UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ATENTO: UM DETECTOR DE DEFEITOS PARA MANETS BASEADO NA POTÊNCIA DO SINAL

DISSERTAÇÃO DE MESTRADO

Miguel Angelo Baggio

Santa Maria, RS, Brasil

2010

ATENTO: UM DETECTOR DE DEFEITOS PARA MANETS BASEADO NA POTÊNCIA DO SINAL

por

Miguel Angelo Baggio

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-Graduação em Informática, Área de Concentração em Sistemas Paralelos e Distribuídos, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Computação

Orientador: Prof. Dr. Raul Ceretta Nunes

Santa Maria, RS, Brasil
2010

B144a Baggio, Miguel Angelo, 1982 –
Atento: um detector de defeitos para MANETS baseado na potência do sinal. / por Miguel Angelo Baggio. Santa Maria, 2010.
79 f.; 30 cm; il.

Orientador: Raul Ceretta Nunes
Dissertação (mestrado) – Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2010.

1. Redes sem-fio 2. Detectores de defeitos 3. Mobilidade 4. Potência do sinal I. Nunes, Raul Ceretta II.Título.

CDU: 004.7

Ficha catalográfica elaborada por
Claudia Carmem Baggio - CRB-10/830

Endereço: Rua Barão do Triunfo, 1630, apto: 302, Santa Maria, RS, 97015-070
Fone (0xx) 55 3026 3381; End. Eletr.: migbaggio@hotmail.com

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A comissão Examinadora, abaixo assinada,
Aprova a Dissertação de Mestrado

ATENTO: UM DETECTOR DE DEFEITOS PARA MANETS BASEADO NA POTÊNCIA DO SINAL

elaborada por
Miguel Angelo Baggio

como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

Raul Ceretta Nunes, Dr.
(Presidente/Orientador)

Marcia Pasin, Dra.
(UFSM)

João Baptista dos Santos Martins, Dr.
(UFSM)

Santa Maria, 30 de Dezembro de 2010.

À minha noiva, Franciele dos Santos Staudt, aos meus pais, Luiz e Tereza, e aos meus irmãos, pela compreensão, amor e estímulo constante, dedico esta obra.

Agradecimentos

Agradeço, em primeiro lugar, ao Professor Doutor Raul Ceretta Nunes, pela oportunidade, estímulo, confiança, pela contribuição na minha formação profissional e disponibilidade para me orientar no decorrer deste trabalho;

À Professora Marcia Pasin, por ter confiado no meu trabalho, pelo estímulo, pela participação nas reuniões e em todas sugestões feitas para o desenvolvimento deste trabalho;

A todos os colegas que conheci e participaram em vários momentos da minha caminhada ao longo dessa pesquisa, por compartilhar todas as vitórias nas etapas que formaram este trabalho e pelo companheirismo que tornou as nossas dissertações grandes aprendizados em equipe;

À Universidade Federal de Santa Maria, pela oportunidade, permitindo o meu aperfeiçoamento profissional;

A todos os amigos do grupo de estudos, meus amigos das jogatinas e todos amigos que fizeram ou fazem parte da minha vida.

A minha família, aos meus pais Luiz e Tereza, por ter dado todo o apoio psicológico e financeiro.

Agradeço também especialmente a Franciele que ficou ao meu lado todos esses anos dando todo suporte necessário.

Agradeço tambem a todos os meus irmãos, especialmente ao meu irmão José que abdicou de vários momentos para me ajudar a fazer o bendito simulador, na escrita do trabalho e inclusive entrando noites a dentro.

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

ATENTO: UM DETECTOR DE DEFEITOS PARA MANETS BASEADO NA POTÊNCIA DO SINAL

Autor: Miguel Angelo Baggio
Orientador: Raul Ceretta Nunes
Data e Local de Defesa: Santa Maria, 30 de Dezembro de 2010.

Detectar falhas é uma tarefa essencial na construção de sistemas distribuídos confiáveis e seu projeto depende fortemente do modelo de sistema distribuído. Diversas soluções tem sido desenvolvidas para tratar a movimentação de nodos em sistemas distribuídos. Este trabalho apresenta um novo detector de defeitos assíncrono não-confiável para redes móveis ad hoc (MANETs). Nodos falhos são diferenciados de nodos móveis através da manutenção de informação sobre a medida de potência do sinal de recepção nos nodos do sistema em um pequeno histórico de regiões.

Neste trabalho também foi desenvolvido um simulador para redes móveis sem fio onde é possível configurar simulações de transmissão de mensagens com movimentos pré-determinados dos nodos na rede. O simulador disponibiliza a potência do sinal recebido em uma mensagem, e permite que sejam configurados diversos aspectos da simulação como freqüência e potência de transmissão de uma mensagem.

O detector de defeitos proposto neste trabalho utiliza um método para suspeitar de um nodo que está dentro de sua região de alcance de transmissão e outro método para um nodo que está movimentando-se para fora do seu alcance de transmissão. Avaliações apresentadas neste trabalho demonstram que é possível uma redução no tempo de detecção de uma falha e um menor número de falsas suspeitas, quando comparados com detectores que utilizam apenas *gossip*.

Palavras-chave: detector de defeitos, redes sem-fio, mobilidade, potência do sinal.

ABSTRACT

Master Dissertation
Computers Science Graduate Program
Federal University of Santa Maria

ATENTO: A SIGNAL POWER BASED FAILURE DETECTOR FOR MANETS

Author: Miguel Angelo Baggio
Adviser: Raul Ceretta Nunes
Santa Maria, December 30, 2010.

Fault detector is an essential component in building reliable distributed systems and its design depends heavily on the model of distributed system, which has demanded several solutions to address the movement of nodes. This dissertation presents a gossip-based unreliable failure detector for mobile ad hoc networks (MANETs) that uses the intensity of received signal to differ faulty nodes from mobile nodes. The differentiation is done by maintaining information on the intensity measurement signal reception in the nodes of the system in a small historic regions.

This work also presents a simulator for mobile wireless networks where is possible configure simulations messaging with pre-determined movements of the nodes in the network. The simulator provides the signal strength received in a message, and allows configuration simulate various aspects such as frequency and intensity transmission of a message.

The failure detector presented in this work provides a new method that uses more than one rule for deciding to suspect a node. Surveys show improvements in service quality of the detector compared with the traditional gossip algorithm.

Keywords: failure detector, wireless networks, mobility and intensity.

Lista de Figuras

SUMARIO

1 INTRODUÇÃO

Nos últimos anos, assim como ocorreu o desenvolvimento dos computadores em termos de velocidade, disponibilidade, redução de custos e redução de volume, a evolução da tecnologia permitiu semelhante melhora nas redes de computadores.

Com o surgimento das novas tecnologias e criação de novos protocolos e padrões, as definições de redes se desenvolveram até o estado atual, onde dois tipos de redes são predominantes: as LANs (*local area-network)* ou redes locais e as WANs (*wide-area networks*) ou redes de longa distância. Neste contexto, um sistema distribuído (SD) é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e independente (TANENBAUM e STEEN, 2007).

Atualmente, esses computadores incluem dispositivos móveis como *PDAs*, telefones celulares, *notebooks*, *netbooks*, sensores, aparelhos de som ou qualquer dispositivo que possua uma maneira de trocar informação com outros dispositivos. Essa variedade de dispositivos acabou transformando o comportamento dos sistemas distribuídos e dos usuários de computadores e dispositivos móveis, fazendo com que os sistemas distribuídos sejam construídos com redes sem fio e apresentam topologia dinâmica. Esta mudança tem demandado esforço da comunidade científica para adequar algoritmos a este novo cenário (MATEUS e LOUREIRO, 2005).

Uma importante característica em SD é o aspecto de falha parcial (TANENBAUM e STEEN, 2007), onde apenas um componente do sistema distribuído pode falhar. Em um sistema distribuído é possível tornar o sistema capaz de conseguir recuperar-se após uma falha parcial, sem afetar o restante do sistema. Uma característica desejável é que o sistema continue operando com as suas principais funcionalidades enquanto o sistema estiver tentando se recuperar da falha que ocorreu. Para construir um sistema confiável, é importante que o sistema consiga controlar as suas falhas, buscando: evitar falhas, remover falhas e prever as falhas.

Mobilidade dos nodos é um novo desafio para detecção de defeitos, pois quando um nodo não falho está em movimento e fora do alcance de transmissão, pode acabar por tornar-se suspeito de falha. Diferenciar um nodo em movimento e um nodo falho pode ser uma tarefa difícil dependendo das características do sistema distribuído. Além disso, como os nodos trocam informações através de troca de mensagens, o atraso ou a perda de mensagens também pode levar a um nodo suspeitar equivocadamente de outros nodos.

Para tratar deste problema, existem algoritmos para detecção de defeitos, que são um bloco de construção básico para a construção de sistemas distribuídos confiáveis (COULOURIS, 2007), tratando do desafio de detectar a falha de um nodo e têm sido amplamente estudados para operar em redes móveis sem fio (HUTLE, 2004)(FRIEDMAN e TCHARNY, 2005)(SRIDHAR, 2006)(SENS et al., 2008).

Detectar defeitos em um sistema distribuído é tarefa dos detectores de defeitos. Felber et al. (1999), descreve que um detector de defeitos é um algoritmo distribuído que fornece informações sobre suspeita de defeitos em componentes monitoráveis. Segundo Gracioli e Nunes (2007), um detector de defeitos é uma parte importante para sistemas tolerantes a falhas. Essas informações são fornecidas através de troca de mensagens entre os nodos do sistema que contem o detector de defeitos acoplado

1.1 Contexto e motivação

Trabalhos atuais utilizam o detector de defeitos para tratar da descoberta de falhas de nodos. Cason e Greve (2009) usam um detector de defeitos para contornar o problema do consenso, em uma rede dinâmica. Nunes e Greve (2009) utilizam um detector de defeitos para garantir propriedades em seu sistema de gestão de filiação ao grupo.

Existem diferentes detectores de defeitos que são projetados para operar com diferentes modelos de sistemas. Alguns detectores são capazes de tratar com partições de rede, parcialmente síncronos, sistemas assíncronos entre outras características que podem diferenciar os tipos de sistemas distribuídos.

Entretanto, tais detectores de defeitos não levam em consideração uma informação sobre a movimentação dos nodos de uma maneira precisa em relação à distância dos nodos. Os nodos apenas sabem que outro nodo vizinho está no seu alcance de transmissão, sem diferenciar se ele está perto ou longe. Sendo assim, um detector de defeitos que consegue saber a localização do nodo e ver uma movimentação para fora do alcance de transmissão ou uma movimentação para uma localização mais próxima pode oferecer informações mais precisas sobre os seus nodos, melhorando a qualidade de serviço do detector.

1.2 Contribuições

Este trabalho apresenta uma estratégia chamada de *Detector de Defeitos Atentos para MANETS*, que usa a potência do sinal recebido em uma mensagem como parâmetro para definir a estratégia a ser utilizada no detector de defeitos. Com a potência do sinal, é possível identificar a mobilidade dos nodos dentro do alcance de transmissão. Identificar um padrão de movimentação dos nodos, dispensando o uso do número de *hops*. O trabalho explora como ajustar o tempo para suspeitar de um nodo, sem modificar o *Timeout* do sistema. Como resultado através da identificação da movimentação, pode-se obter um tempo de detecção de falha menor e mais preciso em alguns casos.

Também foi desenvolvimento um simulador de redes móveis sem fio, que mostra a movimentação dos nodos e inclui, na troca de mensagens, a potência do sinal recebido. O simulador permite que seja possível utilizar rotas aleatória e/ou rotas determinadas pelo usuário para a movimentação dos nodos, conforme é apresentado no capítulo 4.1.

1.3 Organização do trabalho

O capítulo 2 apresenta uma revisão detectores de defeitos para redes sem fio e definições importantes para o estudo dos detectores de defeitos.

O capítulo 3 propõe um novo detector de defeitos para redes sem fio baseado na potência do sinal e na movimentação dos nodos.

O capítulo 4 mostra os testes e resultados obtidos com o detector proposto neste trabalho.

O capitulo 5 apresenta as conclusões deste trabalho.

2 Detecção de Defeitos

Este capítulo apresenta uma revisão sobre os detectores de defeitos e características importantes na definição do modelo de sistema onde esses detectores atuam.

Inicialmente é apresentado um dos ambientes onde os detectores de defeitos são utilizados, que são as redes MANETs (seção 2.1). Na seção seguinte é apresentada uma explicação sobre os tipos de falhas (seção 2.2). Na sequência, são apresentadas as propostas dos principais detectores de defeitos para MANETs (seção 2.5). Finalmente, são apresentadas as métricas para detectores de defeitos (seção 2.7) e algumas considerações sobre as propostas estudadas (seção 2.9).

2.1 Redes MANETs

Segundo Johnson e Maltz (1996), dispositivos móveis e *hardware* de redes sem fio vêm sendo amplamente disponibilizados e utilizados, e um extensivo trabalho vem sendo feito recentemente para integrar estes elementos em redes tradicionais como a internet. Os usuários destes dispositivos móveis querem ter uma comunicação em situações que não existe uma infra-estrutura com fio, porque não seria viável fisicamente devido à restrições de espaço e mobilidade. Desta forma redes MANETS são indicadas para interligar os dispositivos.

Tais situações podem ser exemplificadas por ocasiões onde pesquisadores que gostariam de interagir durante uma leitura, ou por amigos ou sócios que desejam trocar informações ou arquivos durante uma viagem, ou um grupo de resgate em uma situação que precisam passar informações rapidamente após um acontecimento grave, tais como terremoto ou incêndio. Nestas situações, um conjunto de dispositivos móveis com comunicação sem fio pode oferecer um serviço de rede temporário sem o auxilio de qualquer tipo de infraestrutura fixa ou centralizada, denominada rede *ad hoc*.

A necessidade de trocar informações de maneira simples e direta faz com que MANETs, um tipo de rede *ad hoc* sejam cada vez mais utilizadas quando se tratam de celulares, *PDAs*, notebooks ou redes de sensores.

MANET (*Móbile Ad Hoc Networks*) (BASILE et al. 2003) é um tipo particular de rede móvel sem fio, que tem características como ser autoconfigurável (*ad hoc*), onde os dispositivos podem alterar as suas conexões com outros dispositivos com frequência, operando sem uma administração centralizada, como se fosse um roteador. Desta maneira, os dispositivos procuram cooperar com outros dispositivos para prover conectividade.

Este tipo de rede possui uma comunicação direta entre os nodos e não exige infraestrutura para a comunicação. Com esta característica, os nodos não precisam de um protocolo de roteamento ou decisões de roteamento para uma comunicação enquanto o alcance de transmissão de seu rádio permitir comunicação com vizinho(s). A Figura 1 e a Figura 2 são exemplos de uma rede *ad-hoc*.

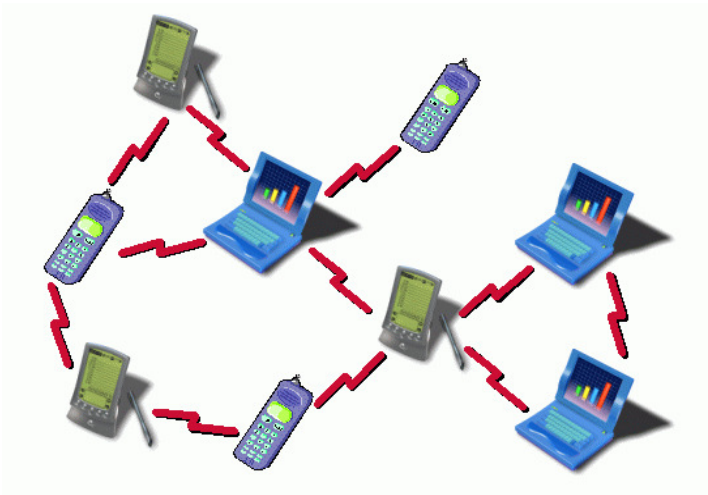

Figura 1 – Notebooks, celulares e PDAs formando uma rede ad-hoc [1]

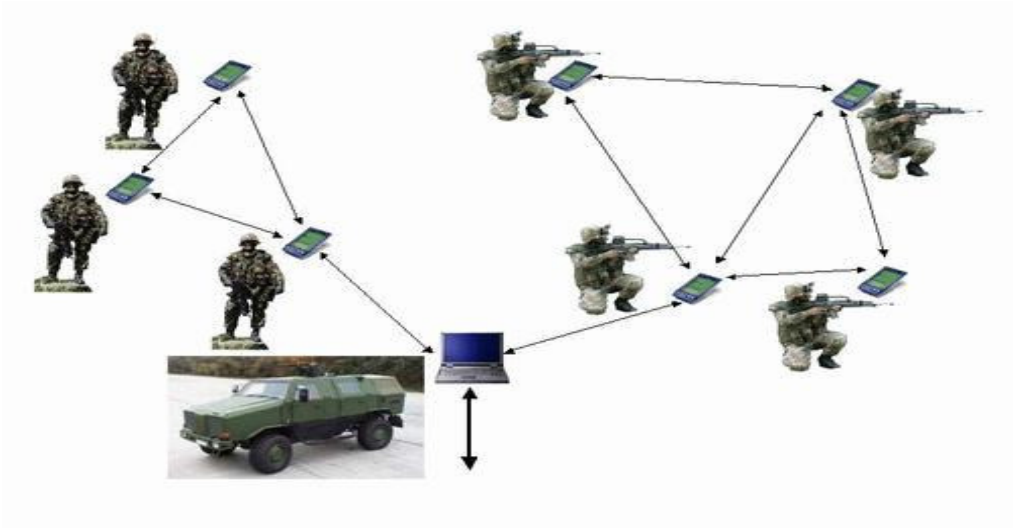

Figura 2 - Exemplo de comunicação entre dispositivos formando uma rede ad-hoc [2]

[1] Figura retirada da url http://www.acorn.net.au/telecoms/adhocnetworks/adhocnetworks.cfm
[2] Figura retirada da url http://www.fgan.de/fkie/site/drucken_c68_f6_en.html

Com a comunicação através de troca de mensagens, um nodo pode falhar, ou ficar temporariamente indisponível para executar sua tarefa por motivos de falha do nodo ou falha da rede. Existem diferentes tipos de falhas que podem classificar o comportamento incorreto ou errado de um nodo.

2.2 Tipos de falhas

Segundo Tanenbaum (2007) temos diferentes situações que podemos classificar quando um componente do sistema distribuído não opera de maneira correta. Um sistema distribuído apresenta um defeito quando não pode cumprir as suas promessas. Se um sistema apresenta diferentes serviços, o sistema falha quando um ou mais desses serviços não pode ser mais fornecido por completo.

Um erro é uma parte do estado de um sistema que pode levar a uma falha, e a causa de um erro é denominada defeito (JALOTE, 1994).

As falhas podem ser transientes, intermitentes ou permanentes. As falhas transientes ocorrem uma vez e desaparecem, enquanto que uma falha intermitente acontece e desaparece de maneira aleatória e geralmente são difíceis de diagnosticar. Já as falhas permanentes são aquelas que continuam a existir até que o componente que está falho seja substituído.

Birman (1996) define os tipos de falhas nos seguintes modelos:

- *Halting failures*: neste modelo, um nodo trabalha de maneira correta ou simplesmente para de executar e trava (*crash*) sem executar nenhuma ação incorreta. Falhas deste tipo podem ser detectadas pelo uso de mensagens do tipo *keep alive*. Se após um determinado tempo não houver resposta, o nodo é considerado falho.
- *Fail-stop failures*: este tipo de falha ocorre de modo que a detecção de falha é precisa. Neste modelo, os nodos falham por *crash*. Entretanto, outros nodos, que interagem com o nodo que sofreu a falta tem um modo preciso para detectar sua falha. Por exemplo, um ambiente com *Fail-Stop* pode usar *timeouts* para monitorar o status dos nodos, e nenhum *timeout* ocorre, a menos que o nodo sendo monitorado tenha realmente falhado.
- *Send-omission failures*: Falhas de omissão de envio são causadas normalmente por falta de *buffer* no sistema operacional ou na interface de rede, o que pode causar uma mensagem ser descartada depois do pedido de envio ser feito pela

aplicação mas antes de sair da máquina. Alguns sistemas operacionais reportam esses eventos para as aplicações que requisitaram o envio.

- *Receive-omission failures*: essas falhas ocorrem quando uma mensagem é rejeitada, pela falta de buffer de recebimento ou pela descoberta de dados corrompidos (alterados) na transmissão da mensagem.
- *Network failures*: falhas de rede ocorrem quando a rede perde mensagens enviadas entre um par de nodos. Neste caso, pode-se dizer que a mensagem perde o seu destino.
- *Network partitioning failures*: estas são consideradas as formas mais graves de falhas da rede, onde *uma* rede *R* particiona-se em sub-redes R_1 até R_x, e as sub-redes estão desconectadas entre si, ou seja, não trocam informações. Quando esta falha é reparada, é preciso fazer uma fusão com as sub-redes até se recompor novamente a rede *R*.
- *Timing failures*: isto ocorre quando uma propriedade temporal do sistema é violada. Um exemplo deste tipo de falha é quando um *clock* (relógio) de um nodo possui um valor inaceitável para os outros *clocks* do sistema distribuído, ou quando uma ação é executada muito cedo ou muito tarde. O atraso de uma mensagem por um tempo maior que o atraso tolerado para uma conexão de rede também é considerado uma falha de tempo.
- *Byzantine failure*: este é o termo que engloba uma grande variedade de outros comportamentos de falhas, incluindo alteração dos dados, programas que atuam de maneira maliciosa para forçar o sistema a violar suas propriedades de segurança e confiabilidade.

Entender bem os tipos de falha é importante para compreender as diferenças no modelo de sistema onde o detector de defeitos atua.

Dependendo do tipo de falha considerada, o algoritmo deve estar preparado para lidar com as diferentes situações que podem acontecer. Um algoritmo pode funcionar perfeitamente para um modelo de sistema, mas alterando uma característica do sistema ou tipo de falha que pode acontecer, o detector de defeitos pode não funcionar tão bem.

Outra característica de um sistema distribuído que é importante para o comportamento do detector de defeitos é a maneira como o sistema é conectado ou troca informações. Desta forma, alguns modos de comunicação são apresentados a seguir.

2.3 Comunicação em Sistemas Distribuídos

Segundo Birman (1996), para a comunicação ou coordenação de redes, um sistema distribuído pode utilizar os seguintes tipos de redes: redes reais, redes assíncronas e redes síncronas.

- Redes Reais: são compostas por *Workstations* (estações de trabalho), computadores pessoais e outros tipos dispositivos interconectados por hardware. As propriedades do *hardware* e *software*, tais como a velocidade da rede, atraso de propagação, frequência de erros dos dispositivos, latência de *software*, velocidade de processamento do *hardware* e precisão dos *clocks* são conhecidas pelo projetista do sistema. Estas informações podem ser importantes para resolver alguns problemas, pois é possível prever o comportamento do sistema distribuído.
- Redes Assíncronas: em redes assíncronas, nenhuma suposição é feita sobre a velocidade relativa de comunicação do sistema, processadores ou nodos na rede. Uma mensagem de um nodo n_i para um nodo n_j pode ser entregue em um tempo nulo ou próximo de zero, enquanto que a próxima pode ser entregue em um intervalo de tempo em segundos, horas, dias, semanas, meses ou anos.
- Redes Síncronas: Em sistemas síncronos, existe uma noção precisa do tempo para todos os processos ou nodos no sistema compartilhado. Geralmente os sistemas síncronos assumem um limite no atraso da comunicação entre os nodos, entre os *clocks* dos nodos e outras propriedades do projeto. Esse tipo de sistema é mais difícil de construir do que os sistemas assíncronos, pois exige uma coordenação perfeita entre todos os dispositivos do sistema. A detecção de falhas neste tipo de sistema é mais simples.

Essa classificação é feita tomando por base o tempo de comunicação, sem levar em consideração as falhas de comunicação no sistema. Além da classificação baseada no tempo, conforme descrito acima, outras classificações de redes podem ser feitas, levando em considerações características desejáveis da rede. Um tipo de classificação utilizada é relativa à confiabilidade do sistema. Nessa classificação, um dos modos mais usados em redes assíncronas é a difusão confiável.

2.4 Difusão Confiável

Uma difusão (*broadcast*) confiável (CHANDRA e TOUEG; 1991) é definida como uma primitiva de comunicação para sistemas assíncronos. Uma difusão confiável garante que (i) todo nodo correto entrega o mesmo conjunto de mensagens, (ii) todas as mensagens enviadas por difusão por nodos corretos são entregues, e (iii) nenhuma mensagem falsa é entregue.

Se um nodo correto n_i envia uma mensagem m, então em um momento a mensagem m será entregue para o seu destinatário n_j. Para qualquer mensagem m que tenha sido entregue para um nodo n_j, então esta mensagem foi previamente enviada por um nodo n_i.

A difusão confiável é utilizada em alguns detectores de defeitos, fazendo com que os detectores não fiquem preocupados para lidar com a situação da perda de mensagem.

2.5 Detector de Defeitos

Um detector de defeitos (CHANDRA e TOUEG, 1991) é um módulo responsável por monitorar os nodos de um sistema distribuído. Ele deve obter as informações sobre os nodos do sistema e ser capaz de identificar os nodos que não estão falhos. Os detectores de defeitos atuam obtendo informações dos nodos vizinhos através de troca de mensagens

Dado a natureza assíncrona do sistema distribuído, o conceito de detectores de defeitos não confiáveis foi introduzido por Chandra e Toueg (1996) para contornar o problema da impossibilidade de resolver o consenso num sistema assíncrono sujeito a faltas (Fischer et al., 1985)

A detecção de defeitos é realizada em cada um dos nodos contidos no sistema distribuído. Isto é realizado de maneira independente, onde cada nodo toma as suas decisões de suspeita dos vizinhos baseado nas informações que tem disponível até o momento da decisão.

Cada nodo no sistema é responsável pelas suas informações, devendo manter sempre atualizada suas informações caso seja necessário enviá-las para outro nodo vizinho. Os detectores de defeitos mantêm uma relação dos conjuntos de nodos que são considerados falhos e de nodos que são considerados corretos. Tal relação é considerada como o resultado da ação de um detector de defeitos. Esses resultados podem mudar continuamente, pois nodos podem ser incluídos e retirados das listas de nodos falhos e nodos corretos.

Deve-se observar que nem sempre as listas de todos os nodos no sistema devem ser iguais, pois em um intervalo de tempo *t* um nodo pode ser suspeito por apenas um nodo. Caso ocorra essa diferença, ela será corrigida através da troca de mensagens, em um tempo futuro.

Existem diferentes estratégias de comunicação para construir a noção de estado no detector (FELBER *et al*. 1999). Em redes cabeadas, as mais tradicionais são: *push*, estratégia baseada no envio periódico de mensagens de estado corrente (*I_am_alive!* ou *heartbeat* (AGUILERA et al. 1997); e *pull*, estratégia baseada no envio periódico de mensagens de solicitação de estado (*Are_you_alive?*) com consequente confirmação (*Yes_I_am!*). Em redes móveis sem fio, incluindo MANETs, a mais tradicional é a estratégia baseada em fofoca (GRACIOLI e NUNES 2007)(FRIEDMAN et al. 2007), a qual é baseada no envio periódico de mensagens *gossip* que são repassadas de vizinhos locais para vizinho remotos. Normalmente uma mensagem *gossip* carrega uma lista de identificadores e contadores de *heartbeat*.

Com o surgimento de propostas de detectores de defeitos para sistemas distribuídos que possuem características e propriedades diferentes, algumas regras foram definidas para ser possível a comparação entre os detectores de defeitos. Através dessas regras, foram determinadas classes de detectores de defeitos.

2.6 Classes de Detectores de Defeitos

Chandra e Toueg (1991) caracterizaram classes de detectores de defeitos especificando completude e precisão, que são propriedades que um detector de defeitos deve satisfazer. A propriedade completude (*completeness*) diz que um detector de defeitos eventualmente suspeita de qualquer nodo que tenha ocorrido uma falha do tipo *crash*. A propriedade precisão (*accuracy*) restringe o número de erros que um detector pode cometer. Esses erros são suspeitas erradas de nodos não falhos. As propriedades foram definidas em duas propriedades *completeness* e quatro propriedades *accuracy*. Com isso, são dadas oito classes de detectores de defeitos. Dependendo de como o detector satisfaz as propriedades, ele vai ser classificado em uma das classes determinadas por Chandra e Toueg.

A propriedade completude é considerada de duas maneiras:

- Completude Forte (*Strong Completeness*): Eventualmente, todo nodo que falhar por *crash* vai ser permanentemente suspeito por todos os nodos ativos.
- Completude Fraca (*Weak Completeness*): Eventualmente, cada nodo que falha por *crash* é permanentemente suspeito por alguns nodos ativos.

A propriedade de precisão é considerada de quatro maneiras:

- Precisão Forte (*Strong Accuracy*): nenhum nodo é suspeito antes deste nodo falhar por *crash*.
- Precisão Fraca (*Weak Accuracy*): algum nodo correto nunca é suspeito.
- Eventual Precisão Forte (*Eventual Strong Accuracy*): após um tempo, cada nodo correto não é suspeito por nenhum nodo correto.
- Eventual Precisão Fraca (*Eventual Weak Accuracy*): após um tempo, algum nodo correto nunca é suspeito por nenhum nodo correto.

Os detectores de defeitos são classificados por essas propriedades. A propriedade completude sozinha não seria de muita utilidade, pois seria fácil fazer um nodo suspeitar sempre dos nodos vizinhos, mesmo que de maneira errada. Isso faria com que o sistema cometesse erros a todo o momento. Com isso, a propriedade precisão deve ser satisfeita para restringir os erros cometidos pelos detectores de defeitos.

A propriedade precisão forte é difícil de ser atingida em sistemas reais, porque existem diversos fatores que podem fazer um nodo correto ser suspeito, por exemplo, atraso na entrega de uma mensagem ou mesmo uma perda de uma mensagem.

As propriedades precisão forte e completude forte, são difíceis de serem atingidas. Tais propriedades podem ser alcançadas após um certo tempo, o que fez surgir as propriedades Precisão Forte Eventual e Precisão Fraca Eventual.

Dadas às propriedades, as seguintes classes foram definidas por Chandra e Toueg (1996), como pode ser visto na Tabela 1.

Completude	**Precisão**			
	Forte	Fraca	Eventual Forte	Eventual Fraca
Forte	*Perfect P* (Perfeito)	*Strong S* (Forte)	*Eventually Perfect* ◊*P* (Eventualmente Perfeito)	Eventually Strong ◊*S* (Eventualmente Forte)
Fraca	*Q*	*Weak W* (Fraco)	◊*Q*	Eventually Weak ◊*W* (Eventualmente Fraco)

Tabela 1 - Oito classes de detectores de defeitos em termos de *Accuracy* e *Completeness*

Além de definir as propriedades e classes dos detectores de defeitos, também foi preciso definir métricas para ser possível avaliar se o detector de defeitos funciona corretamente para o sistema distribuído proposto.

2.7 Métricas de Detector de Defeitos

As métricas definidas por Chen et al. (2002) são utilizadas para medir a qualidade de um detector de defeitos. Essas métricas são importantes, pois definem a *QoS* (qualidade de serviço) do detector.

2.7.1 Métricas Primárias

As métricas primárias medem a velocidade e precisão de um detector de defeitos. São métricas primárias *Detection Time* (T_D), *Mistake recurrence time* (T_{MR}) e *Mistake duration* (T_M).

- Tempo de detecção (*Detection Time* - T_D): Mede a velocidade de detecção de defeitos, ou seja, T_D é o tempo decorrido desde o instante em que n_i falha (colapso, queda, *crash*) até o instante em que n_2 suspeita permanentemente de n_i. Probabilisticamente, T_D é a variável aleatória associada ao tempo decorrido do instante em que n_i falha (colapso) até o instante em que ocorre a última transição do estado Confiável (*Trust*) para o estado de suspeito (*Suspect*), sem haver mais transições. Essa transição entre os estados de *Trust* para *Suspect* é chamada de *S-Transição*.
- Tempo para recorrência ao erro (*Mistake recurrence time* - T_{MR}): Mede o tempo entre dois erros consecutivos cometidos pelo detector (suspeitas incorretas). Probabilisticamente, T_{MR} é a variável aleatória que representa o tempo entre duas *S-Transições* consecutivas.
- Duração de um erro (*Mistake duration* – T_M): Mede o tempo que o detector de defeitos leva para corrigir um erro. Probabilisticamente, T_M é a variável aleatória correspondente ao tempo decorrido entre uma *S-Transição* e a próxima *T-Transição* (passagem do estado Suspeito para o estado Confiável).

2.7.2 Métricas Secundárias

As métricas secundárias são medidas que podem ser geradas através da computação das métricas primárias, e é devido as métricas T_M e T_{MR} gerarem as outras métricas, que estas são chamadas de primárias.

- Taxa de erros média (*Average mistake rate* – λ_M): Mede a taxa que o detector de defeitos erra, isto é, representa o número médio de *S-Transições* por unidade de tempo.
- Probabilidade de uma resposta precisa (*Query accuracy probability* - P_A): É a probabilidade de que a saída do detector de defeitos esteja correta num dado tempo aleatório.
- Duração de um período bom (*Good period duration* - T_G): Mede a duração de um período bom, ou seja, o tempo decorrido entre uma *T-Transição* e a próxima *S-transição*.
- Duração de um período bom à frente (*Forward good period duration* – T_{FG}): É uma variável aleatória que representa o tempo restante do período bom, ou seja, dado um instante aleatório, representa o tempo restante até o momento da próxima *S-Transição*.

2.8 Diferentes Propostas de Detectores de Defeitos

Nesta seção são apresentadas diferentes propostas de detectores de defeitos para sistemas distribuídos.

Gracioli e Nunes (2007) apresentam uma comparação entre estratégias de detecção de defeitos para redes móveis sem fio, onde se percebe que para não depender de hierarquia ou topologia fixa, grande parte das soluções tem como base o algoritmo *Gossip* (RENESSE et al. 1998). O *Gossip* é um algoritmo epidêmico baseado no fenômeno social chamado fofoca, onde um nodo difunde (por *broadcast* ou *multicast*) para seus vizinhos locais (dentro do seu alcance de transmissão) informações sobre um grupo de nodos vizinhos (locais ou remotos). Neste algoritmo, quando um nodo detector de defeitos fica muito tempo sem receber notícias sobre um determinado nodo (equivalente a um tempo nomeado *Tcleanup*), o detector suspeita de falhas neste nodo. O ponto chave do algoritmo é que ele independe de topologia, pois um nodo conhece seus vizinhos ao longo do tempo através de mensagens que passam de vizinhos para vizinhos.

Lorenzi et al. (2007), chegou a conclusão que o detector de defeitos de Sridhar teve um desempenho melhor que os outros detectores de defeitos para detecção de falhas.

2.8.1 Detector proposto por Hutle

Hutle (2004) propôs um algoritmo detector de defeitos para nodos em uma rede *wireless ad-hoc* não totalmente conectada onde cada nodo tem um número de vizinhos limitado. Neste algoritmo, o número de nodos total no sistema não precisa ser conhecido, e as ligações podem falhar fazendo com que as redes fiquem particionadas em componentes. O algoritmo suporta o tipo de falha por *crash.*

Um nodo necessita conhecer o *jitter* (variação de atraso) na comunicação com os seus nodos vizinhos. Esse conhecimento na variação de atraso implica em um sincronismo na comunicação direta entre dois nodos vizinhos. O sincronismo é conseguido reservando um limite na largura de banda para o detector de defeitos e limitando o número de mensagens. Os nodos trocam informações apenas com seus vizinhos através de difusão via *broadcast*. Os nodos possuem uma informação mais precisa sobre os nodos mais próximos do que a informação sobre os nodos que estão à maior distância. Neste algoritmo, cada nodo n_i mantém uma lista que contem as seguintes informações: para todo outro nodo n_2 é mantido um contador de *heartbeats* que tem o mais recente *heartbeat* recebido de n_2, um contador de distância (número de *hops*) que contém a estimativa sobre a atual distância de n_2 e um vetor de *timestamp* que mantém o último *round* que n_i recebeu um *heartbeat* de n_2.

Através do contador de distância, é possível obter uma noção de quantos *hops* de distância está um nodo de outro. Com isso o tempo de detecção de defeito de um nodo pode ser ajustado conforme a distância atual presente no contador. Para detectar um defeito é mantido um conjunto de todos os nodos que o detector de defeitos não suspeita, chamado de lista de detecção. Consequentemente, um nodo irá suspeitar de outro se este não estiver na sua lista de detecção. É possível alterar o algoritmo para os sistemas onde *links* (ligações) podem ser recuperados. Entretanto, em tais sistemas a definição de acessibilidade não é tão óbvia, desde que uma aplicação do detector de defeitos possa usar um algoritmo de roteamento para a comunicação. A relação de informação mais precisa sobre os nodos que estão em outros níveis de sub-redes também dependeria do comportamento deste algoritmo de roteamento. Segundo Hutle, uma solução seria procurar algoritmos que dependem do detector de defeitos, mas operam diretamente nas redes esparsas.

2.8.1.1 Modelo de Sistema

Sistema distribuído consiste de um conjunto de *n* nodos de uma rede não totalmente conectada com um limite no número de vizinhos. Esta limitação é devido às limitações de *hardware* (canais de recebimento ou *buffers*) que permite que apenas alguns nodos mantenham a conexão com um conjunto de outros nodos, esta sendo uma propriedade natural de redes *low-level* (HUTLE, 2004).

O número de nodos em todo o sistema não precisa ser conhecido. As ligações entre os nodos e os nodos podem falhar por falha do tipo *crash*. Existe um *clock* global discreto com valores de um conjunto *T*, usado apenas para análise, mas não disponível para o nodo. Os nodos não precisam conhecer o limite no atraso de comunicação (*delay*) entre nodos arbitrários, mas precisa conhecer apenas o limite no *jitter* na comunicação com os seus nodos vizinhos. Isto implica em uma comunicação síncrona e é uma severa restrição. Este sincronismo é requerido apenas entre vizinhos diretos, o que faz com que a comunicação entre os nodos que não são vizinhos diretos seja parcialmente síncrona.

2.8.1.2 Algoritmo

Cada nodo n_i possui uma tabela *heartbeat* para cada nodo n_j conhecido: (*i*) um contador *heartbeat* que contém o mais recente *heartbeat* de n_j, (*ii*) um contador de distância que contém a estimativa de n_i sobre a atual distância para n_j e (*iii*) um *timestamp* que mantém a última rodada que n_i recebeu um novo contador de n_j. Existe um conjunto chamado $detectedn_i$ que contêm todos os nodos que o detector de defeitos n_i não suspeita, e isto é o resultado do detector de defeitos. Neste algoritmo, n_i envia uma lista para seus atuais vizinhos, e recebe uma tarefa que é atualizar a lista local quando n_i recebe um novo *heartbeat* dos vizinhos. Inicialmente, n_i conhece apenas ele mesmo, e a cada *T* etapas, n_i aumenta o seu próprio contador, que é também usado como um o número do *round* local. A cada Δ^k rounds, todos os nodos conhecidos com distância *k* são colocados no conjunto $unsentn_i$ para que sejam enviadas apenas as mensagens deste conjunto. Com isso ele garante que cada *heartbeat* de um nodo com distância *k* é transmitido no mínimo a Δ^k *rounds*. Deste conjunto $unsentn_i$, o *id* do nodo, *heartbeat*, e a distância Δ+1 dos nodos com menor distância de n_i são enviados e removidos de *unsent* n_i em cada rodada. Se n_i não recebe uma atualização de outro nodo n_j por um tempo muito longo, sendo n_j um nodo que ele previamente detectou, n_i irá suspeitar do nodo n_j. O receptor da tarefa de n_i aumenta o contador de distância para um a cada mensagem

que ele recebe. Se o contador de distância é menor que sua estimativa, ele adapta a nova distância. Quando ele recebe um *heartbeat* mais novo que os seus, ele adapta o seu *heartbeat* para o *heartbeat* mais atual.

2.8.1.3 Propriedades

O algoritmo proposto por Hutle é da classe de detectores de defeitos é denotada por $\Diamond P$ (eventualmente perfeito). Como a maioria dos outros algoritmos, o algoritmo de Hutle tem como saída uma lista de nodos. Visto que não requer que os nodos conheçam todos os outros nodos do sistema, ou mesmo n a priori, o detector de defeitos não terá como saída todos os nodos suspeitos. Ao invés disso, ele tem como saída uma lista de nodos, que é complementar e equivalente à lista de suspeitos, no sentido de que é possível determinar se um nodo específico está correto ou não. Assim, um nodo n_i suspeita de outro nodo n_j, se ele não está na sua lista de detecção.

2.8.2 Detector proposto por Sridhar

Sridhar et al. (2006) propuseram um algoritmo detector de defeitos na presença de mobilidade em sistemas distribuídos. Este algoritmo é voltado para detecção de defeitos em redes locais, sendo um eventual perfeito detector de defeitos local que tolera a mobilidade dos nodos. Este algoritmo tem Completude forte (*Strong Completeness*) e eventual forte exatidão local (*eventual strong local accuracy*). Este detector de defeitos mantém apenas informação sobre seus vizinhos imediatos, reduzindo a quantidade de memória necessária para armazenar dados sobre os nodos vizinhos (que é um recurso escasso em alguns sensores), em comparação com detectores de defeitos que armazenam informações sobre todos os nodos da rede.

Este protocolo é composto por duas camadas independentes. A primeira camada é o detector de defeitos local, responsável por construir a lista de suspeitos entre os vizinhos de um dado nó, que é chamada de Camada de Detector de Falhas Local (*LFD – Local Failure Detection Layer*). A segunda camada é a que detecta a mobilidade dos nodos através da rede, e esta camada é chamada de Camada de Detecção de Mobilidade (*MB – Mobility Detection Layer*). Juntas, essas duas camadas satisfazem as especificações de redes móveis, que são: forte Completude local, exatidão eventual forte local e localização suspeita.

- A forte Completude local (*strong local completeness*) afirma que após um nodo n_i falhar, se o mesmo não retornar em um determinado intervalo de tempo, o nodo n_i é considerado permanentemente suspeito por n_j.
- A exatidão eventual forte local (*eventual strong local accuracy*) refere que deve existir um tempo onde depois que nodos que estão ativos não são mais suspeitos de nenhum outro nodo ativo da vizinhança, os nodos atualizam sua visão de quem são os seus vizinhos.
- A localização suspeita (*suspicion locality*) refere-se há um tempo após o qual nodos não falhos suspeitam apenas de nodos que estão na vizinhança local.

2.8.2.1 Modelo de sistema

É considerado um conjunto de nodos *N* conhecido. Os nodos são organizados em redes de *multi-hop*. Existe uma comunicação direta entre os vizinhos da rede, que forma um Grafo G, que pode ser denotado por G = (*N*, *E*), onde *E* representa o conjunto de *links* entre os nodos de *N*. A topologia do grafo é dinâmica. Cada nodo na rede tem seu próprio *clock*, e não existe um *clock* global no sistema. Não é adotada uma sincronização de *clock* no sistema. Os nodos comunicam-se com os outros nodos através de troca de mensagens. As mensagens podem não ser entregues, e podem ser perdidas com uma probabilidade conhecida. Existe um atraso médio na transmissão da mensagem (*delay*) que é conhecido. Os nodos podem movimentar-se para dentro ou para fora da vizinhança dos outros nodos. É considerado o modelo de mobilidade passiva (*passive mobility*) onde os nodos que estão se movendo não sabem que estão em movimento, e com isto não notificam os outros nodos de sua movimentação. Os nodos movimentam-se lentamente. Um nodo falha parando de enviar mensagens para seus vizinhos.

2.8.2.2 Camada de detecção local

A camada de detecção de falha local funciona da seguinte maneira: em cada nodo n_i, a camada de detector de defeitos mantém informações de quais nodos em um conjunto $nbrsn_i$ (*neighbours* de n_i) são suspeitos de falhar. Esta camada também monitora quando cada um destes nodos foi recentemente suspeito. Este momento é marcado como tdn_j para nodos suspeitos n_j, e este *timestamp* é um *timestamp* local. A camada *LFD* pode ser implementada usando qualquer algoritmo de detectores de defeito da classe $\Diamond P$. Dependendo da necessidade da aplicação, o projetista pode escolher uma de várias estratégias para construir a lista de

suspeitos da lista de vizinhos. Esta camada não tem nenhuma preocupação com os vizinhos. As estratégias mais populares para garantir a detecção de falhas são as estratégias baseadas em *heartbeats* (AGUILERA et al, 1997), *timeouts* adaptativos (FETZER et al, 2005), *pinging* (GUPTA et al, 2001) e *lease* (BOICHAT et al, 2002).

2.8.2.3 Camada de detecção de mobilidade

Cada nodo n_i executa esta camada para compartilhar sua visão de nodos falhos com o resto da rede e corrigir suas suspeitas baseadas nas informações sobre o que os outros nodos conseguem detectar. Devemos lembrar que existe apenas uma mensagem *gossip* na rede em um dado momento. No momento de posicionamento, algum nodo é escolhido como o iniciador. O iniciador para rodadas posteriores é nomeado no final de cada rodada.

Quando um nodo ocioso n_i recebe uma mensagem *gossip* de um nodo n_j, n_i coloca n_j como seu guardião (gerador) e muda para o estado ativo. Depois de acordar, é examinado o grupo suspeito que esteja contido na mensagem. Ele compara a nova lista recebida de suspeitos com sua própria lista de suspeitos para ver se existe conflito de entradas. Caso a lista recebida contenha qualquer vizinho de n_i que não esteja na lista de suspeitos de n_i, então n_i verifica há quanto tempo o nodo está sendo marcado como suspeito. Baseado neste tempo, o módulo detector de defeitos de n_i decide se o nodo suspeito deve permanecer na lista de suspeitos ou não. Se o tempo decorrido desde a última comunicação de n_j é menor que o tempo de suspeita de n_j então exonera n_j, ou seja, este nodo é removido da lista de suspeitos.

Uma situação que pode ocorrer é quando um nodo n_i suspeita de um nodo n_k que é vizinho de n_j, onde n_i e n_k não estão diretamente ligados, assim n_i não enxerga n_k. Quando n_i recebe a lista de n_j ele não suspeita mais de n_k, pois tem informação de que n_k está vivo, e sai da lista de suspeitos de n_i.

Quando um nodo recebe uma mensagem *gossip* e não tem nenhum vizinho para enviar a mensagem, então ele envia a mensagem de volta para quem o enviou. Mais uma vez cada nodo atualiza sua lista de suspeitos baseado na lista de suspeitas recebida pela mensagem *gossip*. Quando um nodo recebe de volta as mensagens de todos os seus filhos, ele envia de volta a mensagem para o seu guardião. Lembrando que cada nodo só espera a resposta dos vizinhos conhecidos. O *round* de *gossip* termina quando o nodo que iniciou escuta a mensagem de volta de todos os seus vizinhos. Assim no final de cada *round*, cada nodo tem uma visão atualizada dos seus vizinhos, se estão ativos ou não.

A fase de expansão do *gossip* é usada para exonerar nodos. Um nodo n_k inferior na árvore de propagação exonera um nodo n_j que é suspeito por um de seus ancestrais n_i. Na fase de encolhimento, n_i corrige sua lista de suspeitos e sua lista de vizinhos para remover n_j. Porém, mesmo se n_j não foi exonerado, n_i ainda vai precisar remover n_j da sua lista de vizinhos, senão a camada de detector de defeitos de n_i vai continuar suspeitando de n_j, e n_i vai continuar em um estado ruim, impossibilitado de fazer um progresso local, mesmo tendo todos os seus vizinhos ativos.

2.8.3 Detector proposto por Pierre Sens

Pierre (2008) propôs um algoritmo de detector de defeitos para redes móveis e dinâmicas e desconhecidas. O algoritmo não é baseado em temporizadores e não requer nenhum conhecimento a respeito da composição do sistema e nem da sua cardinalidade. O processo de detecção de falhas é baseado apenas na percepção local que um nodo tem da rede e não na troca de informações globais.

No inicio do funcionamento do algoritmo, o nodo conhece apenas a si mesmo e periodicamente troca informações com os seus vizinhos em mensagens do tipo *Query-Response*. Com base nas respostas obtidas e no conhecimento parcial que o nodo tem de sua vizinhança, o nodo e capaz de suspeitar de outros nodos ou anular informações incorretas.

As mensagens são enviadas através das mensagens *Query* e podem atingir todos os nodos do sistema, se o sistema atender as condições de conectividade da propriedade de rede com *f-cobertura*. A *f-cobertura* é uma propriedade que garante a existência de um caminho entre quaisquer dois nós ativos da rede, apesar da ocorrência de *f* falhas ($f < n$), onde *n* é o número de nodos do sistema.

O detector proposto por Pierre (2008) implementa as propriedades da classe $\Diamond S$, se os nós do sistema apresentarem certas propriedades comportamentais. As propriedades definidas são quatro: *Propriedade de inclusão, Propriedade de mobilidade, Propriedade de Receptividade, Propriedade de Receptividade para Mobilidade.* A *Propriedade de inclusão* permite que o nó seja reconhecido pelos demais. Para ser reconhecido o nó deve ter interagido e enviado pelo menos uma mensagem na rede. A *Propriedade de mobilidade* estabelece que um nó móvel deve se reconectar a rede por um período de tempo suficiente para atualizar o seu estado quanto a suspeita de falhas e correções de suspeitas equivocadas. A *Propriedade de Receptividade* determina que, após um tempo, a comunicação entre um determinado nó e os demais nós da sua vizinhança serão mais rápidas do que quaisquer outras comunicações na

sua vizinhança. A *Propriedade de Receptividade para Mobilidade* determina que ao menos um nó correto satisfaz a propriedade de receptividade e, além disso, após um tempo, sua vizinhança será composta por nós que não saem da sua área de cobertura.

O modelo de sistema que distribuído considerado é um sistema distribuído dinâmico formado por um conjunto finito de $n > 1$, $\prod = \{n_1,...,n_x\}$ nós móveis. Como a rede é desconhecida, os nós não conhecem o total de n nodos do sistema. Um nodo n_i conhece apenas um subconjunto de $\prod$. Os nós se comunicam pela troca de mensagens através de uma rede sem fio, por rádio frequência. Nenhuma hipótese temporal é feita com respeito a realização das ações efetuadas pelos nós ou pelos canais, ou seja, o sistema é assíncrono.

Um nodo pode falhar por parada (*crash).* Um nodo *correto* não falha durante a execução do sistema. Se o nodo falhar, então é considerado *falho*. Existe um número máximo de nodos autorizados a falhar no sistema ($f < n$), e f é conhecido por todos os nós.

O sistema pode ser representado por um grafo de comunicação $G(V,E)$ onde $V \subseteq E$ representa o conjunto de nós e E, o conjunto de ligações. A comunicação entre os nodos é feita por difusão (*broadcast*) ou diretamente (*ponto a ponto*). Os canais de comunicação são confiáveis e bidirecionais. Os canais não alteram, não criam, nem perdem as mensagens. Assim, uma mensagem m enviada por n_i através de difusão é recebida por todos os vizinhos corretos de n_i.

2.8.3.1 Detector de Defeitos

O detector de defeitos apresentado fornece as informações sobre os nodos que estão falhos. O detector proposto por Pierre é um detector não confiável, pois ele pode se equivocar. O detector de falhas pode continuamente adicionar e remover nodos da sua lista de suspeitos.

Este detector de defeitos garante que todo nodo falho será finalmente suspeito por todo nodo correto (*completude forte)* e existirá um instante a partir do qual um nodo correto não será considerado suspeito por nenhum outro nodo correto (*exatidão fraca após um tempo*).

2.8.3.2 Detector de Defeitos Assíncrono da Classe $\Diamond S$

O principio do algoritmo proposto por Pierre é trocar mensagens na rede através de um mecanismo de *Query-Response* (pergunta e resposta), onde essas mensagens contem informações sobre suspeitas de falhas de nós.

O algoritmo é executado em rodadas e a cada rodada, se o nodo não falhar, o nodo envia a seus vizinhos uma mensagem do tipo *Query*. O intervalo de tempo entre duas rodadas

consecutivas é finito, mas arbitrário. As informações contidas na mensagem *Query* são: conjunto de nós que são atualmente suspeitos de terem falhado (*suspected*) e o conjunto de equívocos (*mistake*). O conjunto *mistake* são os nós erroneamente suspeitos de falhar nas rodadas anteriores.

Cada nó possui um contador atualizado a cada rodada, garantindo que cada nova informação sobre suspeita de falha ou equivoco é etiquetada com o valor corrente do contador de nós. Este mecanismo evita que informações não atualizadas sejam consideradas válidas por outros nós.

A cada mensagem *Query* recebida de um nó da vizinhança, n_i confirma a recepção com uma mensagem *Response*. Uma *Query* é satisfeita por um nó quando este receber pelo menos $d - f$ mensagens *Response*[3]. Deve-se observar que um par *Query-Response* é identificado como um par único no sistema. Um nó enviará outra mensagem *Query* apenas depois que a precedente terminar.

Para implementar o algoritmo que reconheça mensagens de nodos desconhecidos, é definida duas propriedades comportamentais que asseguram que a implementação satisfaça as propriedades da classe $\Diamond S$.

A *Propriedade de inclusão* (*MP*) precisa ser satisfeita por todos os nós do sistema. A propriedade afirma que, para ser membro do sistema, um nodo n_i (correto ou não) precisa interagir pelo menos uma vez com outros nodos da sua cobertura ($rangen_i$) enviando-lhes uma mensagem *Query* por difusão. Esta mensagem deve ser recebida e figurar no estado de pelo menos um nodo correto, além de n_i.

A *Propriedade de receptividade* (*RP*) afirma que depois de um intervalo de tempo finito u, o conjunto *d-f* respostas recebidas por um vizinho de n_i a sua última mensagem *Query* sempre inclui a resposta de n_i.

O algoritmo possui duas tarefas. A tarefa T1 é responsável pela geração das suspeitas e possui laço infinito. A tarefa T2 é responsável pela geração de informações de revogação de suspeitas e pela propagação das informações recentes na rede.

A cada rodada da tarefa T1, uma mensagem *Query* é transmitida a todos os nós da área no alcance de transmissão. O nodo então espera por pelo menos *d-f* respostas a sua pergunta. Um nodo será considerado suspeito de falha se: (i) n_i conhece n_j, (ii) n_i ainda não suspeitou anteriormente de n_j (n_j não pertence ao conjunto de suspeitos) e (iii) n_i não recebeu de n_j um *Response* como resposta a sua última *Query*.

---

[3] d é a densidade da rede e f é o número de caminhos independentes que liga dois nós.

Com esta condição, se uma informação *mistake* referente à n_j existe em $mistaken_i$, a informação será removida. O contador de rodadas $countern_i$ é atualizado com um valor superior ao contador do *mistake* em questão.

A tarefa T2 é responsável por tratar a recepção de mensagens *Query* enviadas por nós vizinhos. Essa tarefa trata as informações referentes a suspeitas de falhas e as referentes a equívocos. Para cada nodo n_k que existe no conjunto $suspectedn_j$, da mensagem *Query* recebida de n_j, n_1 inclui n_k no seu conjunto $suspectedn_i$ somente se a condição é satisfeita: n_i recebeu uma informação sobre o status de n_k que é mais recente do que aquela que ele já possui em seus conjuntos $suspectedn_i$ e $mistaken_i$. A informação é considerada mais recente nos seguintes casos: (i) n_k nunca foi suspeito por n_i ou (ii) n_k pertence a um dos conjuntos de n_i mas com um valor de etiqueta (*counter)* inferior aquele enviado por n_j ($countern_k$) em sua *Query*. Quando isto ocorre, n_i remove também o nodo n_k do seu conjunto $mistaken_i$.

Quando um nodo n_i pertence ao conjunto de suspeitos de uma mensagem *Query* da tarefa T2, ele irá gerar um novo equívoco, adicionando ele próprio ao seu conjunto $mistaken_i$ com um valor de contador mais recente que o associado a sua suspeita.

Foi proposta também uma extensão do detector de falhas para que ele consiga suportar a mobilidade dos nós.

2.8.3.3 Propriedades Comportamentais para Mobilidade

Durante a execução do algoritmo, um nodo n_i pode continuamente se deslocar e se reconectar ou terminar por falhar. Entretanto, para o nodo n_i atualizar suas informações ele deve participar de uma rodada de *Query-Response*. Sem que isto ocorra não existe uma garantida das propriedades de *completude* e *exatidão* do detector. Para garantir esse comportamento, foi definido uma propriedade de mobilidade, que deve ser satisfeita por todos os nós móveis.

2.8.3.4 Propriedade de Mobilidade

A propriedade assegura que, depois de ter se reconectado a rede, haverá um tempo no qual um nodo n_i terá recebido *Query* dos nós de sua nova vizinhança e dentre estas, pelo menos uma foi enviada por um nó correto, diferente de n_i. Dado que estas mensagens contem informações sobre falhas e equívocos, n_i poderá atualizar o seu estado.

Considerando que seja satisfeita a propriedade de inclusão, depois que se reconectar, haverá um instante a partir do qual n_i terá uma interação pelo menos uma vez com outros

nodos de sua vizinhança ($rangen_1$) enviando por difusão uma mensagem *Query* que vai ser recebida pelo menos por um nodo correto em $rangen_i$, além de n_i.

2.8.3.5 Detector de Falhas da Classe ◊*S* para Redes Móveis Desconhecidas

Pierre propôs uma extensão do detector de falhas Assíncrono da Classe ◊*S* para suportar a mobilidade. Na sua proposta, quando um nodo n_i move-se para outro *range*, ele será suspeito de falhar por aqueles nós do *range* que ele saiu, porque a partir do momento que ele sai do *range* antigo, ele não responde mais as mensagens *Query*. Como ele não vai mais responder a mensagem, será colocado sobre suspeita de falha, e essa informação de suspeita de falha de n_i vai ser disseminada pela rede através das mensagens *Query* subjacentes.

No momento que o nodo n_i volta a se reconectar a rede, ele acabará recebendo as mensagens de *Query*. Então ele corrige a falsa suspeita incluindo-se no conjunto $mistaken_i$, que será enviado na próxima mensagem *Query* que ele enviará e que se propagará pela rede. Esse comportamento também leva ao nodo n_i suspeitar dos nodos do seu antigo $rangen_i$, pois estes estão no seu grupo $knownn_i$ (nodos conhecidos pelo nodo n_i). O nodo n_i também acaba suspeitando dos outros nodos do seu antigo $rangen_i$ e incluindo eles em suas mensagens *Query*.

Quando o nodo voltar a reconectar-se na rede, enviara a mensagem para os nodos que da mesma maneira irão incluir-se nos seus grupos *mistake* e irão corrigir o equívoco.

2.9 Considerações sobre as diferentes propostas

Hutle (2004) propõe um detector de defeitos que envia um número conhecido de mensagens por rodada, o que torna fácil de configurar a quantidade de banda necessária para a transmissão de mensagem de cada nodo. O algoritmo também traz a vantagem de ter um *timeout* vinculado ao *jitter* que é adaptado quando um nodo não responde no tempo necessário. Com as informações sobre o *jitter* e largura de banda utilizada, o detector de defeitos consegue ter um sincronismo entre os vizinhos locais.

Sridhar (2006) demonstra uma solução dividida em duas camadas, que são a camada de detecção local e a camada de mobilidade. Sua proposta permite que seja usado qualquer detector de defeitos da classe ◊P para a detecção local. A camada de detecção de mobilidade é responsável por identificar a diferença entre os nodos que falharam dos nodos que estão em movimento.

Pierre (2008) propôs um detector de defeitos que lida com situações onde o número de membros não é conhecido e a conectividade da rede não é composta. O algoritmo proposto por Pierre não precisa de temporizadores para fazer a detecção de falhas, pois este algoritmo lida com uma detecção de falha através de trocas de mensagens *Query-Response*. Esta proposta implementa detectores de falhas da classe $\Diamond S$ se as propriedades comportamentais de receptividade, inclusão e mobilidade propostas forem satisfeitas no sistema onde o detector vai atuar.

As propostas tratam de detecção de defeitos em sistemas distribuídos e redes MANETs. Cada autor propõe uma maneira diferente de identificar os defeitos, tentando melhorar algum aspecto da detecção de defeitos. Porém, os detectores de defeitos para redes MANETs descritos não exploram informações mais precisas que identificam uma movimentação dos nodos que estão dentro do alcance de transmissão. Este trabalho propõe um detector de defeitos que adapta o tempo para suspeitar de um nodo, sem precisar de informação do número de *hops*, e utiliza a informação da potência do sinal para detectar a mobilidade dos nodos e possibilitando, inclusive identificar um padrão de movimentação dentro do alcance de transmissão.

3 Detector de defeitos atento

Neste capítulo, apresenta-se a proposta de Detector de Defeitos Atento (DDA) não confiável para MANETs baseado em troca de mensagens *gossip* entre os nodos da rede e que utiliza a potência do sinal recebido para detectar a mobilidade dos nodos. Através da potência do sinal recebido o algoritmo calcula a região onde um nodo vizinho está, e utiliza esta informação para obter uma maior precisão na detecção de defeitos.

Este algoritmo também identifica um padrão de movimentação dos nodos, dispensando informações como o número de *hops*, atraso na transmissão de mensagens, sem precisar modificar o *Timeout* do sistema na detecção de falha. Cada nodo do sistema é instrumentado com uma instância de um nodo detector de defeitos e no decorrer da apresentação é utilizado apenas "nodo" para se referir ao nodo da rede do detector de defeitos. Os nodos podem movimentar-se pela rede e entrar e sair do alcance de transmissão de outros nodos vizinhos. O detector de defeitos não é confiável, pois é permitido que o detector cometa erros em suas suspeitas de falhas. Isto quer dizer que um nodo pode entrar e sair das listas de nodos ativos e suspeitos, de acordo com a sua decisão em cada instante *T*.

Este capítulo é composto por subseções, onde na subseção 3.1 é apresentado o modelo de sistema que define as características do ambiente ou rede MANET onde o DDA pode atuar. Após a definição do modelo de sistema é apresentado o funcionamento do DDA e suas definições na subseção 3.2. Depois de apresentar o funcionamento são apresentadas as estruturas de dados para suportar mobilidade na subseção 3.3 e após é apresentado o algoritmo básico de detecção de defeitos na subseção 3.4. Por último são apresentadas as propriedades do detector de defeitos na subseção 3.5.

3.1 Modelo de Sistema

O modelo de sistema distribuído considerado neste trabalho é assíncrono e composto por um conjunto finito de nodos móveis $\prod = \{n_i, \ldots, n_k\}$, onde $n > 1$. Cada nodo tem sua própria memória, unidade de processamento e relógio local. Cada nodo possui acesso apenas ao seu próprio relógio (*clock*), e não existe uma sincronização de relógios no sistema.

Os nodos se comunicam através de troca de mensagens, usando difusão (*broadcast*) via rádio com alcance de transmissão finito, onde não existe obstáculos entre os nodos e não existe perda de sinal. Uma mensagem *m* enviada por um nodo n_i somente pode ser recebida por outro nodo n_j que esteja dentro do seu alcance de transmissão quando *m* for enviada. A

topologia de rede é dinâmica e os canais de comunicação são bidirecionais e confiáveis (não alteram, não criam e não perdem mensagens). Os nodos podem movimentar-se pela rede, e é considerado o padrão de mobilidade passiva (Sridhar, 2006), onde um nodo que está em movimento não sabe que está em movimento, e com isso não comunica os outros nodos de sua movimentação. Os nodos podem movimentar-se em velocidades e caminhos aleatórios.

Todo nodo móvel n_i percorre, num tempo finito, pelo menos uma trajetória L tal que ao longo de L o nodo n_i entra no alcance de transmissão de pelo menos um nodo n_j e recebe uma mensagem m de n_j, e pelo menos um nodo n_j recebe uma mensagem de n_i. Devido à mobilidade dos nodos, o sistema pode ter um período de particionamento temporário, onde um ou mais nodos do sistema podem ficar isolados, sem haver uma comunicação entre os nodos. Porém, após um intervalo de tempo, o nodo volta novamente ao alcance de transmissão de outro nodo para fazer a comunicação e assim atualizar as suas informações.

Os nodos são auto-organizáveis e adaptam-se de forma autônoma a falhas do tipo *crash*, onde os nodos param de executar a sua ação, ou seja, param de enviar mensagens para os seus nodos vizinhos. Existe o problema de distinguir entre um nodo que falha e um nodo que sai do alcance de transmissão de todos os outros nodos por estar em movimento. Isto ocorre quando um nodo n_i não tem nenhum nodo n_j em sem alcance de transmissão. Assim, n_i não vai conseguir trocar informações com outro nodo n_j, e vai acabar sendo suspeito de falha por outros nodos n_j, pois para o nodo n_j, n_i parou de enviar mensagens.

Por exemplo, a Figura 3 mostra um nodo n_k que está ativo (não falho), mas está fora do alcance de transmissão dos nodos. Este cenário induz os nodos a suspeitarem de o nodo estar falho após um determinado tempo, mesmo o nodo estando em um estado correto.

Figura 3 - Nodo n_k ativo e fora do alcance de transmissão dos nodos.

Outro exemplo é apresentado na Figura 4, onde o nodo n_k está falho por *crash* e está dentro do alcance de transmissão. Esta é outra situação onde os nodos não sabem se o nodo realmente falhou ou saiu do alcance de transmissão, pois o nodo parou de enviar mensagens. Após um tempo limite, o nodo vai acabar sendo suspeito.

Figura 4 – Nodo n_k está falho e dentro do alcance de transmissão dos nodos.

Como cada nodo no sistema possui seu próprio relógio local e não sincronizado, um nodo n_k que falha no intervalo T pode ser detectado como falho por um nodo n_i em um tempo que pode ser diferente por um nodo n_j devido a cada nodo ter o seu próprio *clock*. Para detectar as falhas, cada nodo no sistema distribuído executa um serviço detector de defeitos.

O alcance de transmissão (*range*) é o limite máximo de distância que uma mensagem emitida por um nodo n_i pode chegar. Este alcance de transmissão é igual para todo $n_i \in \prod$.

Se a distância entre dois nodos n_i e n_j for menor que o alcance de transmissão de n_i, então n_j está no alcance de transmissão de n_i *e* n_i está no alcance de transmissão de n_j. Uma mensagem enviada por um nodo n_i será recebida por todos os nodos que estão no alcance de transmissão de n_i quando a mensagem for enviada por n_i.

Na Figura 5, podemos ver a relação entre distância e Potência de sinal recebido, considerando uma fonte de transmissão com potência de sinal de 100db. Conforme a antena de medição se afasta da fonte do sinal, a potência medida descresce de forma exponencial.

A potência perdida P_L na transmissão ao ar livre é dada por[4]:

$$P_L(dB) = 10.\log\left(\frac{P_T}{P_L}\right) = -10.\log\left(\frac{\lambda^2}{(4\pi d)^2}\right) \quad \textbf{(1)}$$

resolvendo-se a equação(1), tem-se que:

$$P_L = 20.\log\left(\frac{4\pi d}{\lambda}\right) \quad \textbf{(2)}$$

ou

$$P_L = 20\log(d) + 20\log\left(\frac{4\pi}{\lambda}\right) \quad \textbf{(3)}$$

substituindo-se a relação $\lambda = \frac{c}{f}$ na equação (3), tem-se:

$$P_L = 20\log(d) + 20\log\left(\frac{4\pi}{c} f\right) \quad \textbf{(4)}$$

onde *c* é a velocidade da luz, e *f* é a frequência de transmissão.

Separando-se os termos de (4), tem-se:

$$P_L = 20\log(d) + 20\log(f) + 20\log\left(\frac{4\pi}{c}\right) \quad \textbf{(5)}$$

Na equação (5), são usadas as medidas de distância em metros e freqüência em Hertz. Ao se usar a distância em kilômetros e a freqüência em MHz, e substituindo-se a velocidade da luz $c = 3\times10^8 m/s$ em (5), obtém-se:

$$P_L = 20\log(d) + 20\log(f) + 32{,}44^{5\,6} \quad \textbf{(6)}$$

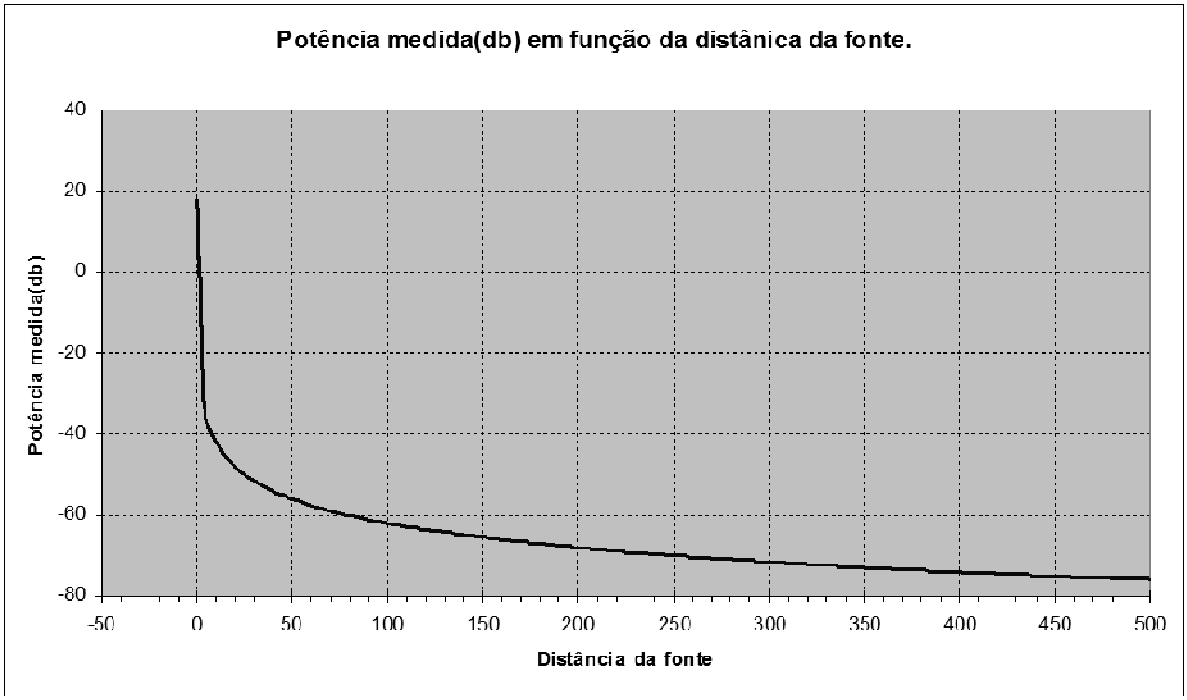

Figura 5 - Potência de sinal em função da distância.

[4] Rappaport (1996), Goldsmith (2005).
[5] Fórmula pode ser encontra no site http://www.osischool.com/protocol/wireless/path-loss
[6] Fórmula pode ser encontra no site http://www.wirelesscommunication.nl/reference/chaptr03/fsl.htm

Da Figura 5, percebe-se que há uma relação entre o alcance de transmissão (distância) e a potência de sinal transmitida.

3.2 Detector DDA e definições

Esta seção apresenta o algoritmo de detecção de defeitos atentos (DDA) e algumas definições e propriedades do algoritmo.

3.2.1 Detector de Defeitos Atentos (DDA)

Para construção da lógica de detecção de falhas assume-se que todo nodo monitora todos outros nodos do sistema através do envio de mensagens *gossip* de acordo com o protocolo *Gossip* básico, porém detecta falhas de maneira diferente.

No *Gossip* básico a detecção de falhas segue o seguinte algoritmo (Renesse *et al.* 1998): a cada intervalo *TGossip*, um nodo escolhe aleatoriamente em uma lista de identificadores um ou mais vizinhos para enviar a mensagem *gossip*; no recebimento da mensagem, a lista de identificadores é unida com a lista que o receptor possui e o maior valor do contador de *heartbeats* de cada nodo é mantido na lista final; cada nodo mantém o instante do último incremento do contador de *heartbeat*; se o contador não for incrementado em um intervalo de *TFail* unidades de tempo então o nodo é considerado suspeito, mas colocado na lista de suspeitos apenas após *TCleanup* unidades de tempo; o *TFail* é escolhido de acordo com a probabilidade de ocorrer uma detecção errônea de falha (*Pmistake*) e o *TCleanup* é escolhido de modo a fazer com que a probabilidade de receber uma mensagem *gossip* que contesta uma suspeita recente seja menor que *PCleanup*.

Burns et al. (1999) diz que é possível simplificar o protocolo *Gossip* permitindo que nodos sejam removidos da lista depois de *TCleanup* sem a necessidade de verificação do *Tfail*. O ponto chave para esta simplificação é que o *TCleanup* pode ser definido como múltiplo do *TGossip* e precisa corresponder ao tempo necessário para que as informações alcançarem outros nós dentro do tempo limite *TCleanup* (Subramaniyan et al. 2005). Desta forma, o presente trabalho também utiliza um método semelhante que simplifica as escolhas dos parâmetros do algoritmo *Gossip* básico fazendo *TCleanup* ser ajustado para *CFail* × *TGossip,* sendo *Cfail* o número de intervalos *TGossip* que devem ser aguardados até que se confirme a suspeita de falha de um dado nodo n_i. Como *CFail* numa MANET depende do padrão de movimentação de cada nodo, existe um *CFail* para cada nodo $n_i \in \prod$.

O princípio básico do DDA consiste na transmissão de mensagens através de difusão (*broadcast*) com informações sobre os nodos vizinhos da rede. O DDA executa tarefas a cada intervalo de tempo *TGossip* e a cada recebimento de mensagens. As tarefas são divididas em: Envio de Mensagem, Recebimento de Mensagem, Verificação de Estado e Atualização de Histório. Cada nodo possui um contador *heartbeat* que é atualizado a cada intervalo de tempo *TGossip* e serve para evitar que informações não atualizadas sejam consideradas válidas por outros nodos. Cada nodo possui uma lista de suspeitos, onde são mantidas informações sobre os nodos que são suspeitos de terem falhado por *crash* e uma lista de vizinhos, que são todos os nodos conhecidos e não suspeitos de falha. A lista de vizinhos mantem informações como o *heartbeat* de cada nodo, o histórico de movimentação responsável por armazenar informação sobre as ultimas regiões onde o nodo esteve, e um contador *CFail* que armazena a quantidade de intervalos *TGossip* que um nodo permaneceu fora do alcance de transmissão ou sem enviar informações.

Uma mensagem enviada por um nodo n_1 é formada pela informação de *heartbeat* do nodo n_i e a lista de vizinhos de n_i. Quando um nodo n_j recebe uma mensagem de n_i, é executada a tarefa de recebimento de mensagens, onde o nodo n_j verifica se o nodo n_i está na sua lista de suspeitos e atualiza as informações sobre o nodo n_i. Ao receber uma mensagem, é armazenada no histórico de movimentação a informação sobre qual região o nodo n_i estava quando enviou a mensagem, o tempo em que a mensagem foi recebida é armazenado em um *timestamp*, levando em consideração o *clock* do nodo n_j, ou seja, o tempo em que o nodo n_j recebeu a mensagem do nodo n_i. Se o nodo estava na lista de suspeitos é então calculado o valor da variável *CFail* e atualizado se for necessário.

Para verificar se um nodo está falho, a tarefa de verificação de estado é executada, e são analisadas as informações referentes a cada nodo na lista de vizinhos, onde é analisado se um nodo é suspeito ou está em movimento. Esta tarefa leva em consideração o histórico de movimentação do nodo.

Para verificar a movimentação do nodo dentro do alcance de transmissão, é necessário que seja identificada qual região o nodo estava nas últimas mensagens que foram transmitidas. Para isto são utilizadas diferentes regiões de cobertura dentro do alcance de transmissão de um nodo.

O funcionamento detalhado das tarefas do DDA é demonstrado na seção 3.4, mas antes são apresentadas as definições utilizadas no detector de defeitos.

3.2.2 Regiões de cobertura.

Uma região de cobertura consiste em um intervalo entre dois valores de potência de sinal. Em uma rede sem fio, a potência de um sinal diminui conforme a distância do emissor aumenta. Quando uma mensagem é enviada de um nodo n_i para um nodo n_j, conforme a distância que a mensagem percorre, perde-se a potência do sinal. Quanto mais longe o receptor n_j estiver do emissor n_i, menor a potência da mensagem recebida e pior o sinal. A potência diminui conforme a distancia da mensagem é percorrida. Conforme mostre a Figura 5 e a Figura 6.

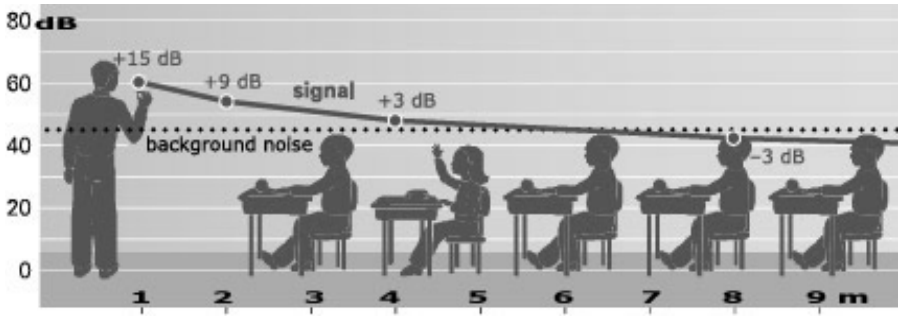

Figura 6 - Exemplo de perda de potência do sinal.[7]

Os detectores de defeitos analisados na seção 2 consideram apenas uma região de cobertura que é igual ao seu alcance de transmissão. Utilizando este tipo de método, podemos verificar apenas que estamos recebendo a mensagem de um nodo dentro do alcance de transmissão, mas não temos a informação de onde o nodo está localizado, como mostra a Figura 7 (a).

Os detectores de defeitos que suportam a mobilidade dos nodos consideram movimentação quando um nodo sai do alcance de transmissão. Essa movimentação considerada acontece quando um nodo n_i sai do alcance de transmissão de um nodo n_j e entra no alcance de transmissão de um nodo n_k, onde n_k é vizinho do nodo n_i e do nodo n_j, conforme as situações de movimento do nodo n_i que é apresentada na Figura 7 (B) (C), onde o nodo n_i sai do alcance de transmissão do nodo n_j.

[7] Figura retirada da url http://www.ufrrj.br/institutos/it/de/acidentes/voz5.htm

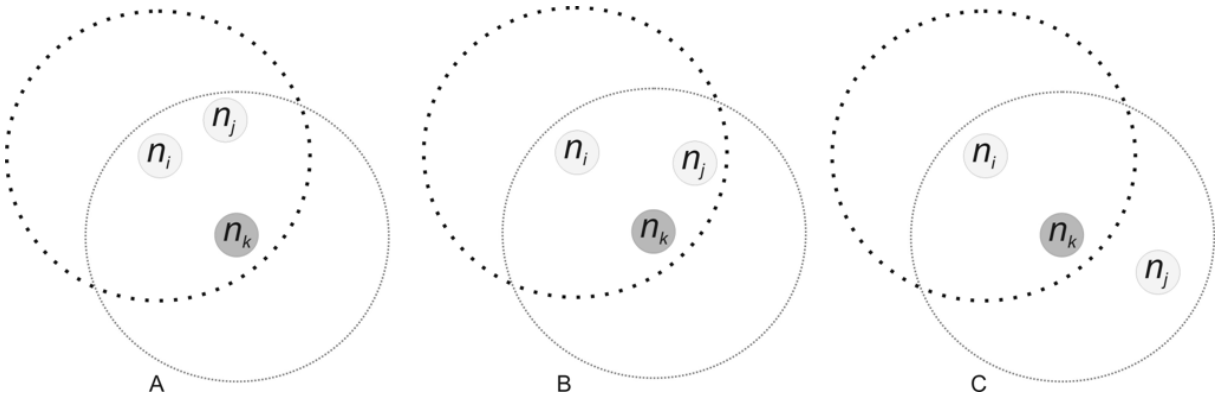

Figura 7 - Exemplo de um alcance de transmissão com apenas uma região. A. Nodo n_j em sua região inicial. B. Nodo n_j movimenta-se e para os nodos n_j e n_k não existe diferença de posição de n_j. C. Nodo n_i sabe que n_j não é mais seu vizinho, mas não sabe diferenciar entre n_j falho ou móvel.

A Figura 7 mostra que a detecção da real movimentação dentro do alcance de transmissão (o nodo n_i sabe que o nodo n_j está saindo do seu alcance de transmissão) não é considerada. Dada essa característica, esses detectores atuam como se não estivessem atentos à movimentação do nodo. A movimentação é uma informação útil para detectar uma possível saída do alcance de transmissão e também para identificar uma falha do tipo *crash* com maior precisão quando este nodo não enviar uma mensagem quando deveria ter enviado.

A informação de distância do nodo pode ser obtida através da análise da potência do sinal, e esta informação pode indicar se um nodo que enviou a mensagem está perto ou longe do nodo que recebeu a mensagem.

O alcance de transmissão de um nodo n_i pode ser dividido em diferentes regiões de cobertura, tal como ilustra a

Figura 8, onde cada região corresponde a um intervalo entre dois níveis de potência do sinal de transmissão.

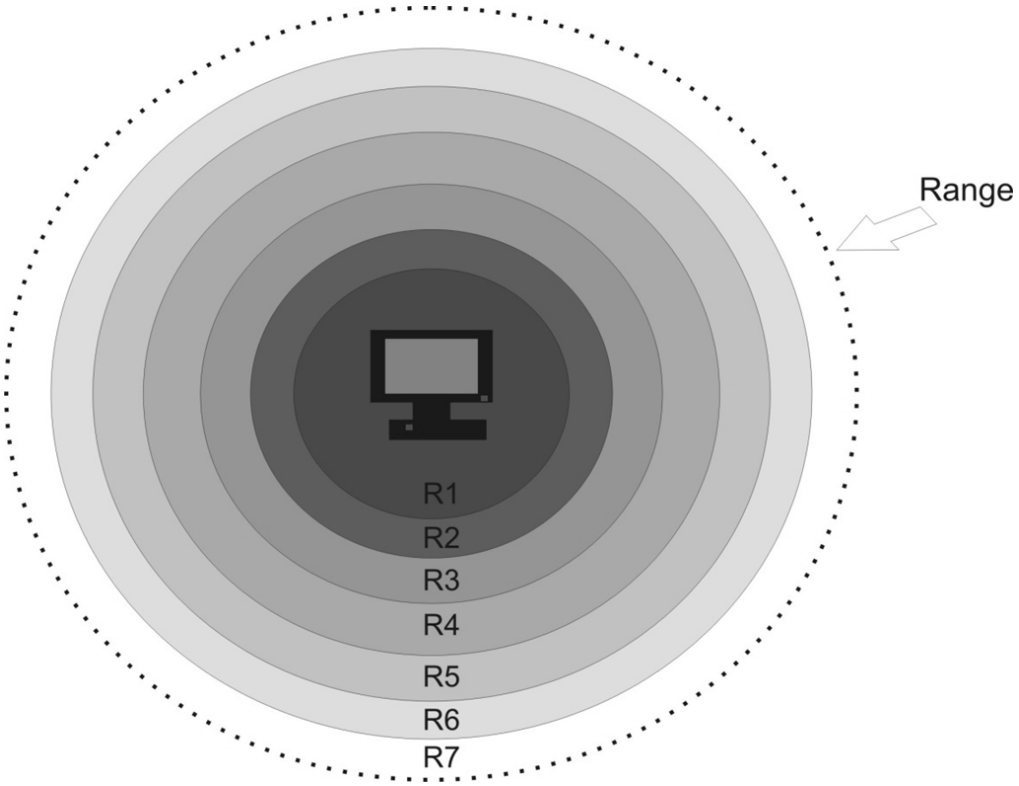

Figura 8 – Diferentes regiões de cobertura em um mesmo alcance de transmissão ou *Range*

A

Figura 8 mostra um exemplo onde existem sete regiões diferentes. A região R_1 possui maior potência de sinal, e conforme as regiões são afastadas do centro, menor o sinal (representado por diferentes tonalidades). A região R_2 apresenta potência de sinal ligeiramente inferior a R_1 e assim por diante. A região R_7 é a ultima região, que considera o *range* máximo para uma mensagem ser entregue.

Considerando o modelo de sistema, onde o alcance de transmissão é igual para todos os nodos, para uma dada região k que está dentro do alcance de transmissão dos nodos n_i e n_j, em um mesmo intervalo de tempo T temos que:

$$\forall \{ n_i, n_j\} \in \prod : \text{se } n_i \subset região_j^k \text{ então } n_j \subset região_i^k.$$

Desta forma, através da potência de cada mensagem recebida, n_i pode então localizar um nodo n_j dentro um nodo n_j dentro de uma de suas regiões de cobertura e saber se n_j está próximo ou longe da localização localização de n_i. Cada região deve ter o mesmo tamanho, ou seja, o intervalo entre dois níveis de potência de potência é igual para todas as regiões definidas. Como o alcance de transmissão de um nodo é igual

nodo é igual para todos os nodos do sistema, se um nodo n_i está na região sete de um nodo n_j, então o nodo então o nodo n_j está na região sete de um nodo n_i como mostra a

Figura 9.

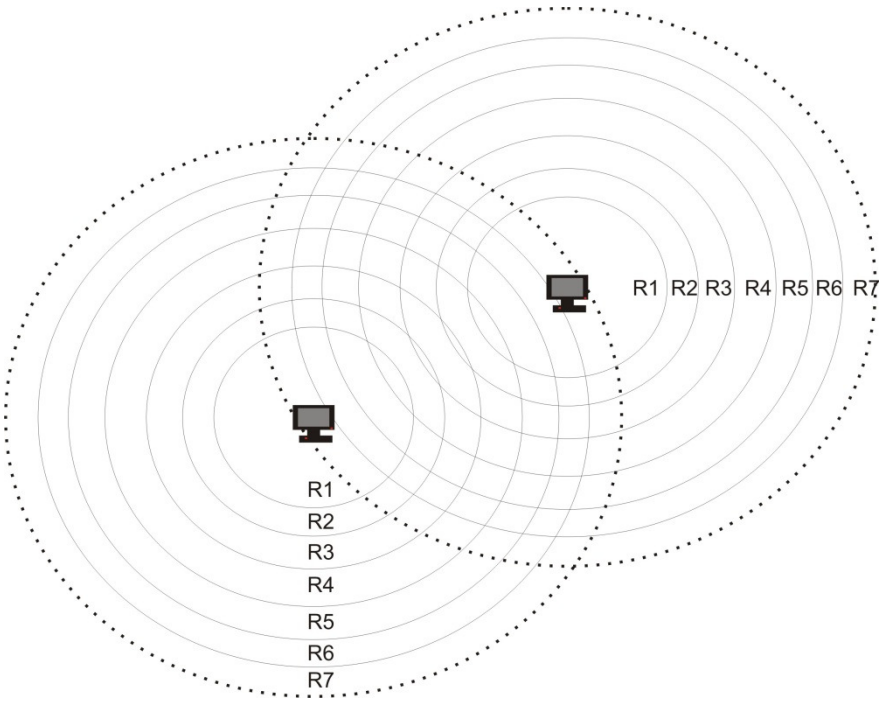

Figura 9 – Nodos n_i e n_j dentro das regiões *R7* devido ao mesmo alcance de transmissão.

Por exemplo, um sistema que define uma região R_1 entre as potências de 25db e 20db, sendo o tamanho do seu intervalo de 5db, tem como padrão o intervalo de potência de 5db para todas as outras regiões. Desta forma, a região R_2 deve ser definida entre as potências 20db e 15db, para manter o mesmo padrão. Qualquer R_x que for definida neste sistema deverá ser do mesmo tamanho de 5db.

Qualquer mensagem que um nodo n_i receber de um nodo n_j vai estar entre uma das regiões, pois a potência da mensagem ($Rpotn_j$) vai estar entre o alcance máximo (*range*) e o alcance mínimo de transmissão.

$$Range_{min} < Rpotn_j <= Range_{min}$$

Com isto, podemos definir que para toda nodo n_i que estiver dentro do alcance máximo de transmissão do nodo n_j, quando o nodo n_i receber a mensagem enviada pelo n_j, n_i irá classificar a mensagem dentro de uma das suas regiões, como por exemplo:

Se: 15db < $Rpotn_j$ <= 20db então *R2*.

Se: 10db < $Rpotn_j$ <= 15db então *R3*.

Esta identificação da região pode mostrar que o nodo n_j está em movimento, pois estará em diferentes regiões dentro do mesmo alcance de transmissão em intervalos de tempo

diferentes. O detector de defeitos em um nodo n_i não trata a movimentação dos nodos que não estão no alcance de transmissão de n_i, pois não pode medir a potência do sinal dos nodos n_k, já que recebe a informação dos nodos n_k que estão fora de seu alcance de transmissão são entregues por nodos intermediários n_j.

3.2.3 Definição do histórico de movimentação

O histórico de movimentação é uma parte importante do DDA, pois através do histórico é possível identificar a movimentação de um nodo dentro do alcance de transmissão. Se um nodo está em movimento, suas regiões no histórico não podem ser iguais, e isto pode identificar um movimento para fora do alcance de transmissão ou para dentro do alcance de transmissão.

Se caso um nodo n_i não enviar uma informação, o histórico do nodo n_i pode ser analisado para saber se o nodo dentro do alcance de transmissão ou estava em movimento (para fora do alcance de transmissão). O algoritmo pode então decidir por suspeitar de falha ou não suspeitar de falha.

Cada nodo n_i mantém um histórico H_i^j para todo n_j que pertence a $\prod$, onde são mantidas as informações sobre as regiões de onde cada nodo vizinho n_j enviou as suas ultimas duas mensagens ou quando o nodo ficou sem enviar mensagens. No caso onde o nodo não enviou mensagens, não é possível identificar sua região.

O histórico é atualizado em um intervalo de tempo finito e a cada atualização a informação mais atual sobre a região de n_j prevalece.

O histórico é atualizado se a informação mais atual, que está sendo analisada, é diferente da informação mais atual do histórico. Se um nodo n_i possui um vizinho n_j que não enviou nenhuma mensagem, então seu histórico vai ser atualizado com nenhuma região, ou seja, com uma informação nula, identificando que o nodo n_j não enviou nenhuma mensagem.

Como definição:

$\forall\, n_j \in \prod$, com $n_i \neq n_j$, existe um H_i^j .

Este histórico possui duas informações, *H[1]*, *H[2]*. Se existir uma atualização de movimentação, a informação mais atual é armazenada em *H[2]*, e a informação que estava em *H[2]* estará em *H[1]*.

Também é possível um nodo n_i receber informação sobre um nodo n_j que está fora do seu alcance de transmissão. Então o histórico é atualizado com uma informação de "Alive", informando que o nodo está vivo.

3.2.4 Definição da lista de suspeitos

Cada nodo n_i mantém uma lista de suspeitos SLn_i. Um nodo n_i é adicionado a SLn_i se e somente se n_i não detecta movimentação de n_j nem recebe nenhuma mensagem de n_j dentro de um intervalo de tempo finito.

3.2.5 Definição da lista de mobilidade

Cada nodo n_i mantém uma lista de mobilidade MLn_i. Um nodo n_j é adicionado a MLn_i se e somente se n_i detecta movimentação de n_j.

A movimentação é detectada analisando as informações do histórico *H[1]* e *H[2]* sobre um nodo n_j. Se houve uma movimentação, o nodo n_j deve possuir histórico *H[2]* e *H[1]* com valores diferentes, e então é adicionado a lista de nodos móveis de n_i.

3.2.6 Definição da lista de vizinhos

A lista de vizinhos $NBLn_i$ é definida como sendo o conjunto de todos os nodos sobre os quais um nodo recebe informação através do recebimento de mensagens e que em sua visão estão corretos (não suspeitos).

Isto implica em um nodo n_i adicionar qualquer outro nodo n_j que n_i tenha recebido informação, direta ou indiretamente. Assim, é importante observar que num dado instante de tempo a lista de vizinhos $NBLn_i$ pode conter tanto nodos vizinhos locais como também os nodos que se deslocaram para outra vizinhança, ou seja, que não estão mais no alcance de transmissão do nodo n_i.

Se um nodo n_i pertence ao alcance de transmissão de um nodo n_j, então o nodo n_i adicionará a sua lista de vizinhos todos os nodos que estão no alcance de transmissão do nodo n_j, mesmo que os nodos vizinhos de n_j nunca tenham entrado na vizinhança de n_i, ou seja, nunca estiveram ao alcance de transmissão de n_i. Isto acontece porque os nodos propagam a sua lista de vizinhos $NBLn_j$.

Como exemplo na Figura 10, temos o caso do nodo n_j que pertence ao alcance de transmissão de n_k e não pertence ao alcance de transmissão de n_i. O nodo n_j acaba sendo

adicionado a lista de vizinhos de n_i, devido à mensagem que será propagada pelo nodo n_k informando que n_j existe e está ativo.

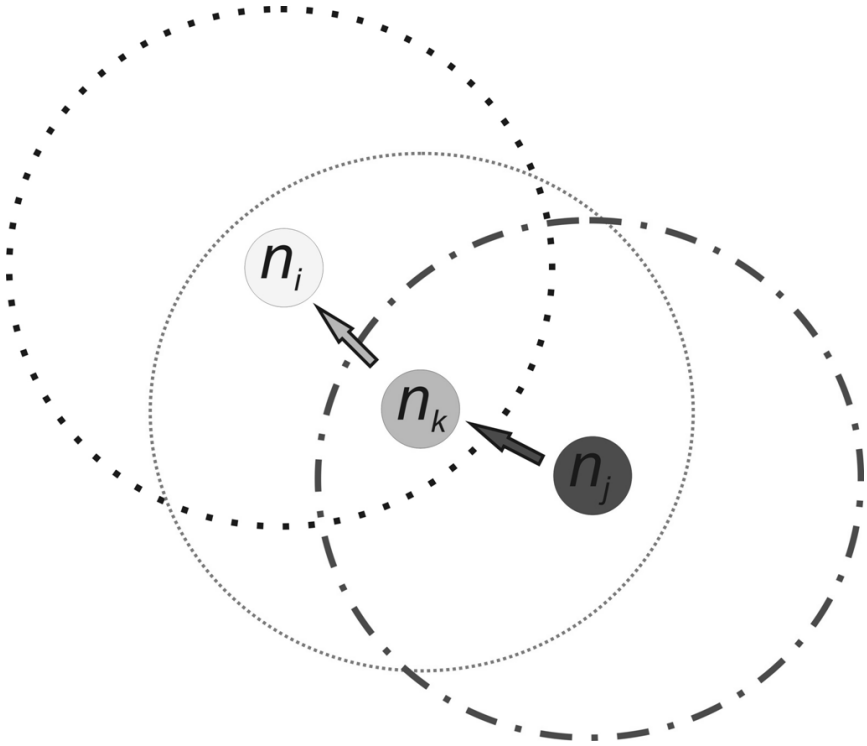

Figura 10 – Nodo n_i adiciona o nodo n_j porque recebe uma mensagem de n_k que possui n_j como vizinho.

3.2.7 Propriedades de comportamento

Todos os nodos que pertencem as MANETs podem mover-se livremente, mas para que seja possível identificar um padrão de mobilidade real e garantir a identificação da movimentação, é preciso definir propriedades para suportar a mobilidade dos nodos.

Segundo Sens et al. (2008), para implementar um detector de defeitos não confiável cujos nodos são desconhecidos, é necessário que eles interajam entre si pelo menos uma vez para se conhecerem, pois de acordo com Fernandez et al. (2006), se existir um nodo no sistema no qual não trocou informações com nenhum outro nodo, não existe algoritmo que possa implementar um detector de defeitos que consiga atingir a propriedade de perfeição fraca, mesmo se os canais forem confiáveis e o sistema síncrono.

3.2.7.1 Propriedade do tamanho da região (*PTR*)

Para definir o tamanho das regiões a serem utilizadas no DDA entre o alcance de transmissão máximo e mínimo de um nodo n_i, é considerada a velocidade máxima em que um nodo pode movimentar-se em um intervalo de tempo *TGossip*. O tamanho de uma região é definido para que um nodo não consiga passar por mais de uma região inteira entre dois intervalos de tempo consecutivos.

Temos que a fórmula da velocidade, em m/s definida por:

$$v = d / t$$ [8]

onde v é velocidade em m/s; d = distância em metros; t = tempo em s.

A distância pode ser medida em relação à velocidade média por um intervalo de tempo, sendo representada pela fórmula:

$$d = v / t$$

Para que um nodo não ultrapasse mais de uma região, consideramos a velocidade máxima de um nodo como a velocidade média, e substituindo na fórmula a distância pelo tamanho da região, a velocidade média pela velocidade máxima que um nodo pode atingir e o tempo pelo intervalo de tempo *TGossip*, definimos que o tamanho da região é dado pela equação:

$$Tamanho\ da\ Região = Velocidade\ Máxima / TGossip$$

Esta definição de tamanho de região garante que se um nodo n_i transmitir uma mensagem *gossip* no início de uma região R_1 e n_j está movimentando-se a velocidade máxima, n_j conseguirá transmitir a próxima mensagem *gossip* no início de outra região R_2.

Para exemplificar, a Figura 11 mostra um comportamento que não é permitido, onde um nodo (PDA) move-se da região R_2 Figura 11.A para a região R_4 na Figura 11.B em apenas um intervalo de tempo *TGossip.*

[8] Fórmula retirada do site http://pt.shvoong.com/exact-sciences/mathematics/1880687-f%C3%B3rmulas-matematica-f%C3%ADsica-movimento/

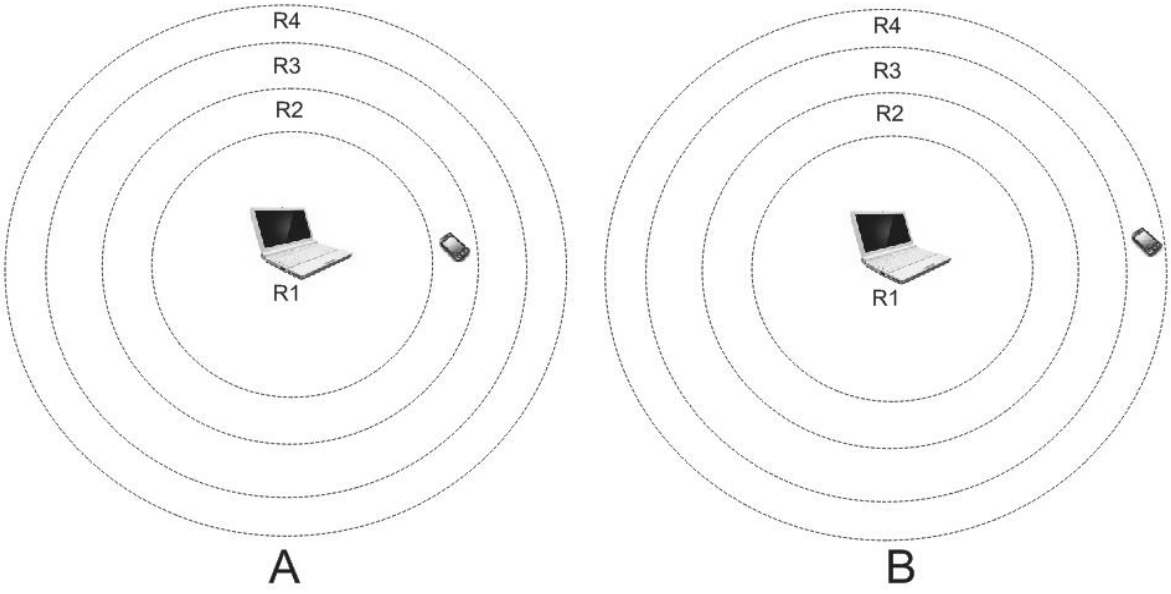

Figura 11- Exemplo de movimento não permitido. A. Nodo está na região R_2 no tempo *TGossip* = 1. B. Nodo está na região R_4 no tempo *TGossip* = 2.

Dada a definição do tamanho da região, devemos ter um número mínimo de regiões que deve ser definido para que seja possível verificar a movimentação do nodo.

O número de regiões é calculado pelo alcance de transmissão dividido pelo tamanho de uma região. Por exemplo, dado um alcance de transmissão de 50 metros, onde cada região tem o tamanho de 10 metros, teremos 5 regiões, sendo a última região a região com menor potência do sinal, abrangendo as mensagens recebidas com distância entre 40m e 50m. Outro exemplo que podemos citar é, dado um alcance de transmissão de 45 metros, onde cada região tem o tamanho de 10 metros, a divisão para calcular o número de regiões resulta em um número não inteiro. Para que o alcance entre 40 e 45 metros seja considerado, é arredondado o resultado da divisão, para que exista um número de regiões onde as mensagens com potência de sinal entre 40 e 45 metros sejam inseridas. Como o tamanho de uma região é 10 metros, qualquer mensagem recebida entre 40 e 50 metros será considerada como recebida na última região.

Se o tamanho da região for igual ao alcance máximo de transmissão, não será possível identificar uma movimentação, pois independente da localização de um nodo dentro do alcance de transmissão, a região a ser armazenada no histórico será sempre a mesma.

Se o número de regiões for igual a duas regiões, utilizando o DDA é possível identificar uma movimentação para fora do alcance de transmissão ou para uma posição mais

próxima. São necessárias no mínimo duas regiões para que o DDA consiga detectar uma falha local.

Com isto, a seguinte equação deve ser satisfeita:

Alcance de Transmissão / Tamanho da Região ≥ 2

O uso de no mínimo duas regiões é uma restrição do DDA e o número de regiões pode variar entre diferentes modelos de sistemas. Em cada configuração de modelo de sistema é calculado o número de regiões de acordo com o alcance de transmissão e o tamanho de cada região para cada modelo.

3.3 Estruturas de dados para suporte à mobilidade

Cada nodo n_i pertencente a $\prod$ mantêm uma lista de suspeitos SLn_i, que descreve todos os nodos que n_i suspeita de falha. Também mantêm uma lista de vizinhos $NBLn_i$, que é a vizinhança atual que n_i acredita ter, e uma lista de nodos móveis MLn_i que mantém a informação de todos os nodos que n_i suspeita de estarem em movimento. O nodo n_i mantém as informações dos seus vizinhos em uma estrutura *NB*, onde para cada vizinho n_j é mantido:

H_i^j - histórico de movimentação do nodo n_j.

TSn_j - *timestamp* da última informação recebida do nodo n_j, referente ao *clock* ou tempo atual de n_i.

$Hcountn_j$ - contador de *heartbeats* referente às mensagens *gossip* do nodo n_j.

$Cfailn_j$ - contador de intervalos *TGossip* que devem ser aguardados para sobrepor o tempo de desconexão local do nodo n_j dado seu padrão de movimentação (trajetória de movimentação).

$Rpotn_j$ - informação sobre a região (localização) de n_j no alcance de transmissão de n_i.

O contador de *heartbeat* é utilizado para que os nodos não recebam mensagens atrasadas e considerem uma informação antiga como sendo uma informação mais atual sobre o estado de um nodo. Por exemplo, se um nodo n_i envia uma mensagem com *heartbeat* $Hcountn_i = 2$ no intervalo de tempo $T = 2$ para um nodo n_j, o nodo n_j ao receber a mensagem armazenaa o *timestamp* TSn_i com o valor do seu *clock* interno $T = 2$. Se posteriormente, em um intervalo de tempo $T = 5$, o nodo n_j receber uma mensagem com informação sobre o nodo n_i com $Hcountn_i = 1$, o nodo n_j não irá considerar a mensagem, pois a informação que ele possuí é mais atual, e com isso não modifica o *timestamp* do nodo n_i, TSn_i. Caso o nodo n_j receba uma mensagem que contém um valor de *hearbeat* de n_i maior que o valor que ele

possui, o *timestamp* do nodo n_i é atualizado com o valor do *clock* de n_j no momento do recebimento da mensagem.

3.4 Algoritmo com suporte à mobilidade

Esta seção apresenta o funcionamento do algoritmo, apresentando as tarefas que são executadas pelo DDA, as quais se dividem em: envio de mensagens, recebimento de mensagens, verificação de estado e atualização do histórico. O algoritmo completo é apresentado na Figura 13 e é dividido em Tarefa 1 e Tarefa 2.

Inicialmente as estruturas de dados são ajustadas para que um nodo n_i conheça todos os outros nodos, sem suspeitar dos nodos. Logo no primeiro intervalo de tempo *TGossip*, ocorre uma varredura e se um nodo n_i não recebe informação de um nodo n_j, ele será adicionado como suspeitos de falha.

A cada intervalo de tempo *TGossip*, cada nodo no sistema incrementa o seu contador de *heartbeat.* Após atualizar o seu *hearbeat*, os nodos atualizam o histórico de todos os seus nodos vizinhos para verificar se houve um recebimento de mensagem entre dois intervalos de tempo *TGossip*. Após a atualização do histórico, o nodo faz uma verificação de estado para identificar os nodos que estão ativos. Esta verificação de estado é responsável por adicionar e remover os nodos nas listas de vizinhos, lista de móveis e lista de suspeitos. Após verificar o estado dos seus nodos vizinhos, os nodos enviam uma mensagem por *broadcast* que contem a sua lista de vizinhos $NBLn_i$ e sua lista de vizinhos considerados móveis MLn_i. É considerado que um nodo n_i inclui ele mesmo na sua lista de vizinhos ativos. Assim em $NBLn_i$ também existe a informação com o *heartbeat* de n_i.

Essas tarefas são executadas em sequência a cada intervalo de tempo *TGossip* por todos os nodos do sistema, e esta sequência de tarefas garante que a mensagem sempre vai contar as informações mais atuais que um nodo possui sobre os seus nodos vizinhos.

Quando um nodo n_i recebe uma mensagem de um nodo n_j, é necessário verificar em qual lista o nodo n_j está inserido. O nodo n_j pode estar na lista de vizinhos, na lista de móveis ou na lista de suspeitos de n_i.

Se n_j está na lista de vizinhos de n_i então n_i compara o valor do *heartbeat* ($Hcountn_j$) recebido de n_j com o valor $Hcountn_j$ que n_i possuí sobre n_j. Esta comparação é feita para verificar se não está recebendo uma informação defasada. Se o *heartbeat* recebido possuir uma informação mais atual, n_i atualiza a sua informação de *heartbeat* com o valor de *heartbeat* recebido, atualiza também a informação do *timestamp* TSn_j com o valor do seu

clock e verifica cada um dos n_k nodos que estão na lista de vizinhos $NBLn_j$, comparando com sua lista de nodos suspeitos, nodos móveis e nodos vizinhos.

Caso ao receber a mensagem, n_j está na lista de nodos móveis de n_i (MLn_i) então o nodo n_i remove o nodo n_j da lista de móveis MLn_i e adiciona na lista de vizinhos $NBLn_i$.

Se o ao receber a mensagem, n_j está na lista de nodos suspeitos de n_i (SLn_i), então o nodo n_i remove o nodo n_j da lista de móveis SLn_i e adiciona na lista de vizinhos $NBLn_i$. Após adicionar o nodo na lista de nodos vizinhos, é calculado o tempo que o nodo n_j permaneceu como suspeito. Este cálculo é feito com tempo atual do *clock* de n_i e a informação do *timestamp* de n_j da última informação recebida de n_j dividido pelo *TGossip,* resultando em um número inteiro, que significa quantos intervalos *TGossip* o nodo n_j ficou como suspeito de n_i. Este resultado é armazenado em $CFailn_j$.

O valor de $CFailn_j$ só é atualizado quando a seguinte condição for satisfeita:

$$CFailn_j < (TLn_i - TSn_j)/TGossip$$

Após verificar se a condição foi satisfeita, é atualizado o valor de TSn_j para com o valor de TLn_i.

Depois de examinar as informações sobre o nodo que enviou a mensagem, o DDA analisa as informações contidas em $NBLn_j$ para conferir se existe informações mais atuais sobre os vizinhos de n_j em comparação com a sua lista de vizinhos $NBLn_i$.

Se um nodo n_k pertence à $NBLn_j$, e está em $NBLn_i$ ou em MLn_i, é verificado qual lista possuí o valor mais atual de *Heartbeat* $Hcountn_k$, e n_i atualiza o *heartbeat* e *timestamp* do nodo n_k se necessário. Se n_k está em SLn_i, é analisado a informação do *heartbeats* do nodo n_k das duas listas para verificar se a informação recebida é mais atual que a existem sobre n_k. Se a informação recebida é mais atual, então é verificado se é necessário atualizar o valor de $CFailn_k$. Depois o nodo n_k é adicionado em MLn_i e removido de SLn_i. Se a informação não for mais atual, n_k não é atualizado em n_i e permanece em SLn_i.

O funcionamento do algoritmo está descrito nas figuras 13 e 14.

1. **Inicialização**:
2. $H_{ni}^{nj} \leftarrow$ [-]; *//O histórico de todos nodos são inicializados.*
3. $CFailn_j \leftarrow 0$; *//O contador CFail é inicializado com valor zero.*
4. $Rpotn_j \leftarrow$ [-]; *//Potência é zero, pois não foi recebida nenhuma mensagem.*
5. $Hcountn_j = 0$; *//Inicializa as informações de hearbeats de todos nodos.*
6. $TSn_j = 0$; *//Inicializa o timestamp de todos nodos.*
7. $NBLn_i \leftarrow n_j$; *//Adiciona todos nodos a lista de vizinhos de n_1.*
8. $MLn_i \leftarrow 0$; *//Lista de Móveis não contém nenhum nodo.*
9. $SLn_i \leftarrow 0$; *//Lista de Suspeitos não contém nenhum nodo.*
10. *RegiãoMáxima* ← *Range / Tamanho da Região*; *//Calcula qual a ultima região com maior alcance de transmissão.*
11. TLn_i = *clock* de n_i; *//valor do clock do nodo n_i.*
12.
13. **Tarefa 1: Tarefas no intervalo *TGossip***
14.
15. **a cada** *TGossip* **faça**
16.
17. $Hcountn_i = Hcountn_i + 1$;
18.
19. *//Atualiza Histórico*
20. $\forall n_j \in NBLn_i \bigcup \forall n_j \in MLn_i$
21. H_{ni}^{nj} [t-1] = [t]
22. H_{ni}^{nj} [t] = [$Rpotn_j$]
23. $Rpotn_j$ = [-];
24.
25. *//Verifica Estado*
26.
27. $\forall n_j \in NBLn_i$ com $n_i \neq n_j$ faça
28. **Se** H_{ni}^{nj} [t] = [-] **então**
29. **se** H_{ni}^{nj} [t-1] < *RegiãoMáxima* **então**
30. adiciona n_j em SLn_i;
31. remove n_j de $NBLn_i$;
32.
33. **se** H_{ni}^{nj} [t-1] = *RegiãoMáxima* **então**
34. **Se** $TLn_i - TSn_j > CFailn_j * TGossip$ **então**
35. adiciona n_j em SLn_i;
36. remove n_j de $NBLn_i$;
37. **senão**
38. adiciona n_j em MLn_i;
39. remove n_j de $NBLn_i$;
40.
41. $\forall n_j \in MLn_i$ com $n_i \neq n_j$ faça
42. **se** $TLn_i - TSn_j > CFailn_j * TGossip$ **então**
43. adiciona n_j em SLn_i;
44. remove n_j de MLn_i;
45.
46. *//Envia Mensagem*
47. *broadcast ($NBLn_1$, $Hcountn_1$);*
48.

Figura 12 - Algoritmo DDA – Parte 1

49. **Tarefa 2: Recebimento de Mensagem**
50.
51. **a cada** mensagem *gossip* de n_j com ($NBLn_j$, $Hountn_j$) **faça**
52.
53. **Se** $n_j \in NBLn_i$ **então**
54. **Se** $Hcount_{ni}^{nj} < Hcount_{nj}$ **então**
55. $Hcount_{ni}^{nj} = Hcount_{nj}$;
56.
57. **Senão Se** $n_j \in MLn_i$ **então**
58. **Se** $Hcount_{ni}^{nj} < Hcount_{nj}$ **então**
59. $Hcount_{ni}^{nj} = Hcount_{nj}$;
60. adiciona n_j em $NBLn_i$;
61. remove n_j de MLn_i;
62.
63. **Senão Se** $n_j \in SLn_i$ **então**
64. **Se** $Hcount_{ni}^{nj} < Hcount_{nj}$ **então**
65. $Hcount_{ni}^{nj} = Hcount_{nj}$;
66. **Se** ($TLn_i - TSn_j$) / $TGossip > CFailn_j$ **então**
67. $CFailn_j = (TLn_i - TSn_j)$ / $TGossip$;
68. adiciona n_j em $NBLn_i$;
69. remove n_j de SLn_i;
70.
71. $\forall\, n_k \in NBLn_j$ com $n_i \neq n_j \neq n_k$ **faça**
72. **Se** $n_k \in NBLn_i, MLn_i$ **então**
73. **Se** $Hcount_{ni}^{nk} < Hcount_{nj}^{nk}$ **então**
74. atualiza $Hcount_{ni}^{nk} = Hcount_{nj}^{nk}$;
75.
76. **Se** $n_k \in SLn_i$ **então**
77. **Se** $Hcount_{ni}^{nk} < Hcount_{nj}^{nk}$ **então**
78. **Se** ($TLn_i - TSn_k$) / $TGossip > CFailn_k$ **então**
79. $CFailn_k = (TLn_i - TSn_k)$ / $TGossip$;
80. adiciona n_k em MLn_i;
81. remove n_k de SLn_i;
82.
83. atualiza TSn_k com LTn_i;
84. atualiza $Rpotn_j$;

Figura 13 - Algoritmo DDA – Parte 2

3.5 Propriedades do detector de defeitos com suporte à mobilidade

O DDA possui as propriedades de completude e exatidão (CHANDRA e TOUEG 1996). O DDA suspeita dos nodos que falham após um tempo sem enviar uma mensagem, pois no momento da verificação do estado de um nodo, o será incluído na lista de suspeitos, conforme pode ser visto entre as linhas 27 e 44 da Figura 13, assim o DDA tem um comportamento que satisfaz a propriedade da completude forte.

Quando um nodo n_i que está suspeito de falha por n_j entrar no alcance de transmissão de n_j em um intervalo de tempo *TGossip*, o nodo n_i que estava suspeito e é nodo correto não será mais considerado suspeito por n_j, pois ao receber uma mensagem é removido da lista de

suspeitos de n_j. As informações sobre n_i são propagadas por difusão e o nodo n_i que antes era considerado falho não será mais suspeito por nenhum outro nodo correto, assim o DDA tem um comportamento que satisfaz exatidão fraca após um tempo. Com essas propriedades, nosso detector de defeitos pode ser classificado na classe ◊S.

3.5.1 Prova de Correção do Algoritmo

Este item apresenta um argumento de prova para o DDA e que mostra que o DDA implementa um detector de defeitos da classe ◊S.

Teorema 1: Algoritmo de detecção de defeitos com mobilidade implementa um detector de defeitos da classe ◊S em uma rede de nodos com mobilidade.

Demonstração 1: Propriedade de *completude forte.*

Para que o algoritmo possua a propriedade de *completude forte*, deve-se demonstrar que eventualmente um nodo $n_i \in \prod$ será incluído na lista de suspeitos SLn_j de todo nodo $n_j \in \prod$, sendo $n_i \neq n_j$. Esta condição segue das seguintes verificações:

Todo nodo n_i pertencente a $\prod$ e fora do alcance de transmissão de n_j está inicialmente em SLn_j. Para fazer parte de $NBLn_j$, n_i deve ter enviado pelo menos uma mensagem *gossip* em um intervalo *TGossip*.

Quando falhar, um nodo n_i não enviará mais mensagens *gossip.* Todo nodo n_j espera $CFailn_i$ intervalos antes de adicionar n_i como falho. Logo, existirá um instante de tempo após o qual n_i será adicionado permanentemente como suspeito em SLn_j de todo nodo $n_j \in \prod$.

Demonstração 2: Propriedade de *exatidão fraca após um tempo.*

Para que o algoritmo possua a propriedade de *exatidão fraca após um tempo*, deve-se demonstrar que existe um instante de tempo *T*, a partir do qual um nodo correto não vai ser suspeito por nenhum nodo correto do sistema, ou seja, onde n_i é um nodo correto e que n_j não vai fazer parte de nenhum conjunto SLn_i, para todo nodo correto $n_j \in \prod$. Esta condição segue das seguintes verificações:

Se n_i está no alcance de transmissão de n_j, em um determinado instante *t*, após um intervalo *TGossip*, n_i enviará uma mensagem *Gossip* para n_j, que adicionará o nodo n_i em sua lista de vizinhos $NBLn_j$. Essa informação será difundida para todos vizinhos n_k do nodo n_j, onde n_k está no alcance de transmissão de n_j. Com isso, todos os nodos eventualmente irão

receber informações sobre o nodo n_i e remover n_i da lista de suspeitos, adicionando n_i em sua lista de vizinhos $NBLn_k$.

Se o nodo n_i estiver na lista de suspeitos de algum nodo n_j pertencente a $\prod$, o nodo n_i terá a sua informação de $CFailn_i$ atualizada, fazendo com que o nodo n_i possa ter o mesmo intervalo de movimentação dentro de sua trajetória sem ser adicionado novamente como suspeito de n_j. Deve-se levar em consideração que se, um nodo estiver ativo fora do alcance de transmissão de todos os outros nodos (partição de rede), não existe uma maneira de detectar que este nodo está ativo, e é necessário que ele envie pelo menos uma mensagem que seja recebida por outro nodo que consiga repassar a informação para todos os outros.

4 Testes e Simulação

Nesta seção, é apresentado o resultado das simulações feitas com o DDA. Para verificar o comportamento do DDA em uma rede sem fio, foi criado um simulador descrito na seção 4.1. Um dos motivos principais foi à necessidade de utilizar a potência do sinal recebido, sendo este um recurso essencial para a proposta do DDA e os demais simuladores existentes não apresentarem esta funcionalidade.

Para mostrar o funcionamento do DDA, foram realizados vários testes para identificar o desempenho do DDA em função do aumento do número de nodos, aumento do alcance de transmissão, velocidade de movimentação dos nodos e diferentes níveis de potência.

Com estes testes, podemos ver o desempenho do DDA em funcionamento utilizando algumas das métricas de qualidade serviço (*QoS*) propostas por Chen et al. (2002).

4.1 Simulador WireS

Para realizar a simulação desenvolvemos o WireS (*Wireless Simulator*). Este simulador foi desenvolvido em *Delphi* na linguagem *Object Pascal*, e tem o objetivo de identificar as mensagens que são transmitidas por nodos dentro de um sistema distribuído que utiliza uma rede sem fio no padrão IEEE 802.11g. Para poder verificar se um nodo n_i recebeu uma mensagem de outro nodo n_j, foi calculada a distância entre os nodos e verificado se a potência da mensagem transmitida alcança a distância máxima permitida entre dois nodos.

Se a distância entre dois nodos for menor que o resultado do alcance de transmissão da mensagem, então o simulador computa a mensagem transmitida entre os dois nodos. Caso contrário, a mensagem não é considerada entregue para o nodo vizinho.

É possível utilizar simulações previamente feitas no WireS, para testar diferentes algoritmos. Isto quer dizer que com uma simulação é possível testar mais de um algoritmo com uma mesma simulação.

O primeiro campo que pode ser configurado em WireS é o tempo de execução da simulação. Este tempo é inserido em segundos, permitindo que a variação do tempo seja medida em um passo definido pelo usuário, que pode variar entre micro-segundos, nano-segundos, ou conforme o usuário achar necessário.

O simulador permite que seja configurada a potência de transmissão (que é a potência máxima emitida por um nodo transmissor), a frequência do sinal (que pode variar de acordo com as especificações da IEEE 802.11g e do dispositivo que faz a transmissão). A potência

máxima válida de leitura é utilizada para expor qual o alcance máximo considerado de uma mensagem que será entregue. Se um sinal chegar com uma potência menor que a potência selecionada na simulação, a mensagem não será considerada.

O WireS permite que sejam utilizados nodos com movimentação controlada e nodos com movimentação aleatória.

Os nodos que possuem movimentação aleatória são controlados de acordo com uma equação *RandomWaypoint* com o sentido de permitir diferentes movimentos com limites de velocidade mínima e máxima conhecidas. Também é possível utilizar uma movimentação definida, onde é possível calcular a posição de um nodo no eixo *x* e no eixo *y* em função do tempo.

Para calcular a posição do eixo *x*, é utilizada uma equação que inclui movimentos matematicamente conhecidos, que podem ser controlados com a variação do tempo. Na equação é possível utilizar a movimentação senoidal[9] representada pelas variáveis *a.sen (2*pi*fr*t),* onde *a* é a amplitude, *pi* é a velocidade, *fr* é a freqüência angular e *t* é o intervalo de tempo. Pode ser utilizado o movimento de aceleração em função do tempo[10] representado pelas variáveis *b*t^2*, onde *b* é a velocidade e *t* o intervalo de tempo. Pode ser utilizada a velocidade constante que é representada pelas variáveis *c*t*, onde *c* representa o valor da velocidade e *t* o intervalo de tempo. A variável *d* no final da equação representa a posição inicial que o nodo vai estar em relação ao eixo *x* da simulação.

Os movimentos de aceleração, movimento senoidal e movimento de velocidade constante podem ser utilizados ao mesmo tempo, ou com uma variação de combinações. É possível também utilizar apenas um dos movimentos, ou nenhum deles, deixando o nodo parado.

Por exemplo, para o nodo ficar parado, é possível utilizar as variáveis *a, pi, fr, b, c* com valor igual a zero. Para não utilizar um dos movimentos, basta à variável que está relacionada com o movimento possuir o valor zero.

A equação da posição do nodo no eixo *x* é o resultado da soma dos movimentos e a posição inicial, definida pela equação:

$$x = a.sen\ (2*pi*fr*t) + b*t\hat{}2 + c*t + d$$

[9] Fórmula retirada do site http://pt.wikipedia.org/wiki/Senoide
[10] Fórmula retirada do site http://www.fisica.ufs.br/CorpoDocente/egsantana/cinematica/rectilineo/rectilineo.htm

Para calcular a posição do eixo *y*, é utilizada uma equação que inclui movimentos matematicamente conhecidos, que podem ser controlados com a variação do tempo. Na equação é possível utilizar uma movimentação representada pela fórmula exponencial representada pelas variáveis *e^(x*t)*, onde *x* representa o valor da velocidade e *t* representa o valor do tempo. Pode ser utilizado o movimento de aceleração em função do tempo representado pelas variáveis *f*t^2*, onde *f* é a velocidade e *t* o intervalo de tempo. Pode ser utilizada a velocidade constante que é representada pelas variáveis *g*t*, onde *g* representa o valor da velocidade e *t* o intervalo de tempo. A variável *h* no final da equação representa a posição inicial que o nodo vai estar em relação ao eixo *y* da simulação.

Os movimentos de aceleração, velocidade constante e exponencial podem ser utilizados ao mesmo tempo, ou com uma variação de combinações. É possível também utilizar apenas um dos movimentos, ou nenhum deles, deixando o nodo parado.

Por exemplo, para deixar o nodo ficar parado, é possível utilizar as variáveis *x, f, g,* e *h* com valor igual a zero. Para não utilizar um dos movimentos no eixo *y*, basta a variável que está relacionada com o movimento possuir o valor zero.

A equação da posição do nodo no eixo *y* é o resultado da soma dos movimentos relacionados com o eixo *y* e a posição inicial *h*, definida pela equação:

$$y = e^{(x*t)} + f*t^2 + g*t + h$$

Para controlar os nodos, são inseridos os valores das variáveis das equações em um arquivo texto. O WireS, no início de cada simulação, lê o arquivo texto e obtém os valores que serão utilizados nas equações do eixo *x* e do eixo *y*.

A seqüência dos valores de entrada do arquivo texto segue conforme a fórmula:

a fr b c d x f g h
0 0 0 0 0 0 0 0 1 0

Este exemplo (seqüência) indica que o nodo movimenta-se um metro por unidade de tempo (*TGossip*) no eixo *y* dentro do campo de simulação. Se todos os valores ficarem em 0, o nodo permanece parado durante toda a simulação.

A velocidade máxima dos nodos com movimentação randômica é o parâmetro utilizado para calcular o *RandomWaypoint* ou próximo local que o nodo assume em relação a sua posição atual nos eixos *x* e *y*.

A velocidade máxima é o valor máximo que uma nova posição pode ser sorteada para fazer sua movimentação, e o valor inverso ou negativo é utilizado para ser o valor mínimo que pode ser sorteado para a nova posição, sendo o valor inverso uma movimentação a posição contrária do valor máximo. Um exemplo é colocar uma movimentação aleatória com velocidade de 2 m/s.

Para calcular a nova posição nos eixos *x* e eixo *y*, é sorteado um valor aleatório no intervalo [-2;2] para os dois eixos. O novo local que o nodo será posicionado após o intervalo de tempo *TGossip* é calculado de acordo com a posição atual do nodo no eixo *x* e no eixo *y*, somado com esse novo valor que foi sorteado aleatoriamente.

Por exemplo, considere um nodo n_1 que estava em sua posição no tempo $t = 1$ com valores nos eixos $x = 2$ e $y = 0,5$. Após o intervalo de tempo *TGossip*, com $t = 2$, é utilizado o método *RandomWaypoint* que sorteia os valores aleatórios dos eixos *x* e *y*. A nova posição do nodo n_1 em relação ao eixo *x* estará no intervalo [0;4] e a nova posição no eixo *y* estará no intervalo [-1,5;2,5].

O próximo parâmetro configurável no simulador é o tempo para trocar de direção. Este tempo para trocar de direção pode ser utilizado caso seja preciso fazer rotas onde os nodos tem que andar em uma linha reta durante um intervalo de tempo. Assim, a cada intervalo definido o nodo escolhe uma nova direção até acabar o intervalo de tempo selecionado.

O próximo parâmetro do simulador faz referência à posição inicial de cada nodo. Este parâmetro permite que sejam iniciados dois tipos de simulações: simulação com posição inicial igual para todos os nodos e simulação com posição inicial aleatória. Se for preciso que em uma simulação todos os nodos tenham o mesmo ponto de início, então esta opção deve ser desmarcada, e todos os nodos serão inicializados com posição no ponto de origem P(x,y) = (0,0). Se esta opção estiver marcada no inicio da simulação, para cada nodo vai ser sorteado um ponto aleatório que vai estar dentro do campo de simulação.

Por exemplo, em um campo de simulação configurado para ter o tamanho em 2000 m^2, cada nodo pode iniciar em uma posição no eixo *x* que varia entre [-1000, 1000] e no eixo *y* pode variar entre [-1000,1000].

O tamanho do campo é o próximo parâmetro a ser configurado antes de começar uma simulação. Este campo define a posição máxima e mínima que cada nodo poder ocupar, delimitando (nodos não podem ultrapassar) a distância que um nodo pode andar e ocupar dentro do espaço da simulação. Se um nodo tentar ultrapassar essa área, o simulador então toma uma decisão, que pode ser: parar a movimentação do nodo neste intervalo (selecionando

a nova posição com o valor igual à posição anterior) ou seleciona uma nova posição que pertença à área interna do campo de simulação definido.

A variação do tempo é o parâmetro que define se os *clocks* dos nodos serão sincronizados ou não. Se não existir uma variação, todos os *clocks* terão o mesmo valor e os nodos terão o seu tempo de *broadcast* igual, todos os nodos transmitem ao mesmo tempo. Se existir uma variação definida, no início de cada simulação é sorteado o intervalo de tempo que cada nodo vai fazer um *broadcast*.

O WireS permite que em uma simulação assíncrona, cada novo tempo seja calculado com uma variação do intervalo de *broadcast*. Este intervalo de tempo é calculado da seguinte maneira:

Novo Tempo = Tempo Anterior + *Random* [*TGossip* - Variação, *TGossip* + Variação]

Por exemplo, se uma variação de tempo for definida em 10% e o intervalo *TGossip* for 1s, para um nodo n_i, o primeiro tempo de broadcast pode ser feito no intervalo entre 0,9 s e 1,10 s, sendo *Tempo de Broadcast* (TB1) = [0,9 ; 1,10]. O segundo tempo de *broadcast* é feito no tempo TB1 + [0,9 ; 1,10]. Esse é o tempo global que cada nodo faz um *broadcast*, mas internamente, cada nodo faz o *broadcast* no seu intervalo de tempo *TGossip*. O nodo n_i tem o seu *clock* interno em 1s e faz o *broadcast* considerando seu *clock* interno, que está relacionado com o *clock* global da simulação em TB1. Como todos os intervalos são calculados com base no tempo anterior, pode haver uma diferença muito grande entre os *broadcasts* de cada nodo, dependendo do tempo total da simulação e dos valores sorteados na função *Random*. Adicionar o valor de *broadcast* anterior para calcular o novo tempo de *broadcast*, é uma maneira de permitir que exista uma diferença de *clock* que pode aumentar ou diminuir com o tempo entre dois *clocks* de nodo n_i e nodo n_j.

Se for necessária uma simulação com diferença de *clock* inicial e os próximos intervalos de transmissão dos nodos serem sempre iguais, ou seja, *TGossip* sem uma variação, basta alterar a variação de tempo com valor igual a zero.

O próximo parâmetro da simulação é o tempo *TGossip*, que é o intervalo de tempo que cada nodo executar o seu *broadcast*. Assim que cada nodo n_i atinge seu TB ele faz o *broadcast* da mensagem e todos os nodos ao seu alcance recebem a mensagem se estiverem dentro do alcance de transmissão definido.

O parâmetro *nodo com falha* é utilizado para inserir uma falha do tipo *crash* dentro da simulação. Este parâmetro é importante no simulador, pois inserindo um nodo n_k falho na

simulação, pode-se saber com certeza que o nodo n_k vai estar falho e em qual momento n_k vai falhar. Para saber o exato momento da falha, o WireS permite que seja definido o parâmetro *tempo da falha*. A partir do tempo definido em *tempo da falha*, o nodo definido no parâmetro *nodo com falha* para de transmitir *broadcasts* devido a uma falha do tipo *crash*. Esse nodo não retorna mais a funcionar. Isto permite que seja medido o tempo de detecção de uma falha com precisão.

Os últimos três parâmetros: Ver Simulação, Salvar Arquivo e Ver Gráfico que podem ser marcados no simulador, definem se no momento da simulação o simulador vai mostrar na tela passo a passo o que está acontecendo (quem está recebendo mensagem, quem está transmitindo), se vai ser gerado um *log* completo, mostrando o intervalo de tempo que cada nodo fez um broadcast, mostrando sua posição e a posição de cada nodo que recebeu a sua mensagem em coordenadas nos eixos *x* e *y*. E o último parâmetro é responsável por gerar um gráfico da simulação, mostrando os caminhos que cada nodo escolheu dentro do intervalo de tempo da simulação. Esses parâmetros são configurados para cada simulação, pois nem sempre é necessário visualizar estas informações, e calcular todas as informações influencia bastante no tempo de cada simulação.

A Figura 14 mostra o simulador depois da execução de uma simulação de uma MANET, mostrando os *broadcasts* realizados pelos nodos.

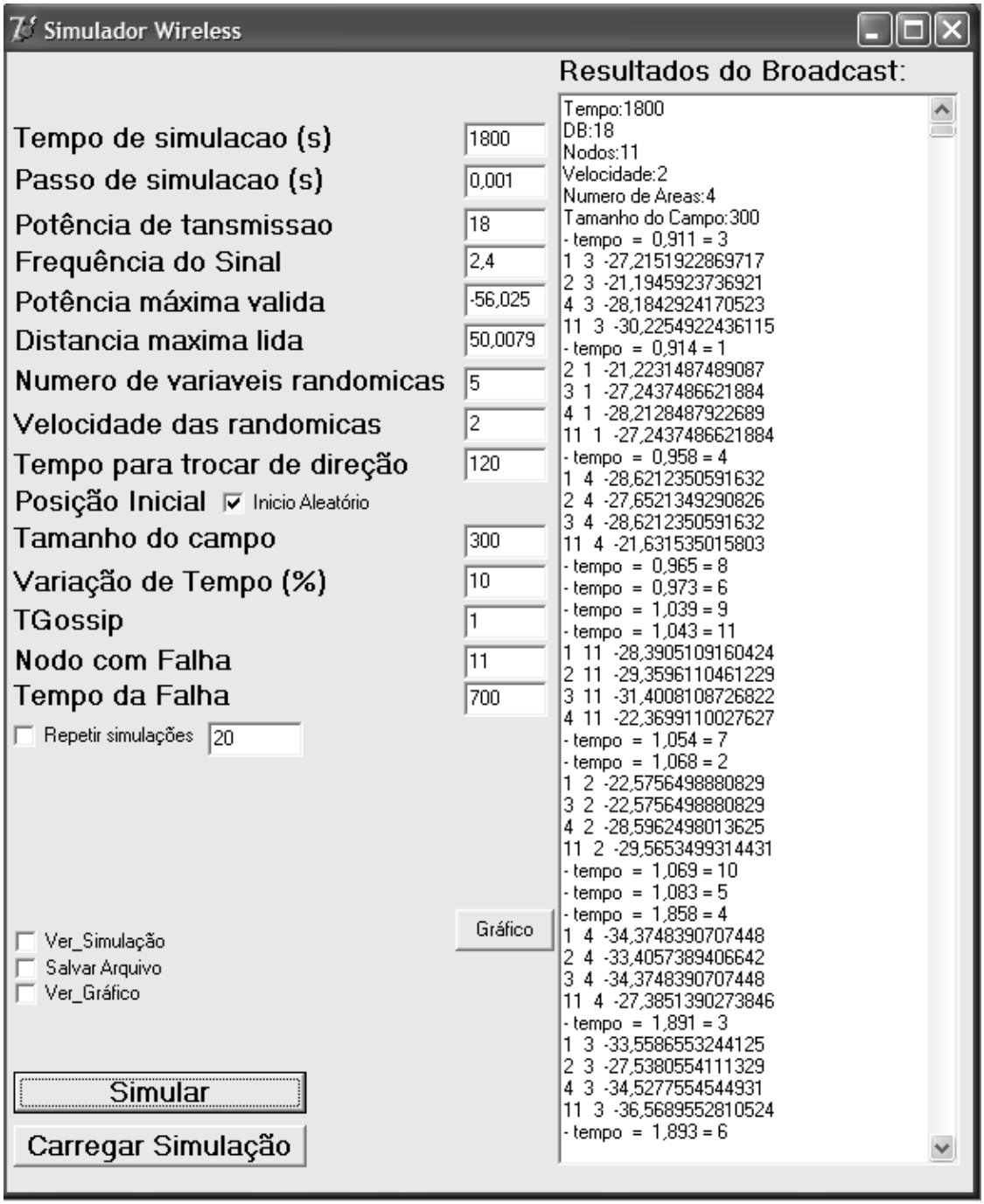

Figura 14 - Tela do simulador WireS após uma simulação

4.2 Modelo de Simulação

Foi definido um campo de 300m^2, contendo variações no número de nodos, alcance de transmissão, mobilidade e diferentes regiões. A potência de sinal (alcance de transmissão de uma mensagem) de cada nodo foi fixada inicialmente em 50 metros, foi feita também uma simulação com alcance de 300 metros. Foram utilizados diferentes valores para as velocidades máxima de movimentação e número de nodos. Para seguir o padrão 802.11 utilizou-se uma

frequência de 2.412 ~ 2.462 GHz (IEEE 802.11g). A largura de banda wireless assumida é de 54 Mbps.

Para os parâmetros de simulação, foram utilizados os mesmos parâmetros previamente estudados e propostos por Friedman e Tcharny (2005).

Os parâmetros de simulação são: o número de nodos, velocidade de movimentação, variação da entrega ou atraso de mensagem.

O número de nodos nas simulações varia de 30 a 60 nodos. As velocidades de movimentação variam de 2 m/s a 8 m/s. Existe uma variação de tempo em 10% em cada intervalo *TGossip* em todas as simulações para que o sistema distribuído apresente um comportamento caótico. Cada nodo conhece todos os seus vizinhos no inicio da simulação, mas não necessariamente são vizinhos diretos (estão dentro da mesma vizinhança). Para testar o DDA, foram utilizados 6 nodos com movimentações definidas e configuradas, e os outros nodos restantes da simulação com movimentações aleatórias.

Utilizamos movimentações configuradas para testar o impacto do Tempo de Detecção de uma falha nos nodos vizinhos, conhecendo assim o momento que um nodo falhou e quais os vizinhos que devem detectar a falha do nodo imediatamente, conforme os parâmetros configurados no simulador.

Os pontos iniciais dos 6 nodos com movimentação conhecida são sempre o ponto de origem nas coordenadas P(x,y) = (0,0). Os nodos restantes que possuem movimentação aleatória possuem ponto inicial aleatório. A cada intervalo *TGossip* os nodos fazem *broadcast* com sua lista de vizinhos, e o intervalo *TGossip* foi definido em 1 segundo em todas as simulações. As simulações foram feitas utilizando tempo total de simulação igual a 30 minutos.

4.3 Simulações

Para verificar se a detecção de movimentação através da potência do sinal afeta o resultado no tempo de detecção e falsas suspeitas do simulador, foi utilizada uma simulação configurada e específica onde é considerada apenas uma área, ou seja, mesmo que receba diferentes potências, é guardado no histórico do simulador apenas uma área R_1 para ver se realmente é importante detectar a movimentação interna do nodo. Foram utilizados 6 nodos controlados. Um nodo n_1 fica parado no ponto inicial, outros dois nodos sobem até a posição máxima do eixo *y* no campo de simulação. Um nodo n_j é configurado para sair do alcance de transmissão durante um intervalo de 50 segundos e voltar repetindo sempre a mesma rota, e

após 700 segundos de simulação, falhar dentro do alcance de transmissão, os outros nodos são configurados para andar para a diagonal saindo do alcance de transmissão e ficando na área máxima do eixo *x* e do eixo *y*. O teste principal é feito em torno do nodo n_i e do nodo n_j.

4.3.1 Testes: Parte 1

A primeira verificação feita é utilizar apenas uma área, onde o DDA atua como um detector de defeitos não atento. A parte do algoritmo que é acionada quando o algoritmo testa a última região e verifica o contador de *CFail*, que é ativado a todo instante. Como o nodo conhece apenas uma área, esta área é sempre a última área do alcance de transmissão do nodo, entrando sempre no teste do contador de *CFail*. Isto faz com que o número de falsas suspeitas seja muito pequeno, com apenas duas falsas suspeitas. A primeira falsa suspeita é quando o nodo n_j atrasa a transmissão e está dentro do alcance de transmissão do nodo n_i. A segunda falsa suspeita é ativada quando o nodo fica um intervalo de tempo fora do alcance de transmissão do nodo n_i, e ao voltar, é calculado o novo valor de *CFail* de n_j.

Após este cálculo, o nodo n_j volta a fazer sua rota, movimentando-se fora do alcance de transmissão e novamente dentro do alcance de transmissão do nodo n_i, mas sem ser considerado como falho. Ao entrar dentro do alcance de transmissão do nodo n_i e falhar no tempo 700 segundos, o nodo n_j será colocado na lista de suspeitas após o intervalo *CFail* de *TGossips*. Isto ocorre porque há apenas uma área de transmissão no algoritmo, entrando na rotina de teste do *CFail*.

Este é um ponto positivo do algoritmo, pois são poucas transições para o estado de suspeita do nodo n_j que está ativo e em movimentação. Mas o ponto que pode ser melhorado é o tempo de detecção. Esta melhora pode ser feita com a entrada de diferentes áreas e detecção de movimentação.

4.3.2 Testes: Parte 2

Foi feito um segundo teste com as mesmas configurações de simulação e movimentação dos nodos, porém foram utilizadas diferentes áreas de potência e foi guardado em um histórico. O nodo n_j ao entrar dentro do alcance de transmissão do nodo n_i e emitir *broadcasts* em diferentes áreas é detectado como movimentando-se. Quando o nodo falha no tempo de 700 segundos, ele não vai estar dentro da ultima área de transmissão, saindo fora da parte do algoritmo que testa o *CFail*. Então é verificado se o nodo estava dentro do alcance de transmissão e se passou o intervalo *TGossip* para adicionar o nodo como suspeito. Este teste

mostrou que o tempo de detecção de falha foi igual a 3,15 segundos, ou seja, o nodo espera apenas dois intervalos para o envio de *broadcast* do nodo n_j para adicionar ele a lista de suspeitos, pois se o nodo estava dentro do alcance de transmissão e não estava na última região, provavelmente o nodo n_j deveria enviar um *broadcast* no máximo em um *TGossip*.

O nodo que testa sua lista a cada intervalo de transmissão, pode cometer muitas falsas suspeita. Entretanto, o DDA utiliza a potência de transmissão, identificando que o nodo está em um ponto interno e não na ultima área, e antes de sair do alcance de transmissão deveria enviar mais uma mensagem dentro da ultima área. Caso uma mensagem não seja enviada e o nodo não estava na última área, então o nodo n_i adiciona o nodo n_j como falho.

Para a detecção de defeitos local, é de extrema importância utilizar a movimentação do nodo (através da detecção com potência de sinal) para ter um tempo de detecção de falha menor.

Finalmente, para testar o impacto da detecção de movimentação local, realizamos diversos testes, sempre considerando diferentes regiões. Em todos os testes realizados, sempre foram realizadas 20 simulações, sendo considerado o valor da média como resultado.

4.3.3 Falsas Detecções vs Velocidade dos Nodos

As simulações realizadas nesta seção pretendem mostrar o comportamento do número de falsas detecções em relação à velocidade de movimentação dos nodos. As simulações da Figura 15 mostraram que o número de falsas detecções cresceu conforme o aumento do número de nodos. Se a velocidade de movimentação dos nodos aumentar, também aumenta o número de falsas detecções.

O aumento relacionado ao número de nodos é devido a configuração inicial do algoritmo onde cada nodo inicialmente conhece todos os nodos da rede, mas não necessariamente faz vizinhança direta com todos os nodos, pois os nodos são iniciados em pontos aleatórios. No inicio da execução do algoritmo, como aumenta o número de nodos, aumenta também o número de suspeita de nodos que ainda não trocaram informações diretamente ou indiretamente.

O aumento do número de falsas suspeitas em função da movimentação dos nodos ocorre devido aos nodos movimentarem-se mais vezes para fora e dentro do alcance de transmissão em intervalos com pouca diferença, onde os nodos vizinhos tem que aumentar e reconfigurar o *CFail* diversas vezes. Como os nodos tendem a movimentar-se muito rápido e entrar e sair de vizinhanças novas onde n_i ainda não conhece n_k, o número de falsas suspeitas

aumenta. Outro fator que pode aumentar o número de falsas suspeitas são as primeiras desconexões da rede. Como um nodo pode movimentar-se para fora da área e para dentro da área, o número de falsas suspeitas tende a crescer. Porem após atingir um determinado padrão de rota, o número de *CFail* tende a mudar muito pouco ou nada, diminuindo a relação de falsas suspeitas conforme o tempo de simulação

Podemos ver que quando existe uma maior movimentação dos nodos, os nodos acabam saindo do alcance de transmissão dos seus vizinhos e acabam sendo suspeitos. Isto aumenta o número de falsas suspeitas.

Quando um nodo entra e sai freqüentemente de um alcance de transmissão tende a ser considerados suspeitos diversas vezes (por diferentes vizinhos), pois a detecção dos nodos é feita localmente. Essa detecção local faz com que as falsas detecções aumentem se um nodo participa de diversos alcances de transmissão de diferentes nodos, pois cada nodo possui um *CFail* que não é compartilhado para cada vizinho.

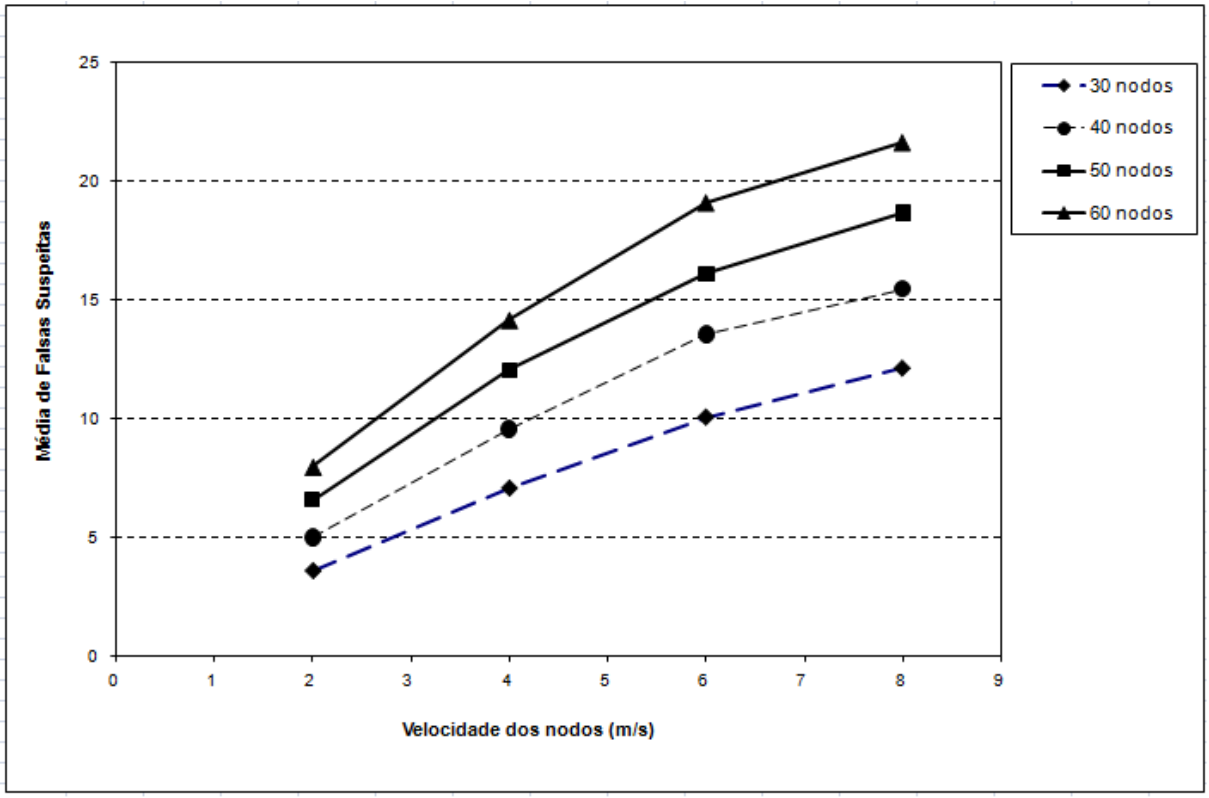

Figura 15 – Aumento da velocidade dos nodos e quantidade de nodos gera um maior número de falsas suspeitas

4.3.4 Tempo de Detecção vs Velocidade dos Nodos

A simulação realizada nesta seção mostra que o tempo de detecção de uma suspeita tende a diminuir quando aumenta a velocidade de movimentação dos nodos e aumenta o número de nodos na simulação, conforme mostra a Figura 16. Como os nodos movimentam-se com maior velocidade, vimos no teste da seção 4.3.1 que o número de falsas suspeitas aumenta. Isto é devido ao modo de detecção de falha, onde após passar um tempo *CFail* um nodo é adicionado como falho. Essa tomada de decisão a todo momento faz com que os nodos fiquem pouco tempo após passar um intervalo *CFail* de *TGossip* sem ser considerado como falho. Devido a este tempo ser curto, o tempo para detecção de uma falha também diminui. A movimentação dos nodos ajuda no sentido que, quando um nodo movimenta-se mais rápido, pode participar da área de transmissão de mais nodos ao mesmo tempo, e como vimos que se um nodo falhar dentro do alcance de transmissão de outro nodo, não sendo a ultima área, facilita a detecção da falha.

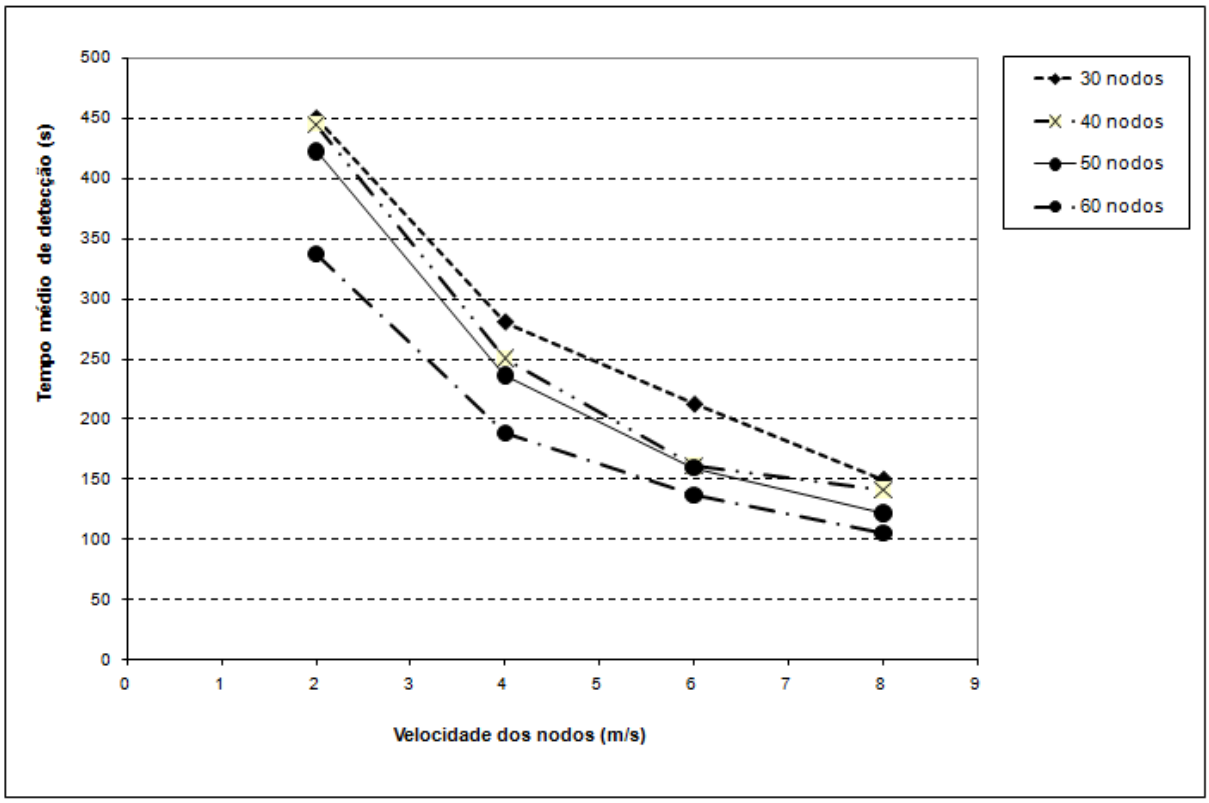

Figura 16 - T_D - Tempo de detecção de uma suspeita diminui quando a velocidade de movimentação aumenta e número de nodos aumenta

4.3.5 Tempo de Detecção vs Média de Falsas Suspeitas

Esta seção mostra os resultados de simulações realizadas, provando que com o aumento do número de falsas suspeitas o tempo de detecção de falhas diminui, conforme mostra a Figura 17. Isso ocorre devido ao nodo fazer uma varredura a cada intervalo *TGossip*. Como o nodo varre sua lista local antes de cada emissão de broadcast para enviar sempre sua lista mais atual, ele tende a suspeitar mais vezes dos nodos vizinhos que não enviaram mensagem no último intervalo *CFail*. Quando se tem menos nodos, ocorre menos comunicação fazendo com que o número do *CFail* aumente. Isto faz com que o tempo de detecção de uma suspeita também aumente, porque quando os nodos não trocam mensagens e estão em movimentação fora do alcance de transmissão é utilizado o contador *CFail* para suspeitar de um nodo.

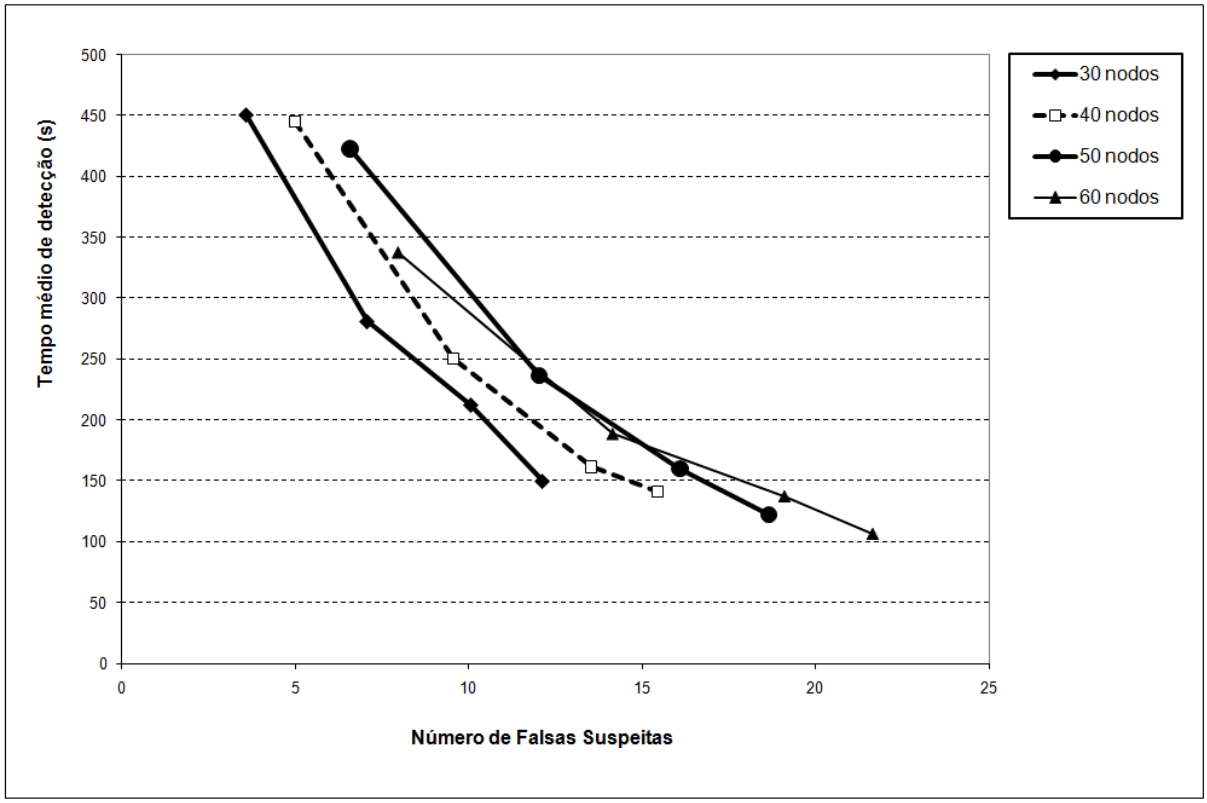

Figura 17 – Maior número de nodos aumenta e falsas suspeitas gera um menor T_D - Tempo de detecção

4.3.6 Tempo de Detecção Mínimo vs Contador *CFail*

Esta seção apresenta uma análise comparando tempo de detecção mínimo para uma falha em relação ao aumento da velocidade dos nodos. Podemos ver na Figura 18 que o DDA possui um tempo de detecção mínimo muito baixo, mesmo aumentado a velocidade dos nodos. Isto deve-se ao fato de que quando uma falha ocorrer dentro do alcance de transmissão, a detecção é quase imediata, levando poucos segundos para detectar a falha comparado com o tempo de detecção quando um nodo está fora do alcance de transmissão. É importante observar que esta é uma abordagem de detecção de falha apenas quando o nodo falhou e estava dentro do alcance de transmissão, e o seu histórico apontava que a região anterior era menor que a ultima região do alcance de transmissão.

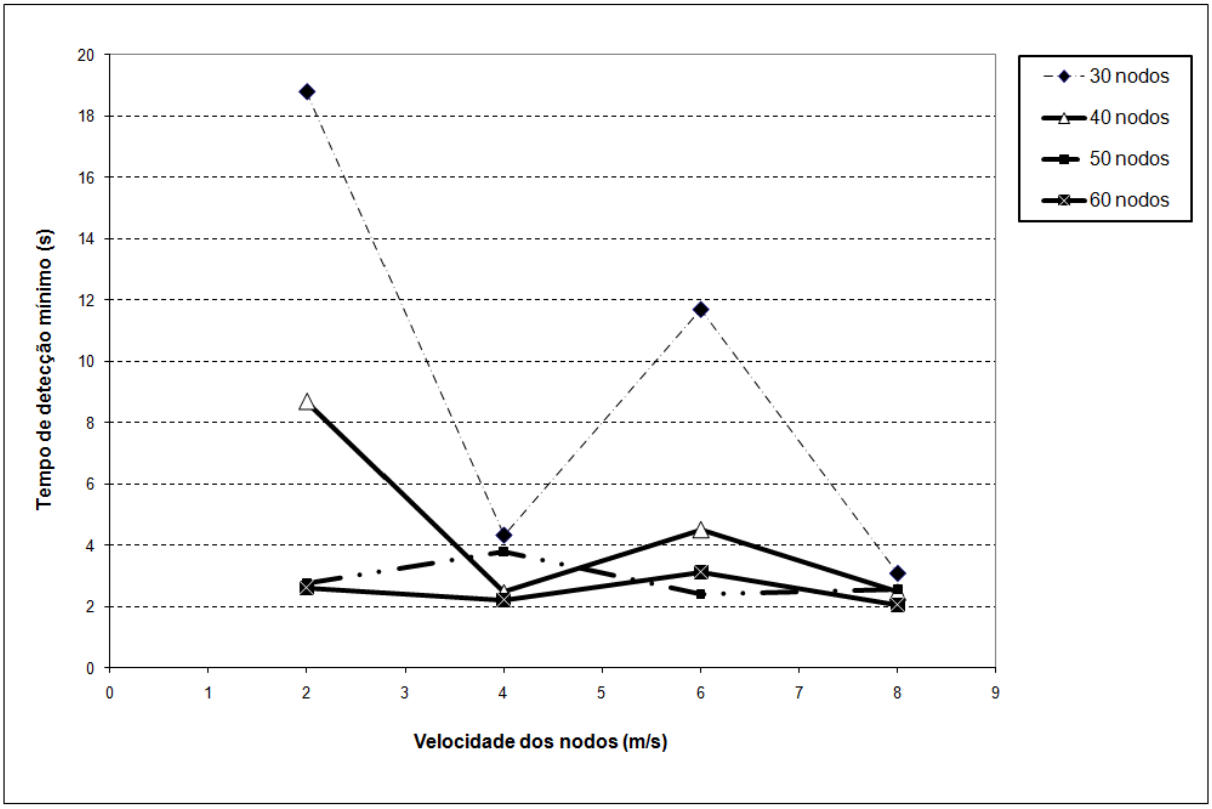

Figura 18 – T_D Mínimo com pouca variação em relação a diferentes velocidades de movimentação e quantidade de nodos

4.3.7 Tempo de Detecção (Máximo) vs Velocidade dos nodos

Esta seção apresenta uma análise comparando tempo de detecção máximo para uma falha em relação ao aumento da velocidade dos nodos. Podemos ver na Figura 19 que o DDA possui um tempo de detecção máximo que diminui conforme aumenta a velocidade dos nodos. Isto deve-se ao fato de que quando aumenta a velocidade de movimentação dos nodos, o *CFail* tende a diminuir, pois se um nodo faz uma rota especifica, ele fica menos tempo fora do alcance de transmissão de um outro nodo vizinho, diminuindo assim o *CFail*. Se o *CFail* diminuir, o tempo de detecção máximo tende a diminuir, pois o DDA utiliza o contador *CFail* como intervalos espera até adicionar um nodo como falho.

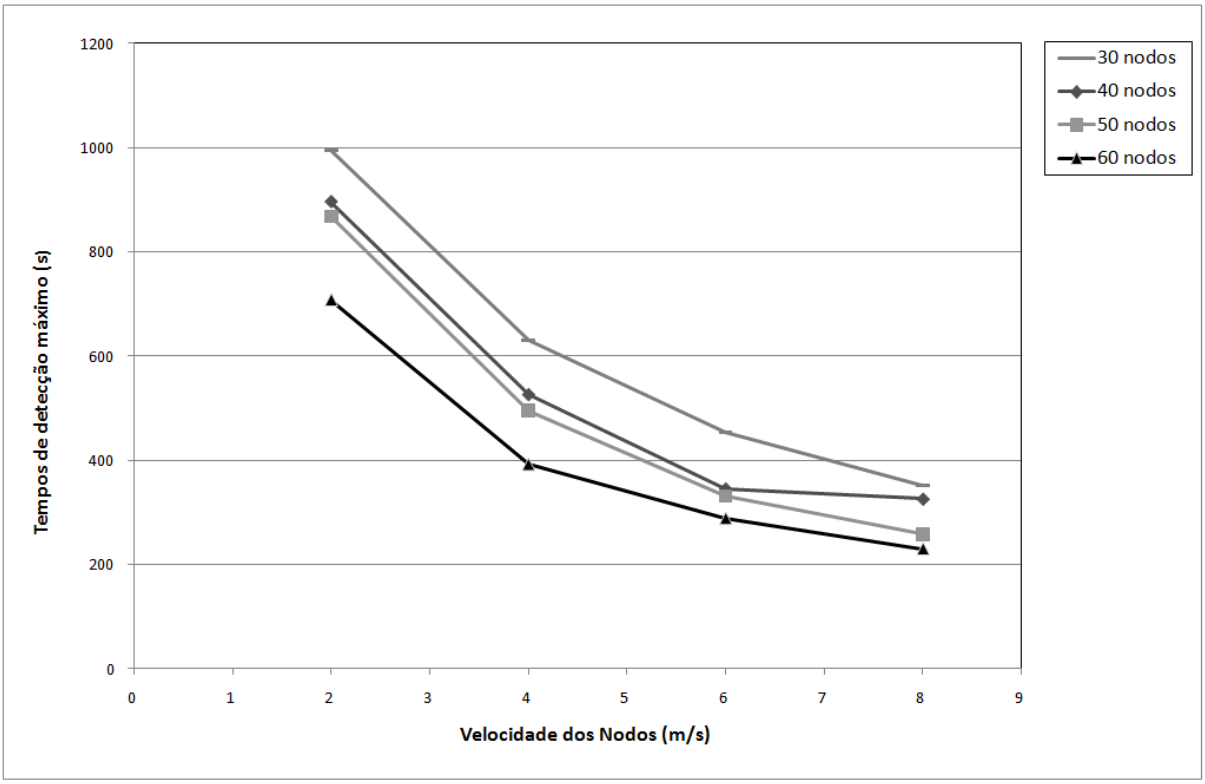

Figura 19 - Tempo de Detecção (Máximo) diminui quando a velocidade de movimentação aumenta e número de nodos aumenta

4.3.8 Comparações entre detectores de defeitos

Esta seção apresenta uma comparação entre detectores de defeitos avaliando o número de falsas suspeitas geradas pelos detectores de defeitos e o tempo mínimo de detecção. Foi considerado a utilização de 30 nodos com velocidade de 2 m/s. A Figura 20 mostra que o DDA gerou menos falsas suspeitas em comparação com os outros detectores de defeitos, ficando com um valor muito próximo ao detector proposto por Sridhar e Friedman. O detector proposto por Pierre Sens corrige este elevado número de falsas suspeitas após suas mensagens de *Query* serem disseminadas na rede, porem o detector comete um elevado número de falsas suspeitas levando em consideração o tempo total de simulação. O DDA consegue esse baixo número de falsas suspeitas em suas simulações devido ao uso do histórico de movimentação e a utilização do contador *CFail*.

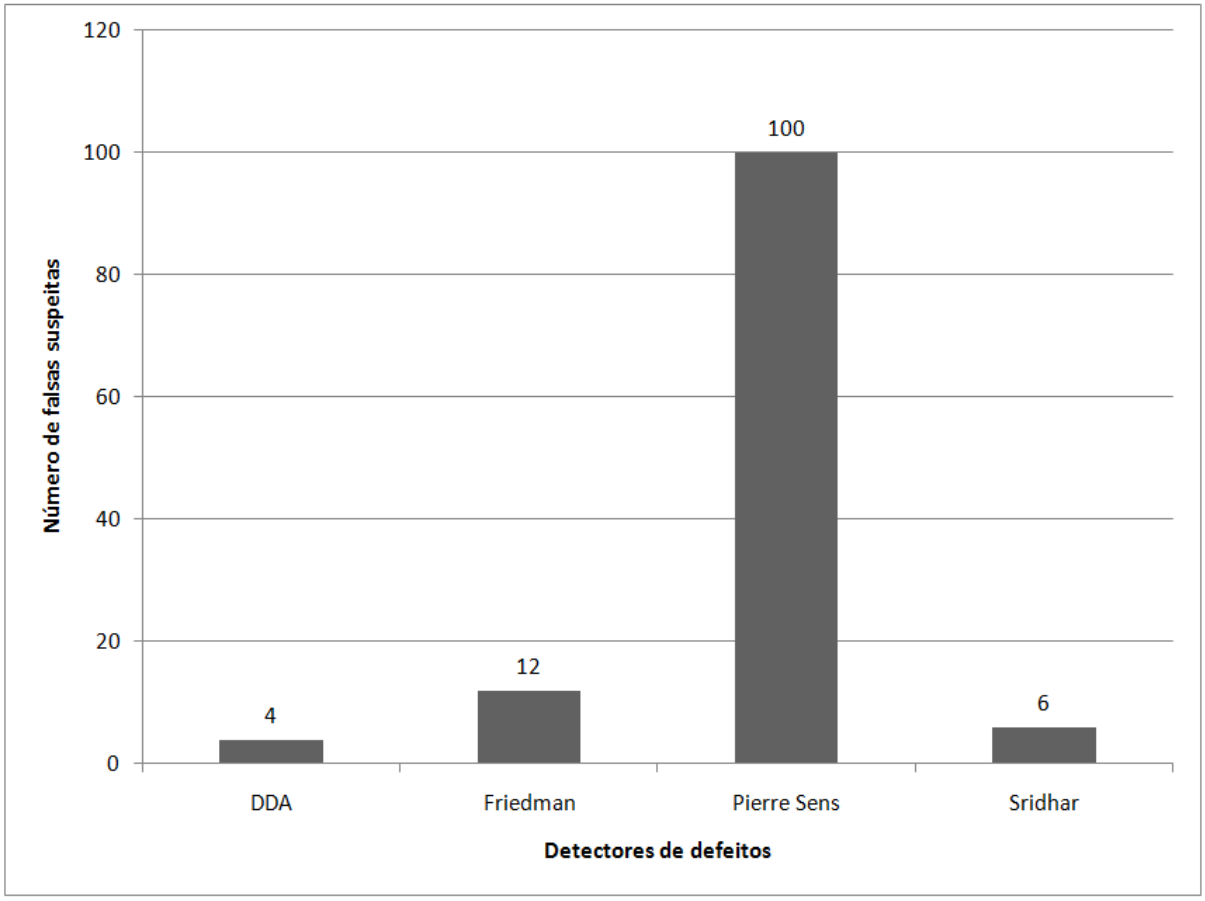

Figura 20 – Comparação entre número de falsas suspeitas entre os detectores de defeitos

A Figura 21 mostra que o detector de Pierre Sens possui um tempo de detecção inferior aos demais detectores, e o DDA só consegue obter um desempenho semelhante com o

tempo de detecção do detector de Pierre Sens com um maior número de nodos na rede, conforme podemos ver na Figura 18. O DDA consegue obter um TD inferior aos detectores de Friedman e Sridhar.

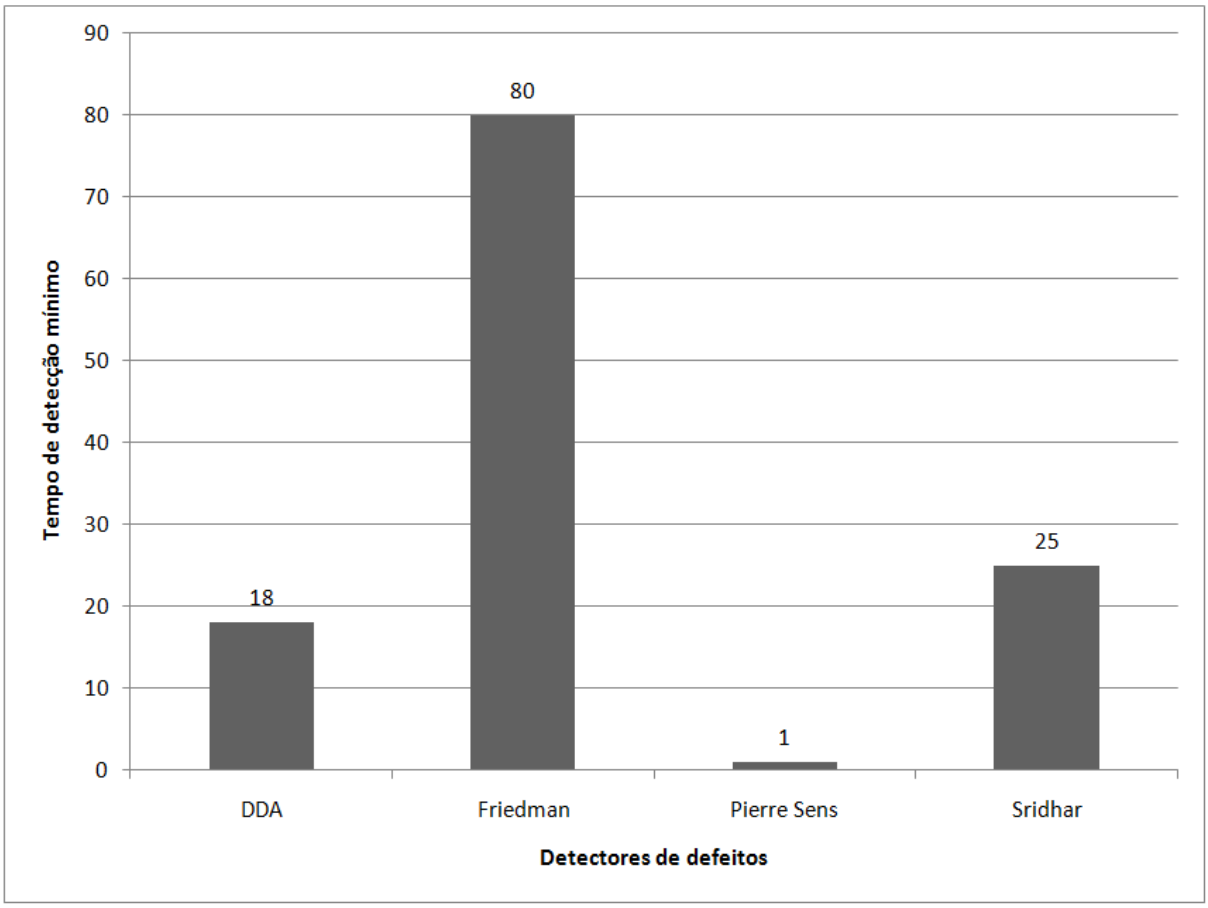

Figura 21 – Comparação do tempo de detecção entre os detectores de defeitos

4.4 Análise de Resultados

Utilizando o tempo médio para detecção da falhas (T_D) e a taxa de erros média, métricas para avaliação de qualidade de serviço de detectores de defeitos [Chen *et al*. 2002], observamos que o DDA possui um bom desempenho em relação aos detectores de defeitos estudados.

O número de falsas suspeitas gerados (transições para o estado de suspeita) pelo DDA é menor em relação aos demais algoritmos estudados na seção 2.8 que utilizam *timeouts*. Quando os algoritmos utilizam *timeouts* com um valor pequeno, acabam gerando muitas falsas suspeitas, devido a atraso na transmissão ou movimentação dos nodos para fora do

alcance de transmissão. O DDA diminui muito o número de falsas suspeitas pois quando um nodo movimenta-se para fora do alcance de transmissão ele adapta o tempo para suspeitar de um nodo pois identifica a movimentação. Essa adaptação não influencia no tempo de detecção quando um nodo está dentro do alcance de transmissão, fazendo com o que o DDA tenha valores muito baixos para detecção quando um nodo falha dentro de sua região de outro nodo. O DDA consegue ter a opção de tomar decisões diferenciadas porque identifica a movimentação e localização de um nodo n_i dentro que está dentro da região de outro nodo n_j

A dificuldade de diminuir o tempo global do tempo de detecção é fazer esta detecção local ser compartilhada sem aumentar o número de mensagens enviadas pela rede.

4.5 Conclusão dos testes

O DDA consegue obter um bom desempenho em relação ao tempo de detecção e média de falsas suspeitas - *Average Mistake Rate* (λ_M) locais porque não suspeita de um nodo p_i que sai do seu alcance de transmissão em uma rota contínua, e isto deve-se a utilização de intervalos de tempo *CFail*. Este intervalo de tempo *CFail* é utilizado quando o nodo percebe uma movimentação de um nodo que está saindo do seu alcance de transmissão, esperando um intervalo de tempo, antes de colocar o nodo na lista de suspeitos. Essa percepção de movimentação só pode ser utilizada se for guardado um histórico de áreas nos últimos 2 intervalos de tempo de transmissão. Esse intervalo pode ser suficiente (se o nodo repetir a sua rota diversas vezes) e é adaptável conforme as novas rotas descobertas (que levam mais tempo). Para adaptar os intervalos de tempo, o DDA não precisa conhecer quantos *Hops* um nodo está do outro, ou informações sobre o *Jitter* da rede, sendo um algoritmo inteligente que consegue adaptar e obter essas informações apenas observando o funcionamento da rede através da troca de mensagens. O algoritmo possui um tempo de detecção mínimo muito baixo para as simulações realizadas e um número de falsas detecções muito pequeno em todas as simulações realizadas.

5 Conclusões

Detecção de defeitos em redes móveis ainda é um assunto que merece algoritmos otimizados e tem sido alvo constante de pesquisa acadêmica. Fatores importantes a serem considerados incluem o dinamismo da rede, a escassez de recursos, a comunicação sem fio por difusão de mensagens e a dificuldade de se diferenciar entre um nodo falho e móvel (sistema assíncrono). Neste trabalho é apresentado um novo detector de defeitos não-confiável com suporte à mobilidade de nodos em redes dinâmicas e um simulador MANET.

Para a implementação do algoritmo proposto, foi desenvolvido um simulador WireS que permite simular nodos se movimentando dentro de uma área definida, e com comunicação ad-hoc. O diferencial do simulador proposto é proporcionar ao desenvolvedor de algoritmos para detecção de defeitos a utilização da potência do sinal, como parâmetro a ser usado em um detector de defeitos. Além de movimentações definidas pelo usuário, é permitida a inclusão de nodos com movimentação com velocidade randômica, bem como posicionamento inicial randômico.

O algoritmo de detecção de defeitos proposto baseia-se no tradicional algoritmo *Gossip* e explora o conhecimento do nível de potência no recebimento de mensagens que os nodos sem fio costumam oferecer. A detecção de mobilidade é feita através da manutenção de um pequeno histórico de movimentação em cada nodo detector de defeitos e decisões sobre falhas são tomadas localmente, sem a necessidade de acordo global, permitindo maior escalabilidade. Resultados de simulação demonstram a eficiência do algoritmo tanto em termos de precisão quanto de desempenho.

Com o uso do DDA, é possível se determinar, com precisão e em um tempo muito curto (o tempo *TGossip*) a detecção de uma falha, quando a mesma ocorre dentro do alcance de transmissão entre os nodos.

Quando a falha ocorre fora do alcance de transmissão de um determinado nodo, o tempo máximo de detecção da falha do nodo, será igual ao tempo *CFail*, neste caso, sendo igual aos demais algoritmos que adaptam seu *Timeout*.

Outra vantagem do DDA é o baixo número de falsas detecções, quando os nodos possuem rotas de movimentação repetitivas, pois o algoritmo acaba adaptando-se a essas rotas, sem suspeitar quando um nodo sai do alcance de transmissão, apenas para fazer a sua rota.

O DDA apresenta uma nova abordagem que pode ser utilizada em outros detectores de defeitos que é a análise da potência do sinal recebido. Com esta análise é possível utilizar decisões diferentes dentro do mesmo detector de defeitos para a detecção de falha dos nodos.

Para trabalhos futuros, sugerimos:

- o estudo da identificação de mobilidade do próprio nodo, onde sabendo de sua movimentação e seu afastamento do alcance de transmissão de outros nodos, informa que está saindo do alcance de transmissão para não ser suspeito.
- análise de um detector de defeitos onde, usando potência de sinal, quando um nodo detectar uma falha local, transmita em broadcast, a todos os demais nodos, a existência deste nodo falho, sem mensagens de broadcast adicionais.
- a utilização de um detector de defeitos com potência do sinal, onde o contador *CFail* seja compartilhado como parâmetro global, ao invés de ser apenas um parâmetro local.
- análise de um detector de defeitos que utilize a potência do sinal, onde a potência do sinal pode ser diferentes entre os nodos do sistema, necessitando assim uma nova estratégia para ao cálculo das regiões e para o tamanho do histórico.

Referências Bibliográficas

AGUILERA, M.K., CHEN, W., TOUEG, S. **Heartbeat: A timeout-free failure detector for quiescent reliable communication**. Workshop on Distributed Algorithms, páginas 126–140. 1997

BIRMAN, K. P. **Building Secure and Reliable Network Applications**. 1996

BOICHAT, R.; DUTTA, P.; GUERRAOUI, R. **Asynchronous leasing** . 7th IEEE Intl. Wksp. On Object-Oriented Real-Time Dependable Systems (WORDS'02), páginas 180–187. 2002.

BURNS, M.; GEORGE, A; WALLACE, B. **Simulative Performance Analysis of Gossip Failure Detection for Scalable Distributed Systems**. *Cluster Computing*. 2(3). 1999

CASON, D.; GREVE, F. G. P. **Resolução do Consenso em Redes Móveis Ad-Hoc a Partir de um Conjunto Dominante Conexo**. Workshop on Tests and Fault Tolerance, LADC'2009: Latin American Symposium on Dependable Computing, João Pessoa, PB, 31 de Agosto. 2009.

CHANDRA, T. D.; TOUEG, S. **Unreliable failure detectors for Asynchronous systems**. Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing, páginas 325–340. ACM Press, Agosto. 1991.

CHANDRA, T. D.; TOUEG, S. Unreliable **failure detectors for reliable distributed systems**. Journal of the ACM, 43(2):225-267. 1996.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos: Conceitos e Projeto**. Bookman, 792p. 2007.

FISCHER , M. J., LYNCH, N. A. , PATERSON, M. S. **Impossibility of distributed consensus with one faulty process**. Journal of the ACM (JACM), v.32 n.2, páginas 374-382, Abril. 1985.

FELBER, P. et al. **Failure detectors as first class objects.** International Symposium on Distributed Objects and Applications (DOA'99). Edinburgh, Escócia. páginas.132–141. IEEE Computer Society. 1999.

FETZER, C.; SCHMID, U.; SUSSKRAUT, M. **On the possibility of consensus in asynchronous systems with finite average response time**. ICDCS ´05 , páginas 271-280, 2005.

FRIEDMAN, R. TCHARNY, G. **Evaluating failure detection in mobile ad-hoc networks**.TRCS-2003-06,Technion,Israel, Outubro.2003.

GOLDSMITH, A. **Wireless Communications**. Stanford University, California. Setembro, 2005.

GUPTA, I.; CHANDRA, T.D.; GOLDSZMIDT, G.S. **On scalable and efficient distributed failure detectors**. PODC'01, páginas 170-179, Nova Iorque, Estados Unidos, 2001.

GRACIOLI, G.; NUNES, R. C. **Detecção de defeitos em redes móveis sem fio: uma avaliação entre as estratégias e seus algoritmos**. Anais do Workshop de Testes e Tolerância a Falhas (SBRC/WTF), Belém/PA. 2007.

HUTLE, M. **An efficient failure detector for sparsely connected networks**. Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2004), Innsbruck, Austria.

JALOTE, P. **Fault Tolerance in Distributed Systems**. Prentice Hall. 1994.

JOHNSON, D. B.; MALTZ, D.A.; **Dynamic Source Routing in Ad Hoc Wireless Network**. Computer Communications Review – Proceedings of SIGCOMM ´96, Agosto. 1996.

NUNES, B. R. P.; GREVE, F. G. P. **Protocolo Assíncrono para a Gestão da Filiação ao Grupo em Redes Móveis Ad Hoc**. Workshop on Tests and Fault Tolerance, with LADC'2009 Latin American Symposium on Dependable Computing. João Pessoa, PB, 31 de Agosto. 2009.

RAPPAPORT , T. S. **Wireless Communications: Principles & Practice**, Prentice Hall, New Jersey, 1996.

RENESSE, R.; MINSKY, Y.; HAYDEN, M. **A Gossip-style failure detection service**. Proceedings of the IFIP International Conference on Distributed Systems and Platforms and Open Distributed Processing (Middleware' 98). 1998.

SENS, P. et al. **Um detector de falhas assíncrono para redes móveis e auto-organizáveis**. XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – SBRC 2008. 2008.

SUBRAMANIYAN, R, et al. **GEMS: Gossip-Enabled Monitoring Service for Scalable Heterogeneous Distributed Systems**. Cluster Computing, 9(1), pp.101-120. 2006.

SRIDHAR, N. **Decentralized local failure detection in dynamic distributed systems**. 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06). pp.143-154. 2006.

TANENBAUM, A. S.; STEEN, M. V. **Sistemas Distribuídos**: Princípios e Paradigmas. 2. ed. São Paulo. Editora Pearson Prentice Hall. 2007.