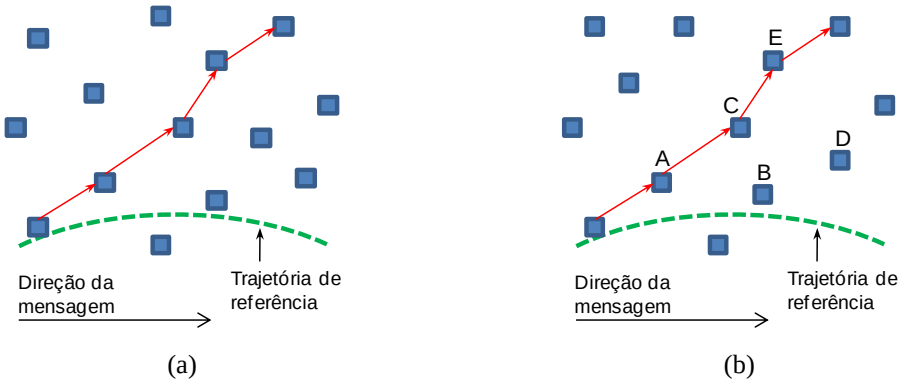


Figura 20 - Loop causado por priorização da proximidade do nó com a trajetória de referência

Por fim, há mais uma situação em que o desvio pode ser finalizado prematuramente. Essa situação já foi comentada antes na seção 3.4.3. Ela ocorre quando a REQD ultrapassa o Limite de Afastamento (LA). A causa desse afastamento pode ser

motivada, assim como em casos anteriores, pela má distribuição dos nós na rede ou pela adoção inapropriada de pesos para a média ponderada que calcula o custo para o próximo salto.

A Figura 21(a) ilustra o primeiro caso. Uma condição pouco provável, mas possível, em que a mensagem se afasta da trajetória de referência por haver um único vizinho válido para o qual ela pode ser encaminhada.



(a) (b)

Figura 21 - Mensagem se afastando indefinidamente da trajetória de referência

Já o segundo caso é ilustrado pela Figura 21(b). Nesse, considere, por exemplo, que o peso da energia remanescente é muito maior do que os demais. Assuma também que os nós A e B foram inicialmente eleitos para fazer parte de um desvio. Após algum tempo propagando mensagens de dados, A recebe uma nova mensagem de REQD para manter o caminho de desvio atualizado ou encontrar novos caminhos. Entretanto, A encaminha a nova REQD para o nó C, e esse para o nó E, e assim sucessivamente, até que a mensagem atinja o LA. Isso ocorre uma vez que o nó B, por fazer parte de um desvio construído anteriormente, possui menos energia do que C. Enquanto que o nó D, por estar mais próximo da trajetória de referência (uma região onde há uma intensa troca de mensagens), consome mais energia do que o nó E.

Entretanto, quando uma REQD atinge o LA, ela não é descartada. Ao invés disso, todas as informações adicionadas anteriormente pelo IDEA ao *request* são removidas, e esse, como uma última tentativa de se alcançar a fonte, volta a ser encaminhado pelo protocolo de roteamento da rede.

CAPÍTULO

4

Simulações e Resultados

A realização de simulações permite avaliar o funcionamento do IDEA. Nesse capítulo, será primeiramente descrita a metodologia utilizada. Dentro dessa sessão será apresentado um estudo para a escolha da trajetória de referência a ser utilizada. Nas sessões seguintes, serão apresentados os cenários de teste e as métricas avaliadas. Serão justificadas algumas das configurações adotadas para a rede e os sensores. Também será discutida a importância dessas simulações e o que se pretende avaliar através das mesmas. Por fim, serão apresentados os resultados das simulações através de gráficos e textos explicativos que demonstram a eficiência do IDEA.

4.1 Metodologia

O principal objetivo das simulações foi avaliar a eficiência do IDEA no isolamento das regiões de anomalia e na construção de múltiplos desvios. Ou seja, avaliar sob que condições o IDEA apresenta os melhores resultados e a partir de quais condições ele perde em desempenho. Para isso, foi realizada a implementação do IDEA no simulador de redes ns-2. Os detalhes dessa implementação podem ser consultados no Apêndice A.

4.1.1 Implementação do modelo

A fim de se avaliar o funcionamento do IDEA, foi realizada a implementação do mesmo na versão 2.32 do simulador ns-2. Esse simulador foi desenvolvido em C++ e utiliza scripts TCL para descrever e controlar os ambientes de simulação. A implementação do IDEA, por sua vez, é um conjunto de classes (também escritas em C++) as quais interagem com as classes no ns-2.

Complementarmente à implementação do IDEA, foram adotados os códigos do Directed Diffusion e do GEAR, disponíveis ns-2, como os protocolos de roteamento da rede. Entretanto, para isso, foram necessárias algumas modificações nesses códigos.

Essas modificações podem ser consultadas no Apêndice C. Adicionalmente a isso, no Apêndice A e B podem ser consultadas explicações básicas sobre o funcionamento do DD e do GEAR respectivamente. A escolha desses protocolos foi motivada pelo fato deles dois atenderem à maioria dos pré-requisitos necessários para o funcionamento do IDEA.

No DD as mensagens de *request* são conhecidas como interesses. Uma vez recebido um interesse, a fonte foi incumbida de enviar 1000 mensagens de dados para o *sink*, a uma freqüência de duas mensagens a cada segundos. Os tamanhos das mensagens de dados foram equivalentes aos adotados em [54], ou seja, 64 *Bytes*. Os parâmetros de freqüência de envio das mensagens de interesse e das mensagens de dados exploratórias também foram determinados com base nos utilizados em [54].

Para poder realizar as simulações e avaliações desejadas, foi necessário definir primeiramente alguns parâmetros da rede. Um desses parâmetros é o raio de alcance dos sensores. Em [55] são utilizados diferentes cenários de teste para se realizar um estudo sobre o raio de alcance de sensores CC2420 da Texas Instrument [56]. Através desse, foi possível constatar que o raio de alcance dos sensores pode variar muito de acordo com a freqüência, o terreno e a altura em que eles são posicionados. Por exemplo, o alcance de um CC2420 operando a 900MHz pode variar de 5 a 80 metros. Dessa forma, para os experimentos, foi definido que a interface 802.11 padrão do ns-2 possuiria um raio de alcance de 40 metros.

Porém, para o consumo de energia, foi adotado o padrão já utilizado em diversos trabalhos, como em [54,7]. Ou seja, o cartão PCM-CIA WLAN padrão do ns-2. Esse cartão consome 0,660 W durante as transmissões, 0,395 W durante as recepções e 0,035 W quando em *idle*.

4.1.2 Escolha da trajetória de referência

Na seção 3.4.2, foram discutida quais características as trajetórias de referências devem ter para que elas possam ser utilizadas na construção dos desvios. Nessa mesma seção, já foi declarado que a definição da melhor equação para descrever as trajetórias de referência não está no escopo desse trabalho. Contudo, para a implementação do IDEA no ns-2, era necessário definir uma trajetória de referência a ser utilizada. Para isso foi necessário realizar um estudo comparativo sobre os gráficos de algumas equações.

Gráficos de funções polinomiais

Intuitivamente, talvez pela simplicidade e familiaridade, o primeiro tipo de equação avaliada foram as das funções de segundo grau. Uma vez que o seu gráfico é uma parábola, sabe-se que o mesmo cruza o eixo das abscissas em apenas dois pontos,

os quais podem ser associados ao PIN e o PIF. Uma parábola também possui um único ponto extremo, o qual pode ser de máximo ou de mínimo (dependendo dos coeficientes da equação). Outra característica interessante das equações de segundo grau é a possibilidade de se variar facilmente a coordenada y do seu ponto de máximo. Se isso for feito e forem mantidos os mesmos pontos de intercessão com o eixo das abscissas, será possível obter sempre um novo gráfico e, conseqüentemente, uma nova trajetória (ver Figura 22).

Em seguida foram pesquisados outros gráficos que possuíssem características similares aos das equações de segundo grau. Algumas análises matemáticas nos conduziram a uma família de equações do seguinte tipo:

$$y = a|x|^3 + bx^2 + cx + d \tag{3}$$

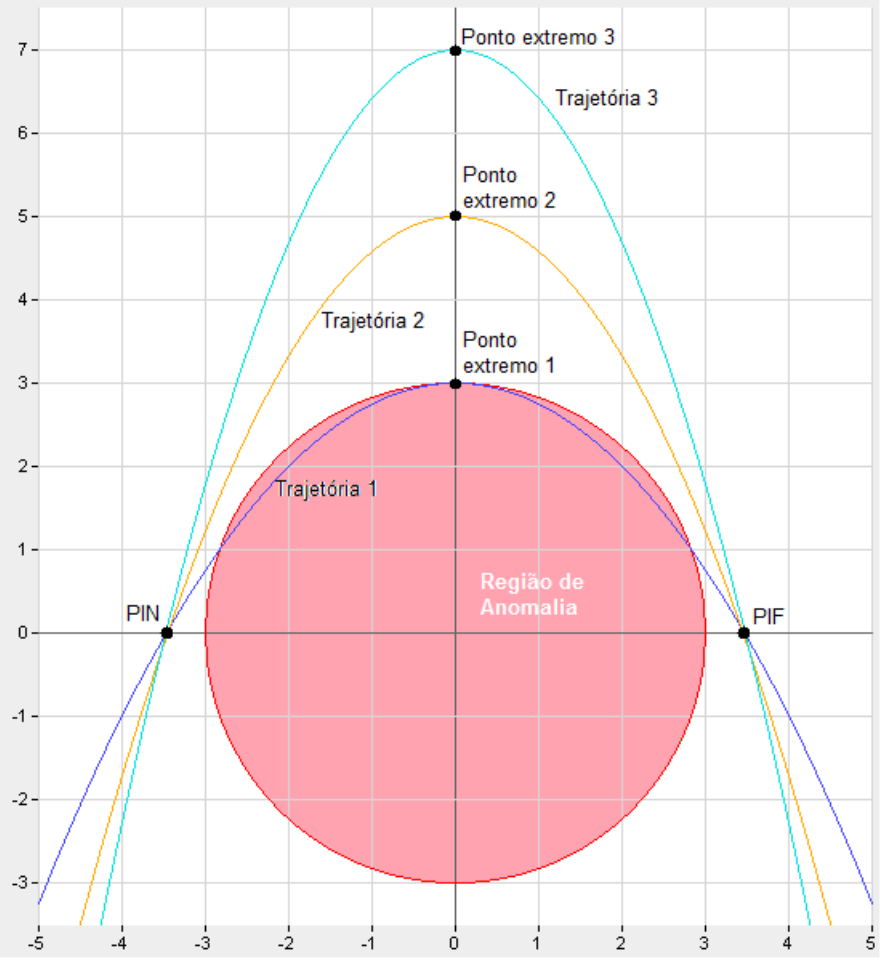


Figura 22 - Trajetórias definidas por equações do segundo grau

Foi possível perceber que esse tipo de equação é capaz de fornecer trajetórias as quais, diferentemente das parábolas, não interceptam a região de anomalia e ainda assim podem tangenciar essa mesma região em seu maior valor de y. Se forem considerados

os coeficientes b e c da equação (3) igual a zero e variarmos os valores de a e d, será possível obter gráficos como os apresentados através da Figura 23.

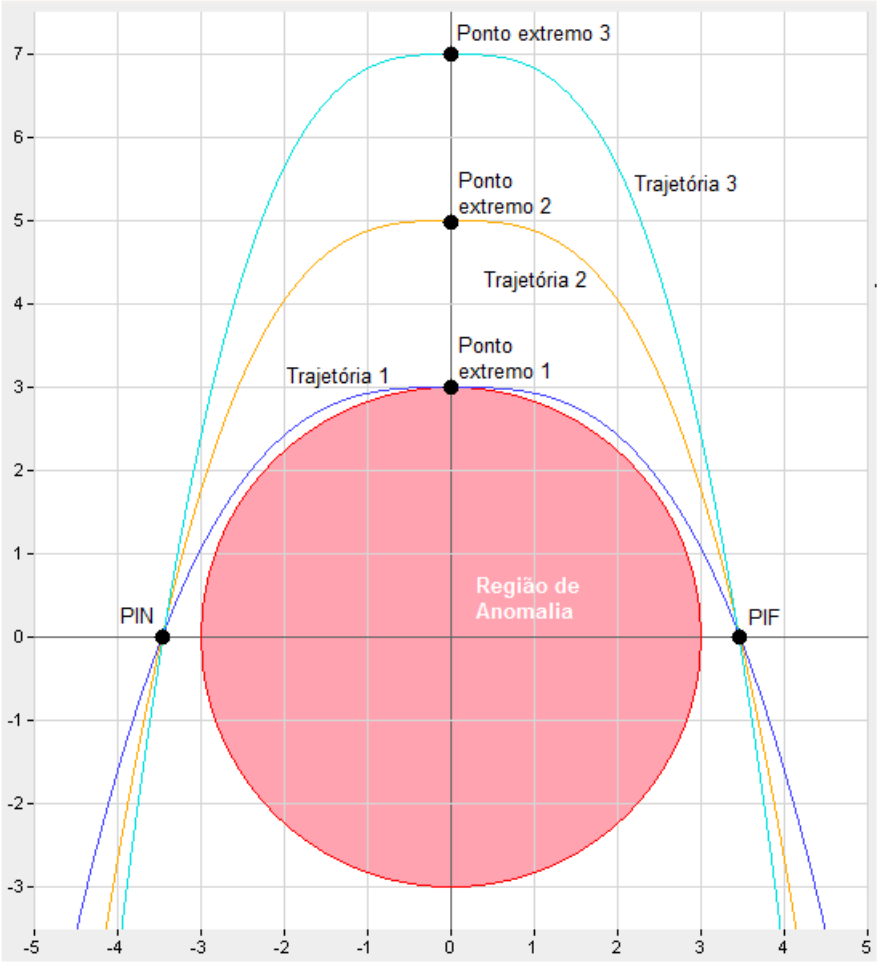


Figura 23 - Trajetórias definidas por equações do tipo $a|x|^3 + c = 0$

Motivados pelos benefícios alcançados pelas equações do tipo (3), foi dado continuidade aos estudos a fim de se obter equações que tivessem uma distribuição no plano cartesiano ainda mais uniforme. Nesse sentido, foram encontradas as equações do seguinte tipo:

$$y = a|x|^4 + bx^3 + cx^2 + dx + e \qquad (4)$$

Assim, similarmente como foi feito para as equações do tipo (3), foi atribuído o valor zero aos coeficientes b, c e d e variados os valores de a e e para se obter os gráficos apresentados na Figura 24.

Apesar de não estarem ilustrados nesse trabalho, posteriormente também foram analisados outros gráficos dados por equações similares às (3) e (4). Para isso, apenas foi aumentado o grau da variável x. Assim, foi possível perceber que quanto mais se aumentava o grau de x, mais o formato das curvas se parecia com o de uma letra "U".

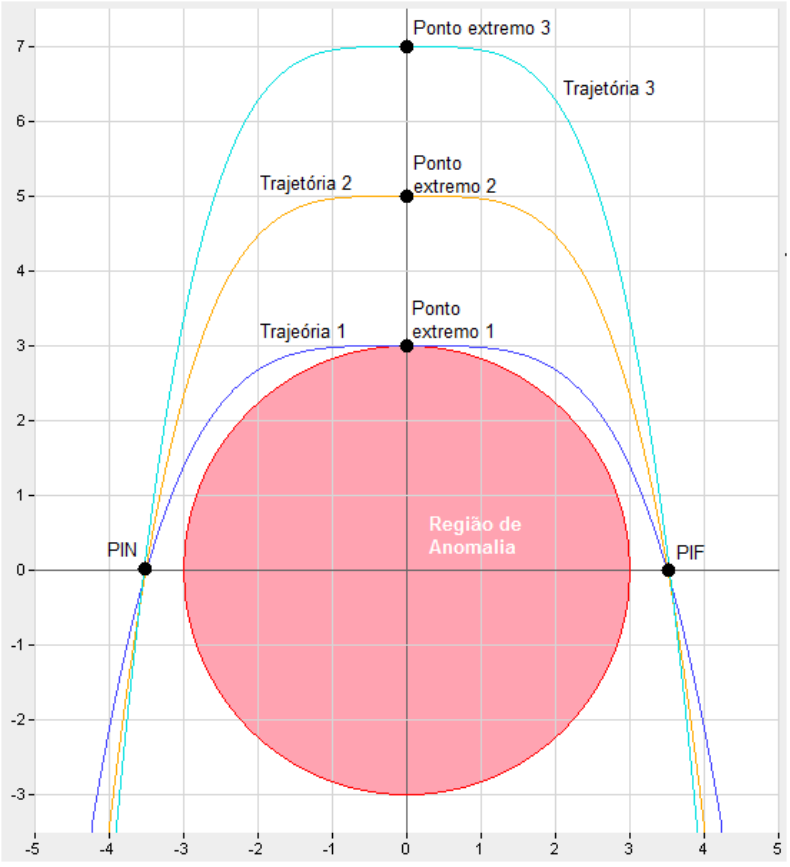
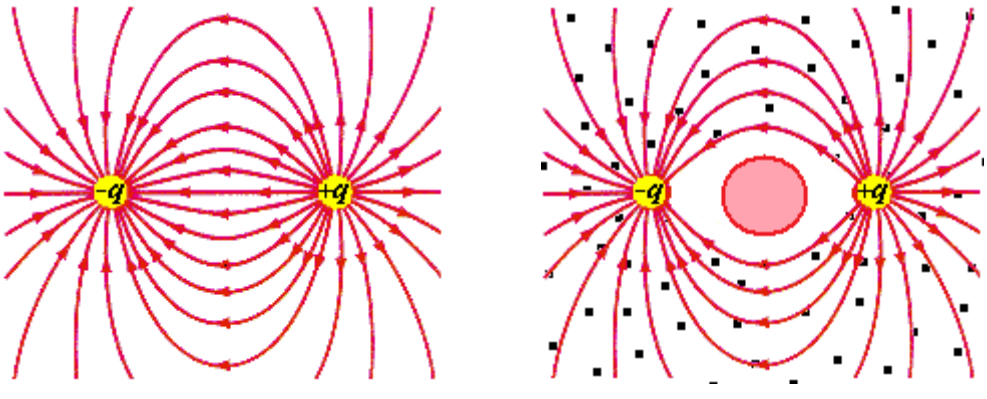


Figura 24 - Trajetórias definidas por equações do tipo $a|x|^4 + e = 0$

Gráficos de linhas de força

Inesperadamente, durante os estudos, foi possível observar a representação gráfica das linhas de força eletrostática formadas por duas cargas de sinais opostos (ver Figura 25(a)). A partir dessa imagem, foi intuitivo associar o PIN à carga $-q$, o PIF à carga $+q$ e as trajetórias de referência às linhas de força. Se esses elementos forem colocados em uma rede de sensores onde há uma região de anomalia, seria experimentada a situação apresentada na Figura 25 (b).



(a) **(b)**

Figura 25 - Associação das linhas de força com as trajetórias de referência

Dessa forma, foram estendidos os estudos sobre os gráficos das funções polinomiais aos das linhas de força e, assim, foi possível obter a Figura 26. Através dessa, foi possível perceber que as linhas de força possuem uma melhor distribuição no plano cartesiano do que os casos analisados anteriormente. Isso pode ser percebido principalmente nos dois pontos em que as curvas convergem. Diferentemente das linhas de força, nos outros gráficos as trajetórias já se aproximam muito antes de chegarem perto do PIN e do PIF.

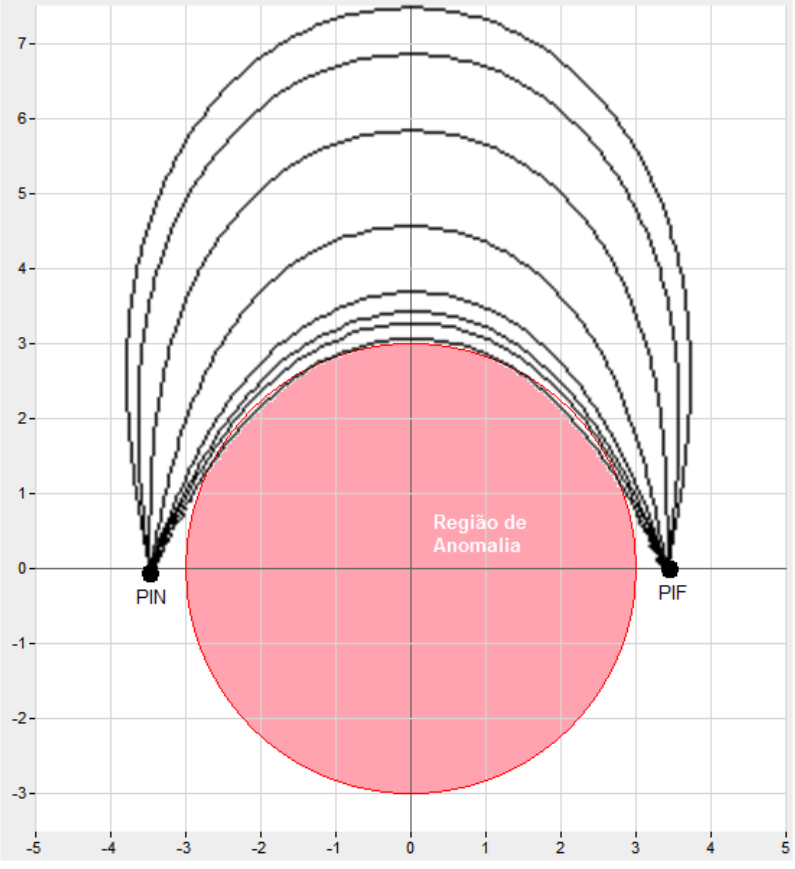


Figura 26 - Trajetórias definidas por linhas de força eletrostáticas

Esses primeiros resultados gráficos estimularam o aprofundamento dos estudos sobre as linhas de força eletrostáticas. Assim, foi constatado que várias propriedades dessas linhas são interessantes para o IDEA. Essas propriedades já foram comentadas na seção **Erro! Fonte de referência não encontrada.**, porém dentre é possível citar ovamente:

- Dadas duas cargas de sinais opostos, infinitas linhas de força saem do pólo positivo e chegam ao pólo negativo.
- As linhas de força são necessariamente disjuntas. Ou seja, dado um ponto no plano cartesiano, uma única linha de força passa por esse ponto.

- A partir da localização de um ponto qualquer do espaço, da intensidade das duas cargas e da localização das mesmas, é possível determinar a linha de força que passa por esse ponto.

Em [57], Dan Alin Muresan realizou um conjunto de cálculos matemáticos a partir de equações diferenciais para obter a expressão (5). Essa expressão é capaz de descrever as linhas de força em um plano cartesiano. Para a realização desses cálculos, Muresan precisou realizar as seguintes considerações:

- Dadas duas cargas, $-q$ e $+q$, elas foram associadas respectivamente ao ponto A e B de um plano cartesiano (ver Figura 27).
- A origem desse plano cartesiano foi posicionada no ponto médio do segmento $\overline{AB}$, cujo comprimento é igual a $2a$
- Por fim, o eixo das abscissas foi determinado como a reta que passa pelos pontos A e B.

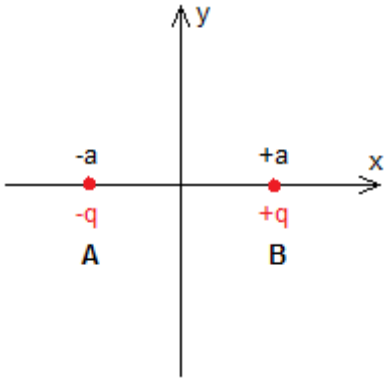


Figura 27 - Considerações sobre o posicionamento das cargas no plano cartesiano

Dessa forma, o coeficiente a da equação (5) é dado pela coordenada x do ponto B, enquanto que C é uma constante real contida no intervalo $[0,2q]$. Assim, após se atribuir um valor para a, é possível variar C para se obter todas as linhas de força que partem de uma carga para a outra. Ou seja, para cada valor real de $C \in [0,2q]$, duas linhas de força são determinadas: uma situada no primeiro e segundo quadrantes do plano cartesiano e outra, simétrica a essa primeira em relação ao eixo das abscissas, situada no terceiro e quarto quadrantes. Isso significa que se um ponto T do plano for escolhido e as suas coordenadas atribuídas ao x e y da expressão (5) será possível calcular o valor de C para esse ponto T, ou seja, será possível calcular o C_T.

$$q\left(\frac{x+a}{\sqrt{(x+a)^2+y^2}} - \frac{x-a}{\sqrt{(x-a)^2+y^2}}\right) = C \qquad (5)$$

Assim, para se encontrar os demais pontos que compões a linha de força que passa pelo ponto T basta encontrar todos os valores de x e y que satisfação à equação (5) com C igual a Ct.

Um dos benefícios em se adotar as linhas de força no nesse trabalho é que, se for considerado que o PIN e o PIF estarão posicionados sobre o ponto A e B respectivamente, o ponto extremo da linha de força sempre estará posicionado sobre o eixo das ordenadas. Dessa forma, será possível obter várias trajetórias de referência alternando apenas o posicionamento desse ponto extremo.

Essa abordagem, além de simplificar muitos cálculos, ela também facilita o armazenamento do histórico de trajetórias adotadas pelo bastião (seção 3.4.2). Pois, nesse caso, ele só terá que gravar um único valor: o da coordenada y de cada ponto extremo adotado.

Dessa forma, esse estudo forneceu os indícios necessários de que, dentre as equações analisadas, as linhas de força devem proporcionar as trajetórias de referências mais adequadas para a construção dos desvios. E, assim, elas foram adotadas na implementação do IDEA no ns-2, como será apresentado a seguir.

4.1.3 Cenários

A escolha dos cenários foi realizada a partir da determinação das dimensões da região sobre a qual os nós seriam distribuídos. Para isso, foi levado em consideração o raio de alcance dos sensores, 40 metros. Além disso, a necessidade de se realizar simulações com regiões de anomalias de diferentes tamanhos também influenciou na determinação das dimensões da rede. Assim, a partir desses elementos, foi possível concluir que uma área de 500x500 metros seria adequada para distribuir os sensores aleatoriamente.

Além disso, foi necessário definir a quantidade de nós que deveriam ser utilizados. Inicialmente foram considerados alguns indícios de que o IDEA deveria funcionar melhor em redes densas. A definição do que é um rede de sensores densa foi introduzida por Gupta *et al.* em [58] e depois adotada em vários outros trabalhos. Assim, uma RSSF pode ser considerada densa se a seguinte expressão for atendida:

$$N > \left(4 \times {}^{s}/_{t}\right)^{2} \quad (6)$$

Onde, N é o número de nós na rede, s é a metade do comprimento de uma das dimensões (considerando regiões quadradas) e t é o raio de transmissão do sensor.

Dessa forma, aplicando os valores dos parâmetros definidos anteriormente nessa fórmula, será possível obter o resultado $N > 625$. Isso significa que, se uma rede

possuir nós capazes de transmitir a 40m e as suas dimensões forem de 500x500 metros, ela só será considerada densa se a quantidade de sensores distribuídos aleatoriamente for superior a 625. Por isso, para as simulações iniciais desse trabalho, foram adotadas e analisadas redes com 600 nós.

Porém, para se avaliar a densidade limite na qual o IDEA ainda funciona adequadamente, a quantidade de sensores foi variada ao passo de 100 nós, tanto para mais como para menos. Assim, a eficiência do IDEA na construção dos desvios foi verificada em redes com 300, 400, 500, 600, 700 e 800 nós. Ou seja, redes com densidades de 1,2; 1,6; 2; 2,4; 2,8 e 3,2 nós por 1.000 m^2.

Contudo, a eficiência do IDEA não depende apenas da densidade da rede. Também é necessário levar em consideração o tamanho da região de anomalia. Por isso, para cada densidade avaliada, também foi variado o raio da região de anomalia em 50, 60, 70, 80 e 90 metros. Então, para cada um desses raios adotados, foram gerados 20 cenários onde os nós foram distribuídos aleatoriamente.

Será mostrado na seção seguinte que em uma rede de 500x500 metros e 500 nós o IDEA conseguiu construir, em 100% dos cenários avaliados, pelo menos um caminho de desvio. Entretanto, apenas a partir de 600 nós (2,4 nós / 1.000 m^2), foi possível observar em todos os cenários a construção de múltiplos desvios. Dessa forma, para algumas avaliações foram utilizados apenas cenários com 600 nós.

Porém, para se verificar a perda de mensagens, o aumento do número de saltos, o aumento do atraso e a quantidade de energia consumida global e localmente, também foram simuladas redes com anomalias superiores a 90 metros de raio. Assim, para cada anomalia de 110, 130, 150 e 170 metros de raio foram gerados mais 20 cenários com densidade de 2,4 nós / 1.000 m^2.

Apesar dos sensores terem sido distribuídos de forma aleatória, o *sink*, a fonte e a região de anomalia foram fixados em uma mesma posição. Isso foi necessário para se induzir a passagem das mensagens através da região de anomalia e, conseqüentemente, provocar a construção dos desvios.

4.1.4 Métricas

Devido à grande quantidade de métricas a serem avaliadas, foram realizados estudos iniciais para se determinar a **densidade mínima na qual o IDEA consegue construir os desvios**, para em seguida, se restringir o escopo das simulações.

Se for considerada uma disposição aleatória dos sensores no plano e for levado em conta que os mesmos possuem um raio de alcance limitado, espera-se que quanto menor a densidade da rede, mais difícil será estabelecer um caminho entre o *sink* e a fonte. Isso

se agrava ainda mais quando o roteamento geográfico é utilizado, pois, nesse caso, poderá haver na rede vários buracos (ver seção 2.2).

Outra variável que influi no sucesso da construção dos desvios é o **tamanho da região de anomalia**. Quanto maior o raio dessa região, maior será a quantidade de nós isolados e indisponíveis para o encaminhamento das mensagens. Além disso, mesmo em redes muito densas, podem ocorrer casos extremos, em que a anomalia é muito grande, ela poderá se aproxima simultaneamente de duas bordas opostas da rede (considerando os nós distribuídos em uma região quadrática). Nesse caso, é provável que ocorra o particionamento da rede. Conseqüentemente, se o *sink* e a fonte estiverem em partes distintas, será impossível estabelecer, não só desvios, mas também qualquer tipo de rota entre os mesmos.

A eficiência do IDEA também foi analisada através de simulações que forneceram a **quantidade de desvios distintos** criados pelo mesmo. Essa métrica é importante para que seja possível avaliar o quanto que o mecanismo de variação de caminhos apresentado na seção 0.0.0 e a utilização da energia remanescente na escolha do próximo salto influenciam na quantidade de desvios construídos.

Por outro lado, é importante avaliar o impacto da construção desses múltiplos desvios na quantidade de mensagens de dados recebidas com sucesso, no aumento do número de saltos, no aumento do atraso e na quantidade de energia consumida global e localmente.

Uma redução na **quantidade de mensagens de dados recebidas com sucesso** é esperada nos casos em que uma anomalia é detectada onde passa uma rota entre a fonte e o *sink*. Para explicar esse fenômeno, é importante lembrar que o início da construção do desvio depende do recebimento de uma mensagem de interesse em um bastião (seção 0.0.0). Dessa maneira, até que essa mensagem de interesse seja recebida e o desvio seja construído, todas as mensagens de dados transmitidas através da rota que passa pela região de anomalia serão descartadas. Dessa forma, é importante avaliar a quantidade de mensagens perdidas durante esse processo.

A construção dos desvios também implica necessariamente em um **aumento do número de saltos**. Por se apoiar nos paradigmas do roteamento geográfico, o protocolo de roteamento sempre tentará descobrir o caminho mais direto entre a fonte e o *sink* e, conseqüentemente, que necessite do menor número de nós. Logo, se uma anomalia bloquear os caminhos mais curtos, será necessário estabelecer rotas mais longas para poder contorná-la. Por isso, foi avaliado o aumento no número de saltos da fonte para o *sink* após a detecção de diferentes tamanhos de anomalias.

Um maior número de saltos, por sua vez, implica em um **aumento do atraso** para que um dado seja enviado pela fonte e recebido no *sink*. Isso ocorre, pois, cada nó gasta

pelo menos algumas frações de segundo para receber, processar e retransmitir a mensagem. Desse modo, foi avaliado em quanto o atraso de uma mensagem de dados aumenta à medida que são construídos desvios e ocorrem variações no raio da região de anomalia.

Além do aumento do atraso, é possível prever que um maior número de saltos consuma mais energia da rede como um todo. Se for considerado que um sensor gasta uma quantidade de x Watts para encaminhar uma mensagem e inicialmente eram suficientes y sensores para levar a mensagem da fonte para o *sink*, haveria um consumo de $x \times y$ Watts para cada mensagem transmitida. Contudo, se o número de saltos aumentar em z sensores, o consumo de energia para executar essa mesma tarefa passará a ser $x \times (y + z)$ Watts.

Toda via, uma vez que o IDEA é capaz de construir múltiplos desvios e o envio das mensagens de dados é balanceado entre esses desvios, era de se esperar que o consumo individual de cada sensor fosse muito menor do que se fosse adotada uma única trajetória de referência. Além disso, uma vez que as trajetórias definidas pelo IDEA são espaçadas, também era de se esperar que o consumo de energia dos nós fosse significativo apenas quando os mesmos participassem do encaminhamento das mensagens de dados. Ou seja, quando a trajetória que passa por eles fosse utilizada.

Porém, simulações preliminares revelaram que o gasto de energia com a transmissão das mensagens representa apenas uma pequena fração do consumo total de energia de um sensor. Isso ocorre, pois, na maior parte do tempo os sensores permanecem no estado *idle*. Contudo, ao se avaliar a quantidade de energia gasta apenas com as transmissões, foi possível perceber uma grande economia de energia. Dessa forma, foram realizados vários experimentos para se avaliar **o consumo de energia gasta com as transmissões dos sensores**.

A fim de se comparar o consumo de energia quando são utilizados múltiplos caminhos com o obtido quando se utiliza um único caminho, foi necessário modificar a implementação do IDEA para que apenas um dos desvios construídos fosse adotado para o encaminhamento das mensagens de dados.

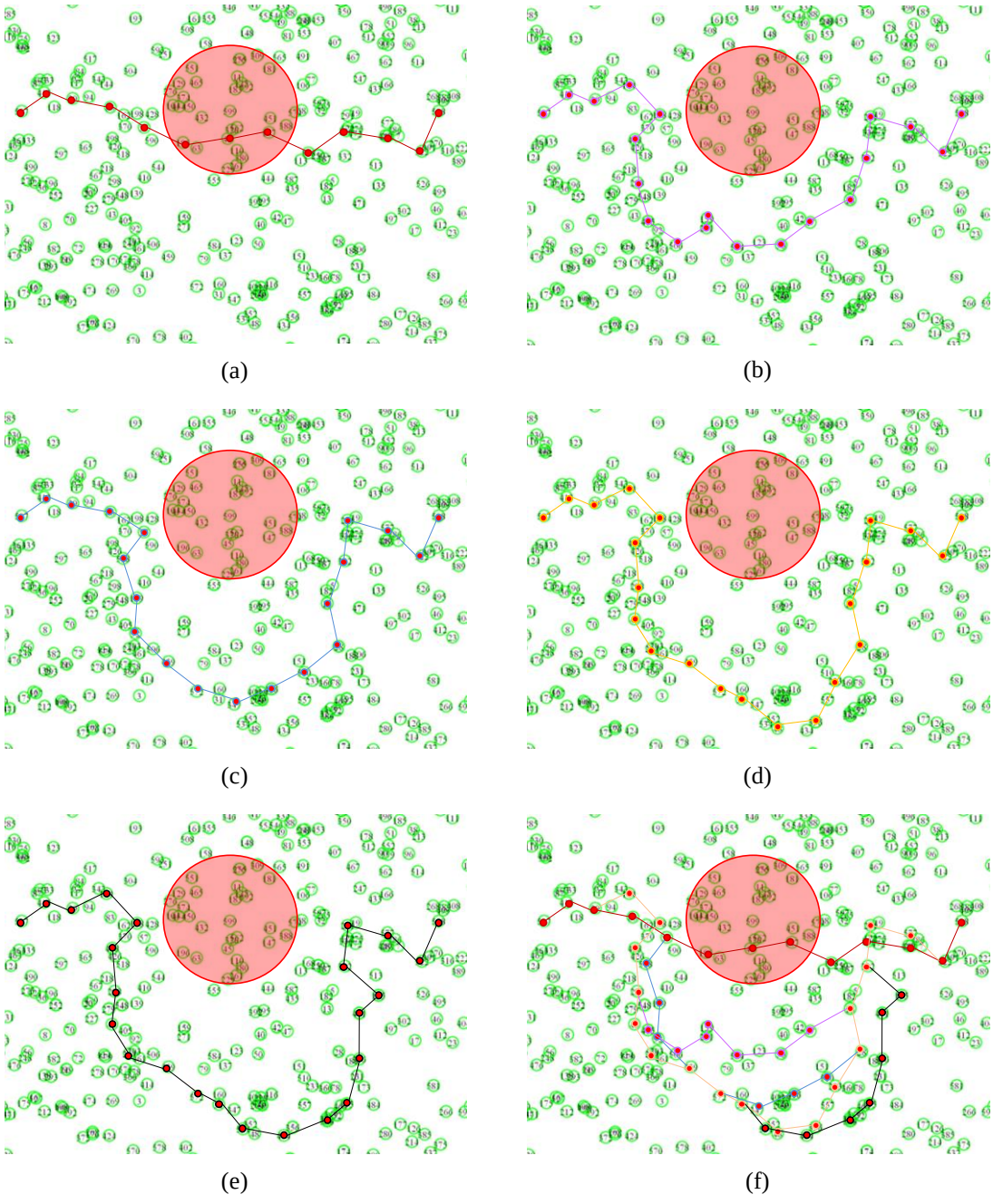
4.2 Análise dos resultados

Nessa seção serão apresentados os resultados das simulações realizadas no ns-2 através de alguns gráficos. Além disso, será discutido o significado de cada um dos resultados obtidos.

A partir da ferramenta de animação gráfica do ns-2, o *Network Animator* (nam), foi extraído o fundo da Figura 28. Esse mostra a distribuição dos nós no plano para uma

das nossas simulações. Em seguida, através de uma análise de *logs*, foi possível traçar as linhas dos desvios estabelecidos pelo IDEA após a detecção de uma anomalia.

A Figura 28(a) mostra o caminho inicialmente determinado pelo GEAR e a região de anomalia. Já as ilustrações (b), (c), (d) e (e) da Figura 28 mostram cada um dos desvios construídos em torno da região dessa mesma região. Por fim, através da Figura 28(f) é possível visualizar em conjunto todas as rotas e a região de anomalia.



(a) (b) (c) (d) (e) (f)

Figura 28 - Desvios construídos pelo IDEA em um dos cenários de teste

Como já foi citado na seção anterior, e era esperado, apesar de serem utilizadas apenas três trajetórias de referência, foram obtidos quatro desvios distintos. Ainda

através Figura 28(f), é possível observar que esses desvios, por sua vez, não são disjuntos. Ou seja, alguns nós participam de mais de um desvio.

Se a distância entre os pontos extremos for aumentada ou for dado um peso maior para o vizinho que estiver mais próximo da trajetória de referência, é possível imaginar que a diferença entre os caminhos possa aumentar. Contudo isso não foi avaliado nesse trabalho e pode gerar efeitos colaterais os quais não é possível prever sem simulações adicionais.

4.2.1 Influência da densidade

Através das simulações, foi possível perceber que a densidade da rede é um fator determinante para a construção de desvios com sucesso.

Após analisar os dados dos experimentos, foi possível verificar que a partir de 500 nós, em todos os nossos cenários de teste, o IDEA conseguiu estabelecer pelo menos um desvio em torno da região de anomalia. É importante lembrar que as dimensões das redes utilizadas foram de 500x500 metros. Isso significa que, a partir de uma densidade de 2 nós / 1.000m² (considerando nós com raio de alcance de 40m), o IDEA será capaz contornar uma região de anomalia definida por uma área circular de até 25.400 m².

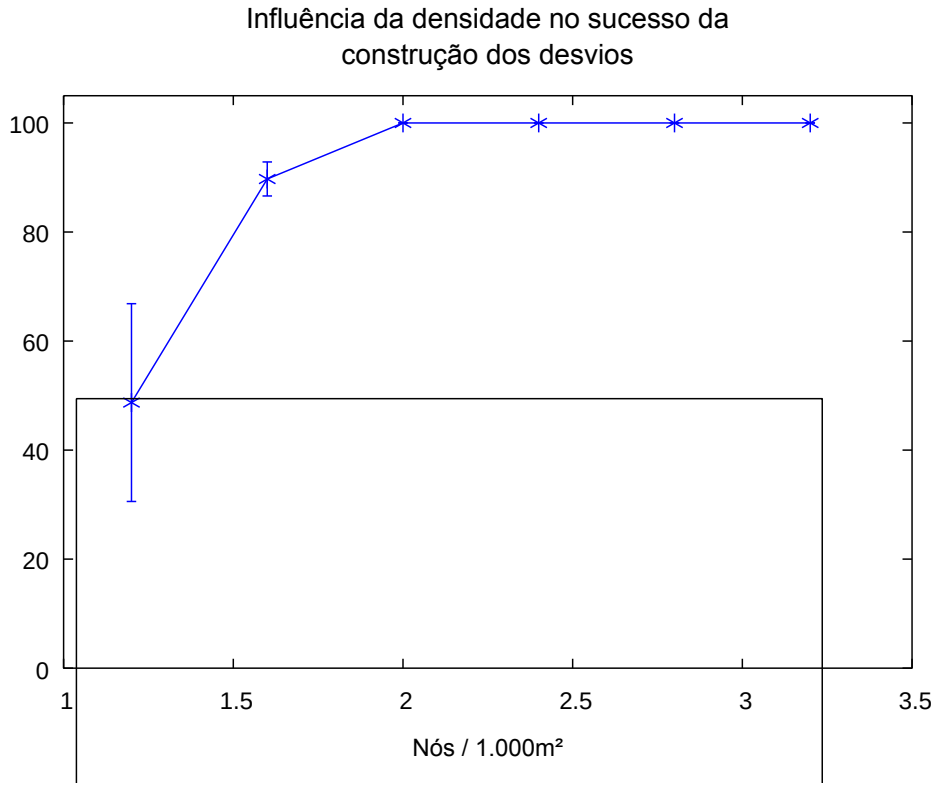


Figura 29 - Influência da densidade da rede no sucesso da construção dos desvios

Através da Figura 29 é possível perceber uma tendência logarítmica do gráfico. Dessa forma, abaixo da densidade de 1 nó / 500m², uma diminuição de 100 nós na rede já reduz em aproximadamente 10% as chances de se construir um desvio com sucesso. Além disso, as simulações também mostraram que em vários cenários de 400, 500 e até 600 nós o GEAR não foi capaz de determinar uma rota entre a fonte e o *sink*. Entretanto, após a detecção da anomalia, o IDEA, através do seu algoritmo de desvio, conseguiu estabelecer um ou mais caminhos entre esses dois nós.

Outra constatação positiva sobre esses resultados é o de que o IDEA não necessita de uma rede densa para alcançar o seu principal objetivo. Essa afirmação pode ser feita se for levada em consideração a definição de redes densas realizada por Gupta *et al.* em [58]. Na seção anterior, foi demonstrado que são necessários pelo menos 625 nós para uma rede com as características das utilizadas nas nossas simulações ser considerada densa. Entretanto, com 125 nós a menos do que esse valor, o IDEA se mostrou capaz de encontrar pelo menos uma rota de desvio.

4.2.2 Influência da construção dos desvios

Foi observado que a densidade da rede, além da influenciar no sucesso da construção dos desvios, também interfere na quantidade de dados recebida pelo *sink*. A perda das mensagens de dados ocorre no momento em que a anomalia é detectada e a fonte continua a enviar dados. Isso acontece uma vez que, para economizar recursos dos sensores, a fonte não é notificada sobre a ocorrência da anomalia. Conseqüentemente, até que uma mensagem de interesse seja recebida por um bastião e o desvio seja construído, a fonte continuará a enviar dados a uma taxa de duas mensagens por segundo e todas essas mensagens serão descartadas.

A Figura 30 mostra que, para os parâmetros de redes adotados nas simulações, a quantidade de mensagens recebidas pelo *sink* se aproxima de 86% na proporção que o número de nós utilizados é superior a 500.

Em alguns cenários com 500 e 600 nós, o GEAR não foi capaz de encontrar um caminho entre a fonte e o *sink* antes do IDEA. Conseqüentemente, nesses casos, não há perdas de mensagens entre o momento em que a anomalia é detectada e o primeiro desvio é estabelecido.

Esse fenômeno também é observado nas simulações com 300 e 400 nós. Entretanto, na maioria desses cenários nem o IDEA e nem o GEAR não conseguem encontrar um único caminho entre a fonte e o *sink* (como foi mostrado na seção anterior). E isso termina provocando um equilíbrio no cálculo da média geral.

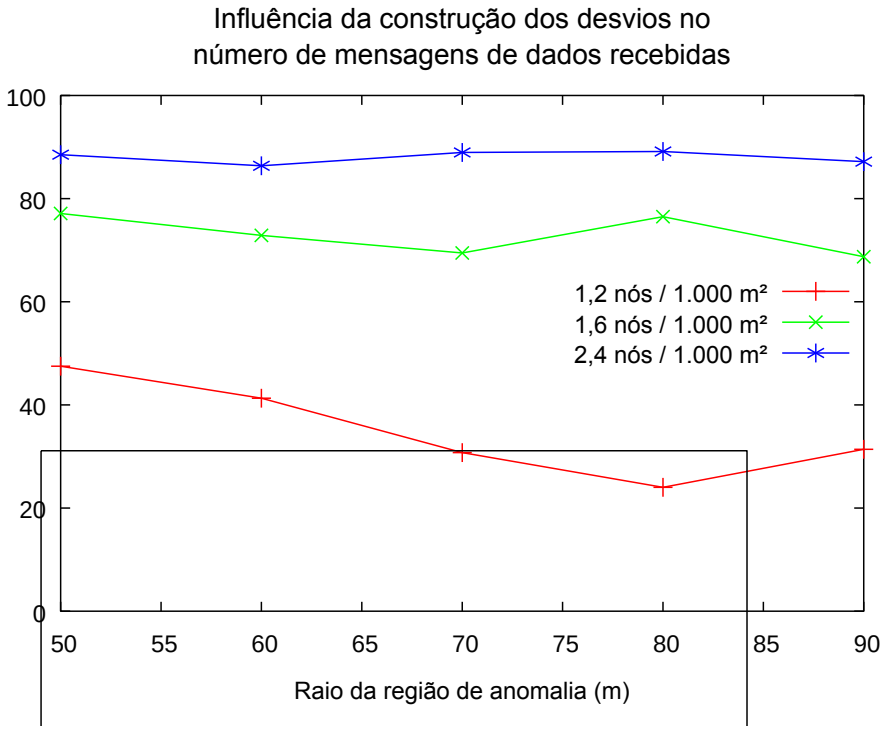


Figura 30 - Influência da densidade da rede no recebimento das mensagens de dados

4.2.3 Influência do tamanho da região de anomalia

Na análise da influência da densidade da rede no sucesso da construção dos desvios, já foi levada em consideração a variação do raio da região de anomalia. Para isso, foi adotado um incremento de 10 metros e avaliado o comportamento do IDEA com anomalias de 50 a 90 metros de raio.

Entretanto, para observar mais detalhadamente a influência do tamanho da região de anomalia no recebimento das mensagens de dados, foram aproveitados os resultados apresentados na seção anterior para primeiramente fixar o número de nós em 600. Em seguida, foram realizadas simulações adotando regiões de anomalia com raio igual a 110, 130, 150 e 170 metros. Os resultados dessas simulações podem ser observados na Figura 31.

Através dessa, foi comparado o aumento do número de nós dentro da região de anomalia com a quantidade de mensagens recebidas pelo *sink*. Dessa forma, foi possível observar que em redes nas quais a região de anomalia possui até 130 metros de raio a quantidade de mensagens recebidas no *sink* se mantém em torno de 87% das mensagens transmitidas pela fonte. Como já foi comentado na seção anterior, a perda de aproximadamente 13% das mensagens de dados ocorre no momento em que a anomalia é detectada e a fonte continua a enviar dados.

Quando o raio da anomalia é modificado para 150 ou 170 metros, pode ser observado um grande aumento na quantidade de mensagens de dados perdidas

(chegando a até 40%). Entretanto, isso já era esperado, pois, quanto maior a região de anomalia, menor será a probabilidade de que a primeira trajetória de referência adotada consiga estabelecer um desvio. É importante lembrar que, apenas na segunda e na terceira trajetória de referência o bastião utilizará um ponto extremo mais afastado da região de anomalia. Entretanto, até que o terceiro interesse seja recebido, muitas mensagens de dados terão sido enviadas pela fonte e descartadas no bastião responsável por isolar a região de anomalia.

Através da Figura 31, também é possível observar que, quando a quantidade de mensagens de dados recebidas pela fonte começa realmente a diminuir, cerca de 20% dos nós da rede já estão dentro da região de anomalia. Em termos absolutos, isso representa um total de 120 nós indisponíveis.

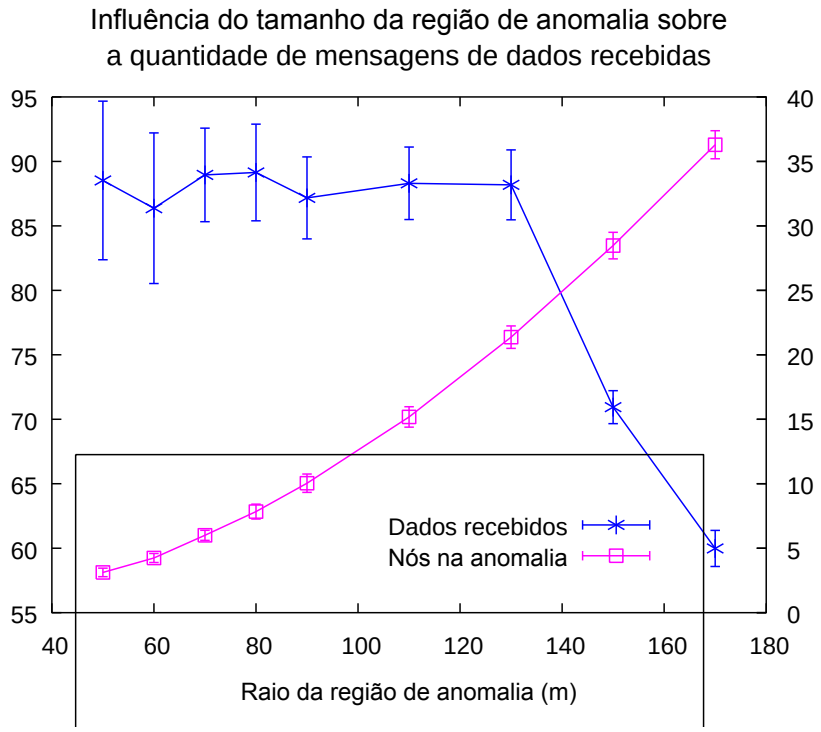


Figura 31 - Influência da quantidade de nós na região de anomalia para o recebimento de dados

Outro fator que influencia fortemente na quantidade de mensagens descartadas, é a freqüência com a qual as mensagens de dados são enviadas. Uma vez que o período do envio das mensagens de interesse é fixo e a perda de pacotes acontece enquanto o desvio não é estabelecido, quanto maior a vazão das mensagens de dados, maior será a quantidade de mensagens barradas na fronteira da anomalia.

4.2.4 Quantidade de desvios construídos

Devido ao mecanismo de variação de caminhos apresentado na seção 0.0.0 e à utilização da energia remanescente do vizinho como uma das variáveis para se escolher

o próximo salto, a quantidade de desvios construídos e que se diferenciam em pelo menos um salto é, na maioria dos casos, superior ao das três trajetórias de referência determinadas pela variação dos pontos extremos. Contudo, através das simulações foi possível perceber que essa quantidade diminui à medida que o raio da região de anomalia aumenta.

A partir do gráfico da Figura 32 é possível constatar que a média de desvios construídos se mantém entre 5 e 5,5 para regiões de anomalia com até 110 metros de raio. Entretanto, para valores acima de 110 metros, há uma queda brusca da quantidade de desvios construídos.

Toda via isso já era esperado. Na seção anterior já foi demonstrado que, em cenários com regiões de anomalia acima dos 130 metros de raio, há uma grande redução do número de nós disponíveis para a construção dos desvios. Nessa seção não foi diferente. Também foi observado que a maior redução do número de desvios construídos ocorreu para anomalias com raios acima de 130 metros.

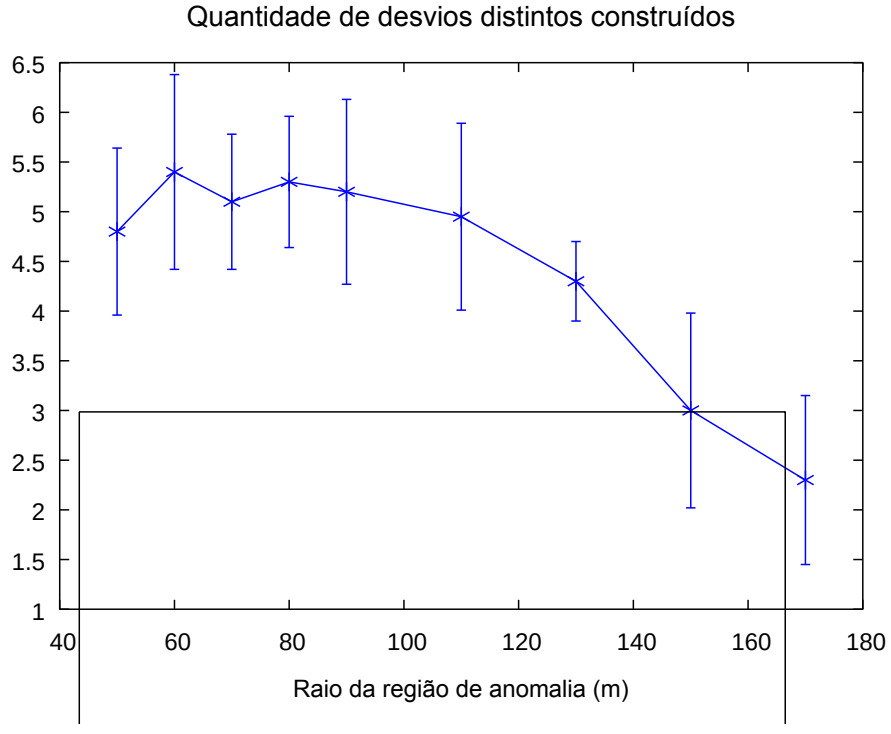


Figura 32 - Quantidade de desvios distintos construídos

4.2.5 Variação do número de saltos

Nas sessões anteriores foram apresentados alguns resultados positivos sobre o IDEA. Contudo, nessa seção e nas seguintes será discutido o impacto da construção dos desvios.

As barras verticais do gráfico apresentado na Figura 33, ao contrário dos demais, não representam o intervalo de confiança. Nesse caso, a o ponto inferior dessas barras representa o menor número de saltos utilizados para se construir um desvio. Enquanto que o ponto superior representa o maior número de saltos para se atingir esse mesmo objetivo.

Além disso, perceba que os pontos que conectam cada segmento das curvas estão quase sempre mais próximos de um dos extremos da barra vertical. Isso fornece um recurso visual para que seja observado com quantos saltos a maioria dos desvios foram construídos.

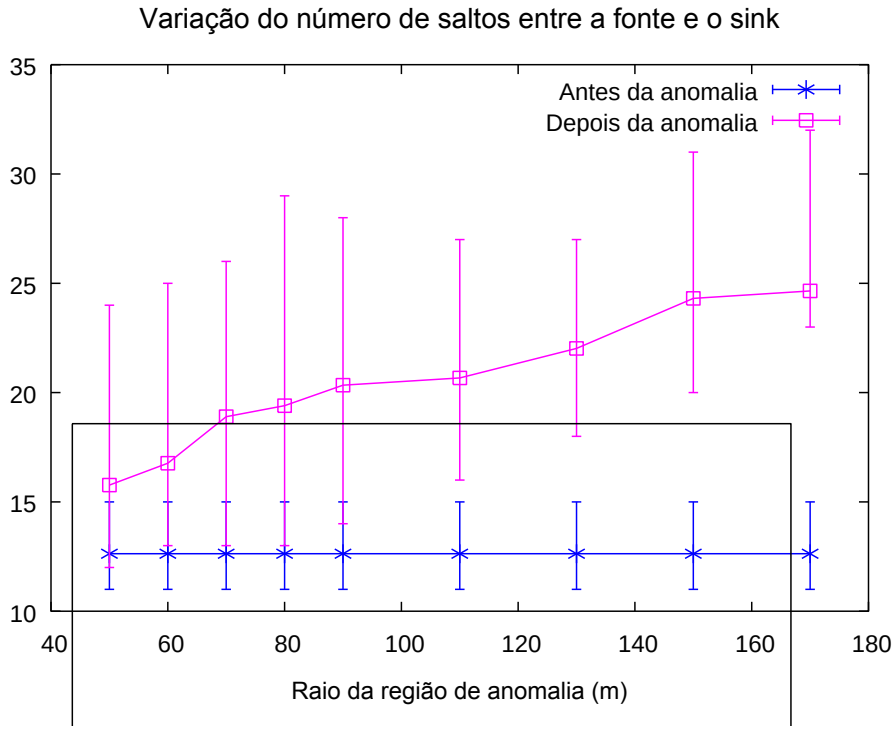


Figura 33 - Comparativo do número de saltos antes e depois da anomalia

Um exemplo pode ser utilizado para explicar isso melhor. No gráfico a baixo, considere a parte que ilustra o número de saltos necessário para se levar a mensagem da fonte ao *sink* depois de detectada uma região de anomalia com 170m de raio. O ponto superior da barra vertical indica que uma mensagem passou por no máximo 32 nós antes de chegar no *sink*. Já o ponto inferior dessa mesma barra, indica que foram necessários no mínimo 23 saltos. Por outro lado, o pequeno quadrado sobre a barra vertical, indica que, em média, foram necessários 25 saltos (aproximadamente). Porém, como o ponto que marca a média está bem mais próximo do ponto inferior da barra vertical, isso significa que a grande maioria dos desvios foram construídos com menos do que 25 saltos.

Após essas elucidações, é possível analisar o significado do gráfico. Uma vez que o número de nós na rede foi fixado em 600 e os 20 cenários utilizados foram os mesmos (independente do raio da região de anomalia), já era esperado que a média do número de saltos antes da detecção da anomalia fosse sempre a mesma. Além disso, também era esperado que o número de saltos necessários para que o *sink* alcançasse a fonte fosse menor antes da detecção da anomalia do que depois.

Ainda em relação à Figura 33, é possível observar que, na medida em que o raio da região de anomalia aumenta, há um crescimento suave da média do número de saltos necessários para se desviar da mesma. Observe ainda que quando a região de anomalia começa a atingir um tamanho muito grande (150 e 170 metros de raio), a média do número de saltos começa a se manter em torno de 24 e o seu máximo não ultrapassa os 32 saltos. Isso ocorre, pois, na proporção em que o raio da região de anomalia aumenta também aumenta a distância do desvio e o número de saltos. Contudo, a partir de certo limite, os nós disponíveis tornam-se escassos e apenas os desvios mais longos continuam disponíveis. O que faz a média do número de saltos convergir para a quantidade de nós necessários para a construção desses desvios mais longos.

4.2.6 Variação do atraso

Uma das conseqüências diretas de um maior número de saltos é o aumento do atraso.

Através da Figura 34 pode-se observar que o à medida que o raio da região de anomalia cresce o atraso também cresce. Isso ocorre, pois, o tamanho do desvio necessário para contornar a região de anomalia também aumenta.

Entretanto, esse gráfico revela um ponto positivo: através da sua curva, é possível observar que o seu comportamento apresenta um crescimento linear e suave. Note que quando o raio da anomalia passa de 50 para 90 metros, ou seja, há uma variação positiva de aproximadamente 45%, o atraso aumenta apenas 22%.

Uma importante comparação que pode ser feita é a do aumento do número de saltos antes e depois da anomalia com o aumento do atraso nessas duas situações. Nesse caso, será possível notar que o impacto da detecção de uma anomalia é maior no número de saltos do que no atraso.

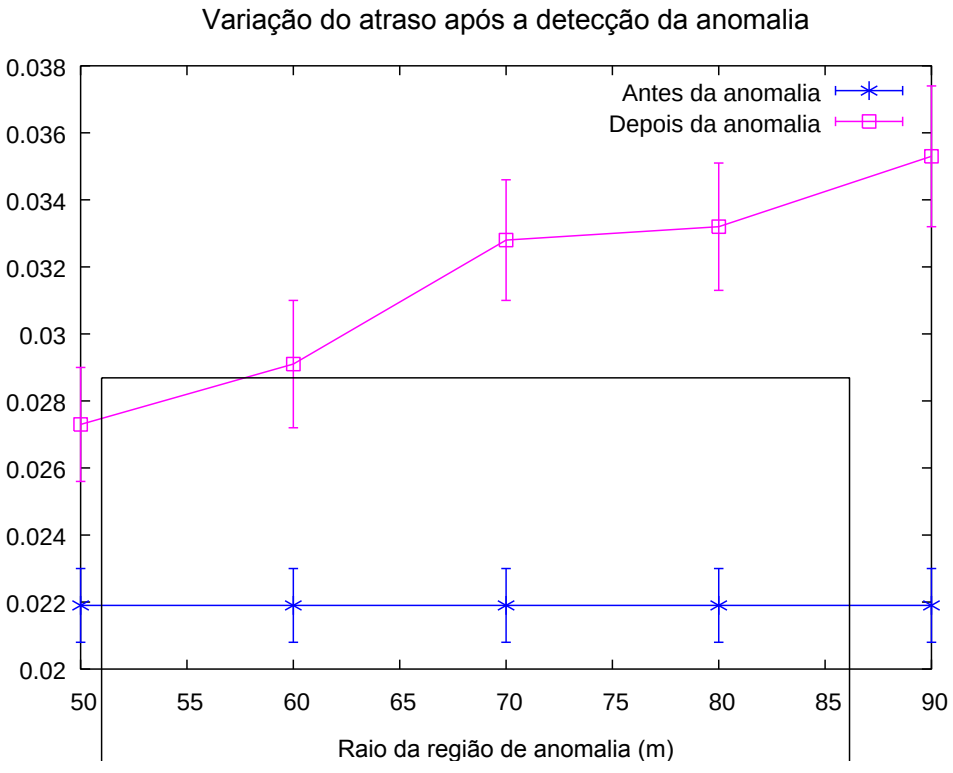


Figura 34 - Variação do atraso após a detecção da anomalia

4.2.7 Consumo de energia

Um dos principais diferenciais do IDEA para outros trabalhos que realizam o desvio de buracos em redes de sensores é na questão do consumo de energia. Ao contrário dos demais trabalhos, o IDEA é capaz de construir múltiplos caminhos de desvio e alternar o envio das mensagens de dados através desses caminhos.

Contudo, ao contrário do que poderia ser esperado, a variação do consumo total de energia é muito pequena. Isso se deve ao fato de que na maior parte do tempo os sensores são colocados no estado *idle*. Esse fenômeno pode ser melhor compreendido através dos seguintes valores fornecidos pelo *EnergyModel* de um sensor que fez parte de um desvio:

- Energia total consumida: 29,652785 J
- Energia consumida em *idle*: 20,474 J
- Energia consumida pelas transmissões: 0,227 J
- Energia consumida pelas recepções: 8,952 J

Isso mostra que a energia gasta quando o sensor está em *idle* é equivalente a 69,04% do total consumido. Enquanto que a energia desprendida nas transmissões representa apenas 0,76%. Isso comprova que o tempo que um sensor passa no estado *idle* é muito maior do que nos demais estados. Pois, apesar do consumo de energia de um nó nas transmissões ter sido definido igual a 0,660 W e em *idle* igual a 0,035 W (ou

seja, 94,7% menor), o total de energia gasta nesse segundo estado é 9000 vezes maior do que o consumido no primeiro.

Entretanto a abordagem utilizada pelo IDEA não se preocupa com o consumo de energia do nó quando no estado *idle*. Mas sim, com a energia gasta nas transmissões das mensagens. Dessa forma, essa métrica será utilizada nas demais comparações que serão feitas daqui para frente.

Como já foi comentado na seção 4.1, para se avaliar a economia de energia proporcionada por múltiplos caminhos, foi adotada uma versão simplificada do IDEA, na qual apenas um dos desvios é reforçado. Assim, foi possível comparar o consumo de energia provocado pelas transmissões de um sensor em uma rede com e sem múltiplos desvios.

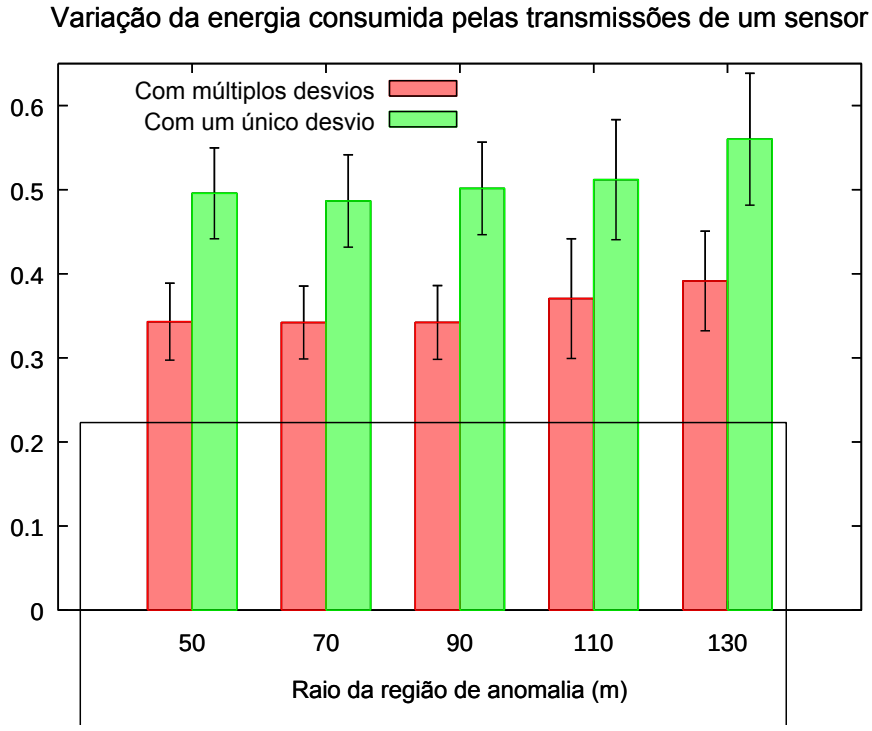


Figura 35 - Quantidade de energia consumida pelas transmissões de um sensor

A Figura 35 ilustra os resultados dessa comparação. Através dela é possível constatar que o consumo de energia médio com as transmissões teve uma redução de aproximadamente 30% em todos os cenários analisados.

Outra análise que pode ser feita sobre o consumo de energia é relacioná-lo com a quantidade de desvios. Já mostrado na seção 4.2.4 que quanto maior o raio da região de anomalia, menor será a quantidade de desvios distintos capazes de serem determinados pelo IDEA. Quando isso acontece, um menor número de sensores participará do

encaminhamento das mensagens de dados e, conseqüentemente, esses terão que transmitir um maior número de mensagens.

Isso justifica o comportamento das duas curvas na Figura 36. Através dessa, pode ser observado que o consumo de energia de energia varia de forma inversa à da quantidade de saltos. Até mesmo nas pequenas variações, quando a linha que indica o número de desvios construídos tem um comportamento crescente, a linha do consumo de energia tem um comportamento exatamente contrário. E o mesmo ocorre no caso inverso, em que a linha do consumo de energia é crescente quando a linha de número de desvios tem um comportamento decrescente.

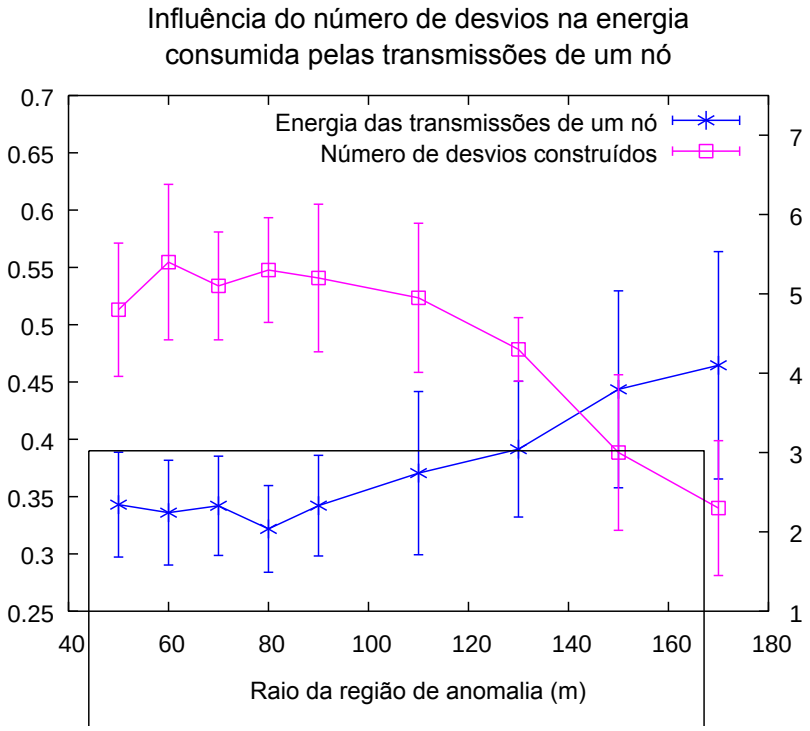


Figura 36 - Influência da quantidade de desvios no consumo de energia

CAPÍTULO

5

Conclusões

O objetivo deste último capítulo é apresentar as considerações finais sobre os principais conceitos e tópicos existentes neste trabalho, bem como os objetivos que foram transformados em contribuições e os possíveis trabalhos futuros. Destaque principalmente para essa última parte, através da qual será possível vislumbrar uma grande quantidade de estudos que poderão potencializar ainda mais o IDEA.

5.1 Considerações finais

Através desse trabalho foi possível perceber que muitas aplicações podem ser desenvolvidas aproveitando o potencial das redes de sensores. Entretanto, para que esse potencial pudesse ser obtido, foram necessárias algumas restrições no hardware dos sensores. Essas restrições, acompanhada pelos cenários em que as redes são implantadas contribuem para a ocorrência de anomalias que não podem ser tratadas através de uma interferência direta nos nós da rede.

Visando solucionar esse problema, foi proposto o IDEA, um protocolo capaz de isolar e construir desvios em torno de anomalias. Para isso, foram considerados alguns outros trabalhos os quais já mostraram ser possível detectar as regiões de anomalia e determinar o posicionamento dos nós da rede. Além disso, foi mostrado que o problema de se desviar de anomalias já vem sendo estudado há algum. Entretanto, nos outros trabalhos a anomalia é estudada apenas como um buraco na rede. A abordagem proposta nesse trabalho, diferentemente das demais, é mais ampla.

Contudo, para isso foi preciso introduzir o conceito de faixa de isolamento. Através dessa, os bastiões impedem a passagem das mensagens de dados para o núcleo da anomalia e determinam se um desvio deve ser construído. Outra inovação desse trabalho é a forma como esses desvios são construídos. Foi mostrado que através de

uma modelagem matemática é possível determinar trajetória de referência as quais servirão de guias para a construção de desvios.

Ao contrário de outros trabalhos, o IDEA garante que todas as interações ocorrem localmente. Isso reduz significativamente a sobrecarga causada por trocas de mensagens desnecessárias e, assim, o consumo de energia. Entretanto, o protocolo apresentado nessa dissertação ainda vai além. Foi desenvolvida uma técnica capaz de criar múltiplos desvios e, conseqüentemente, distribuir a quantidade de energia necessária para encaminhar as mensagens de dados. Dependendo da densidade da rede e do tamanho da região de anomalia, o IDEA conseguiu construir até seis desvios distintos.

Como prova de conceito das idéias apresentadas, foi implementada uma instância do IDEA no ns-2. Entretanto, para isso, foram adotadas linhas de força eletrostáticas para as trajetórias de referência. Após um estudo gráfico de algumas equações, foi possível perceber que as linhas de força possuem características bastante adequadas ao IDEA.

As simulações mostraram que o IDEA é um protocolo eficiente na construção dos desvios. Pois, mesmo em redes com um número de nós inferior às consideradas densas, o IDEA foi capaz de construir desvios com uma pequena perda de mensagens durante esse processo.

As simulações também mostraram que, mesmo nos cenários em que vários nós da rede são comprometidos pela anomalia, o IDEA ainda consegue atingir o seu objetivo. Ainda mais, foi constatado que em alguns casos o GEAR não foi capaz de estabelecer um caminho entre o *sink* e a fonte, enquanto que o IDEA tornou isso possível.

Por fim, também foi avaliado o impacto da proposta em métricas como o atraso e o número de saltos. Dessa forma, foi observado que esse impacto pode ser considerado pequeno em situações nas quais a região de anomalia não é muito grande.

5.2 Contribuições

Nessa seção iremos listar de forma sucinta e direta várias das contribuições que podem ser extraídas desse trabalho.

1. Introdução de uma nova abordagem para isolar anomalias em redes de sensores através do conceito de faixa de isolamento
2. Desenvolvimento de um protocolo geográfico capaz de construir caminhos em torno de uma região de anomalia
3. Introdução do conceito de ganho angular, utilizado para a escolha do próximo salto

4. Criação de um novo modo de eleição do próximo salto, o qual relaciona através de uma média pondera as grandezas distância, ganho angular e energia remanescente.
5. Utilização de trajetórias de referência como guias para a construção dos desvios em torno da região de anomalia
6. Utilização de equações matemáticas para determinar as trajetórias de referência
7. Implementação flexível e modular capaz de permitir a utilização de outros tipos de trajetórias e diferentes regiões de anomalia

5.3 Trabalhos futuros

Um dos pontos positivos do IDEA é a sua potencialidade para trabalhos futuros. Ao longo das nossas pesquisas, vislumbramos diversas variáveis que poderiam ser modificadas e comparadas com as que utilizamos na nossa implementação.

Algumas dessas já foram citadas anteriormente, como a equação das trajetórias de referência e o peso dos coeficientes utilizados no cálculo do custo de um vizinho. Entretanto, outras modificações e experimento que poderiam ser realizados.

Uma dessas seria avaliar o IDEA considerando erros de localização. A maioria das técnicas utilizadas para a determinação da localização dos sensores em uma região são imprecisas. Entretanto, as simulações realizadas nesse trabalho consideram que os nós são capazes de conhecer exatamente os seus posicionamentos. Dessa forma, seria interessante avaliar a influência de erros de localização no desempenho do IDEA.

Tanto a elaboração conceitual como a implementação do IDEA levaram em consideração apenas uma fonte se comunicando com um *sink*. Entretanto, esse é um caso bastante particular e pouco comum nas redes de sensores. Dessa maneira, seria interessante realizar experimentos com mais de uma fonte e mais de um *sink*. Nesse sentido os seguintes casos de testes poderiam ser estudados: múltiplas fontes e um único *sink*; múltiplos *sinks* e uma única fonte; e múltiplas fontes e múltiplos *sinks*

Na seção 3.4.3 foi apresentada a expressão (2) através da qual é calculado o custo para se eleger um vizinho como próximo salto. Um possível trabalho futuro seria ajustar os pesos P_1, P_2 e P_3 para que a distância para a trajetória de referência, ou o ganho angular, ou a energia remanescente tenham uma influência maior ou menor na escolha do próximo salto.

Avaliar o IDEA em redes com várias regiões de anomalia. O IDEA foi desenvolvido levando em consideração um caso simples em que existe apenas uma região de anomalia. Entretanto, uma rede de sensores está sujeita a várias anomalias simultâneas.

Além desses, um estudo comparativo mais detalhado entre diversas trajetórias de referência pode indicar alternativas melhores do que as linhas de força. Porém, uma escolha dinâmica da trajetória de referência pode ser ainda mais interessante. Considere uma rede em que regiões de anomalias de diferentes formatos são detectadas. Assim, quando uma mensagem de interesse chegasse a um bastião, ele saberia o formato da região de anomalia e seria capaz de determinar o tipo de trajetória de referência mais adequado para o formato da região de anomalia em questão. Ou seja, se o bastião estivesse responsável por isolar uma região descrita por uma circunferência, um tipo de trajetória seria adotada. Já se fosse um triângulo, a trajetória de referência a ser utilizada seria outra.

Uma abordagem ainda mais ousada é utilizar diferentes trajetórias de referência para um mesmo tipo de região de anomalia. Ou seja, imagine que a região de anomalia é descrita por uma circunferência. Assim, para o primeiro ponto extremo o bastião adotaria um arco como trajetória de referência. Já para o segundo ponto extremo, poderia ser adotada uma parábola, e assim sucessivamente. Essas abordagens poderiam proporcionar, por exemplo, uma melhor distribuição das trajetórias de referência e uma economia ainda maior do consumo de energia.

Por fim, uma abordagem completamente diferente poderia ser utilizada para as trajetórias de referência. Ao invés de se utilizar linhas contínuas, poderiam ser adotadas linhas poligonais. Essas linhas poligonais poderiam ser uma ampliação do polígono que descreve a região de anomalia e centradas no baricentro da anomalia. Assim, cada trajetória de referência seria determinada por um fator de ampliação.

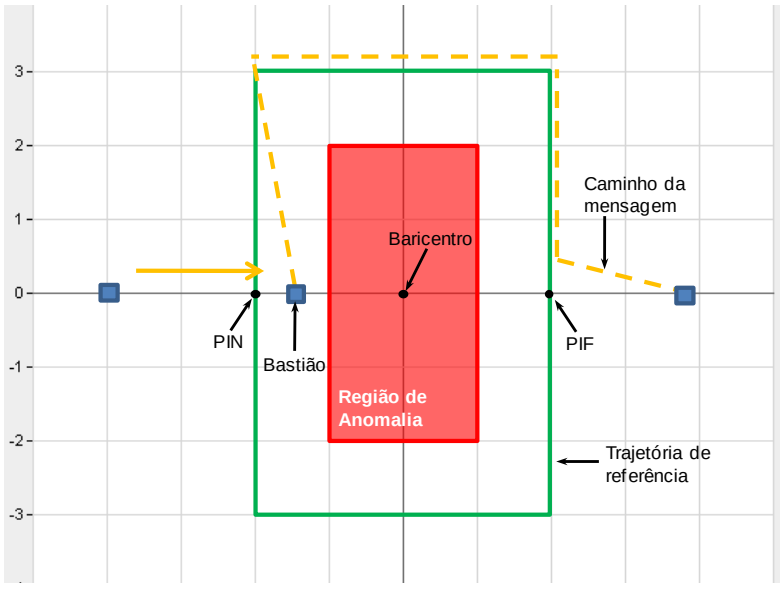


Figura 37 - Trajetória de referência determinada por uma linha poligonal

A Figura 37 ilustra bem esse caso. Repare que a mensagem é enviada de um vértice para outro da trajetória de referência (o retângulo verde) até alcançar o PIF. Além disso, o PIN não coincidiria mais com o bastião e sim a uma distância da região de anomalia determinada pelo fator de ampliação adotado para a trajetória de referência.

Referências

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 38, no. 4, pp. 393-422, Mar 2002.

[2] UC Berkely. (2009, Ago) Campus News. [Online]. http://www.berkley.edu/news/media/releases/2002/08/05_snsor.html

[3] Paulo Goncalves, "Sensor networks," in *Management, Control and Evolution of IP Networks*, Guy Pujolle, Ed. Paris, França: Wiley-ISTE, 2007, ch. 23, pp. 531-552.

[4] Richard R. Brooks and Sundararaja S. Iyengar, *Multi-Sensor Fusion: Fundamentals and Applications with Software*, 1st ed.: Prentice Hall, 1997.

[5] Dragos Niculescu, "Positioning in ad hoc sensor networks," *IEEE Network*, vol. 18, no. 4, pp. 24-29, Jul-Ago 2004.

[6] Dragos Niculescu, Nath, and Badri, "Trajectory based forwarding and its applications," in *International Conference on Mobile Computing and Networking*, Nova Iorque, 2003, pp. 260-272.

[7] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin, "Directed Diffusion: a Scalable and Robust Communication Paradigm for Sensor Networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Boston, 2002, pp. 56-67.

[8] Yan Yu, Ramesh Govindan, and Deborah Estrin, "Geographical and Energy Aware Routing: a recursive data dissemination protocol for wireless sensor networks," Universidade da Califórnia, Los Angeles, Relatório técnico UCLA/CSD-TR-01-0023, 2001.

[9] Ilker Onat and Ali Miri, "A Real-Time Node-Based Traffic Anomaly Detection Algorithm forWireless Sensor Networks," in *Proceedings of the 2005 Systems Communications* , Siena, 2005, pp. 422-427.

[10] Gaurav Gupta and Mohamed Younis, "Fault-Tolerant Clustering of Wireless Sensor Networks," in *IEEE Wireless Communications and Networking (WCNC'03)*, vol. Vol.3, New Orleans, EUA, 2003, pp. 1579- 1584.

[11] Farinaz Koushanfar, Miodrag Potkonjak, and Alberto Sangiovanni-Vincentelli, "Fault tolerance techniques for wireless ad hoc sensor networks," in *Proceedings of IEEE on Sensors*, vol. Vol. 2, Orlando, EUA, 2002, pp. 1491-1496.

[12] Joanne Bechta Dugan and Kishor Trivedi, "Coverage modeling for dependability analysis of fault-tolerantsystems," *IEEE Transactions on Computers*, vol. 38, no. 6,

pp. 775-787, Jun 1989.

[13] Linnyer Beatrys Ruiz et al., "Fault management in event-driven wireless sensor networks," in *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, Veneza, Itália, 2004, pp. 149-156.

[14] Sergio Marti, T J Giuli, Kevin Lai, and Mary Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, Boston, EUA, 2000, pp. 255-265.

[15] Stefano Chessa and Paolo Santi, "Crash faults identification in wireless sensor networks ," *Computer Communications*, pp. 1273-1282, Set 2002.

[16] Yongguo Mei, Changjiu Xian, Saumitra Das, Y. Charlie Hu, and Yung-Hsiang Lu, "Replacing Failed Sensor Nodes by Mobile Robots," in *Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems*, Washington DC, EUA, 2006, pp. 87-92.

[17] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, pp. 11 - 25, Out 2001.

[18] Jessica Staddon, Dirk Balfanz, and Glenn Durfee, "Efficient tracing of failed nodes in sensor networks," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, Atlanta, EUA , 2002, pp. 122 - 130.

[19] Yonggang Jerry Zhao, Ramesh Govindan, and Deborah Estrin, "Residual energy scan for monitoring sensor networks," in *IEEE Wireless Communications and Networking Conference (WCNC2002)*, vol. vol. 1, Orlando, EUA, 2002, pp. 356-362.

[20] Mohamed Younis, Poonam Munshi, and Ehab Al-Shaer, "Architecture for efficient monitoring and management of sensor networks," in *Proceedings of the IFIP/IEEE Workshop on End-to-End Monitoring Techniques and Services (E2EMON'03)*, Belfast, Irlanda do Norte, 2003.

[21] Gayathri Venkataraman, Sabu Emmanuel, and Srikanthan Thambipillai, "A cluster-based approach to fault detection and recovery in wireless sensor networks," in *4th International Symposium on Wireless Communication Systems (ISWCS 2007)*, Trondheim, Noruega, 2007, pp. 35-39.

[22] Jerry Zhao, Ramesh Govindan, and Deborah Estrin, "Computing aggregates for monitoring wireless sensor networks," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, Anchorage, Alasca, 2003, pp. 139- 148.

[23] Tao Yang, Yue Khing Toh, and Lihua Xie, "Run-time Monitoring of Energy Consumption in Wireless Sensor Networks," in *IEEE International Conference on Control and Automation*, 2007, pp. 1360-1365.

[24] Tuan Le, Nadeem Ahmed, Nandan Parameswaran, and Sanjay Jha, "Fault Repair Framework for Mobile Sensor Networks," in *IEEE First International Conference on Communication System Software and Middleware (Comsware 2006)*, Nova Deli, Índia, Jan 2006, pp. 1-8.

[25] Chih-fan Hsin and Mingyan Liu, "A distributed monitoring mechanism for wireless sensor networks," in *Proceedings of the 1st ACM workshop on Wireless security*,

Atlanta, EUA, 2002, pp. 57 - 66.

[26] Kuo-Feng Ssu, Chih-Hsun Chou, Hewijin Christine Jiau, and Wei-Te Hu, "Detection and diagnosis of data inconsistency failures in wireless sensor networks," *The International Journal of Computer and Telecommunications Networking*, pp. 1247-1260 , Jun 2006.

[27] Min Ding, Dechang Chen, Kai Xing, and Xiuzhen Cheng, "Localized fault-tolerant event boundary detection in sensor networks," in *IEEE Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, Miami, Mar 2005, pp. 902-913.

[28] Bhaskar Krishnamachari and Sitharama Iyengar, "Distributed Bayesian algorithms for fault-tolerant event region detection in wireless sensor networks," *IEEE Transactions on Computers*, pp. 241-250, Mar 2004.

[29] Chaiporn Jaikaeo, Chavalit Srisathapornphat, and Chien-Chung Shen, "Diagnosis of Sensor Networks," in *IEEE International Conference on Communications*, Helsinki, Finland, 2001, pp. 1627-1632.

[30] Andreas A. Strikos, "A full approach for Intrusion Detection in Wireless Sensor Networks," School of Information and Communication Technology, Estocolmo, 2007.

[31] Anthony D. Wood and John A. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, no. 10, pp. 54-62, Out 2002.

[32] Anthony D. Wood and John A. Stankovic, "A Taxonomy for Denial-of-Service Attacks in Wireless Sensor Networks," in *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, Laurie Kelly, Mohammad Ilyas, and Imad Mahgoub, Eds.: CRC Press, 2004, ch. 32, pp. 1-17.

[33] Chris Karlof and David Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures ," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, Anchorage, 2003, pp. 113-127.

[34] Anthony D. Wood, John A. Stankovic, and Sang H. Son, "JAM: A Jammed-Area Mapping Service for Sensor Networks," in *Proceedings of the 24th IEEE International Real-Time Systems Symposium* , Cancun, 2003, p. 286.

[35] Mario Cagalj, Srdjan Capkun, and Jean-Pierre Hubaux, "Wormhole-Based Anti-Jamming Techniques in Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 1, pp. 100-114, Jan 2007.

[36] Adrian Perrig, Robert Szewczyk, J.D. Tygar, Victor Wen, and David E. Culler, "SPINS: Security Protocols for Sensor Networks," in *Proceedings of the 7th annual international conference on mobile computing and networking*, Roma, 2001, pp. 189-199.

[37] Chris Karlof, Naveen Sastry, and David Wagner, "TinySec: a link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, Baltimore, 2004, pp. 162-175.

[38] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig, "The sybil attack in sensor networks: analysis & defenses," in *Proceedings of the 3rd international symposium on Information processing in sensor networks*, Berkeley, 2004, pp. 259-268.

[39] Zdravko Karakehayov, "Using REWARD to Detect Team Black-Hole Attacks in Wireless Sensor Networks," in *Workshop on Real-World Wireless Sensor Networks*

(REALWSN'05), Estocolmo, 2005, pp. 20-21.

[40] Haowen Chan, Adrian Perrig, and Dawn Song, "Random key predistribution schemes for sensor networks," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy* , California, 2003, p. 197.

[41] Laurent Eschenauer and Virgil D. Gligor, "A key management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications security*, Washington, 2002, pp. 41-47.

[42] Wenliang Du et al., "A pairwise key pre-distribution scheme for wireless sensor networks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 8, no. 2, pp. 228-258, Mai 2003.

[43] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia, "LEAP+: Efficient security mechanisms for large-scale distributed sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 2, no. 4, pp. 500-528, Nov 2006.

[44] Nadeem Ahmed, Salil S. Kanhere, and Sanjay Jha, "The holes problem in wireless sensor networks: a survey," *Mobile Computing and Communications Review* , vol. 9, no. 2, pp. 4-18, Abr 2005.

[45] Qing Fang, Jie Gao, and Leonidas J. Guibas, "Locating and bypassing holes in sensor networks," *Mobile Networks and Applications*, vol. 11, no. 2, pp. 187-200, Abr 2006.

[46] Weijia Jia, Tian Wang, Guojun Wang, and Minyi Guo, "Hole Avoiding in Advance Routing in Wireless Sensor Networks," in *Wireless Communications and Networking Conference*, Sidiney, 2007, pp. 3519-3523.

[47] Brad Karp and H. T Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, Massachusetts, 2000, pp. 243-254.

[48] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger, "Geometric ad-hoc routing: of theory and practice," in *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, Boston, 2003, pp. 63-72.

[49] Ben Leong, Sayan Mitra, and Barbara Liskov, "Path vector face routing: geographic routing with local face information," in *IEEE International Conference on Network Protocols*, Boston, 2005, pp. 12 pp.-.

[50] Hannes Frey and Ivan Stojmenovic, "On delivery guarantees of face and combined greedy-face routing in ad hoc and sensor networks," in *International Conference on Mobile Computing and Networking*, Los Angeles, 2006, pp. 390-401.

[51] Fucai Yu, Soochang Park, Ye Tian, Minsuk Jin, and Sang-Ha Kim, "Efficient Hole Detour Scheme for Geographic Routing in Wireless Sensor Networks," in *IEEE Vehicular Technology Conference*, Calgary, Alberta, 2008, pp. 153-157.

[52] Ye Tian et al., "Energy-Efficient Data Dissemination Protocol for Detouring Routing Holes in Wireless Sensor Networks," in *IEEE International Conference on Communications*, Pequim, 2008, pp. 19-23.

[53] Yingqi Xu and Wang-chien Lee, "PSGR: priority-based stateless geo-routing in wireless sensor networks," in *IEEE Conference in Mobile Ad-hoc and Sensor Systems*, 2005, pp. 7-10.

[54] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva, "Directed Diffusion for Wireless Sensor Networking," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 2-16, Fev 2003.

[55] C.H. See, R.A. Abd-Alhameed, Y.F. Hu, and K.V. Horoshenkov, "Wireless sensor transmission range measurement within the ground level," in *Antennas and Propagation Conference*, Loughborough, 2008, pp. 225-228.

[56] Texas Instruments. Cc2420: 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver. [Online]. http://focus.ti.com/lit/ds/symlink/cc2420.pdf

[57] Muresan and Dan Alin, "Electrostatic fields of point charges studied on the computer," in *Scientific Communications Session*, Bucareste, Maio, 1996.

[58] Himanshu Gupta, Samir R. Das, and Quinyi Gu, "Connected Sensor Cover: Self-Organization of Sensor Networks for Efficient Query Execution," *IEEE/ACM Transactions on Networking*, vol. 14, no. 1, pp. 55-67, February 2006.

[59] The Network Simulator - ns-2. [Online]. http://www.isi.edu/nsnam/ns

[60] Fabio Silva, John Heidemann, and Ramesh Govindan, "Network Routing Application Programmer's Interface (API) and Walk Through 9.0.1," Universidade da Califórnia do Sul, California, 2002.

[61] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward, "A distance routing effect algorithm for mobility (DREAM)," in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, Dalas, 1998, pp. 76-84.

[62] Young-Bae Ko and Nitin H. Vaidya, "Location aided routing (LAR) in mobile ad hoc networks," *Wireless Networks*, vol. VI, no. 4, pp. 307-321, Jul 2000.

[63] Mahesh K. Marina and Samir R. Das, "On-Demand Multi Path Distance Vector Routing in Ad Hoc Networks," in *Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP)*, California, 2001, pp. 14-23.

[64] Mehdi Kalantari and Mark Shayman, "Design Optimization of Multi-Sink Sensor Networks by Analogy to Electrostatic Theory," in *IEEE Wireless Communications and Networking Conference*, Las Vegas, 2006, pp. 431-438.

[65] Nam T. Nguyen, An-I Andy Wang, Peter Reiher, and Geoff Kuenning, "Electric-field-based routing: a reliable framework for routing in MANETs," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, no. 2, pp. 35-49, Abr 2004.

[66] Mehdi Kalantari and Mark Shayman, "Energy Efficient Routing in Wireless Sensor Networks," in *Conference on Information Sciences and System*, Princeton, 2004.

[67] S. Toumpis, "Mother nature knows best: A survey of recent results on wireless networks based on analogies with physics," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 52, no. 2, pp. 360-383, Fev 2008.

[68] S. Toumpis, "Optimal design and operation of massively dense wireless networks: or how to solve 21st century problems using 19th century mathematics," in *Proceedings from the 2006 workshop on Interdisciplinary systems approach in performance evaluation and design of computer & communications sytems*, Pisa, 2006.

[69] S Toumpis and L Tassiulas, "Packetostatics: deployment of massively dense sensor networks as an electrostatics problem," in *Proceedings of the 24th IEEE Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Miami, 2005, pp. 2290-2301.

[70] Mehdi Kalantari and Mark Shayman, "Routing in Wireless Ad Hoc Networks by Analogy to Electrostatic Theory," in *IEEE International Conference on*

Communications, Paris, 2004, pp. 4028-4033.

[71] Mehdi Kalantari and Mark Shayman, "Routing in Multi-Commodity Sensor Networks Based on Partial Differential Equations," in *40th Annual Conference on Information Sciences and Systems*, Princeton, 2006, pp. 402-406.

[72] J. M. Kahn, R. H. Katz, and K. S. J. Pister, "Next Century Challenges: Mobile Networking for Smart Dust," in *Proceedings of the ACM/IEEE international conference on Mobile computing and networking*, Washington, 1999, p. 271–78.

[73] Curt Schurgers, Vlasios Tsiatsis, Saurabh Ganeriwal, and Mani Srivastava, "Optimizing sensor networks in the energy-latency-density design space," *IEEE Transactions on Mobile Computing*, vol. 1, no. 1, pp. 70-80, Jan-Mar 2002.

[74] Crossbow Technology. (2009, Ago) Crossbow Technology : Wireless Home Page. [Online]. http://www.xbow.com/Home/wHomePage.aspx

[75] Tia Gao, Dan Greenspan, Matt Welsh, Radford R. Juang, and Alex Alm, "Vital Signs Monitoring and Patient Tracking Over a Wireless Network," in *International Conference of the Engineering in Medicine and Biology Society*, Xangai, 2006, pp. 102-105.

[76] Jamal N. Al-Karaki and Ahmed E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 6-28, Dez 2004.

[77] Matthias Handy, Haase Marc, and Dirk Timmermann, "Low energy adaptive clustering hierarchy with deterministic cluster-head selection," in *International Workshop on Mobile and Wireless Communications Network*, 2002, pp. 368-372.

APÊNDICE

A

O Directed Diffusion

Dentre os diversos protocolos de roteamento plano para redes de sensores, o Directed Diffusion [7] se destaca pelo seu modelo *data centric* inovador. A idéia principal é difundir esquemas de dados pela rede a fim de se obter os dados relativos a esse esquema. A utilização de esquemas de dados foi desenvolvida principalmente para minimizar o consumo de recursos com operações desnecessárias. Assim, todos os nós em uma rede baseada em Directed Diffusion são orientados a aplicação. Isso possibilita poupar energia selecionando empiricamente bons caminhos através de *chaching* e processamento de dados da rede.

No DD uma *query* de um operador humano é transformada em uma **mensagem de interesse**, a qual é difundida pela rede. Uma *query* pode ser algo do tipo: "Quantos carros você observou passar em uma região x,y?". Assim, quando um nó (fonte) de uma região x,y recebe uma mensagem de interesse, ele ativa os seus captadores para coletar as informações sobre a *query* especificada e enviar os dados coletados de volta para o nó (*sink*) que elaborou a *query*. Além disso, no DD, nós intermediários podem realizar agregação de dados combinando respostas de vários sensores. Além de economizar energia, a agregação de dados também pode ocorrer com o objetivo de fornecer respostas mais precisas sobre a *query*. Por exemplo, através de um processo de a agregação de dados, a resposta de vários sensores podem ser combinadas para informar com maior precisão a quantidade de carros que passaram na região x,y.

Uma das características mais importantes do DD é a de que tanto a difusão de interesses, como a propagação de dados e a agregação ocorrem de forma localizada. Ou seja, os sensores não armazenam o estado da rede como um todo. Todas as decisões sobre o encaminhamento das mensagens são tomadas com base apenas nas informações do próprio nó e de seus vizinhos diretos.

Esquemas de nomes

Selecionar um esquema de nomeação é o primeiro passo em planejar o DD para a rede. Ou seja, cada rede deve ter o seu próprio esquema de nomeação. Isso é feito através de um esquema atributo-valor capaz de descrever o interesse. Aproveitando o exemplo anterior, um possível esquema de nomeação seria:

- Veículo = carro
- Intervalo = 0,5
- Duração = 20
- Região = [50;30;10]

Esse esquema de nomeação poderia ter o seguinte significado: deseja-se observar carros passando em uma região circular centrada no ponto (50,30) e com raio de 10 metros. Além disso, deseja-se que os dados sejam transmitidos a cada 0,5 segundos e durante 20 minutos.

Ou seja, o esquema de nomeação define quais atributos serão utilizados. No exemplo em questão, os atributos são: veículo, intervalo, duração e região. Assim, antes do sensor enviar uma mensagem de interesse para a rede ele deverá atribuir um valor para cada um desses atributos.

Interesses e gradientes

Os interesses podem ser originados em qualquer um dos nós da rede, geralmente através de uma tarefa programada por uma pessoa. O nó que origina o interesse é chamado de *sink* e ele recebe os dados coletados pelos nós da fonte através da transdução dos sensores.

Quando um *sink* é incumbido de uma tarefa, ele difunde na rede (daí o nome *Diffusion*) mensagens de interesse. O interesse inicial pode ser pensado como exploratório, pois ele irá tentar encontrar nós que possam fornecer os dados descritos através de seus atributos. Diferentes técnicas podem ser utilizadas para a difusão dos na rede. Duas bastante estudadas são a de *flooding* e a de encaminhamento geográfico. Na primeira, as mensagens são propagadas através de broadcasts consecutivos. Já na segunda, algum sistema de coordenadas é utilizado para direcionar as mensagens diretamente para a fonte. Essa segunda técnica foi utilizada, por exemplo, pelo GEAR, o qual será analisado mais detalhadamente na sessão seguinte.

Quando um nó recebe um interesse, ele verifica se esse o mesmo já está na sua *cache*. Caso isso não se confirme, um gradiente será criado nesse nó. O gradiente armazena as informações contidas no interesse (atributos e respectivos valores) e o nó através do qual esse interesse foi recebido. Dependendo do método adotado para a difusão das mensagens de interesse, um nó poderá conter mais de um gradiente. Se um

mesmo interesse for recebido a partir de diferentes vizinhos, para cada um desses vizinhos será criado um gradiente.

Cada gradiente tem um tempo de vida útil, o qual pode ser atualizado através do recebimento de um novo interesse vindo do mesmo vizinho. A freqüência com que os interesses são enviados e a duração de um gradiente na *cache* de um nós são parâmetros configuráveis no DD.

Essa operação se repete até que o interesse alcance a fonte. Quando isso acontece, essa também registra um interesse, porém, além disso, ela envia uma **mensagem de dados exploratória** através de todos os gradientes registrados em sua *cache*. Esse tipo de mensagem, como o próprio nome indica, já contém dados sobre o evento observado. Porém, ela também serve para determinar o melhor caminho através do qual as demais mensagens de dados serão transmitidas.

Reforço positivo

A fim de determinar o melhor caminho através do qual as mensagens de dados serão transmitidas, cada nó intermediário que receber uma mensagem dados exploratória irá procurar na sua *cache* quais gradientes estão associados ao tipo de dado contido nessa mensagem. Em seguida, ele irá enviar uma cópia dessa para cada um dos gradientes encontrados.

Assim, uma ou mais mensagens de dados exploratórias serão encaminhadas de volta ao *sink*. Se esse nó receber mais de uma dessas mensagens, ele irá registrar apenas a primeira recebida e descartar as demais. Essa foi a forma encontrada para determinar o melhor caminho entre a fonte e o *sink*.

Logo na seqüência, o *sink* irá reforçar esse caminho. Para isso, ele envia uma **mensagem de reforço de caminho** para o nó o qual encaminhou a mensagem de dados exploratória registrada (ou seja, aquela que foi primeiramente recebida). Esse nó, por sua vez, irá encaminhá-la para o nó do qual ele recebeu essa mesma mensagem de dados exploratória e assim sucessivamente até que a mensagem de reforço seja recebida pela fonte.

Em cada nó pelo qual a mensagem de reforço passar, o respectivo gradiente receberá o status de reforçado. A partir desse momento, quando a fonte enviar uma mensagem de dados, o nó que a receber irá procurar dentre os seus gradientes aquele que possui o status de reforçado e encaminhará para o mesmo a mensagem de dados recebida.

A fim de manter ativas as rotas previamente definidas, o sink difunde na rede novas mensagens de interesse. Assim, todo processo de envio de mensagens

exploratórias de dados e de reforço de caminho é realizado novamente. A freqüência com que isso ocorre são parâmetros configuráveis do Direted Diffusion.

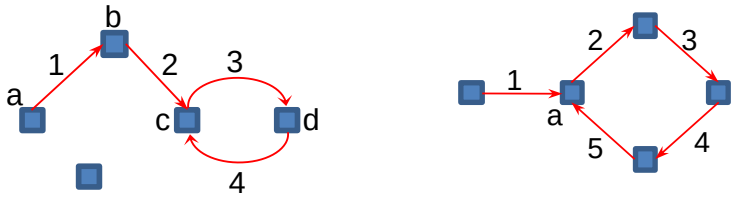
Reforço negativo

Se o caminho inicialmente reforçado se degradar (ou for interrompido), provavelmente as novas mensagens de dados exploratórias enviadas através desse caminho, e que antes eram recebidas primeiramente, serão descartadas em detrimento das outras mensagens que chegarão previamente. Quando isso acontece, um novo caminho será reforçado e a fonte passará a enviar dados por dois caminhos distintos.

Ao perceber esse fato, o *sink* envia uma **mensagem de reforço negativo** para o caminho através do qual os dados são recebidos com uma menor vazão. Assim, todos os nós desse caminho irão remover dos gradientes associados ao mesmo o status de reforçado. Conseqüentemente, a fonte passará e enviar os dados por apenas um único caminho.

Remoção de loops

A fim de evitar loops de roteamento, o DD adota um mecanismo através do qual cada nó registra em sua *cache* as mensagens por ele encaminhadas. Ao receber novamente uma mesma mensagem, o sensor simplesmente a descarta. A **Erro! Fonte de referência não encontrada.**(a) ilustra essa situação. Uma mensagem é encaminhada do nó a até o nó d. Entretanto, o único vizinho desse último nó é c. Assim, a mensagem é descartada quando ela é encaminhada de volta para c.



(a) **(b)**

Figura 38 - Detecção de loops realizada pelo GEAR

Além desse caso extremo, em que um nó só possui como vizinho o último salto, há outras situações em que uma mensagem poderá efetivamente retornar a um nó e ser descartada. Normalmente isso acontece quando há na rede uma região de baixa densidade. Esse caso pode ser observado através da **Erro! Fonte de referência não encontrada.**(b). Nessa, a mensagem é descartada pelo nó a, após entrar em uma região na qual ela só possui um único vizinho fora ao salto anterior. Assim, considerando que

a só está ao alcance de b e d, a mensagem será obrigada a seguir os saltos 1, 2, 3, 4 e 5. Ao chegar em a novamente, ela será descartada.

APÊNDICE

B

O GEAR

O *Geographical and Energy Aware Routing* (GEAR) é um protocolo que foi desenvolvido para tornar mais eficiente o processo de difusão das mensagens de interesse pelo DD. O GEAR foi concebido a partir das restrições de energia que recaem sobre os sensores. Assim, a idéia principal é restringir o número de mensagens necessárias para que o *sink* encontre a fonte. Dessa forma, esse protocolo atua meramente no processo de propagação de interesses do DD.

O GEAR apóia-se no fato de que é cada vez maior o número de técnicas capazes de fornecer a localização geográfica de um dispositivo em um plano. No caso do trabalho em questão, seus autores citam o GPS como uma dessas possíveis técnicas. Assim, eles adotam um mecanismo de encaminhamento geográfico para levar os interesses do *sink* até a fonte.

Nas redes em que o GEAR é utilizado, cada um dos seus nós envia periodicamente uma mensagem de requisição (um *beacon*) através de *broadcast* contendo as suas coordenadas geográficas e a sua energia remanescente. Os nós que receberem essa requisição respondem através de *unicast* com as suas informações de localização e energia. Assim, o GEAR cria uma lista de vizinhos com os respectivos dados sobre o posicionamento e a energia restante de cada um.

Essas informações são armazenadas em *cache* e têm um tempo de vida útil restrito. Se após certo período o GEAR não receber nenhuma mensagem contendo informações atualizadas sobre um dos nós já registrados na sua *cache*, a respectiva entrada será eliminada da sua lista de vizinhos.

Determinação do próximo salto

Quando uma mensagem de interesse é recebida em um nó, a seguinte regra é aplicada para a eleição do próximo salto:

a) Quando há um ou mais vizinhos mais próximos da fonte, será eleito aquele que possuir a menor distância para a mesma.
b) Quando todos os vizinhos estão mais afastados, será eleito aquele que possuir o menor custo. Onde esse custo é determinado por dois tipos de custos: um estimado e outro aprendido.

O custo estimado de um nó é calculado a partir da sua energia restante e da sua distância para o centro da região de interesse. Já o custo aprendido representa o custo para se chegar nessa região através de um vizinho específico. Dessa forma, em um nó há um custo aprendido para cada um dos seus vizinhos. Porém, o cálculo do custo aprendido para um vizinho é iniciado através de uma consulta direta ao mesmo sobre o seu custo aprendido para chegar à fonte. Em seguida, o nó adiciona a esse valor o seu custo estimado e armazena o resultado como o novo custo aprendido para esse vizinho. Assim, o vizinho que possuir o menor custo aprendido será eleito como próximo salto.

No caso inicial em que o valor do custo aprendido ainda não está disponível, o custo estimado é utilizado.

APÊNDICE

C

Modelagem do IDEA no ns-2

A fim de avaliarmos o funcionamento do IDEA, nós realizamos uma implementação do mesmo na versão 2.32 do simulador ns-2 [59]. O ns-2 é um simulador de redes escrito em C++ e que utiliza scripts TCL para descrever e controlar os ambientes de simulação. A implementação do IDEA, por sua vez, é um conjunto de classes (também escritas em C++) as quais interagem com as classes no ns-2.

Complementarmente à nossa implementação, adotamos os códigos do Directed Diffusion e do GEAR (seção **Erro! Fonte de referência não encontrada.**), disponíveis ns-2, como os protocolos de roteamento da rede. Essa escolha foi motivada pelo fato desses dois protocolos atenderem à maioria dos requisitos necessários para o funcionamento do IDEA (seção **Erro! Fonte de referência não encontrada.**).

Dessa forma, nesse capítulo iremos descrever primeiramente alguns aspectos do DD no ns-2 e como o GEAR e o IDEA se integram ao mesmo. Na seção seguinte, serão apresentadas as classes de C++ relacionadas com o desenvolvimento do nosso protocolo e o papel desempenhado por cada uma. Destaque para o diagrama em UML através do qual será possível observar o relacionamento dessas classes. Logo depois, descreveremos seqüencialmente o funcionamento do IDEA desde o momento em que a rede é inicializada, passando pela detecção de uma anomalia, até o instante final em que os desvios são construídos. Nessa descrição serão utilizados os nomes dos métodos e atributos das classes do ns-2 para que o leitor possa ter uma noção mais concreta da implementação.

A integração do IDEA com o DD e o GEAR no ns-2

A versão 3.2 do DD no ns-2 foi desenvolvida de forma a facilitar a implementação e integração de novos módulos com o seu núcleo de roteamento. Esses módulos são

também conhecidos como filtros. Quando uma mensagem é recebida em um nó, ela é encaminhada seqüencialmente para cada um dos filtros existentes. Um filtro pode modificar, descartar ou apenas analisar e repassar a mensagem para o próximo filtro. Após passar por todos os filtros, a mensagem é então encaminhada de volta para a rede (ver Figura 39). O GEAR, por exemplo, possui uma implementação na versão 2.32 do ns-2 a qual foi realizada através de dois filtros do Directed Diffusion.

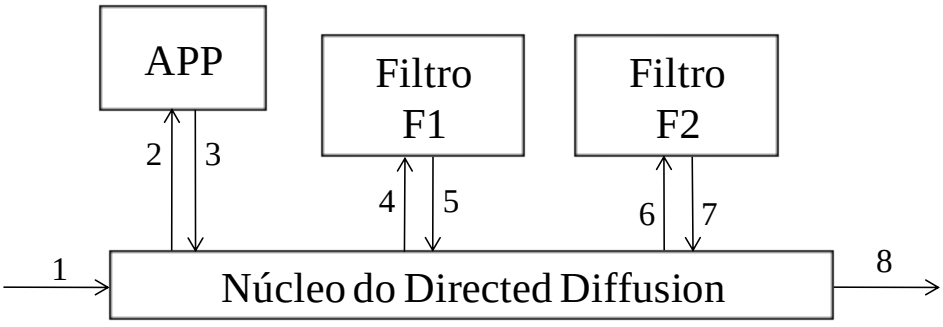


Figura 39 - Esquema de filtros adotado pelo Directed Diffusion

Cada um desses filtros é executado de acordo uma ordem de prioridade, a qual é definida por um atributo numérico. Quanto maior o valor desse atributo, maior será a prioridade de execução do filtro. A determinação de quais filtros serão utilizados em uma simulação é configurada no script TCL.

Assim como o GEAR, o IDEA foi implementado como dois filtros do Directed Diffusion. A Figura 40 ilustra a ordem de encaminhamento de uma mensagem entre os filtros utilizados nas nossas simulações. O primeiro filtro do IDEA, também chamado de pré-filtro (passo 4), é responsável por executar o algoritmo que atua na recepção das mensagens (ver seção 3.1). Assim, se uma mensagem vier núcleo de uma anomalia, por exemplo, ela será descartada no pré-filtro do IDEA e os demais filtros nada saberão sobre a mesma. Caso contrário, a mensagem será encaminhada para o pré-filtro do GEAR (passo 6) e, em seguida, para o *GradientFilter* (passo 8). Dentre as várias funções desempenhadas por esse último filtro, podemos destacar: a criação e manutenção dos gradientes, o gerenciamento das tabelas de encaminhamento e o envio periódico de mensagens de interesse, de dados exploratória e de reforço positivo. Ou seja, ele é o filtro incumbido de realizar as operações de roteamento do DD.

Após esse processamento, a mensagem retorna ao GEAR (passo 10) para que o pós-filtro do mesmo possa eleger o próximo salto. Por fim, a mensagem é repassada para o pós-filtro do IDEA (passo 13). Nesse ponto, é executado o algoritmo que impede o envio das mensagens pra dentro da região de anomalia (ver seção 3.1). Além disso, todas as rotinas definidas na seção 3.4, ou seja, aquelas compreendidas desde a decisão

de se iniciar um desvio até o momento de finalizá-lo, também são processadas quando a mensagem é recebida por esse pós-filtro.

Apesar do IDEA e do GEAR serem filtros distintos, ambos dependem das informações sobre a localização e a energia restante dos seus vizinhos. Contudo, o GEAR possui todos os métodos necessários para obter essas informações, armazená-las e mantê-las atualizadas. Dessa forma, não faria sentido o IDEA possuir funções e estruturas de dados replicados. Isso apenas consumiria os já escassos recursos de memória e processamento. Também é importante lembrar que, se esses dois filtros fossem implementados em algum *hardware*, eles compartilhariam a mesma memória física. Dessa forma, na inicialização das nossas simulações, a instância do IDEA em cada sensor recebe uma referência para respectiva instância do GEAR nesse mesmo sensor. Assim, toda vez que o IDEA necessita de informações sobre o posicionamento ou a energia restante de algum nó, ele invoca os métodos do GEAR capazes de fornecê-las.

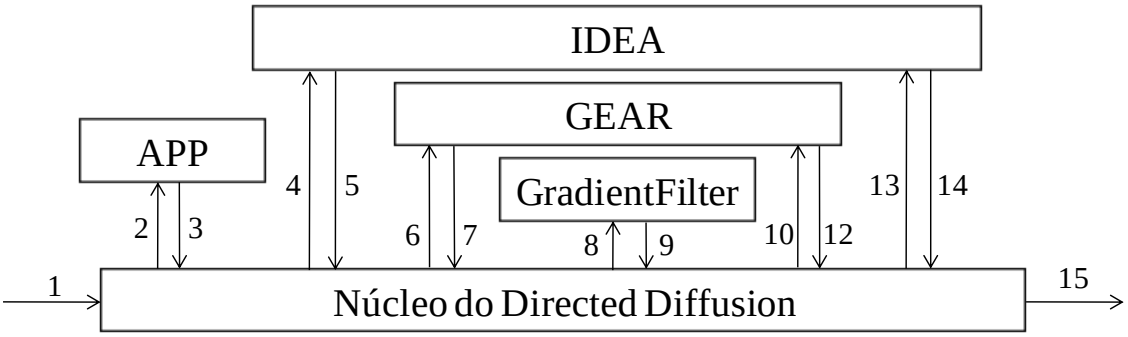


Figura 40 - Interação do filtro do IDEA com os demais filtros do Directed Diffusion

Contudo, a implementação do GEAR na versão 2.32 do ns-2 é incompleta e incapaz de simular o consumo de energia dos nós. Dessa forma, foi necessário realizarmos algumas modificações nesse protocolo para que ele passasse a utilizar a classe *EnergyModel* do ns-2. Dentre essas modificações, podemos destacar a inclusão do atributo *energy_model_* e a implementação do método *getRemainingEnergy* na classe *GeoRoutingFilter*, como veremos na próxima seção. Entretanto, para que o *EnergyModel* fosse utilizado, foi necessário configurarmos os scripts em TCL para que cada sensor recebesse uma instância do mesmo.

Toda vez que um nó realiza uma transmissão, uma recepção ou entra em um estado de inatividade (*idle*), a classe *EnergyModel* percebe cada um desses eventos e decrementa do total de energia restante do sensor um valor associado ao consumo de energia para cada um desses estados. Os valores da energia inicial e da quantidade de energia consumida durante a transmissão, a recepção e a inatividade também são

configurados no script TCL. Os valores adotados nas nossas simulações serão apresentados na seção 4.1.

Ainda sobre a implementação do DD no ns-2, é importante comentar que os seus desenvolvedores criaram uma API [60] através da qual é possível utilizar a classe *NRSimpleAttributeFactory* para cria objetos do tipo *NRSimpleAttribute*. No DD, esses objetos são utilizados para se definir o esquema de nomeação da rede. Ou seja, cada *NRSimpleAttribute* e o seu respectivo valor definem um atributo-valor que é utilizado para descrever os interesses (ver seção **Erro! Fonte de referência não encontrada.**). Além disso, esses objetos também são utilizados para armazenar e retornar os dados da fonte para o *sink*.

No caso do IDEA, os seguintes *NRSimpleAttribute* foram criados para auxiliar na troca de mensagens desse protocolo:

- *IdeaMsgTypeAttr*: armazena um identificador do tipo da mensagem. Nesse trabalho, o IDEA possui um único tipo de mensagem: a REQD, cujo identificador é a constante *DETOUR_REQUEST*.
- *IdeaAnomalyRegionAttr*: armazena uma cópia do *anomalyRegion_*.
- *IdeaPINLongitudeAttr*: armazena o valor da coordenada x do PIN
- *IdeaPINLatitudeAttr*: armazena o valor da coordenada y do PIN
- *IdeaPIFLongitudeAttr*: armazena o valor da coordenada x do PIF
- *IdeaPIFLatitudeAttr*: armazena o valor da coordenada y do PIF
- *IdeaDeterminantsAttr*: armazena um *array* do tipo *double* onde cada determinante da trajetória de referência é inserido.

O armazenamento dos determinantes em um *array* possibilita que, independente da quantidade de elementos necessários para a determinação da trajetória de referência, esses possam ser passados de um nó para o outro sem a necessidade de se modificar a estrutura da mensagem utilizada para isso. Por exemplo, considere que um tipo de equação Z foi escolhido para a construção dos desvios. E que a determinação das trajetórias definidas por Z necessita dos elementos a, b, c,..., n. Para que um nó possa passar os valores desses elementos para um dos seus vizinhos, é necessário programar uma ordem de inserção de cada um desses valores no *array* de determinantes. Dessa forma, uma possibilidade para o exemplo em questão, seria incluir o valor de a na posição 0 do *array*, o de b na posição 1, e assim sucessivamente, até o do elemento n, cujo valor seria incluído na posição $n - 1$. Quando outro nó da rede receber esse *array* ele irá extrair o valor do elemento a da posição 0, o do elemento b da posião 1 e assim por diante. Essa abordagem é bastante flexível, porém, para todo tipo de trajetória que

for implementada, será necessário definir uma ordem de inserção dos seus determinantes no *array*.

É importante observar que as coordenadas do PIN e do PIF, por serem necessárias em várias operações, não são incluídas no *array* de determinantes. Ao invés disso, elas são declaradas como *NRSimpleAttribute*. Dessa forma, é possível extraí-las da mensagem de uma forma mais simples e direta do que se elas estivessem no *array* de determinantes.

As classes do IDEA

Nessa seção, serão detalhadas as classes desenvolvidas em C++ para as simulações do IDEA no ns-2. Além disso, também serão apresentadas, mas de forma resumida, algumas classes do GEAR e do Directed Diffusion com as quais o IDEA se relaciona.

A classe mais importante do IDEA é o *IDEAFilter*. Nessa foram implementados todos os métodos necessários para o isolamento das regiões de anomalia e a construção dos desvios. O GEAR possui a classe *GeoRoutingFilter*. Ela é responsável por descobrir e manter atualizadas as informações sobre localização e energia restante dos vizinhos, além de realizar o encaminhamento geográfico. Enquanto isso, a classe *GradientFilter* executa as principais funções de roteamento do Directed Diffusion, como a manutenção dos gradientes, o reforço dos melhores caminhos e o envio periódico das mensagens de interesse. A classe *DiffusionRouting* é o núcleo do Directed Diffusion. Ela possui todos os métodos necessários para gerenciar os filtros que serão acoplados a cada sensor. Por fim, as classes *GearSenderApp* e *GearReceiverApp* são implementações no nível de aplicação para o envio e recebimento das mensagens de dados.

Visando um código fonte bem estruturado, desenvolvemos algumas classes para a execução de operações auxiliares e/ou representação de estruturas de dados. Uma dessas classes foi a *AnomalyRegion*. Ela é responsável por armazenar as dimensões e localização da região de anomalia, determinar se um dado ponto está dentro dessa região e definir o posicionamento do PIF. Entretanto, para que essas duas últimas funções possam ser executadas corretamente, a *AnomalyRegion* tem que ser informada sobre a geometria da região de anomalia que ela irá armazenar. Nesse trabalho, a geometria adotada para a região de anomalia é igual à de um círculo. Entretanto, a fim de permitirmos que em trabalhos futuros a anomalia possa ser representada por qualquer polígono (ver seção **Erro! Fonte de referência não encontrada.**), nós desenvolvemos a classe virtual *Polygon*.

Dessa forma, é através do atributo *polygon_* que a classe *AnomalyRegion* armazena a região de anomalia. O atributo *polygon_*, por sua vez, é do tipo *Polygon*, o

qual possui dois métodos virtuais: o *isInside* e o *calcPIF*. Assim, quando *polygon_* é inicializado, ele deverá receber uma classe que herde de *Polygon* e implemente esses dois métodos de acordo com as suas características geométricas. Nesse trabalho, a classe a classe *Circle* foi desenvolvida justamente para atender a esses requisitos. Por fim, também desenvolvemos a classe *Point*, através da qual o IDEA pode representar qualquer coordenada geográfica e calcular a distância dessa coordenada para outra através do método *calcDistance*.

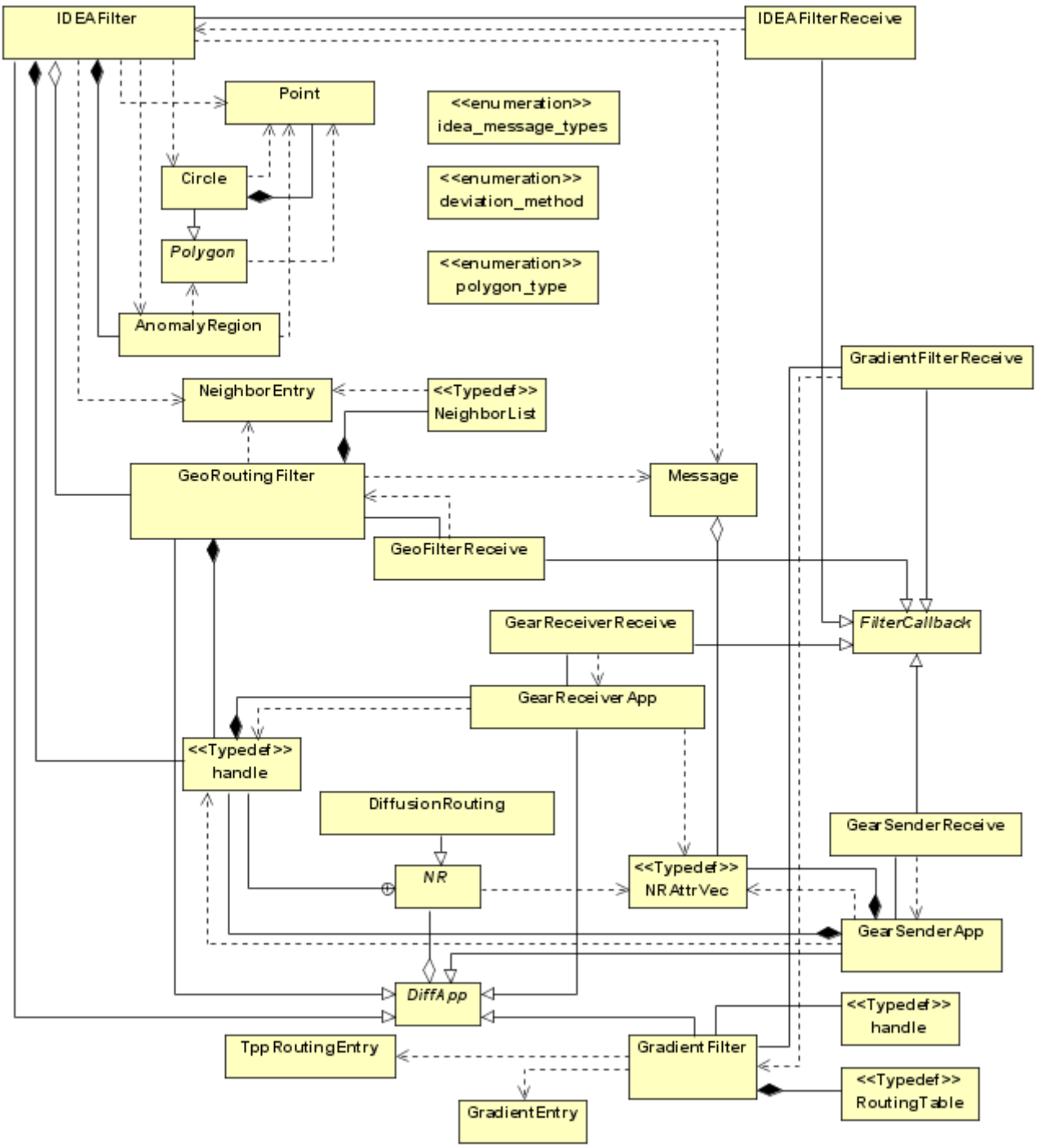


Figura 41 - Diagrama de relacionamento das classes do IDEA

A Figura 41 é um diagrama de classes simplificado em UML, através do qual é possível visualizar o relacionamento das principais classes utilizadas pelo IDEA. Além desse diagrama, no Apêndice A são apresentadas as classes do IDEA com todos os seus atributos, métodos e parâmetros. Algumas classes do GEAR e do Directed Diffusion também podem ser observadas nesse mesmo apêndice. Entretanto, nesses dois últimos casos, listamos resumidamente os atributos e métodos, a fim de destacar apenas os mais relevantes para a compreensão do funcionamento do IDEA.

Execução seqüencial do IDEA no ns-2

Nessa seção, nós iremos descrever a execução do IDEA no ns-2. Ou seja, iremos apresentar a seqüência de métodos invocados desde o momento em que uma rede é iniciada até o momento em que os desvios em torno de uma região de anomalia são construídos.

Assim, considere uma rede de sensores cujos nós foram configurados para utilizar o Directed Diffusion, o GEAR e o IDEA. Quando a rede é iniciada, para cada nó é criada uma instância das classes *DiffusionRouting*, *GradientFilter*, *GeoRoutingFilter* e *IDEAFilter*. Nesse momento, várias opções de configuração, declaradas no script TCL, são utilizadas para inicializar alguns dos atributos dessas classes. Por exemplo, no caso do *IDEAFilter*, há o atributo *detourMethod_* o qual é inicializado com um valor especificado no script TCL.

Em seguida, cada um dos filtros invoca os seus respectivos métodos para se registrarem no núcleo do DD. No caso do IDEA, os métodos *setupPreFilter* e *setupPostFilter*. Cada um desses instanciará um *FilterCallback* e um *handle*. O primeiro é acionado toda vez que uma mensagem é recebida do núcleo do DD. Enquanto que o segundo é utilizado para devolver a mensagens ao núcleo do DD.

Nos nós que desempenharão o papel da fonte e do *sink*, é criada uma instância da classe *GearSenderApp* e uma da classe *GearReceiverApp* respectivamente. Nessa primeira, o tipo de evento observado pela fonte é configurado através do método *setupPublication*. Já na segunda, a operação de *setupSubscription* é utilizada para configurar o interesse do *sink*. Tanto a operação de *setupPublication* como a de *setupSubscription* retornam um *handle*.

Após essa faze de inicialização da rede, várias mensagens de descoberta de vizinhos são trocadas entre os *GeoRoutingFilters*. Para cada vizinho descoberto, é criado um objeto *NeighborEntry* com a localização e a energia restante desse vizinho. Então, o *GradientFilter* do *sink* passa a enviar periodicamente mensagens de interesse para a rede. Cada uma dessas mensagens é do tipo *Message*, a qual o atributo

*msg_attr_vec_* do tipo *NRAttrVec*. Esse, por sua vez, é um *Vector* de *NRSimpleAttribute* que seve para descrever o interesse.

Contudo, antes de serem enviados para a rede, esses interesses passam pelo pós-filtro do *GeoRoutingFilter*. Ao receber uma nova mensagem (de qualquer tipo), esse pós-filtro chama o método *findNextHop*. Esse é responsável por determinar o próximo salto de acordo com as regras de encaminhamento definidas pelo GEAR.

Quando o interesse é recebido no nó seguinte, o *DiffusionRouting* extrai os seus atributos e, através do método *checkPublication*, os confronta com a sua lista de tipos de dados publicados. Se houver alguma coincidência, é porque a mensagem de interesse chegou à fonte e deve ser iniciado o envio de dados para o *sink*. Caso contrário, a mensagem será repassada para o pré-filtro do *IDEAFilter*, o qual consultará o seu atributo *isBastian_* para verificar se alguma anomalia foi detectada. Caso isso não se confirme, a mensagem será repassada para o pré-filtro do *GeoRoutingFilter*. Nesse, dentre outras ações, será verificado se a mensagem já foi recebida anteriormente. Em caso afirmativo, o GEAR assume a ocorrência de um loop e a mensagem é descartada. Porém, se isso não acontecer, o interesse é conduzido para o *GradientFilter*. Nesse, será criado um gradiente para o salto anterior com base nos atributos do interesse. Em seguida, é a vez do pós-filtro do *GeoRoutingFilter* invocar o método *findNextHop* para determinar o próximo salto. Quando o interesse é repassado para o pós-filtro do *IDEAFilter*, verifica-se se a mensagem recebida é uma REQD. Para isso, ele busca no *msg_attr_vec_* da mensagem o atributo *IdeaMsgTypeAttr*. Ao perceber que esse atributo não está presente, a mensagem é simplesmente devolvida para o *DiffusionRouting*, o qual a encaminha para o vizinho definido pelo GEAR.

Perceba que até agora não consideramos a existência de nenhuma região de anomalia na rede. Dessa forma, a rotina descrita no parágrafo anterior se repetirá até que a mensagem de interesse seja recebida na fonte e essa comece a transmitir dados para o *sink*. Observe que o *IDEAFilter* fica de certa forma inerte durante todo esse processo. Ou seja, o pré-filtro apenas verifica se o dado veio de uma região de anomalia e o pós-filtro avalia se a mensagem é uma REQD. Ao constatarem que não é nenhum desses casos, a mensagem é simplesmente conduzida para o próximo filtro. Assim, na medida em que não há anomalias na rede ou nenhuma mensagem é encaminhada para um bastião, os filtros do IDEA não interferem na forma como o GEAR encaminha as mensagens de interesse ou como o Directed Diffusion conduz os dados para o *sink*.

Considere agora que desejamos introduzir na rede uma região de anomalia. A forma que encontramos para simular a detecção de uma anomalia na rede foi programar o script TCL para que, em um dado momento, fosse invocado o método *anomalyDetected* do *IDEAFilter* em todos os nós da rede. Esse método recebe como

parâmetro uma *string* formada por valores separados por dois pontos. Na nossa implementação, essa *string* assume o seguinte formato:

polígono:*coordenada_x*:*coordenada_y*:*raio*

Onde, *polígono* é um número natural predefinido que está associado ao tipo de polígono que será utilizado para representar a região de anomalia. Nesse trabalho, por exemplo, nós associamos o número 1 a um círculo. Em outros trabalhos, o número 2 poderia significar um quadrado, e assim por diante. A *coordenada_x* e a *coordenada_y* são dois números reais os quais determinam a posição do centro da anomalia na rede. Por fim, o *raio* (também um número real) especifica o raio da região de anomalia. Vale a pena lembrar que nesse trabalho nós sempre utilizaremos o círculo como polígono para descrever a região de anomalia.

Ao receber essa *string*, o método *anomalyDetected* invoca a rotina *extractAnomalyRegion* para transformar os valores recebidos em formato texto para valores no formato numérico. Além disso, é criado um objeto da classe *AnomalyRegion* a partir desses valores. Em seguida, cada nó executa o método *isInAnomalyRegion* do objeto *AnomalyRegion* para determinar se o mesmo encontra-se dentro da região de anomalia. Ao executar essa rotina, o atributo *polygon_* será utilizado para chamar o método *isInside* pertencente à classe *Polygon*. Entretanto, por esse ser um método virtual, o programa precisa verificar em tempo real qual tipo de objeto foi instanciado para o atributo *polygon_*. Como já comentamos antes, nesse trabalho será sempre a classe *Circle*. Assim, indiretamente, o método *isInside* da classe *Circle* é utilizado para verificar se o próprio nó está na região de anomalia. Caso isso se confirme, o valor *true* será conferido ao atributo *isInAnomalyRegion_*, a ocorrência da anomalia será gravada no atributo *detectedAnomaly_* e o objeto *AnomalyRegion* será definitivamente armazenado pelo atributo *anomalyRegion_*.

Quando um nó percebe que ele está na região de anomalia, ele executa o método *isInIsolationZone* para avaliar se ele está na faixa de isolamento. Essa avaliação é realizada da seguinte forma:

1. Calcula-se o raio do núcleo da região de anomalia. Para se obter esse resultado, é subtraído do raio da região de anomalia o valor igual à largura da faixa de isolamento (a qual é igual ao alcance de transmissão dos sensores, como foi discutido no capítulo 1).
2. Em seguida, verifica-se se a distância do nó em questão para o centro da região de anomalia é maior do que o raio do núcleo da região de anomalia. Em caso positivo (e considerando que já se sabe que o nó está dentro da região de

anomalia) é possível concluir que o mesmo está na faixa de isolamento. O resultado dessa verificação é, então, armazenado no atributo *isInIsolationZone_*.

Entretanto, se o nó não estiver dentro da região de anomalia, ele invocará o método *isBastian* para determinar se ele é um bastião. Para isso se confirmar, a diferença entre o raio da região de anomalia e a distância do nó para centro dessa região tem que ser menor do que o alcance da transmissão dos sensores. Dessa forma, se o nó concluir que ele é um bastião, ele irá armazenar essa informação no atributo *isBastian_* e, similarmente como os nós que estão na faixa de isolamento, a ocorrência da anomalia é registrada no atributo *detectedAnomaly_*. Os demais nós que não são bastiões e nem estão na faixa de isolamento, não armazenam nenhum registro sobre a região e anomalia.

Nesse momento, a rede está preparada para isolar a região de anomalia e criar os desvios necessários. A partir de então, quando uma mensagem chega um pré-filtro do *IDEAFilter* cujo atributo *detectedAnomaly_* possui o status verdadeiro, é executado o pré-algoritmo de isolamento (seção 3.1). Esse algoritmo foi implementado no método *preProcessFilter*. Porém, a decisão sobre qual parte do pré-algoritmo será executada é tomada a partir dos atributos *isInIsolationZone_* e *isBastian_*. No fim da execução do pré-filtro do *IDEAFilter* será determinado se a mensagem será descartada ou repassada para o filtro do GEAR.

Se a mensagem for repassada para o GEAR, o pós-filtro do *GeoRoutingFilter* irá invocar normalmente o seu método *findNextHop* para definir o próximo salto e aplicar o *id* do mesmo ao atributo *next_hop_* da mensagem. Logo depois, no pós-filtro do IDEA, será decidido se a construção de um desvio deve ser iniciada (ver seção 3.4.1). Para isso, o nó eleito pelo GEAR como próximo salto será passado como parâmetro para o método *isInAnomalyRegion* do atributo *anomalyRegion_*. Assim, é possível verificar se o encaminhamento natural do interesse seria em direção à região de anomalia. Em caso positivo, a rotina *startDetour* do *IDEAFilter* será invocada para iniciar a construção de um desvio. Em caso negativo, a mensagem será encaminhada para o vizinho definido pelo GEAR.

As rotinas do *startDetour* englobam os procedimentos descritos em uma parte da seção 3.4.1 e nas sessões 3.4.2 e 3.4.3. Agora iremos apresentar os detalhes da implementação desses procedimentos. Primeiramente, o método *extractSourceLocation* é utilizado para extrair da mensagem a localização da fonte. Em seguida, essa localização é passada como parâmetro para o método *isInAnomalyRegion*. Se o IDEA perceber que a fonte está dentro da região de anomalia, a mensagem é descartada. Caso contrário, um procedimento para criar um objeto do tipo *Point* representando o PIN é realizado a partir das coordenadas do nó atual. Logo depois, o atributo *anomalyRegion_*

é utilizado para invocar o método *calcPIF* da classe *AnomalyRegion*. Ao fazer isso, o método *calcPIF* utilizará o atributo *polygon_* para chamar um outro método *calcPIF*, o da classe *Polygon*. Apesar de o atributo polygon_ ser do tipo *Polygon*, o interpretador perceberá em tempo de execução que a instância associada a tal atributo é da classe *Circle*. Assim, o método *calcPIF* dessa classe é que determinará a localização do PIF.

Toda via, o *calcPIF* poderá perceber que a reta determinada pelo PIN e pela fonte não intercepta a região de anomalia (ver seção 3.4.1). Nesse caso, não é possível determinar o PIF. Se isso ocorrer, a inicialização do desvio será abortada e o IDEA utilizará o seu atributo *gearFilter_* para chamar o método *findNextHop* do *GeoRoutingFilter*. Assim, outro vizinho, diferente do primeiro, será eleito como próximo salto conforme as regras do GEAR.

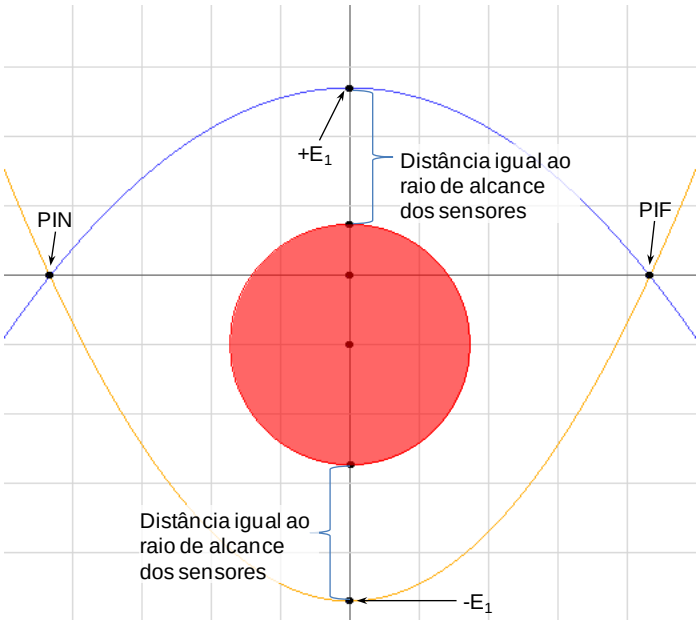


Figura 42 - Determinação do ponto extremo

Por outro lado, se a reta determinada pelo PIN e pela fonte interceptar a região de anomalia, serão processadas as rotinas do *calcTrajectoryDeterminants*. Nesse método, primeiramente é verificado através do atributo *detourMethod_* o tipo de desvio previamente definido como o padrão para a rede. Logo após, é determinada a trajetória de referência que será utilizada para a construção do desvio. Na nossa implementação, a primeira trajetória adotada sempre será aquela que passa pelo ponto extremo o qual está à menor distância recomendada para a região de anomalia (conforme discutido na seção 3.4.2). Ou seja, é determinado o ponto E_1 que está sobre o eixo das ordenadas e a uma distância para a região de anomalia igual ao raio de alcance dos sensores. Entretanto, em alguns casos haverá dois pontos que atendem a essa condição: um situado na parte positiva do eixo das ordenadas $+E_1$ e outro na parte negativa $-E_1$ (ver Figura 42).

Assim, na nossa implementação, sempre é escolhido aquele que determinar a menor trajetória. Ou seja, cuja coordenada y possui o menor valor absoluto.

Nesse momento, é importante lembrar que, a partir dos estudos apresentados no capítulo **Erro! Fonte de referência não encontrada.**, adotamos as linhas de força como trajetórias de referência nesse trabalho. Além disso, nesse mesmo capítulo, nós já comentamos que um único ponto no plano cartesiano é capaz de determinar uma única linha de força e, conseqüentemente, uma única trajetória de referência.

Dessa forma, o único elemento que irá compor o conjunto de determinantes da trajetória é a coordenada y do ponto $+E_1$ ou $-E_1$. As coordenadas do PIN e do PIF, também necessárias para determinar a trajetória de referência, já farão parte do conjunto de atributos da mensagem de interesse e, por isso, não serão incluídos no *array* de determinantes. Assim, esse *array* será utilizado como o retorno do método *calcTrajectoryDeterminants*.

Após esse retorno, a seqüência de execução volta para o *startDetour*. A próxima etapa desse método é incluir na mensagem de interesse um indicador através do qual será possível identificá-la como uma REQD. Isso é feito adicionando-se ao *msg_attr_vec_* do interesse outro elemento do tipo *NRSimpleAttribute*: o *IdeaMsgTypeAttr*. Esse, por sua vez, é inicializado com o valor igual ao da constante *DETOUR_REQUEST*. A fim de completar a mensagem de REQD, cada um dos demais *NRSimpleAttributes* do IDEA são inseridos no *msg_attr_vec_*. Como resultado, a mensagem de REQD conterá informações sobre as dimensões da região de anomalia, os coeficientes da trajetória de referência e a localização do PIN e do PIF.

No final do *startDetour* é invocado o método *findDetourNextHop*. Esse método é responsável por eleger como próximo salto o vizinho que possui os melhores valores para o ganho angular, para a distância até a trajetória de referência e para a energia remanescente (seção 3.4.3). Para se alcançar esse objetivo, primeiramente é encontrada a projeção ortogonal P do centro da anomalia sobre a reta definida pelo PIN e pelo PIF (ver seção 3.4.1).

Em seguida, é iniciado um *loop* no qual, para cada *NeighborEntry* que não está na região de anomalia e não é o salto anterior são realizadas várias análises. Primeiro é verificado se ele é a fonte através do método *isInside* da classe *Circle*. Caso isso se confirme o loop é imediatamente abortado, o valor *true* é atribuído ao parâmetro *isNextHopInsideSourceRegion* e esse nó é retornado como próximo salto. Caso contrário, será dada continuidade ao loop e será calculado para esse nó o seu ganho angular, a sua distância para trajetória de referência e a sua energia restante

O cálculo do ganho angular sempre é precedido de algumas transformações lineares no plano cartesiano sobre o vizinho em questão. Essas transformações atuam

como uma função $g(x, y)$ a qual associa cada ponto geográfico da rede a um único ponto do plano cartesiano C definido na seção 3.4.2. As coordenadas do vizinho obtidas após essas transformações, as quais chamaremos de coordenadas virtuais, são passadas para o método *calcAngularGain*, onde efetivamente é calculado o ganho angular.

Ainda dentro do *loop*, o método invocado na seqüência é o *calcDistanceToTrajectory*. Esse método primeiramente consulta o atributo *detourMethod_* para verificar qual método de desvio foi adotado. Ao perceber que esse atributo possui o valor igual ao da constante *LINES_OF_FORCE*, ele irá invocar outro método, o *calcDistToForceLine*. A partir do *array* de determinantes, do PIN e do PIF passados como parâmetro, esse último método é capaz de encontrar exatamente a mesma trajetória que foi anteriormente definida. Em seguida, algumas operações matemáticas são realizadas de forma a obter a distância do vizinho (cujas coordenadas também são passadas como parâmetro) para a trajetória de referência.

Nas últimas rotinas do loop, é obtida a energia restante do vizinho em análise através do atributo *remaining_energy_* da classe *NeighborEntry*. Logo depois, esse valor junto com o ganho angular e a distância para trajetória de referência são passados como parâmetro para o método *calcNextHopCost*, no qual é calculada a média ponderada com esses três valores (ver expressão (2) na seção 3.4.3). Uma vez que nesse trabalho os pesos adotados para o cálculo dessa média são iguais, calculamos, na verdade, uma média aritmética com o ganho angular, a distância para trajetória de referência e a energia remanescente. Assim, no fim do loop, será eleito como próximo salto o vizinho que possuir o menor custo.

O fim do loop também marca o término do método *findDetourNextHop*. Ao retornar para o método *startDetour*, o atributo *next_hop_* da mensagem é sobrescrito com o *id* do vizinho eleito pelo IDEA. Agora, a mensagem de interesse, transformada em uma REQD, está pronta para ser enviada ao próximo salto.

Desse momento, até o término do desvio, cada nó que receber a REQD irá proceder da mesma forma que a descrita nos próximos cinco parágrafos.

Quando a mensagem chega no próximo salto, primeiramente ela é encaminhada para o pré-filtro do *IDEAFilter*. Nesse, são realizados os mesmos procedimentos que os descritos anteriormente para determinará se a REQD deverá ser descartada ou repassada para o pré-filtro do *GeoRoutingFilter*. Se o segundo caso for considerado, o *rdm_id_* da mensagem será registrado para prevenir futuros loops e ela será encaminhada para o *GradientFilter*.

Já citamos anteriormente que esse filtro é responsável por administrar os gradientes em cada sensor. Entretanto, é importante lembrar que a criação dos gradientes é realizada a partir dos atributos contidos nas mensagens de interesse. Dessa

forma, se o *GradientFilter* utilizasse integralmente o *msg_attr_vec_* da REQD para criar os gradientes, seriam adotados, não só os atributos originais do interesse, mas também os atributos do IDEA. Conseqüentemente, foi necessário realizarmos algumas modificações no *GradientFilter* para que o mesmo pudesse gerar os gradientes a partir dos atributos corretos. Essas modificações foram implementadas para extrair do *msg_attr_vec_* os atributos do IDEA antes das operações com os gradientes e, depois, incluí-los de volta na mensagem.

Em seguida, o pós-filtro do *GeoRoutingFilter* recebe a mensagem e executa normalmente as suas rotinas para eleição do próximo salto. Quando o pós-filtro do *IDEAFilter* é acionado na seqüência, o método *postProcessFilter* verificará se a mensagem recebida é uma REQD. Para isso, ele busca no *msg_attr_vec_* o atributo *IdeaMsgTypeAttr* e, após confirmar a presença do mesmo, ele verifica se o mesmo possui um valor igual ao da constante *DETOUR_REQUEST*. Caso isso se confirme, informações sobre o PIN, o PIF, a região de anomalia e os determinantes da trajetória são obtidas a partir dos respectivos atributos contidos do *msg_attr_vec_*. Logo depois, o método *findDetourNextHop* é executado do mesmo modo que o descrito anteriormente para que o IDEA selecione o próximo salto.

Ao retornar do método *findDetourNextHop*, o *postProcessFilter* irá verificar se o desvio deve ser finalizado. Para isso, primeiramente é verificado se foi atribuído o valor *true* ao parâmetro *isNextHopInsideSourceRegion*. Se esse for o caso, isso significa que o próximo salto é a fonte e o desvio deverá ser interrompido.

Caso contrário, será verificado se a distância do nó atual para o PIF é menor do que a distância desse mesmo nó para o vizinho eleito pelo IDEA (seção 3.4.4). Se isso se confirmar, os atributos do IDEA são removidos do *msg_attr_vec_* e o interesse original é encaminhado para o próximo salto determinado pelo IDEA. Quando isso acontece, os saltos seguintes passam a ser determinados pelo GEAR.

Até esse ponto, descrevemos apenas a execução seqüencial do IDEA para a construção da primeira trajetória de referência. Entretanto, a construção das trajetórias alternativas é realizada de forma similar. A única diferença será no processo para determinar a trajetória de referência no método *calcTrajectoryDeterminants*. O evento que inicia a construção das trajetórias alternativas é o recebimento de outra mensagem de interesse em um bastião que já construiu um primeiro desvio. Na nossa implementação, a definição das trajetórias alternativas será realizada através da variação do ponto extremo das linhas de força sobre o eixo das ordenadas. Essa variação ocorrerá da seguinte forma:

1. Considere que um bastião já recebeu $1, 2, 3, \ldots, (n-1)$ mensagens de interesse que, a princípio, iriam em direção à região de anomalia. Mas que o IDEA impediu isso e iniciou a construção de um desvio para cada uma.
2. Também considere que a mensagem de interesse n acabou de ser recebida nesse mesmo bastião.
3. Por fim, considere D a distância, sobre o eixo das ordenadas, entre a região de anomalia e o ponto extremo que será associado ao interesse n.
4. Assim, a distância D será calculada através da seguinte expressão:

$$D = j \times [Raio\ de\ alcance\ do\ sensor]$$

$$\text{Onde, } \begin{cases} j = 3, \ se\ n(mod\ 3) = 0 \\ j = n(mod\ 3), \ se\ n(mod\ 3) \neq 0 \end{cases}$$

Dessa forma, a partir da distância D e de outros elementos conhecidos (como o posicionamento do plano na rede, a posição da região de anomalia e as dimensões da mesma) será possível determinar o ponto extremo e a trajetória alternativa associados ao interesse n.

Modificações no Directed Diffusion

Na seção anterior, foi detalhada a seqüência de rotinas executadas no ns-2 para que fossem construídos múltiplos desvios em torno da região de anomalia. Contudo, a tarefa de alternar o encaminhamento das mensagens de dados por esses desvios cabe ao protocolo de roteamento da rede, ou seja, o Directed Diffusion. Porém, esse protocolo não foi desenvolvido para essa finalidade. Assim, foram necessárias algumas modificações no DD a fim de se avaliar os benefícios e o impacto da utilização do IDEA em uma RSSF.

As alterações realizadas do Directed Difusion concentram-se na classe *GradientFilter*. Mais especificamente foi adicionado o atributo *lastReinforced_* e modificados os métodos *forwardExploratoryData*, *forwardData* e *processNewMessage*.

Na implementação original do DD, o *forwardExploratoryData*, toda vez que recebe uma mensagem de dados exploratória, encaminha uma cópia da mesma para cada um dos gradientes do nó. Porém, na grande maioria dos casos, a cópia enviada através de um desses gradientes sempre será a aquela primeiramente recebida pelo *sink* (e as demais descartadas). Isso significa que o caminho inverso ao do recebimento dessa cópia será sempre o único reforçado. E isso acarretará, posteriormente, no envio das mensagens de dados através de um único caminho.

Dessa forma, a fim de se permitir que diferentes caminhos sejam alternadamente reforçado, a solução adotada foi permitir ao nó realizar um escalonamento *round-robin* entre todos os seus gradientes. Esse escalonamento foi implementado no método *forwardExploratoryData* aproveitando o fato de que, no código do DD, os gradientes para um mesmo interesse são armazenados em um único *vector*. Dessa forma, quando houver mais de um gradiente, aquele que estiver na última posição do *vector* será escolhido para encaminhar a mensagem de dados exploratória. Logo em seguida, esse gradiente é movido para primeira posição do *vector* e os demais são deslocados uma posição para frente. Essa operação se repete para cada nova mensagem de dados exploratória recebida.

Entretanto, quando um caminho é reforçado, ele permanece com esse status durante certo tempo. Conseqüentemente, se antes desse tempo expirar, outra mensagem de dados exploratória for recebida e enviada por um caminho diferente, haverá simultaneamente entre o *sink* e a fonte mais de um caminho reforçado. Nesse momento, é importante lembrar que, quando um nó recebe uma mensagem de dados, o *forwardData* original do DD envia uma cópia dessa mensagem para cada um dos caminhos reforçados que convergem nesse nó.

A fim de se evitar um desperdício de energia com essa replicação de mensagens de dados, foi necessário modificar os métodos *processNewMessage* e *forwardData*. No primeiro, foram realizadas as modificações necessárias para que, quando uma nova mensagem de reforço seja recebida em um nó, esse registra no seu atributo *lastReinforced_* o *id* do último salto. Já no *forwardData*, as nossas modificações possibilitaram que, ao receber uma mensagem de dados, um nó a encaminhe para um único vizinho: aquele registrado no atributo *lastReinforced_*.

Assim, de um ponto de vista mais global, essas modificações permitem que a mensagem de dados seja encaminhada através do último caminho reforçado. Ou seja, elas permitem que cada um dos desvios construídos seja alternadamente reforçado e utilizado um por vez para o encaminhamento dos dados. Porém, uma vez que a abdicação de um desvio em detrimento de outro só é realizado quando uma nova mensagem de reforço é recebida, a freqüência com que isso ocorrerá vai depender da periodicidade na qual as mensagens de dados exploratórias são enviadas.

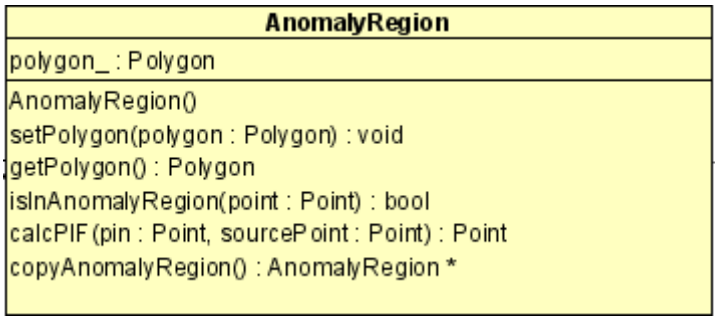
APÊNDICE

D

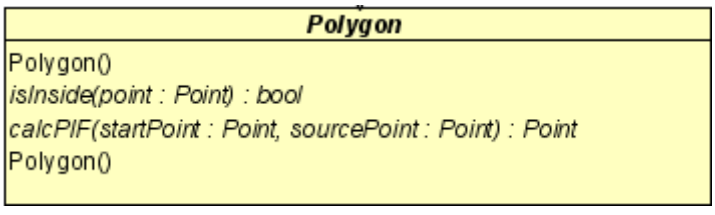
Diagramas de classes

Nesse apêndice serão apresentados os diagramas de classes do IDEA desenvolvidas para as simulações no ns-2. Além das classes do IDEA, também serão apresentados, mas de forma mais resumida, os diagramas das classes do Directed Diffusion e do GEAR que se relacionam com o IDEA. Uma melhor explicação sobre o papel de cada uma das classes apresentadas aqui pode ser consultada no Apêndice C.

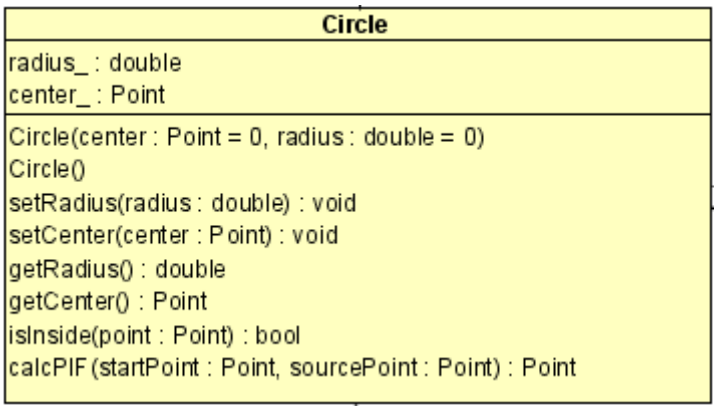
A classe AnomalyRegion



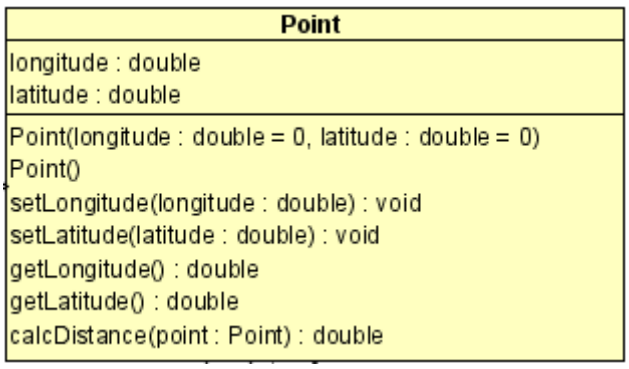
A classe Polygon



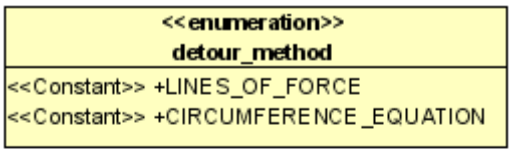
A classe Circle

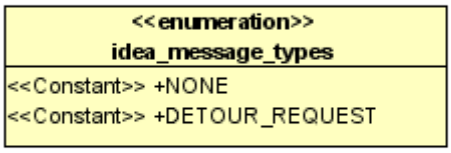


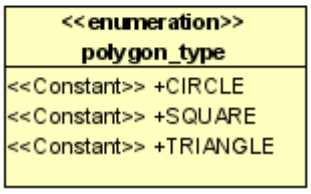
A classe Point



As constantes



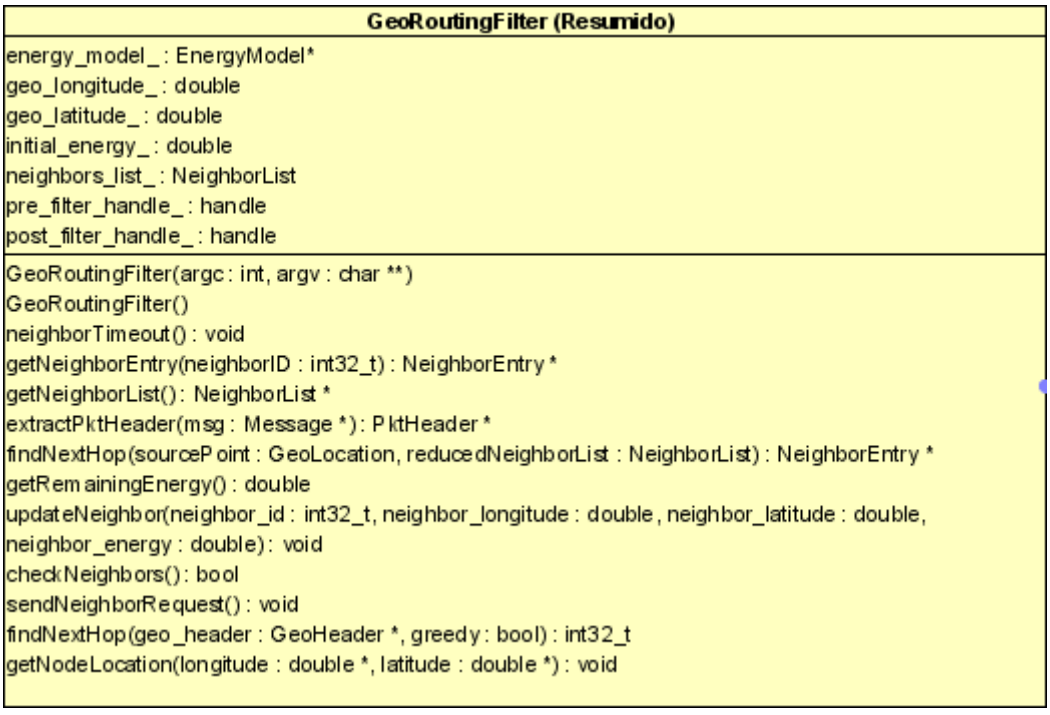




A classe IDEAFilter

IDEAFilter
numbOfTrajectorys_ : int lastTrajectoryUsed_ : int transmissionRange_ : double geoLongitude_ : double geoLatitude_ : double preFilterHandle_ : handle postFilterHandle_ : handle anomalyRegion_ : AnomalyRegion detectedAnomaly_ : bool isBastian_ : bool isInIsolationZone_ : bool isInAnomalyRegion_ : bool gearFilter_ : GeoRoutingFilter* detourMethod_ : int maxDistanceToTrajectory_ : double filterCallback_ : IDEAFilterReceive*
IDEAFilter(argc : int, argv : char **) IDEAFilter() run() : void recv(msg : Message *, h : handle) : void anomalyDetected(anomalyRegionChar : char *) : void startDetour(msg : Message *) : void findDetourNextHop(lastHop : int, pin : Point, pif : Point, sourceRegion : Circle, anomalyRegion : AnomalyRegion *, detourMethod : int, determinants : double [], isNextHopInsideSourceRegion : bool &) : NeighborEntry * calcTrajectoryDeterminants(detourMethod : int, pin : Point, pif : Point) : double * calcDistanceToTrajectory(nextHop : Point, detourMethod : int, pin : Point, pif : Point, determinants : double []) : double calcAngularGain(nodeLocation : Point) : double reachedEndNode(nextHop : Point, pif : Point) : bool extractAnomalyRegion(coordinatesChar : char *) : void getNeighborLocation(nodeID : int) : Point extractSourceLocation(msg : Message *) : Circle isBastian() : void calcNextHopCost(angularGain : double, distanceToCurve : double, remainingEnergy : double) : double getInitialEnergy() : double isInIsolationZone(nodePosition : Point) : bool calcDistToForceLine(pin : Point, pif : Point, neighbor : Point, extremePoint : double) : double setupPostFilter() : handle setupPreFilter(geoLongitude_ : double, geoLatitude_ : double) : handle preProcessFilter(msg : Message *) : void postProcessFilter(msg : Message *) : void getNodeLocation(longitude : double &, latitude : double &) : void

A classe GeoRoutingFilter



A classe GradientFilter

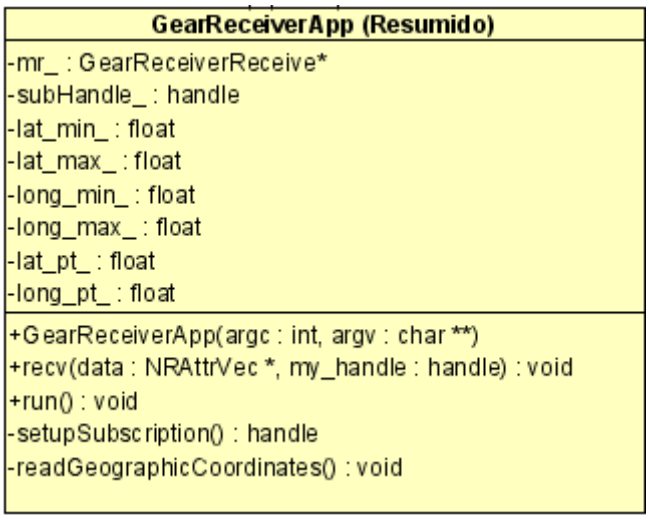
A classe DiffusionRouting

DiffusionRouting (Resumido)
pub_list_ : HandleList sub_list_ : HandleList filter_list_ : FilterList
DiffusionRouting(port : u_int16_t) run(wait_condition : bool, max_timeout : long) : void subscribe(subscribe_attrs : NRAttrVec *, cb : Callback *) : handle unsubscribe(subscription_handle : handle) : int publish(publish_attrs : NRAttrVec *) : handle unpublish(publication_handle : handle) : int send(publication_handle : handle, send_attrs : NRAttrVec *) : int addFilter(filter_attrs : NRAttrVec *, priority : u_int16_t, cb : FilterCallback *) : handle removeFilter(filter_handle : handle) : int sendMessage(msg : Message *, h : handle, priority : u_int16_t = FILTER_KEEP_PRIORITY) : int recvPacket(pkt : DiffPacket) : void recvMessage(msg : Message *) : void sendMessageToDiffusion(msg : Message *) : void sendPacketToDiffusion(pkt : DiffPacket, len : int, dst : int) : void processMessage(msg : Message *) : void processControlMessage(msg : Message *) : void checkSubscription(attrs : NRAttrVec *) : bool checkPublication(attrs : NRAttrVec *) : bool checkSend(attrs : NRAttrVec *) : bool removeHandle(my_handle : handle, hl : HandleList *) : HandleEntry * findHandle(my_handle : handle, hl : HandleList *) : HandleEntry * deleteFilter(my_handle : handle) : FilterEntry * findFilter(my_handle : handle) : FilterEntry *

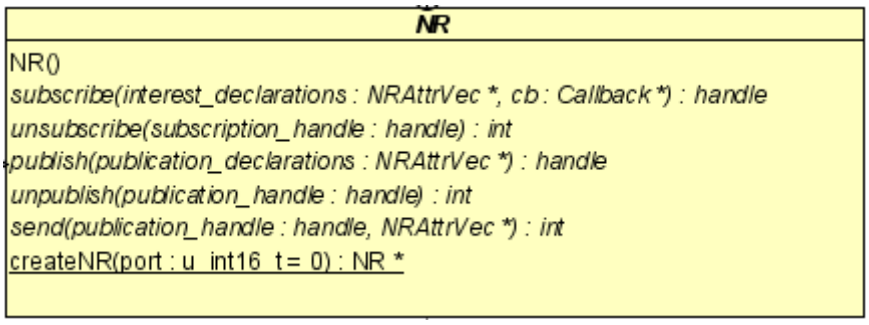
A classe GearSenderApp

GearSenderApp (Resumido)
-mr_ : GearSenderReceive* -subHandle_ : handle -pubHandle_ : handle -num_subscriptions_ : int -last_seq_sent_ : int -lat_min_ : float -lat_max_ : float -long_min_ : float -long_max_ : float -lat_pt_ : float -long_pt_ : float -data_attr_ : NRAttrVec
+GearSenderApp(argc : int, argv : char **) +GearSenderApp() +run() : void +recv(data : NRAttrVec *, my_handle : handle) : void -setupSubscription() : handle -setupPublication() : handle -readGeographicCoordinates() : void +recv(data : NRAttrVec *, my_handle : handle) : void

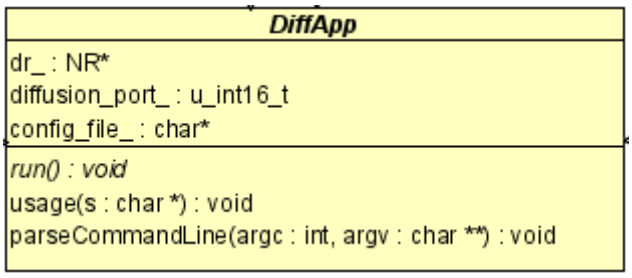
A classe GearReceiverApp



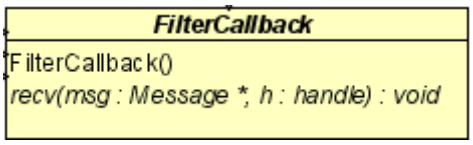
A classe NR



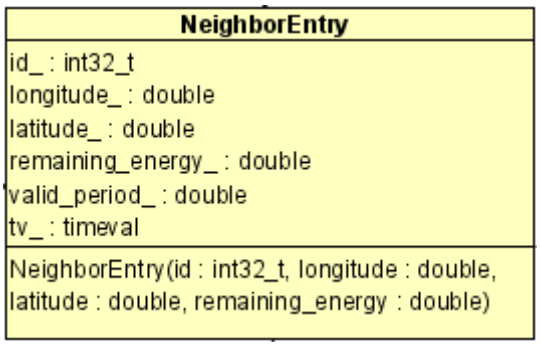
A classe DiffApp



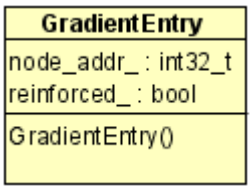
A classe FilterCallback



A classe NeighborEntry



A classe GradientEntry



A classe TTPRoutingEntry