ESCOLA POLITÉCNICA
DE PERNAMBUCO

d.s.c. UPE
Departamento de Sistemas e Computação

IMPLEMENTAÇÃO E AVALIAÇÃO DE NOVOS MODELOS DE MOBILIDADE PARA O SIMULADOR JIST/SWANS

Trabalho de Conclusão de Curso

Engenharia da Computação

Fagner Pereira de Araújo
Orientador: Prof. Renato Mariz de Moraes

UNIVERSIDADE
DE PERNAMBUCO

FAGNER PEREIRA DE ARAÚJO

IMPLEMENTAÇÃO E AVALIAÇÃO DE NOVOS MODELOS DE MOBILIDADE PARA O SIMULADOR JIST/SWANS

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, novembro de 2008.

A minha família.

Agradecimentos

A minha família pelo apoio sempre constante.

Ao Prof. Renato Moraes pela ótima orientação.

A minha querida irmã Flavia Araujo por sempre me ajudar.

Quero também agradecer aos funcionários do TJPE André Poroca e Norma de Miranda.

Resumo

A crescente utilização de dispositivos sem fio tem sido motivação para vários estudos relacionados com a mobilidade. Um desses estudos é a avaliação de desempenho dessas redes sob diferentes cenários de mobilidade com o objetivo de investigar a comunicação de nós que as compõem visando alcançar uma tendência em seu comportamento, possibilitando análises posteriores para uma determinada configuração de tráfego. Este presente trabalho tem por objetivo propor implementações de novos modelos de mobilidades até então inexistentes nos simulador JiST/SWANS e uma avaliação de desempenho do protocolo AODV para os modelos propostos em redes com densidade crescente de nós. O JiST/SWANS é um simulador de redes sem fio altamente escalável com a capacidade de simular rede com milhares ou milhões de nós. A análise de desempenho redes aqui realizada se baseia em três métricas comumente usadas para medir o desempenho de redes: vazão, overhead e atraso de descobertas de rotas. Com os resultados obtidos foi possível verificar que a crescente densidade de nós nas redes é fator limitante da comunicaçao, onde os valores médios das métricas demonstraram uma degradação do desempenho dessas redes.

Abstract

The proliferation of wireless network technologies motivate several studies about mobility. One of them is the evaluation of peformace these networks upon varied scenarios varied of mobility aiming to explore the behavior of the nodes communication in order to achieve a tendency in its behavior, making it possible to obtain further analysis or a certain traffic configuration. The goal of this study is to propose implemetations of new mobility models for the JiST/SWANS simulator and their performace evaluation for AODV protocol in wireless networks with increasing nodes density. The performance analysis presented here is based on three metrics commonly used to measure network performance: throughput, overhead and route discovery delay. The JiST/SWANS is highly scalable wireless network simulator with capacity of simulating large networks. Using the results obtained, it was possible to check that increasing nodes density is a limited factor of communication, where the mean values of those metrics showed a degradation performance of the tested networks.

Sumário

Índice de Figuras

Índice de Tabelas

Índice de Listagens

Tabela de Símbolos e Siglas

JiST/SWANS – Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator

NS-2 – Network Simulator version 2 (Network Simulator versão 2)

CBR – Constant Bit Rate

Mbps – Mega bits per second (Mega bits por segundo)

MAC – Media Access Control (Controle de Acesso ao Meio)

NAM – Network Animator

UDP – User Datagram Protocol (Protocolo de Datagrama do Usuário)

MANET – Mobile Ad Hoc Network (Rede Ad Hoc Móvel)

AODV – Ad hoc On-Demand Distance Vector (Vetor de Distância Ad hoc Sob Demanda)

GHz – Gigahertz

s – segundos

m/s – metros por segundos

RWP – Random Waypoint

GPRS – General Packet Radio Service

GloMoSim – Global Mobile Information Systems Simulation Library

VANET – Vehicular ad-hoc Network

STRAW – STreet RAndom Waypoint

RREP – Route Repley

RREQ – Route Request

RERR – Route Error

JVM – Java Virtual Machine

RPGM – Mobility Reference Point Group

Capítulo 1

Introdução

A importância da comunicação sem-fio tem crescido cada vez mais na sociedade. A cada dia ela viabiliza atividades mais importantes, que trazem benefícios e agregam valor ao meio no qual atua. Geralmente, novos avanços nessa tecnologia surgem a cada dia. A tendência é que o mundo se torne cada vez mais dependente da comunicação sem-fio. De fato, Redes *Ad hoc* Móveis (MANETs) *Mobile Ad hoc Networks*, é uma importante área de redes sem-fio que está tendo um evidente crescimento nos últimos anos.

MANETs são redes de comunicações sem-fio não infra-estruturadas, que comportam certo número de dispositivos móveis e/ou estáticos com capacidade de comunicação restrita e consumo de energia limitado. A topologia da rede, devido à mobilidade, é, em geral, dinâmica e pode mudar rapidamente e imprevisivelmente ao longo do tempo. Para entender e analisar o desempenho das MANETs através de seus protocolos de comunicação, modelos de mobilidade, modelos de consumo de energia, etc., são usados estudos em simuladores para representar os comportamentos realístas dos dispositivos da rede e para refletir a dinâmica natural das MANETs.

1.1 Motivação

Para realizar pesquisas principalmente em redes sem-fio é necessário ter eficientes e confiáveis sistemas de simulação. Outro importante aspecto que deve ser bem investigado é a mobilidade em MANETs sob diferentes cenários de movimento. O modelo de mobilidade representa a sucessiva movimentação das estações comunicantes ou não numa rede sem fio móvel. Diversos estudos [1], [2], [3], [4], [5] e [6] têm demonstrado que o modelo de mobilidade escolhido em termos de simulação tem efeito significativo no desempenho de protocolos de comunicação sob investigação nessas redes. Em outras palavras, um protocolo

pode ter um bom desempenho sob um padrão de mobilidade, mas ter um mau desempenho sob outro padrão. O padrão de mobilidade de nós é, portanto um fator chave que impacta diretamente nos resultados da simulação, e, conseqüentemente no mundo real. Então, a escolha de um modelo de mobilidade é crucial e deve refletir nos cenários de aplicação.

Outro fator impactante na análise das MANETs é a escolha do simulador para validar seus experimentos. Os simuladores mais utilizados em pesquisas são NS-2 [7], GloMoSim [8]. Estudos anteriores [9] e [10] demonstram um enorme potencial no simulador JiST/SWANS. Por outro lado, observou-se em [11] que não há muitas implementações de modelos de mobilidade no simulador JiST/SWANS.

As duas principais razões pelos quais o simulador JiST/SWANS foi escolhido como software de simulação a ser usado neste trabalho, são sua boa aceitação na comunidade acadêmica e, principalmente, pela sua grande capacidade de processar simulações em redes densas, devido a sua eficiente utilização da máquina virtual Java [12]. Adicionamente, trata-se de uma ferramenta de código aberto, possibilitando estudos posteriores mais aprofundados e, até mesmo, implementação de novos trechos de código e alterações no código-fonte original, caso seja necessário ou conveniente.

1.2 Trabalhos Relacionados

Antes de iniciar o desenvolvimento desse trabalho, foi efetuada uma revisão da literatura com o objetivo de identificar outros trabalhos que tivessem as mesmas características relacionadas a esse. Entre eles, destaca-se a entidade vanet.info [13], que realiza pesquisas na área de redes veiculares, mais conhecidas como *Vehicular Ad Hoc Networks* (VANETs). A venet.info incorporou diversas melhorias ao simulador JiST/SWANS, a exemplo do suporte ao Java 5.0, e acrescentou o modelo de mobilidade *STreet RAndom Waypoint* (STRAW). O projeto AquaLab desenvolveu uma extensão do JiST/SWANS chamada Swans++ [14], que acrescenta uma implementação para GPRS [15] e para o protocolo DSR [4]. Em [16], um modelo de energia até então inexistente nesse simulador foi proposto. Em [17], foi proposto

um novo protocolo da camada de acesso ao meio (MAC) baseado no consumo eficiente de energia, que se chama Sensor-MAC, que foi implementado no simulador JiST/SWANS.

1.3 Objetivos

Devido ao evidente crescimento do simulador JiST/SWANS, vislumbrado na comunidade acadêmica, o objetivo principal desse trabalho é desenvolver uma extensão para esse simulador acrescentando mais três modelos de mobilidade que ainda não estão presentes no simulador em estudo, disponibilizando dessa forma novas possibilidades de avaliações para pesquisas.

Nesse sentido, os objetivos específicos consistem em:

- Implementar três modelos de mobilidade, dois individuais e um em grupo, para redes móveis no simulador JiST/SWANS ainda não presentes nele (Uniforme, *Gauss-Markov e* Grupo Uniforme). Esses modelos ainda não estão disponíveis nele. Os trabalhos [18] e [19] são usados como base para essa implementação.
- Comparar os modelos de mobilidade acrescentados em relação ao modelo *Random Waypoint* (RWP) existente no simulador JiST/SWANS a partir do protocolo de roteamento AODV, como parâmetro de avaliação.

Para todos os modelos incluídos no simulador foi realizada uma avaliação baseada nas medidas de desempenho, definidas na próxima seção, em relação ao modelo RWP já existente no JiST/SWANS. Os resultados foram analisados a partir da média das métricas utilizadas em 10 (dez) amostras de cada cenário testado.

1.3.1 Métricas Avaliadas

Para estudar o comportamento das redes testadas diante dos modelos de mobilidade propostos, e, para analisar o desempenho do protocolo de comunicação sob os diferentes padrões de mobilidade foram estabelecidas as seguintes métricas.

- **Taxa de Entrega de Mensagens;**
- **Número de Pacotes de Sobrecarga;**

- **Atraso Médio de Descoberta de Rotas.**

De acordo com os objetivos aqui definidos, uma parcela de tempo considerável foi investida na investigação dos modelos de mobilidade existentes e não existentes no simulador JiST/SWANS. Posteriormente, foi feito um estudo do funcionamento do simulador JiST/SWANS antes dos experimentos de interesse serem realmente iniciados. Esse estudo abrangeu principalmente o funcionamento básico do JiST/SWANS, incluindo arquitetura desse simulador, seus formatos de arquivos de trace, usados para posterior extração das métricas escolhidas para análise. Finalmente, foi feito um levantamento das principais características do protocolo AODV. Portanto esse trabalho propõe acrescentar novas formas de mobilidade ao simulador JiST/SWANS, colaborando com a comunidade acadêmica que utiliza o simulador JiST/SWANS e, ao mesmo tempo, avaliando esses novos modelos acrescentados utilizando o protocolo AODV.

1.4 Estrutura da Monografia

Este documento foi estruturado em capítulos da seguinte maneira:

O **Capítulo 2 –** Estudo da Mobilidade das MANETs no Simulador JiST/SWANS – Apresenta uma visão geral sobre mobilidade em MANETs, mostrando modelos que existem no JiST/SWANS e os modelo propostos nesse trabalho.

O **Capítulo 3** – Visão Geral do Simulador JiST/SWANS – Apresenta os conceitos básicos da arquitetura do simulador JiST/SWANS.

O **Capítulo 4** – Implementação da Mobilidade e da Simulação com o Protocolo AODV – Com base no que foi apresentado nos capítulos anteriores, este capítulo propõe a implementação dos modelos de mobilidade propostos e da simulação para realização dos testes propostos nesse trabalho.

O **Capítulo 5** – Considerações Finais e Trabalhos Futuros – Apresenta as conclusões além de propor trabalhos futuros.

Capítulo 2

Estudo da Mobilidade das MANETs no Simulador JiST/SWANS

Este capítulo demonstra algumas características básicas das MANETs. Em seguida, uma análise dos modelos que existem implementados no simulador JiST/SWANS e os modelos propostos nesse trabalho. As MANETs são uma tecnologia promissora onde se encontram bastantes desafios a ser superados. A eficiência na comunicação nesse tipo de rede é um deles e a mobilidade é um componente essencial na avaliação de desempenho em MANETs. É comum usar modelos de mobilidade para representar o padrão de movimentação dos dispositivos na rede, uma vez que estas redes, sob diferentes cenários de mobilidade, podem apresentar variados níveis de desempenho, por exemplo, em um protocolo de roteamento ou na camada de acesso ao meio (MAC) de acordo com [1], [2], [3], [4], [5] e [6].

2.1 MANETs e Mobilidade

As redes sem fio são divididas em duas categorias: infra-estruturadas e não infra-estruturadas. As redes infra-estruturadas são aquelas que possibilitam um grau de mobilidade aos usuários em um ambiente que já possui uma rede fixa. Através de Estações de Rádio Base o usuário consegue ter mobilidade dentro da área de cobertura sem-fio. As redes não infra-estruturadas, conhecidas por redes *ad hoc*, não necessitam de conexão com uma estrutura fixa.

As redes *ad hoc* são dinâmicas e, em alguns casos, temporárias, onde as estações integrantes da rede de comunicação são responsáveis pelo gerenciamento e manutenção da mesma. As redes *ad hoc* ainda podem ser móveis (MANETs – Mobile Ad Hoc Networks). Neste caso, os módulos integrantes da estrutura de comunicação movimentam-se arbitrariamente em diferentes sentidos e velocidades.

As MANETs proporcionam a existência de estruturas de comunicação em ambientes com muitos obstáculos para criação de uma estrutura de rede. Um cenário de operação militar pode ser visto como uma situação em que as MANETs são requeridas para proporcionar a comunicação em um ambiente hostil e geograficamente acidentado, no qual uma estrutura de rede fixa é inviável.

2.1.1 Desafios da Mobilidade

Varios trabalhos [1], [2], [3], [4], [5] e [6] indicam que a mobilidade é um aspecto que não está completamente investigado em redes dinâmicas. O efeito da mobilidade nos dispositivos nesse tipo de rede é um componente essencial na avaliação de desempenho das MANETs. Geralmente, o campo de investigação sobre esse tema tem se detido, basicamente, ao desempenho de protocolos de roteamento usando limitados modelos de mobilidade para representar o padrão de movimentação dos dispositivos. Entretanto, são poucos os trabalhos que se voltam para estudar a mobilidade usando padrões de movimentação mais fiéis à realidade em cenários onde possa haver maior exatidão na avaliação dos protocolos de comunicação em redes sem-fio. Há um grande número de pesquisadores que utilizam mobilidade em suas avaliações, entretanto estes modelos não representam um comportamento fiel da realidade. Os modelos propostos neste trabalho foram escolhidos por que representam melhor a dinâmica de uma rede prática, o que possibilita uma avaliação mais precisa dos protocolos de comunicação, os quais se baseiam nos trabalhos de [18] e [19].

O desempenho das MANETs é altamente influenciado pelo padrão de mobilidade dos nós na rede [3], [20]. De fato, o comportamento da mobilidade é um dos mais importantes fatores ambientais que determina o desempenho e influencia no projeto da rede. Dessa forma, é importante que sejam desenvolvidos modelos que tenham precisão na captura dos dados em movimento e, ao mesmo tempo, eficiência em simulação e comunicação.

A comunidade que estuda comunicação sem fio conta com modelos muito simples tais como *Random Walk* e *Random Waypoint* [19]. Embora esses modelos continuem sendo usados em análise e simulação, entretanto, recentemente foi descoberto que estava sendo utilizados de forma errônea em simulações [1]. Esse estudo é um dos que aponta que RWP

não descreve fielmente a realidade influenciando diretamente na precisão dos dados em redes móveis. Nos últimos anos, grande esforço tem sido feito no desenvolvimento de modelos mais fiéis à realidade.

2.2 Modelos de Mobilidades no JiST/SWANS

Como já mencionado o simulador JiST/SWANS, foco desta pesquisa, o qual será descrito no próximo capítulo, possui poucos modelos de mobilidade implementados de acordo com a documentação descrita por [21]. Existem apenas os seguintes modelos de mobilidade listados abaixo:

- *Random Walk*;
- *Random Waypoint*;
- *Random Direction*;
- *Boundless Simulation Area*;
- *Random Street Waypoint*.

2.2.1 *Random Walk*

Trata-se de um modelo de mobilidade simples baseado em direções e velocidade aleatórias. Possui independência temporal, ou seja, a não existência de memória que significa que as sucessivas posições dos são indepedentes das posições anteriores dos nós. Conforme [19], funciona da seguinte forma: em um intervalo de tempo constante ΔT, um nó move-se de sua posição atual para outra posição escolhendo aleatoriamente uma direção D e uma velocidade constante V. Geralmente a velocidade constante V do nó é escolhida uniformemente entre *vmin* e *vmax*. Já a direção D é escolhida uniformemente entre $(0,2\pi]$. Sucessivamente, ao alcançar o destino, novamente uma direção e uma velocidade são escolhidas e o nó se move até esta posição. A cada novo instante de tempo, o valor das variáveis de direção e velocidade

não possui nenhuma relação com os valores anteriores, ocasionando movimentos bruscos dos nós. Um exemplo do padrão de movimentação de um nó pode ser visto na Figura 1.

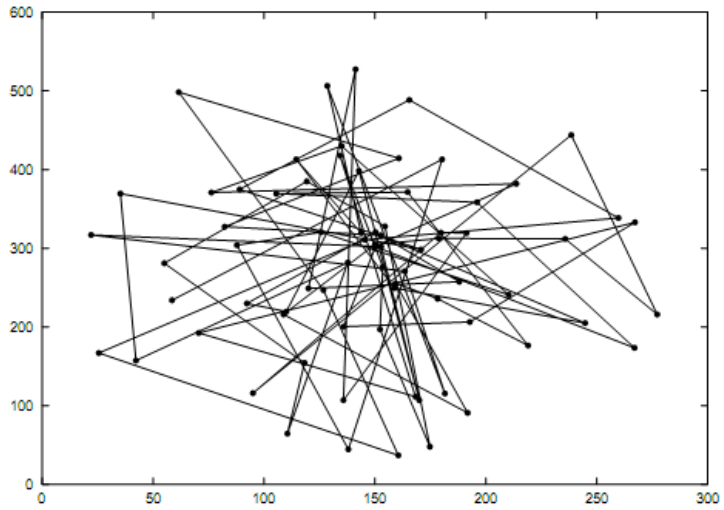

Figura 1. Exemplo de movimentação gerada no padrão *Random Walk* [19].

2.2.2 *Random Waypoint* (RWP)

Modelo de mobilidade que apresenta o funcionamento a partir do posicionamento inicial das estações que é aleatório e, usualmente, segue uma distribuição de probabilidade uniforme dentro da área de simulação. A estação permanece nesta posição por um intervalo de tempo aleatório chamado tempo de pausa. Após o término deste período a estação escolhe uma nova posição uniformemente dentro da área de simulação. A velocidade de deslocamento do nó *V* entre a origem e o destino é uniformemente distribuída entre *vmin* e *vmax*. O nó percorre o seu caminho até o destino com esta velocidade constante *V*. Uma vez alcançado o destino, fica estático durante o tempo de pausa e após o término deste período o processo é reiniciado. Um exemplo de um rastro de movimentação quando uma estação se desloca de acordo com o modelo RWP pode ser encontrado na Figura 2.

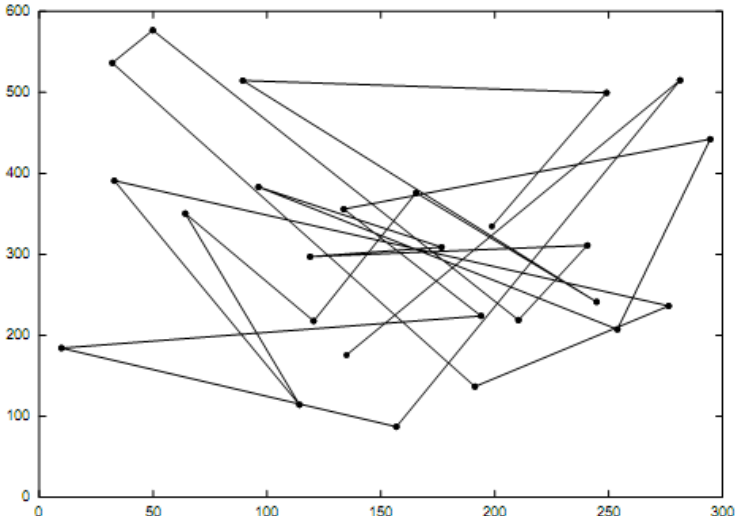

Figura 2. Exemplo de movimentação gerada pelo padrão *Random Waypoint* [19].

2.2.3 *Random Direction*

Esse modelo de mobilidade força os nós a deslocar-se até a borda da área de simulação e depois trocar de direção e velocidade. Nesse modelo os nós selecionam aleatoriamente uma direção e deslocam-se na direção escolhida até alcançar o limite da área de simulação. Quando esse limite é alcançado, o nó pára por um determinado tempo, escolhe aleatoriamente outra direção angular (entre 0º e 180º) e continua o processo [19]. Na Figura 3, é demonstrado um exemplo dos caminhos percorridos por uma estação no padrão *Random Direction*, considerando que os movimentos são executados até o limite da área de movimentação (o limitador do movimento é a borda).

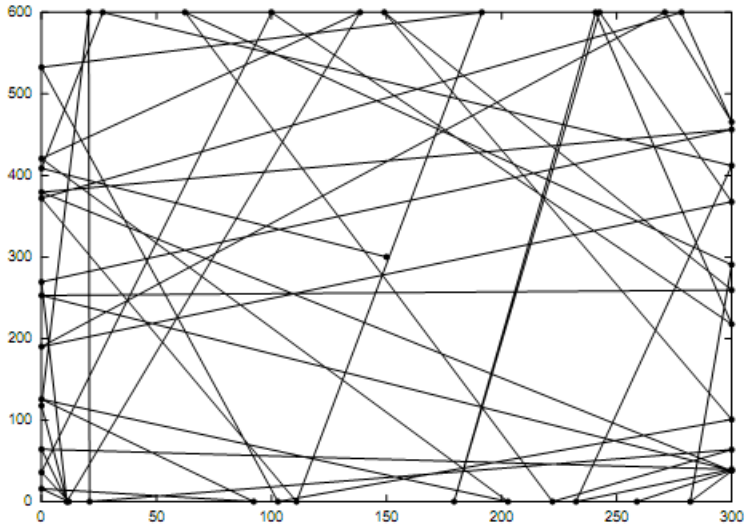

Figura 3. Exemplo de movimentação gerada pelo padrão *Random Direction* [5].

2.2.4 *Boundless Simulation Area*

Os padrões demonstrados até agora apresentam características de não ter memória, ou seja, um novo movimento do nó não depende dos movimentos anteriores. O padrão de mobilidade *Boundless Simulation Area,* proposto em [22], mantém uma relação entre um novo movimento e os anteriores. Cada nova combinação de velocidade e direção é gerada a partir da combinação anterior, gerando um padrão de movimentação como demonstrado na Figura 4.

O movimento de cada nodo móvel na área de movimentação é determinado por dois elementos em V = (v,θ), onde v é a velocidade e θ é a direção, os quais são atualizados a cada período de tempo Δt. A definição de novas velocidade e direção é obtida através do conjunto de equações abaixo [19]:

;

é a velocidade máxima definida na simulação, é a variação da velocidade uniformemente distribuída entre , é a aceleração máxima de um dado nó móvel, é a variação na velocidade que é uniformemente distribuída entre , e é a máxima variação angular na direção de um nó móvel.

Outra característica deste padrão é quanto à área de movimentação. Nos outros padrões, se o nó móvel atinge uma borda de limite da área de movimentação, este inicia um novo movimento, com nova direção, de modo a manter-se dentro desta área, como um efeito de reflexão. Neste padrão, o nodo ao atingir uma borda é transferido para o outro lado da área. Desta forma, seria como se a borda esquerda fosse ligada à borda direita, assim como a superior a inferior, formando um toro, como pode ser visto na Figura 5.

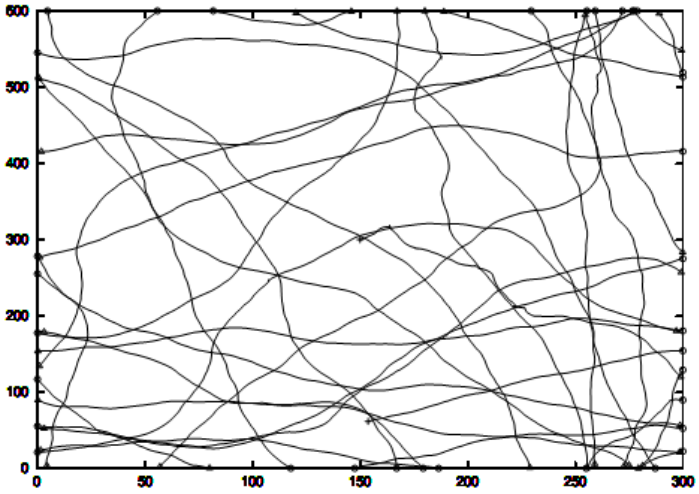

Figura 4. Exemplo de movimentação gerada pelo Padrão de Mobilidade *Boundless Simulation Area* [19].

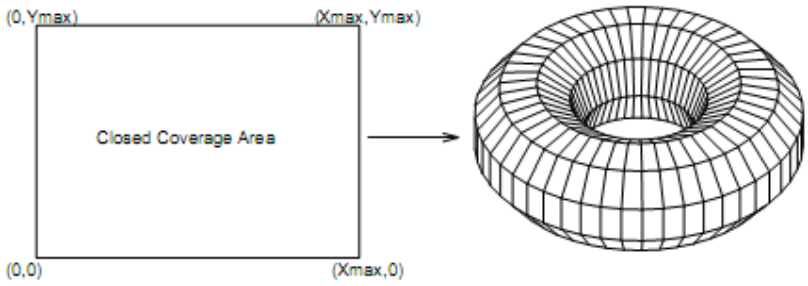

Figura 5. Área de movimentação redesenhada para uma área sem fronteiras [19].

2.2.5 *STreetRAndom Waypoint*

O padrão de mobilidade *Street Random Waypoint* [14] se baseia no modelo *Random Waypoint* apontando resultados mais precisos. Esse modelo de mobilidade veicular, baseado nas operações de tráfego real de veículos, emula os percursos dos carros pelas ruas de uma cidade, demonstrando ser um modelo mais realísta. Um exemplo padrão de mobilidade pode ser visto na Figura 6. Funcina baseado na escolha aleatória de rotas entre dois pontos na área de simulação que os nos tem que se desloca pelas ruas da cidade.

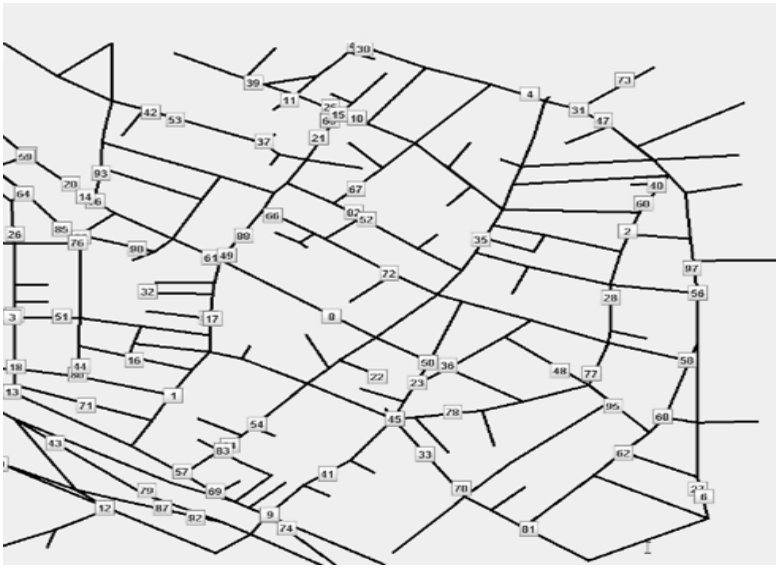

Figura 6. Exemplo de movimentação gerada pelo padrão [14].

2.3 Modelos de Mobilidades Propostos para JiST/SWANS

O entendimento das características que envolvem mobilidade se faz necessário para projetar e analisar o correto funcionamento das MANETs sob diferentes padrões de mobilidade. Como visto anteriormente modelos de mobilidade visam representar o comportamento de movimentação dos dispositivos móveis numa MANET. Esses modelos são usados na avaliação do desempenho de aplicações e sistemas de comunicação, permitindo analisar o impacto causado pela mobilidade no funcionamento dos mesmos. Além disso, outro aspecto é a pouca disponibilidade e imprecisão dos modelos de mobilidade existentes no simulador JiST/SWANS.

Dentro desse contexto, a implementação dos três modelos de mobilidade individual e em grupo para redes móveis e sua inclusão no simulador JiST/SWANS são o foco deste trabalho. Os três modelos acrescentados foram Uniforme, *Gauss-Markov e* Grupo Uniforme, que se baseiam em [18], [19] e, indicam maior liberdade de movimentação dos dispositivos móveis tornando viável uma avaliação das MANETs sob diferentes cenários de mobilidade. Os modelos propostos e implementados, a partir deste trabalho, possibilitou a avaliação com maior precisão do protocolo de roteamento AODV no simulador em estudo, os quais são descritos nas subseções a seguir.

2.3.1 Uniforme

O Modelo de Mobilidade Uniforme [18] resolve o problema da variação de vizinhos que ocorre no padrão de *Random Waypoint* mostrado na Figura 7, mantendo sempre a distribuição das estações uniforme na área de simulação. Nesse modelo, cada um dos nós locomove a uma velocidade *V* constante uniformemente entre *vmin* e *vmax* e numa direção aleatória *D* entre (0,2π) até uma distancia *d* escolhida aleatoriamente utilizando uma distribuição exponencial com média μ. O processo se repete após cada nó alcançar a distancia *d*. Se um nó bater na borda da área *A*, ele é refletido com o mesmo ângulo de incidência, como demonstrado na Figura 8. Outro problema que ocorre o modelo *Random Waypoint* que não

ocorre com modelo Uniforme é concentração dos nós ao redor do centro da rede como pode ser visualizado na Figura 17 na Seção 5.1.1

Figura 7. Variação do número de vizinhos no *Random Waypoint* [19].

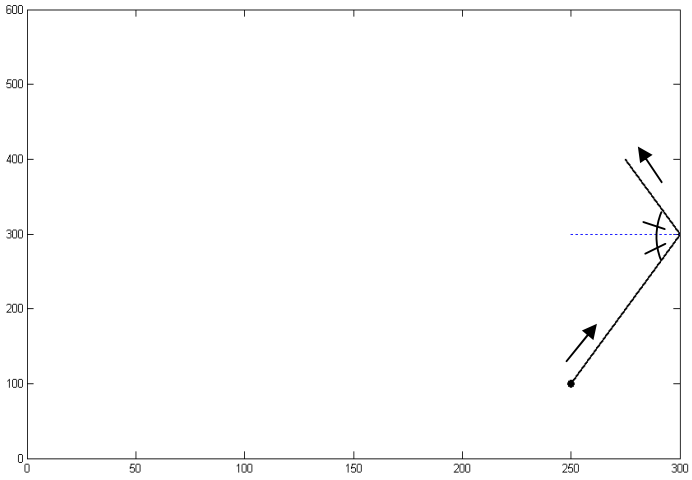

Figura 8. Exemplo de trajeto de um nó quando alcança a borda no modelo uniforme. Ângulo de reflexão é igual ao ângulo de incidência.

2.3.2 *Gauss-Markov*

O padrão de mobilidade *Gauss-Markov*[23] foi originalmente proposto para simulação de redes de celulares, porém pode ser aplicado também em simulação de redes *ad hoc* [20]. Este modelo usa intervalos de tempo discretos para dividir seu movimento. A próxima localização de um nó é gerada em função da localização e velocidade anteriores. Esse modelo foi projetado para se adaptar a diferentes níveis de aleatoriedade via um parâmetro de ajuste. A relação entre um novo movimento e um movimento anterior é definida por equações, conforme indicado por [23]:

$$s_n = \alpha s_{n-1} + (1-\alpha)\bar{s} + \sqrt{1-\alpha^2}s_{x_{n-1}};$$

$$d_n = \alpha d_{n-1} + (1-\alpha)\bar{d} + \sqrt{1-\alpha^2}d_{x_{n-1}};$$

$$x_n = x_{n-1} + s_{n-1}\cos d_{n-1};$$

$$y_n = y_{n-1} + s_{n-1}\sin d_{n-1};$$

onde s_n e d_n são as novas velocidade e direção de nó no intervalo de tempo n; α, onde $0 \leq \alpha \leq 1$, é o parâmetro de ajuste usado para variar a aleatoriedade; s e d são constantes representando o valor médio da velocidade e direção quando n $\rightarrow \infty$; e $s_{x_{n-1}}$ e $d_{x_{n-1}}$ são variáveis aleatórias que segue uma distribuição Guasiana.

Para evitar que o nó saia da área de movimentação, quando este atinge as regiões próximas às bordas (partes marcadas em cinza, delimitadas pelas linhas pontilhadas indicadas na Figura 10), a variável *d* (direção) do novo movimento do nó receberá os valores das constantes m_i (são as setas m_1=*0*, m_2=*45*, ..., m_8=*315*), conforme a região em que o nó estiver localizado. Desta forma, o nodo deverá retornar em direção ao centro da área de movimentação. Na Figura 9 é apresentado um exemplo dos caminhos gerados por um nó seguindo o padrão de movimentação *Gauss-Markov*.

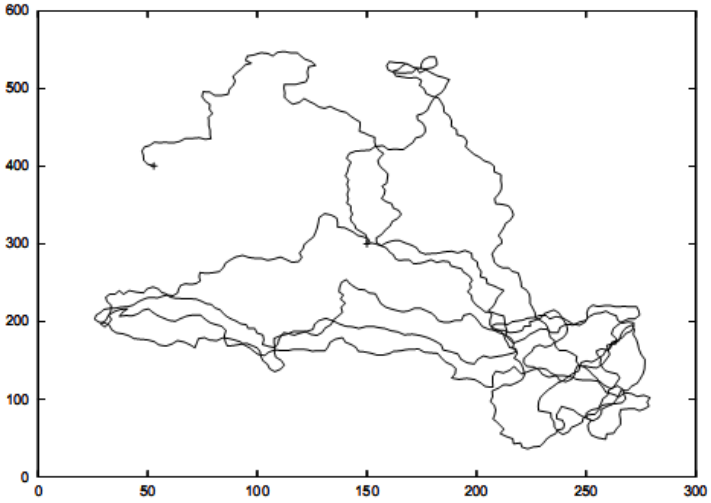

Figura 9. Exemplo de movimentação gerada pelo movimento Gauss-Markov [23].

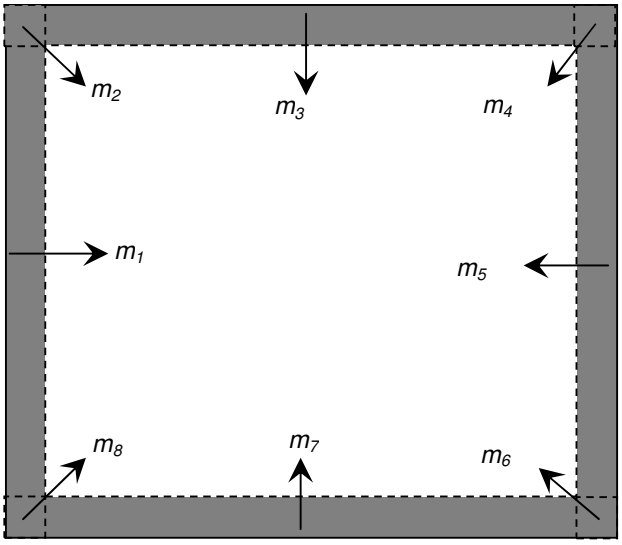

Figura 10. Movimentação no *Gauss-Markov*.

2.3.3 RGPM

A movimentação por Ponto de Referência (Reference Point Group Mobility - RPGM), proposto em [31] é um padrão muito flexível para modelagem de movimentação em grupo. Neste, os nós são agrupados de acordo com uma lógica ou padrão de comportamento de

movimentação entre eles. Este comportamento definem as variáveis velocidade, direção, localização e aceleração. O padrão também permite movimentação dos nós de forma aleatória e independente, além da movimentação do grupo. Com este padrão é possível modelar variados comportamentos de movimentação de grupo, definindo diferentes padrões de lógica de movimentação dos grupos. Em [31] são descritos vários cenários onde pode ser aplicado este padrão, como: movimentação de tropas em um campo de batalha, onde cada grupo possui um objetivo; equipes de salvamento (paramédicos, policiais, bombeiros, ...) em uma área de desastre, cada equipe possui certo comportamento e está espalhada por toda a área. O modelo RGPM não será implementado nesse trabalho, mas sim um modelo baseado nele e no modelo Uniforme apresentado na próxima seção.

2.3.4 Grupo Uniforme

Este é um novo modelo, proposto neste trabalho, que se baseia no modelo de mobilidade RPGM [31] e no modelo uniforme [18]. Para esse modelo existem 3 (três) tipos de nós e, conseqüentemente, 3 (três) tipos de movimentos, descritos a seguir:

- **nó-referência**: são os nós que são escolhidos para servir como referência para os nós internos. É parâmetro do modelo a quantidade de nós-referências que devem existir na simulação, sendo importante salientar que as posições desses nós, inicialmente, são geradas aleatoriamente na área de simulação. Também é um parâmetro do modelo a distância *D* que é a diagonal de um quadrado, com origem partindo da posição de um nó-referência. Esta área determinada por esse quadrado é onde possíveis nós internos podem se mover em relação ao nó-referência. O nó-referência é livre para se mover dentro da área de simulação, mas quando ele se move todos os nós internos à área limite desse nó-referência se movem junto com ele.
- **nó-interno**: é um nó que está dentro da área limite de um nó-referência e pode se mover dentro dessa área livremente. Quando o seu nó-referência move-se, ele acompanha o movimento seguindo o mesmo trajeto do seu nó-referência.
- **nó-externo**: o nó externo não está relacionado a nenhum outro nó. Esse nó é livre para se movimentar pela área de simulação.

Todos os movimentos que ocorrem em cada tipo de nó nesse modelo seguem o padrão do modelo de mobilidade uniforme com área de movimentação diferente de acordo com o tipo do nó. A Figura 11 exemplifica o esquema de movimentação de cada tipo de nó nesse padrão. O tipo de cada nó é determinado anteriormente ao início do movimento do nó, sendo proibido sair ou entrar nos grupos após o início do movimento.

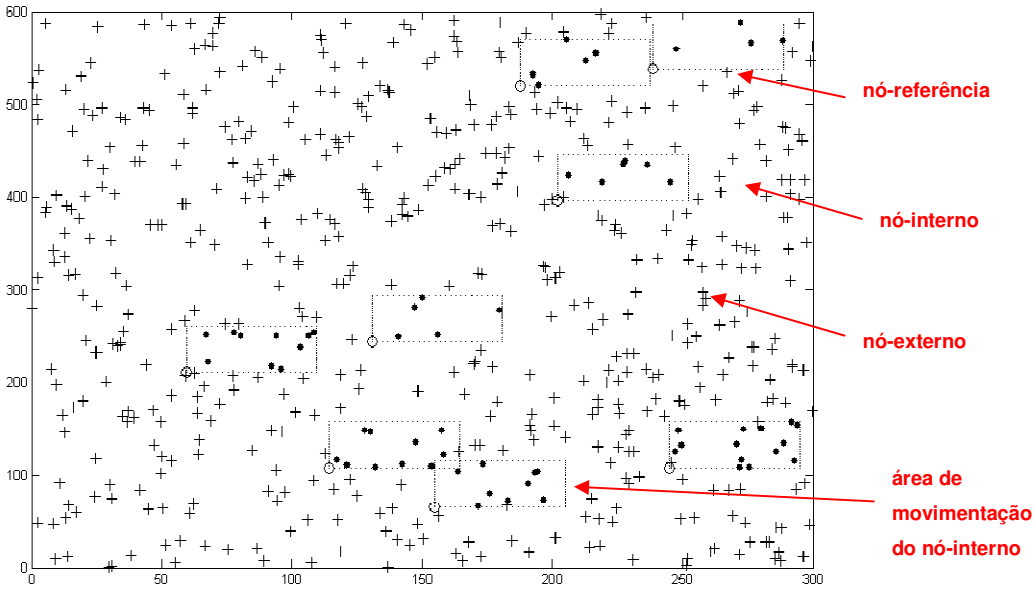

Figura 11. Exemplo de movimentação gerada pelos três tipos movimentos do modelo Grupo Uniforme.

Capítulo 3

Visão Geral do Simulador JiST/SWANS

Neste capítulo é descrito o simulador JiST/SWANS. Aqui são analisadas a arquitetura geral do simulador, instalação, testes iniciais realizados e suas configurações. Finalmente, são mostrados os arquivos de *trace* gerados pelos testes de análise das simulações apresentadas no próximo capítulo.

3.1 JiST

JiST [24], é uma abreviação de *Java in Simulation Time*, trata-se de uma plataforma e um simulador de eventos discretos de alto desempenho [12]. Esse simulador de código aberto é um protótipo desenvolvido por Rimon Barr na Universidade de Cornell. A idéia principal dele é transformar a máquina virtual Java em um escalonador de eventos modificando a maneira como as chamadas de métodos entre as entidades de simulação são conduzidos.

Os principais critérios levados em conta para a escolha desse simulador para nosso estudo foram sua escalabilidade e seu desempenho. São apresentados em [9] e [10] várias comparações desse simulador com outros populares simuladores utilizados em redes. O desempenho do JiST/SWANS tem se mostrado surpreendente para redes densas, inclusive mais adequado até que o NS-2[7] e o GloMoSim[8]. Como um dos objetivos do presente trabalho é avaliar os modelos de mobilidade implementados sob densidade crescente de nós na rede, esse simulador foi escolhido para estudo, tendo em vista que ele é eficiente tanto em termos de utilização de memória quanto em tempo de processamento.

3.1.1 Estrutura Geral

As entidades de simulação do JiST são executadas e compiladas como classes regulares de Java. Portanto, todas as vantagens da plataforma Java como a independência, o grande apoio de bibliotecas, familiaridade com a linguagem, se aplica também para o JiST. Para introduzir a semântica de "tempo simulação", o JiST divide as simulações em "Entidades", que são na realidade classes Java normais marcadas com a interface *JistAPI.Entity*, cujos métodos públicos são sempre chamados através do escalonador. Antes da execução da simulação, as classes de uma simulação JiST são transformadas ou reescritas usando o editor de *bytecodes*. Em seguida, a simulação é executada pelo núcleo da simulação que é composto por uma máquina virtual Java padrão.

3.1.2 Arquitetura

A arquitetura do sistema é composta por quatro componentes como mostra a Figura 12:

- um compilador;
- (re)editor de *bytecode*;
- um núcleo de simulação;
- uma máquina virtual Java(JVM).

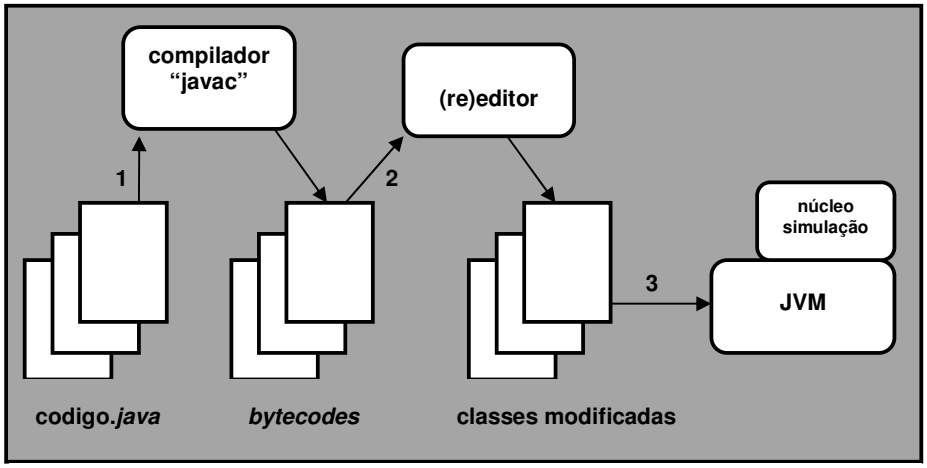

Figura 12. Arquitetura do simulador JiST/SWANS [21].

Primeiro os programas de simulação são escritos em linguagem Java normal e compilado em *bytecodes* usando um compilador Java padrão. Em seguida, essas classes compiladas são modificadas dinamicamente através do re-editor de *bytecode* para acrescentar a semântica de "tempo de simulação", pois cada entidade no JiST possui seu próprio tempo de simulação que determina qual o tempo de chamadas de outras entidades que estão escalonadas.

Nessa etapa, as chamadas para métodos públicos de entidades são desaclopados, isto é, as chamadas do método irão retornar imediatamente e serão colocadas numa pilha de eventos. O código correspondente ao método do objeto invocado é executado somente quando o tempo de entidade alcança o mesmo tempo de simulação. Para uma entidade avançar o seu tempo é utilizado o método *JistAPI.Sleep()*. Após os *bytecodes* serem reescritos, a simulação segue para o núcleo da simulação que juntamente com a máquina virtual Java executa a simulação.

3.2 SWANS

Na base do JiST, encontra-se o SWANS ou JiST/SWANS [11], acrônimo em língua inglesa de *Java in Simulation Time/Scalable Wireless Ad hoc Network Simulator*, um simulador resultante das necessidades de investigações atuais nas grandes redes que demandou a criação dessa ferramenta compatível que provê todos os mecanismos necessários para simular

MANETs escaláveis. O JiST/SWANS é um software de simulação desenvolvido em Java, o que torna possível instalá-lo e executá-lo em diversas plataformas como Linux, Windows e Solaris.

O SWANS é organizado em componentes independentes de software que podem ser compostos de forma a completar as configurações de uma rede sem fio ou rede de sensores mostrando-se uma arquitetura bastante modular. Este simulador apresenta capacidades similares às do NS-2 [7] e do GloMoSim [8], além de dar suporte à simulação de redes maiores, ou seja, com uma maior quantidade estações. A comunidade acadêmica vem investindo muito nesse simulador, desenvolvendo novas funcionalidades como protocolos de camada de acesso ao meio (MAC) e de roteamento, novos tipos de modelos de mobilidade e modelos de energia [13], [14], [16] e [17].

3.2.1 Estrutura Geral

Como JiST/SWANS é dividido em componentes, existem instâncias de entidades de cada tipo de componente disponível que estão esquematizada na Figura 13. Há componentes para protocolos da camada de rede, roteamento, acesso ao meio (MAC), rádios de transmissão, recepção, ruído, assim como modelos de propagação do sinal e de desvanecimento (*fading*), além de modelos de mobilidades.

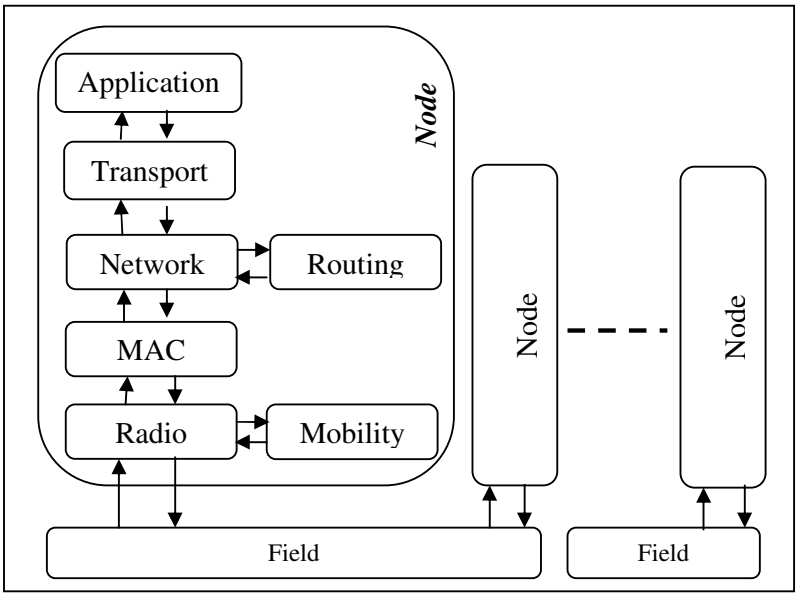

Figura 13. Arquitetura de uma simulação no JiST/SWANS.

Como mencionado anteriormente, o software SWANS é projetado com diferentes e independentes módulos que podem ser combinados para formar uma rede. Cada entidade JiST é responsável por armazenar seu estado interno e interagir com outros componentes via interfaces. Cada nó SWANS é uma entidade. Os nós também são formados por outras entidades da pilha da camada do modelo OSI, por exemplo, aplicação, transporte, rede, MAC e física. Há ainda entidades para mobilidade, rádio e roteamento. A arquitetura básica que um nó JiST/SWANS pode ter é ilustrada na Figura 14. Nesta figura encontra-se as diferentes camadas do modelo OSI implementada.

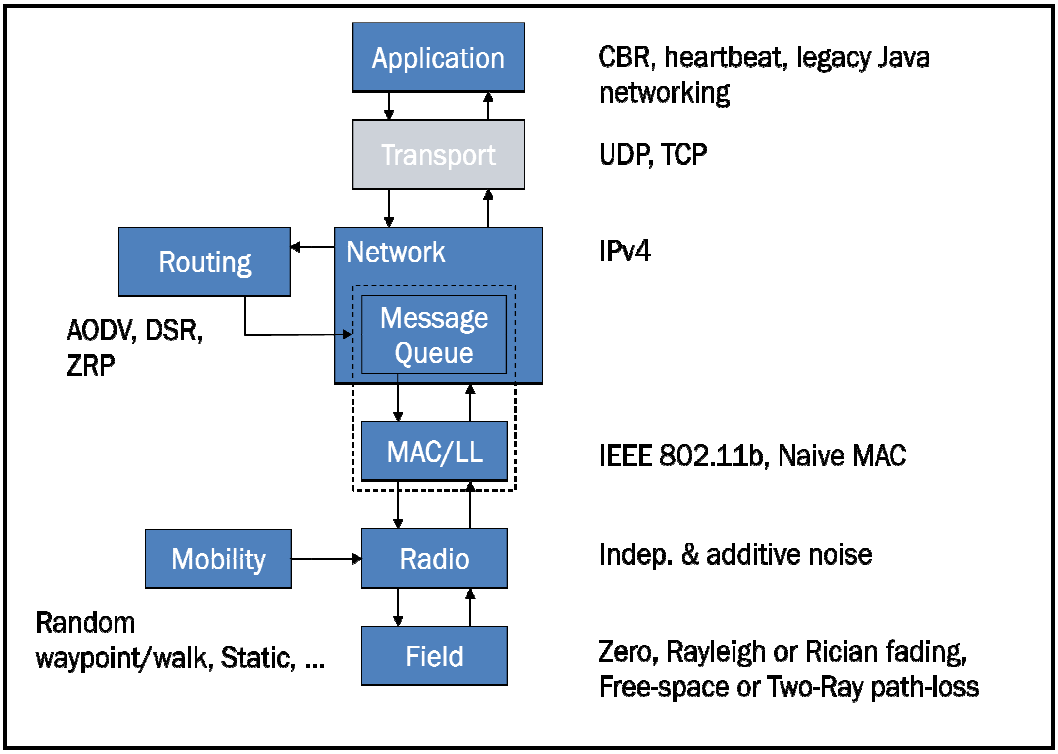

Figura 14. Instâncias de componentes disponíveis no JiST/SWANS [11].

O interesse desse trabalho está especificamente na interface de mobilidade chamada Mobility que será descrita no próximo capítulo. É através da implementação dessa interface que são adicionados novos modelos de mobilidade ao simulador JiST/SWANS.

3.3 Instalação e Configuração

O JiST/SWANS é compatível com diversas plataformas como Linux, Windows e Solaris. O presente trabalho utilizou a versão do JiST/SWANS 1.0.6 com o auxilio do ambiente de desenvolvimento Java Eclipse para facilitar a codificação e depuração na plataforma Windows.

A instalação do JiST/SWANS pode ser feita de maneira bem simples. Primeiramente fazendo descarga do JiST/SWANS em formato .zip que pode obtido em [24]. Para descompactar o arquivo, é preciso ter algum programa descompactador *zip* instalado no

sistema. Um programa desse tipo pode ser encontrado em [25]. Qualquer ambiente de programação de desenvolvimento Java pode ser usado. Nesse trabalho, foi utilizada a versão do Ecplise 3.2 que pode ser encontrado em [26]. Depois de instalados o descompactador zip e o Ecplise, bastar seguir os passos descritos abaixo para assegurar a correta instalação do JiST/SWANS:

1. Descompacte o código fonte do JiST/SWANS em algum diretório;
2. Inicie o Eclipse;
3. Do menu "*File*" selecione "*New => Project*";
4. Selecione *"Java Project*", e clique em *Next*;
5. Defina um nome para o seu projeto JiST/SWANS. Selecione "*Create project from existing source*" da opção "*Contents"* na mesma janela. Navegue até o diretório do código fonte do JiST/SWANS que você criou no passo 4;
6. Clique *Finish* na última janela. Seu projeto deve aparecer na guia *Package Explorer*.

Após esses passos é possível desenvolver e executar programas de simulação assim como alterar o código do simulador pelo Eclipse.

3.3.1 Exemplo de Simulação

Nesta seção é apresentado um exemplo simples de simulação e execução no JiST/SWANS contendo apenas um entidade que imprime mensagem na saída. Existe duas formas de executar a classe mostrada na Listagem 1:

Listagem 1. Exemplo "aloMundo.java" de uma simulação JiST/SWANS.

```
import jist.runtime.JistAPI;

public class aloMundo implements JistAPI.Entity
{
   public void myEvent()
   {
      JistAPI.sleep(1);
      myEvent();
      System.out.println("hello world, t="+JistAPI.getTime());
      try
      {
         Thread.sleep(500);
      }
      catch(InterruptedException e)
      {}
   }
   public static void main(String[] args)
   {
      System.out.println("starting simulation.");
      hello h = new hello();
      h.myEvent();
   }
}
```

A primeira forma de executar a simulação mostrada acima é executar como uma classe normal de Java, conforme demonstrado na Listagem 2. Nesse caso, uma exceção irá ser lançada devido ao estouro da pilha, provocada pela recursividade.

Listagem 2. Executação da o "aloMundo.java" como uma classe normal Java

```
java aloMundo
```

A segunda forma de executar é carregando o núcleo do simulador JiST/SWANS demonstrado na Listagem 3. Nesse caso, entre outras coisas, o núcleo carrega uma classe na JVM chamada (re)editor de *bytecode*, que dinamicamente reescreve os *bytecodes* da classe 'aloMundo' à medida que ela é carregada. As entidades marcadas com a interface *JistAPI.Entitity* servem para direcionar o (re)editor de bytecodes, conforme mencionado anteriormente, para introduzir a semântica de tempo de simulação.

Listagem 3. Executando o primeiro exemplo carregando através do simulador *JiST/SWANS*.

```
java jist.runtime.Main aloMundo
```

Essencialmente, nessa execução a chamada de método é transformada em um evento de simulação na entidade "aloMundo". Ele é escalonado e invocado pelo núcleo da simulação no tempo da simulação. Note que a classe 'aloMundo' é uma entidade e não um objeto comum, pois implementa a interface *JistAPI.Entity*. É possível observar também da saída, que o método *JistAPI.sleep()* serve para avançar o tempo de simulação. É por isso que não há nenhum estouro de memória.

3.3.2 Arquivo de *Trace*

O JiST/SWANS disponibiliza um formato básico de *trace* para simulações, como pode ser visto na Listagem 4. Existem 6 eventos que são registrados nesse arquivo de *trace* quando uma simulação é executada. O formato do arquivo de trace do JiST/SWNAS é suportado pelo Network Animator [27] é composto de eventos que registram tudo que ocorre na simulação.

Listagem 4. Exemplo de saída do arquivo de trace do JiST/SWANS

```
d -t 10.875866 -s 5 -d 3 -p aodv -e 44 -i 7698 -a 0
r -t 10.938671 -s 1 -d 14 -p aodv -e 44 -i 7700 -a 0
n -t 11.000001 -s 6 -S UP -v circle -c red -x 61.32594 -y 92.68963 -z 4
n -t 11.000001 -s 5 -S UP -v circle -c red -x 49.533245 -y 385.06702 -z 4
+ -t 11.000543 -s 16 -d -1 -p aodv -e 44 -i 7701 -a 0
- -t 11.000543 -s 16 -d -1 -p aodv -e 44 -i 7701 -a 0
r -t 11.001009 -s 16 -d 10 -p aodv -e 44 -i 7701 -a 0
h -t 11.001009 -s 16 -d 10 -p aodv -e 44 -i 7701 -a 0
```

Esse arquivo de *trace* é composto pelos eventos detalhados abaixo, como por exemplo, um evento associado ao estado do nó que é:

'n'

'n' denota o evento do estado do nó. O atributo '-t' indica tempo e '-a' e '-s' denota endereço e a identificação do nó. O '-S' indica o status do nó que pode ser UP ou DOWN, isto é, representando o estado ativo ou inativo do nó. O '-v' representa a forma do nó podendo ser quadrado (*square*) ou círculo (circle) e '-c' que representa a cor do nó.

Por outro lado, existem cinco eventos associados a pacotes no arquivo de trace do JiST/SWANS que são:

'h'

Hop (salto) – pacote transmitido de um nó-fonte a um nó-destino e redirecionado para o próximo salto em direção ao seu destino;

'r'

Receive (Recepção) – pacote que foi recebido no destino com sucesso;

'd'

Drop (Perda) – pacote que foi perdido da pilha ou link entre a fonte e destino. Perda aqui não distingue entre perda de pilha ou de link;

'+'

Entrada na pilha – pacote que entrou na pilha do nó-fonte para ser transmitido para nó-destino;

'-'

Saída da pilha – pacote que deixou a pilha do nó-fonte para ser transmitido para o nó-destino.

Os outros atributos têm os seguintes significados:

-t <time> – tempo em que o evento ocorreu.

-s <src> – nó fonte.

-d <dst> – nó destino.

-p <pkt-type> – nome descritivo do tipo de pacote.

-e <extent> – tamanho em bytes do pacote.

-i <id> – identificação do pacote na transmissão.

-a <attr> – uma opção ou atributo para o pacote, que é atualmente usado como identificação da cor do pacote.

Capítulo 4

Implementação da Mobilidade e da Simulação com o Protocolo AODV

Este capítulo tem como foco principal a descrição geral da implementação dos modelos de mobilidade juntamente com a simulação das redes testadas com protocolo AODV. Este capítulo demonstra o desenvolvimento dos modelos de mobilidade no JiST/SWANS utilizando a interface Mobility. Em seguida, são exibidas algumas características básicas do protocolo AODV. Em seguida, é mostrado a estruturação e configuração de uma simulação da rede testada com protocolo AODV no JiST/SWANS para o ambiente proposto.

4.1 Interface Mobility do JiST/SWNAS

Como mostrado na Figura 13 na Seção 2.2.1 a mobilidade no simulador JiST/SWANS é implementado através de um módulo representado pela interface Mobility, conforme demonstrada na Listagem 5. Para acrescentar um modelo de mobilidade no JiST/SWANS é preciso seguir o padrão da interface Mobility.

Listagem 5: Interface Mobility do JiST/SWANS.

```
public interface Mobility
{
MobilityInfo init(FieldInterface f, Integer id, Location loc);

void next(FieldInterface f, Integer id, Location loc, MobilityInfo info);
}
```

Existe uma estrutura auxiliar que pode ser usada para ajudar na implementação do modelo de mobilidade, conforme demonstrada na Listagem 6.

Listagem 6: Interface que auxilia na implementação da mobilidade no JiST/SWANS.

```
public interface MobilityInfo{
}
```

Nessa interface existem dois métodos que precisam ser implementados. O primeiro método, *init()*, é chamado quando o nó é criado pela classe *Field* do simulador JiST/SWANS. Sua função é inicializar um objeto de alguma classe que implemente a interface *MobilityInfo*. Essa classe pode ser usada como estrutura auxiliar para desenvolver o padrão de mobilidade do nó. O segundo método, *next(),* é o método principal da interface *Mobility* usado para gerar as sucessivas posições de cada nó ao longo de uma simulação.

4.1.2 Implementação do Modelo Uniforme (ver também Seção 2.3.1)

Conforme o apêndice A, primeiro foi criada a estrutura auxiliar de mobilidade denominada *UniformeInfo*. Esta classe é responsável por guardar algumas informações referentes ao estado do nó durante a simulação, informações quanto à velocidade, à direção e à distância corrente do movimento para cada nó. Além disso, a classe *UniformeInfo* possui os seguintes atributos: precisão, ou seja, a quantidade de passos que serão realizados pelo trajeto do movimento corrente (*passos*), e tempo entre os passos do movimento (*tempoPasso*).

A velocidade, a distância e a direção são calculados todas as vezes que é chamado o construtor da classe *UniformeInfo.* No construtor da classe *UniformeInfo* são passados como parâmetro as velocidades mínima(*vMax*) e máxima(*vMin*) e a média μ(*mu*) utilizada na função *exprnd(),* gerando as distâncias a ser percorridas pelos nós. Para ter acesso à função geradora de números aleatórios distribuídos exponencialmente foi preciso usar a biblioteca Java *Simulation Stochastique en Java* (SSJ) [28].

O modelo uniforme éimplementado pela classe *Uniforme do* Apêndice A. No código primeiramente é definido o construtor da classe Java com a função de inicializar os parâmetros do modelo uniforme: limites da área de simulação (*limites*), a média μ*(mu)* usada para gerar as distâncias aleatórias usadas na distribuição exponencial, velocidade mínima (*vMin*) e máxima (*vMax*).

Em seguida, o método *init(),* que tem a função de atribuir para cada nó um objeto da classe *UniformeInfo*. No método *next(),* no qual a lógica do modelo realmente é implementada. Neste método, primeiro é calculada a distância a ser percorrida pelo nó no intervalo de tempo determinado pela variável (*tempoPasso*), então, são calculadas as novas coordenadas do nó. O trecho de código da Listagem 5 foi extraído do Apêndice A, e é responsável pelo tratamento das bordas quando um nó ultrapassa os limites da área de simulação.

Listagem 7. Trecho de código responsável pelo tratamento de bordas no modelo Uniforme.

```
......
while(novoX<0 || novoX>limites.getX() || novoY<0 || novoY>limites.getY()){
          double deltaXExt = novoX-loc.getX();
          double deltaYExt = novoY-loc.getY();
          double a = deltaYExt/deltaXExt;
          double b =  deltaYExt - (a*deltaXExt);
          Location2D lastPoint = null,reflexPoint=null;
          double deltaXInt, deltaYInt;

          if(novoY<0){
            lastPoint = new Location2D((float)((0 - b)/a),0);
            reflexPoint = new Location2D((float)novoX,(float)(-1*novoY));
            novoY = -1*novoY;
            deltaXInt = reflexPoint.getX() - lastPoint.getX();
            deltaYInt = reflexPoint.getY() - lastPoint.getY();

            if(uinfo.direcao>3*Math.PI/2 && uinfo.direcao<2*Math.PI)

              uinfo.direcao =  Math.atan(deltaYInt/deltaXInt);
            if(uinfo.direcao>Math.PI && uinfo.direcao<3*Math.PI/2)
              uinfo.direcao = Math.PI + Math.atan(deltaYInt/deltaXInt);

          }else if(novoY>limites.getY()){
            lastPoint = new Location2D((float)((limites.getY() -
b)/a),limites.getY());
            reflexPoint = new
Location2D((float)novoX,(float)(2*limites.getY() - novoY));
            novoY = 2*limites.getY() - novoY;
            deltaXInt = reflexPoint.getX() - lastPoint.getX();
            deltaYInt = reflexPoint.getY() - lastPoint.getY();

            if(uinfo.direcao>0 && uinfo.direcao<Math.PI/2)
              uinfo.direcao =  2*Math.PI + Math.atan(deltaYInt/deltaXInt);
            if(uinfo.direcao>Math.PI/2 && uinfo.direcao<3*Math.PI)
              uinfo.direcao = Math.PI + Math.atan(deltaYInt/deltaXInt);

          }
          else if(novoX<0){
            lastPoint = new Location2D(0,(float)(a*0 + b));
            reflexPoint = new Location2D((float)(-1*novoX),(float)novoY);
            novoX = -1*novoX;
            deltaXInt = reflexPoint.getX() - lastPoint.getX();
```

```
            deltaYInt = reflexPoint.getY() - lastPoint.getY();

            if(uinfo.direcao>Math.PI && uinfo.direcao<3*Math.PI/2)
              uinfo.direcao = 2*Math.PI + Math.atan(deltaYInt/deltaXInt);
            if(uinfo.direcao>Math.PI/2 && uinfo.direcao<Math.PI)
              uinfo.direcao = Math.atan(deltaYInt/deltaXInt);
          }else{
            lastPoint = new
Location2D(limites.getX(),(float)(a*limites.getX() + b));
            reflexPoint = new Location2D((float)(2*limites.getX() -
novoX),(float)novoY);
            novoX = 2*limites.getX() - novoX;
            deltaXInt = reflexPoint.getX() - lastPoint.getX();
            deltaYInt = reflexPoint.getY() - lastPoint.getY();

            if(uinfo.direcao>0 && uinfo.direcao<Math.PI/2)
              uinfo.direcao = Math.PI + Math.atan(deltaYInt/deltaXInt);
            if(uinfo.direcao>3*Math.PI/2 && uinfo.direcao<2*Math.PI)
              uinfo.direcao = Math.PI  + Math.atan(deltaYInt/deltaXInt);
          }
        }

.........
```

No trecho acima, primeiro é verificado se as novas coordenadas do nó ultrapassaram os limites da área de simulação. Se sim, o percurso do nó é rebatido com um ângulo de reflexão igual ao ângulo de incidência em relação à reta normal ao último ponto que toca a área de simulação como mostra a Figura 8 (Seçao 2.3.1). Após isso, as coordenadas e direção do movimento do nó são atualizadas. Em seguida, são decrementados os passos do movimento, então o nó é movido para as novas coordenadas e o tempo de entidade é avançado através do método *JistAPI.sleep()* pela quantidade de tempo do intervalo do passo. Finalmente, é verificado se o trajeto corrente terminou, ou seja, se o atributo *passos* do objeto *UniformeInfo* é igual a zero. Se for o caso é criado um novo objeto *UniformeInfo* para esse nó, representando uma nova distância, uma nova direção e velocidade a ser percorrida por esse nó.

4.1.2 Implementação do Modelo Gauss-Markov (ver Seção 2.3.4)

Para a implementação desse modelo, cujo código é apresentado no Apêndice B, primeiramente foi criada a classe *GaussMarkovInfo* que implementa a interface *MobilityInfo.*

Esta classe possui os seguintes atributos: velocidade (*velocidade*), direção (*direcao*), direção média dos nós (*direcao_media*) e velocidade média do nós (*velocidade _media*). Em seguida, a classe *GaussMarkov* que implementa a interface *Mobility* foi definida. Esta classe possui os atributos do modelo *Gauss-Markov* que são a velocidade média inicial (*velocidade_media_inicial*), a direção media inicial (*direcao_media_inicial*), o limite de borda (*borda*), a velocidade mínima inicial (*velocidade_min_inicial*), a velocidade máxima inicial (*velocidade_min_inicial*), o parâmetro de ajuste α (*alpha*) e os limites da área de simulação (*limites*).

O método *init*() é responsável pela criação de um objeto da classe *GaussMarkovInfo* para cada nó da simulação. No trecho de código do método *next()*, mostrado na Listagem 8, é verificado se o nó chegou aos limites das bordas da área de simulação. Se sim, uma nova direção média dos nós é calculada baseada na posição em que o nó se encontra de acordo com o esquema apresentado na Figura 10 da Seção 2.3.2.

Listagem 8. Tratamento das bordas para o modelo Guass-Markov.

```
.............

  if (x > 0 && x < limiteXinf && y > 0 && y < limiteYinf)
         direction_medio = Math.PI/4;
       else if (x > limiteXinf && x < limiteXsup && x > 0 && y <
limiteYinf)
         direction_medio = Math.PI/2;
       else if (x > limiteXsup && x < X && y > 0 && y < limiteYinf)
         direction_medio = 3*Math.PI/4;
       else if (x > limiteXsup && x < X && y > limiteYinf  && y <
limiteYsup)
         direction_medio = Math.PI;
       else if (x > limiteXsup && x < X && y > limiteYsup  && y < Y)
         direction_medio = 5*Math.PI/4;
       else if (x > limiteXinf && x < limiteXsup && y > limiteYsup  && y <
Y)
         direction_medio = 3*Math.PI/2;
       else if (x > 0 && x < limiteXinf && y > limiteYsup  && y < Y)
         direction_medio =7*Math.PI/4;
       else if (x > 0 && x < limiteXinf && y > limiteYinf  && y<
limiteYsup)
         direction_medio = 0;
       else
         direction_medio = gminfo.direcaoMedia;

.........
```

Em seguida, a velocidade e a direção do nó são calculadas de acordo com o conjunto de equações apresentadas na Seção 2.3.2. Feito isto, são calculadas as novas coordenadas do nó baseadas também no conjunto de equações apresentadas na Seção 2.3.2. Em seguida, o nó é movido para as novas coordenadas e o tempo de entidade é avançado utilizando o método *JiST.Sleep()*.

4.1.2 Implementação do Modelo Grupo Uniforme

Para implementação desse modelo, demonstrado no Apêndice C, inicialmente foi definida a classe *GrupoUniformeInfo* responsável por guardar o estado de cada nó durante a simulação. O modelo Grupo Uniforme é composto de 3 tipos de movimentos, todos uniformes e com área limite de movimentação diferente para os três tipos de nós, como pode ser visto na Figura 15 abaixo.

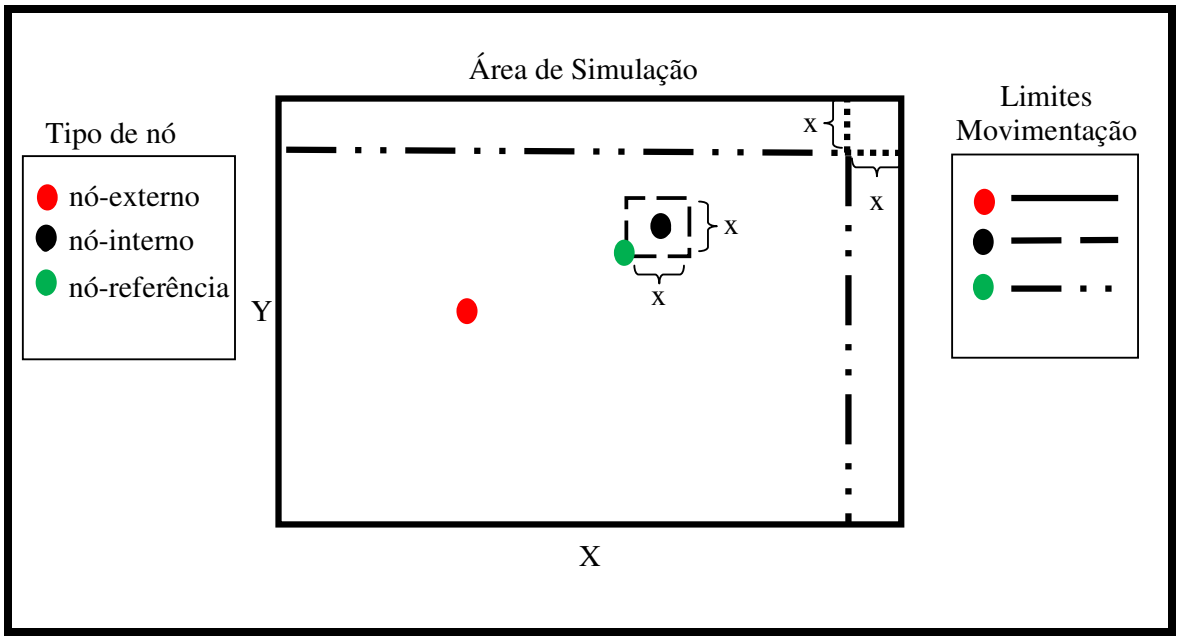

Figura 15. Área de movimentação dos 3 tipos de nós para modelos Grupo Uniforme.

A classe *GrupoUniformeInfo* possui os mesmos atributos da classe *UniformeInfo* e três atributos a mais que são a localização do nó referência, o tipo do nó, um lista dos nós internos. Esta lista guarda a identificação dos nós internos ao nó referência correspondente. Esta classe tem a função de ajudar a controlar o movimento dos 3 tipos de nós que esse modelo

implementa. Existe ainda nessa classe um método chamado *renew()* que é invocado quando o nó chega ao destino e tem a função de escolher rota para o nó mover-se.

A classe *GrupoUniforme* implementa, de fato, o modelo de mobilidade. Nessa classe destacam-se, em relação à implementação do modelo Uniforme, os atributos: quantidade de pontos de referência (*qtdPointReference*), vetor/lista dos pontos de referências (*nosReferencias*), diagonal da área do quadrado dos pontos de referências (*diagonalGrupo*). O restante dos atributos é idêntico ao modelo Uniforme.

No construtor dessa classe são inicializados os atributos. O método *init()* é responsável por definir o tipo de cada nó e retornar um objeto da classe com essa informação. Por convenção, os nós com número de identificação 1 até *qtdpointReference* (o número da quantidade de nós referencias) são definidos como nós do tipo 1, ou seja, nó-referência. Os demais nós são classificados como nó tipo 2 se estiverem localizados dentro da área limite de algum nó tipo 1, nesse caso é inserida a identificação do na lista de nós internos do nó-referência correspondente. Aqueles que não estiverem em nenhuma área limite de algum nó-referência são classificados como nós tipo 3. O método *next()* realiza os movimentos uniformes para todos os tipos de nós com o auxilio do método *moveno()* mostrado simplificadamente nas Listagem 9, a qual foi extraída do Apendice C.

Listagem 9. Trecho do código do metodo *moveNo()* responsável por mover os diferentes tipos de nó do modelo Grupo Uniforme.

```
private Location2D moveNo(FieldInterface f, Integer id, Location loc,
GrupoUniformeInfo uinfo) {
        GrupoUniformeInfo headinfo = getHeadReference(id);
        double xnode = loc.getX();
        double ynode = loc.getY();
        double X = limites.getX();
        double Y = limites.getY();
        if (uinfo.tipo == 1){
          X = limites.getX()-diagonalGrupo;
          Y = limites.getY()-diagonalGrupo;
        }

.............
```

Para realizar o movimento dos nós, primeiramente verifica-se qual é o tipo de nó, se o nó corrente for um tipo 1, os nós tipo 2 que estiverem dentro da área limite desse nó movem-se

junto com o mesmo deslocamento do nó tipo 1. O método *moveno()* trabalha da mesma forma para os três tipos de nós. As únicas diferenças são área limite de movimentação e o ponto de referência que está relacionado, como ilustrado na Figura 15.

4.3 Visão Geral do Protocolo AODV

Como um dos objetivos desse trabalho é investigar o desempenho das redes testadas sob os três modelos de mobilidade implementados. Para testar esse desempenho adotou-se como principal meio de análise o protocolo de roteamento *Ad hoc On-Demand Distance Vector* (AODV) [29] que foi desenvolvido para ser utilizado por nós móveis especialmente para MANETs. Este protocolo oferece uma rápida adaptação às condições dinâmicas dos enlaces, baixo processamento e reduzida taxa de utilização da rede, e determina rotas para destinos dentro de uma rede *ad hoc*. Também utiliza números de seqüências, associados a cada destino para garantir a ausência de *loops* de roteamento, evitando problemas característicos dos protocolos de vetor de distância, como a contagem para infinito.

Para permitir a formação de redes que tenham uma grande dinâmica de topologia, como é caso de MANETs, o protocolo AODV é que mais se utiliza [3], [19] e [20]. Este protocolo permitiu aos nós encontrar e manter uma rota para outros nós na rede não importando quando tais rotas serão necessárias. O protocolo AODV é reativo, ou seja, atua sob demanda fazendo com que uma rota para um determinado nó de destino só seja descoberta quando se deseja enviar um pacote a este nó.

4.3.1 Pacotes de Controle AODV

Também é interesse do presente trabalho avaliar a sobrecarga de pacotes de controle do protocolo AODV. Para isso é preciso entender os tipos de pacotes de controle que os nós utilizam para estabelecer os enlaces de comunicação. O AODV usa quatro tipos de pacotes de controle. Os pacotes de controle *Route Request* (RREQ) e *Route Reply* (RREP) são usados para descobrir rotas, enquanto os pacotes Route Error (RERR) e (HELLO) são usados para manter as rotas.

- RREQ – são pacotes reponsáveis pela inundação da rede por *broadcast*. Sempre que um nó fonte deseja se comunicar com outro nó para o qual ele ainda não possui uma entrada em sua tabela de roteamento, ele inunda a rede com um RREQ.

- RREP – são pacotes de respostas que são enviados via *unicast* quando um nó recebe um pacote RREQ e possui em sua tabela de roteamento uma rota requerida.

- REER – nó que detecta alguma falha na comunicação e tem a função de informar a todos os nós que dependiam da conexão sobre a quebra do *enlace*. Isso é possíıvel graças ao pacote de erro chamado Route Error (RERR). Uma vez verificados na tabela de roteamento os destinos que utilizam o *enlace*, o nó notifica a quebra do *enlace* enviando um pacote RERR aos nós afetados. O nó que recebe a notificação, por sua vez, encaminha o pacote RERR para todos os seus predecessores.

- HELLO – Para se detectar a queda de um *enlace*, podem ser usadas mensagens periódicas de *hello*. Os nós podem conhecer seus vizinhos de duas maneiras: quando um nó escuta um *broadcast* de outro nó, ele atualiza sua informação de conectividade local ataualizando este vizinho na tabela de roteamento; outra maneira é quando um nó não tenha mandado pacotes para todos seus vizinhos no período de um intervalo *hello-interval*, então ele deve explicitamente se mostrar vivo e acessível como vizinho, mandando através de *broadcast* aos vizinhos uma mensagem de *hello*.

4.4 Implementação da Simulação com o Protocolo AODV

A rede testada nesse trabalho é uma rede sem-fio utilizando o protocolo de roteamento AODV [29]. Para desenvolver uma simulação de rede no ambiente JiST/SWANS foi necessário unir os diferentes módulos que implementam as camadas do modelo OSI. O ambiente de simulação que foi utlizado no desenvolvimento da simulação é mostrado na Tabela 2.

Tabela 1. Ambiente de simulaçao das redes testadas.

Nós	16-32-64-128-256-512
Campo	500x500 m^2
Velocidade Máxima	20 m/s
Velocidade Mínima	0 m/s
Tempo de Pausa	0s
Tempo de Duração	500s
Potência de Transmissão	15 dBm
Sensitividade de Recpação	-91.0 dBm
Taxa de Transmissão	1 Mpbs
Frequência	2.4 GHz
Tráfego	30 mesagens por minuto
Protocolo de Roteamento	AODV
Tamanho de Mensagem	64 bytes

O JiST/SWANS disponibiliza implementações de alguns dos principais protocolos de roteamento para redes *ad hoc* propostos na literatura. Dentre os protocolos disponíveis, escolheu-se AODV por ser um dos protocolos mais estudados da área e por possuir uma boa documentação. Além disso, o AODV possui um implementação para o JiST/SWANS mais estável e confiável, por já ter sido testada em diversos trabalhos [9], [10] e [32].

Como pode ser visto no Apêndice D, a simulação foi estruturada numa classe chamada *AodvSim* contendo 4 partes principais que são:

- Os atributos do ambiente de simulação;
- O método *constroiRede()* – responsável pela construção do campo de simulação, criação dos nós, atribuição dos modelos de mobilidade aos nós, criação da camada física, além de atribuir modelo de desvanecimento, inicializa o arquivo de trace e gera a transmissão do tráfego UDP CBR aleatoriamente entre os nós presentes na simulação;
- O método *adicionaNo()* – responsável pela criação de cada nó da simulação, inicializando a mobilidade, criação da camada MAC, de rede e de transporte de cada nó;

- O método principal *main()*: responsável por atribuir o tempo de simulação e executar a simulação chamando o método *construirRede()*.

Os resultados dos testes realiazados com as implementações neste capítulo são mostrados no próximo capítulo.

Capítulo 5

Simulação e Análise dos Resultados

Neste capítulo são descritos os estudos e experimentos de interesse principal deste trabalho. Aqui estão descritas as validações dos modelos de mobilidade implementados, as configurações do ambiente de simulação e testes realizados com esses modelos. E finalmente, são mostrados os resultados das análises das simulações estudadas.

5.1 Validação dos Modelos de Mobilidade Implementados

Para validar os modelos implementados foi utilizado percentagem média de vizinhos e distribuição dos nós na área de simulação. Foram realizadas simulações durante um tempo de 1000 segundos, numa área de simulação igual 500x500m^2 com um total de 512 nós com velocidades escolhidas uniformemente entre 0 e 20m/s. Para computar esta percentagem de vizinhos utilizou-se uma circuferência de raio igual 70m, ou seja, o nó que estiver dentro da área da circuferência é considerado um vizinho. Para os modelos Uniforme e Grupo Uniforme foi utilizada um média μ=100, empregada na na distribuição exponencial para gerar as distâncias a ser percorridas pelos 'nós'. Foi escolhido esse valor por ser o mais próximo das distâncias percorridas pelo RWP.

5.1.1 Validação dos Modelos de Mobilidade

Para validar o modelo Gauss-Markov foi realizada uma comparação do cálculo de vizinhos utilizando o MobiSim[30], simulador desenvolvido para gerar *traces* de vários modelos de mobilidade, com o nosso modelo de Gauss-Markov implementado. A partir da ilustração da Figura 16, nota-se bastante semelhança na curva de vizinhos gerados a partir da implementação do JiST/SWANS e do MobSim. É evidente que existem pequenas diferenças

entre essas duas curvas devido às diferenças na aleatoridade em cada simulação. Entretanto, é importante observar a semelhança no comportamento das curvas da percentagem média de vizinhos, tanto para JiST/SWANS quanto para MobSim, validando o modelo Gauss-Makov implementado. Nas Figuras 17(A) e 17(B) são ilustradas as ditribuições dos nós para o modelo Guass-Markov implementado e o MobiSim para tempo *t=1000s*.

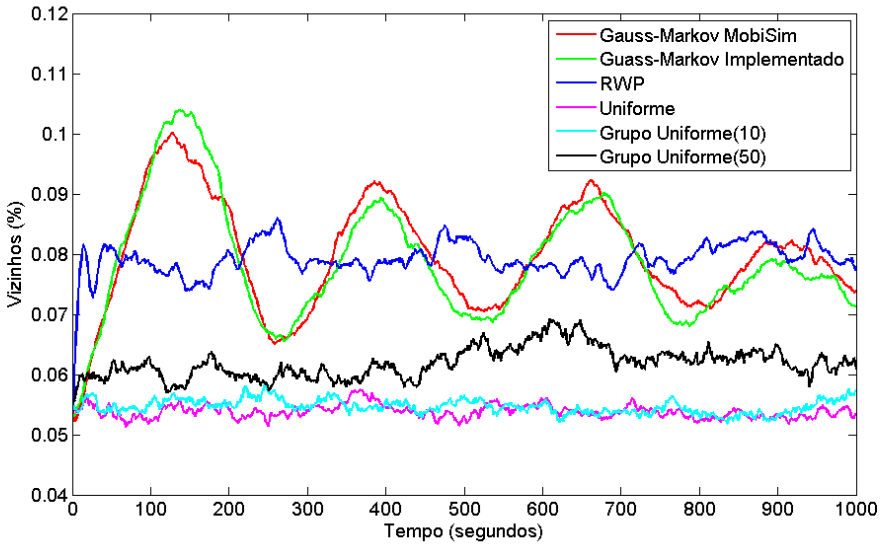

Figura 16. Percentagem de vizinhos para os modelos de mobilidades implementados.

A validação dos modelos de mobilidade Uniforme e Grupo Uniforme tornou-se inviável com outros simuladores disponíveis na *Internet*, devido à falta de software compatível com os modelos implementados. Contudo, foi realizada uma validação da precisão dos valores da percentagem média de vizinhos computados para esses algoritmos de mobilidade através de uma comparação em relação ao modelo RWP mostrado na Figuras 16.

A percentagem média de vizinhos para o RWP possui uma alta variabilidade [19]. A idéia principal do modelo Uniforme é corrigir esta variação mantendo ao longo de uma simulação uma distribuição uniforme dos nós na área de simulação. Esta mesma observação pode ser vista para o modelo Grupo Uniforme que é uma extensão do modelo Uniforme, o que indica que pode haver diferenças no comportamento das curvas da percentagem média de vizinhos tanto para o modelo Uniforme quanto para o modelo *Random Waypoint*. Como

observado na Figura 16, a curva do *Random Waypoint* apresenta uma evidente variabilidade nos primeiros segundos de simulação, inciando com percetangem em torno de 6% e mantendo uma estabilidade ao redor de 8%. Na curva de percentagem do modelo Uniforme, essa variação é bem menor, pois mantém uma percentagem de vizinhos estável em torno de 6%.

No caso do RWP , isso ocorre porque para o tempo *t=0*, os nós estão distribuídos uniformente, entretanto ao longo da simulação esses nós tendem a se concentrarem no centro da área da simulação, o que provoca esse efeito na percentagem de vizinhos.

Para o modelo Grupo Uniforme foi feito o mesmo cálculo de percentagem média de vizinhos e observaram-se dois cenários relevantes nesse modelo. Como ilustrado na Figura 16, a simulação depende do valor da diagonal que determina a área de um quadrado a partir de cada nó-referência, onde é permitido a movimentação dos nós internos em relação a esse ponto de referência. Portanto, observa-se que quanto menor for a diagonal mais a curva do modelo Grupo Uniforme se aproxima do modelo Uniforme. Á medida que a diagonal aumenta o modelo Grupo Uniforme se distancia do modelo Uniforme. Isto pode ocorrer porque, dependendo do tamanho da diagonal, há uma variação da quantidade de 'nós' dentro dos grupos, provocando uma maior dinâmica dos 'nós' ao mesmo tempo.

A Figura 17 ilustra as distribuições dos 'nós' para o tempo decorrido de *1000s* nos cenários simulados da Figura 16. Nas Figuras 17(B) e 17(C), percebe-se que os modelos RWP e Gauss-Markov concentram os nós em torno do centro da área de simulação. Enquanto nas Figuras 17(C) e 17(E), cuja ilustração mostra, respectivamente, os modelos Uniforme e Grupo Uniforme com uma diagonal igual 10, percebe-se uma grande uniformidade na distribuição dos nós. E, finalmente a Figura 17(F), ilustra as falhas de distribuição de nós que uma diagonal grande provoca na área de simulação para o modelo Grupo Uniforme.

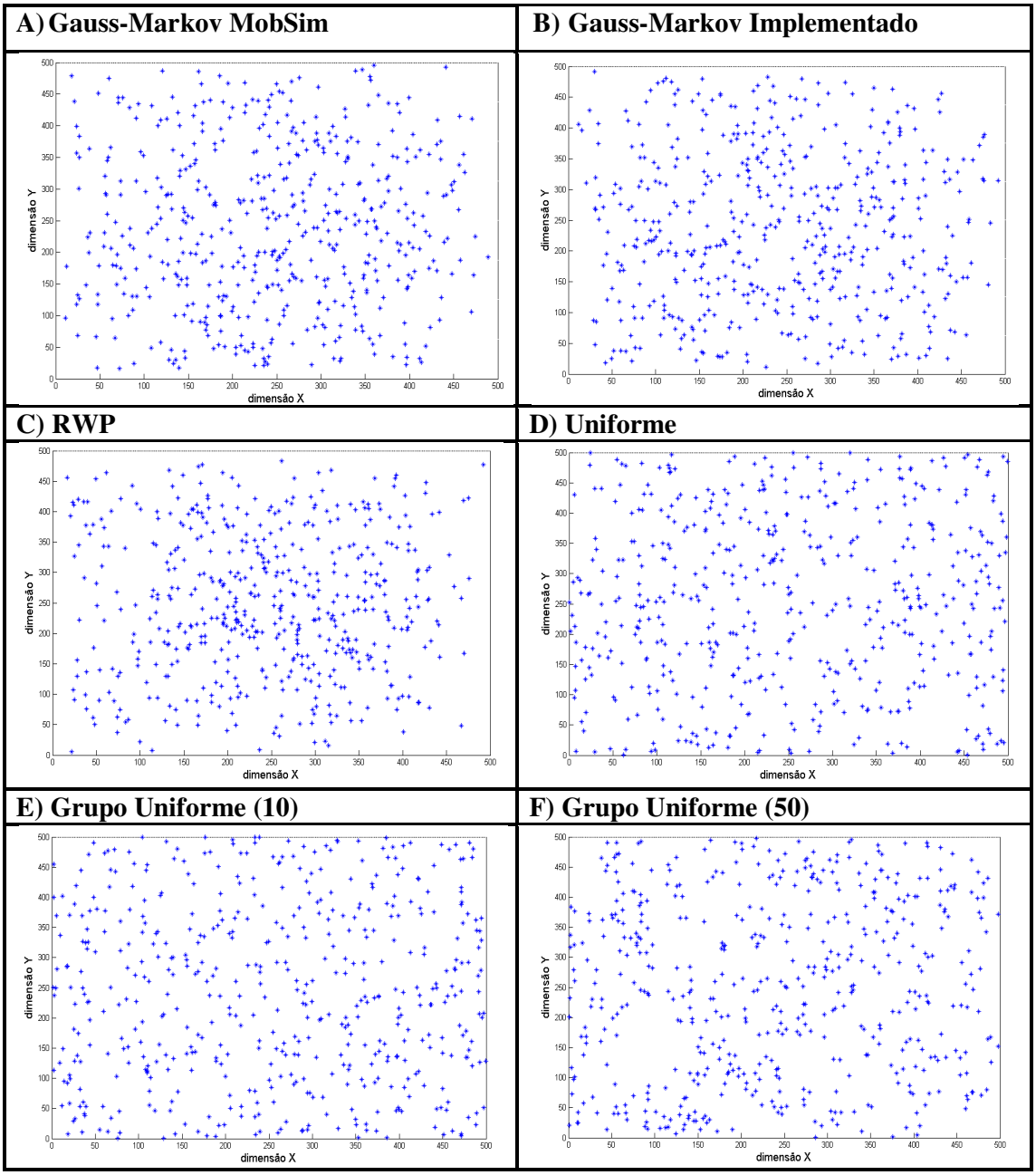

Figura 17: Distribuição dos nós para os modelos implementados, o RWP do JiST/SWANS e Guass-Markov do MobiSim.

5.2 Configuração dos Experimentos

5.2.1 Configuração de Hardware da Máquina Utilizada para Realizar as Simulações

Todas as simulações realizadas durante o desenvolvimento deste trabalho foram feitas numa máquina com processador Intel Core2Quad Q6600 com quatro núcleos, cada qual com freqüência de clock de 2.4GHz, 8GB de memória RAM e 6GB de swap.

5.2.2 Scripts de Análise das Métricas de Interesse ao Trabalho

Para o desenvolvimento dos artefatos de software utilizados durante este trabalho para análise dos arquivos de *trace* gerados pelas simulações executadas no JiST/SWANS, optou-se por utilizar a linguagem de programação C++. Esta linguagem é bastante conhecida por sua eficiência na execução de rotinas que requerem grandes quantidades de processamento. Conforme definido no capítulo 1, as métricas de interesse do presente trabalho são: taxa de sobrecarga de pacotes de controle AODV; taxa de entrega de mensagens (pacotes UDP); atraso médio de mensagens e percentagem de vizinhos. Para fazer extração destas métricas utilizou-se o programa C++, apresentado no Apêndice E.

A taxa de sobrecarga de pacotes de controle AODV é calculada fazendo uma contagem simples dos pacotes AODV que são enviados no *trace* gerado pela simulação. Em seguida, são contados os pacotes de UDP que são enviados no mesmo *trace*. Essa taxa de sobrecarga é dada pela relação de todos os pacotes de controle AODV enviados pelo total de pacotes gerados.

A taxa de entrega de mensagens é calculada da mesma forma que a anterior, entretanto essa taxa é a razão entre total de pacotes de mensagens UDP recebidos pelo total de pacotes gerados durante a simulação.

O atraso médio de descoberta de rotas é calculado pela diferença entre o momento do empilhamento de um pacote de *broadcast route request* de algum nó fonte e o tempo de

recebimento da primeira resposta a essa requisição de rota pelo nó fonte. Por exemplo, na linha 1 da Listagem 10 há um envio ou empilhamento de pacote (*route request*) AODV por *broadcast* indicado pelo '–d -1' com identificação do nó fonte igual 4 '-s 4'. Na linha 19, há uma primeira recepção de resposta (*route reply*) pelo nó fonte igual a 4, logo o atraso de descoberta de rotas é igual a diferença do tempo de recepção do pacote de resposta (*route reply*) pelo tempo de envio do pacote de requisição (*route request*).

Listagem 10. Exemplo de descrobrimento de rota no *trace* gerado pelo JiST/SWANS.

```
1.    + -t 50.000362 -s 4 -d -1 -p aodv -e 44 -i 800 -a 0
2.    - -t 50.000362 -s 4 -d -1 -p aodv -e 44 -i 800 -a 0
3.    r -t 50.00083 -s 4 -d 10 -p aodv -e 44 -i 800 -a 0
4.    r -t 50.00083 -s 4 -d 16 -p aodv -e 44 -i 800 -a 0
5.    r -t 50.00083 -s 4 -d 9 -p aodv -e 44 -i 800 -a 0
6.    r -t 50.00083 -s 4 -d 15 -p aodv -e 44 -i 800 -a 0
7.    r -t 50.00083 -s 4 -d 2 -p aodv -e 44 -i 800 -a 0
8.    r -t 50.00083 -s 4 -d 6 -p aodv -e 44 -i 800 -a 0
9.    r -t 50.00083 -s 4 -d 1 -p aodv -e 44 -i 800 -a 0
10.   d -t 50.00083 -s 4 -d 14 -p aodv -e 44 -i 800 -a 0
11.   d -t 50.00083 -s 4 -d 3 -p aodv -e 44 -i 800 -a 0
12.   r -t 50.00083 -s 4 -d 5 -p aodv -e 44 -i 800 -a 0
13.   r -t 50.00083 -s 4 -d 11 -p aodv -e 44 -i 800 -a 0
14.   + -t 50.001167 -s 11 -d 4 -p aodv -e 40 -i 801 -a 0
15.   - -t 50.001167 -s 11 -d 4 -p aodv -e 40 -i 801 -a 0
16.   r -t 50.001602 -s 11 -d 4 -p aodv -e 40 -i 801 -a 0
17.   + -t 50.00172 -s 4 -d 11 -p udp -e 92 -i 0 -a 0
18.   - -t 50.00172 -s 4 -d 11 -p udp -e 92 -i 0 -a 0
19.   r -t 50.00257 -s 4 -d 11 -p udp -e 92 -i 0 -a 0
```

A percentagem de vizinhos utilizada para validar os modelos de mobilidade implementados é calculada pela relação dos nós que estiverem localizados dentro da área da vizinhaça determinada por uma circuferência de raio *r* pelo total de nós, por exemplo, se de um total de 50 nós, um 'nó' tem 10 vizinhos, logo sua percentagem de vizinhos é igual a 20%.

5.2.3 Validação dos Scripts de Análise das Métricas de Interesse ao Trabalho

Para validar a extração das métricas foi realizada uma verificação da corretude dos valores computados pelo programa através de uma inspeção visual e manual do arquivo de *trace* dos valores das métricas calculadas.

5.3 Análise dos Resultados Obtidos

Nesta seção são mostrados os gráficos gerados a partir da extração dos dados dos arquivos de *trace* produzidos pelas simulações de interesse principal deste trabalho, as quais visam reproduzir o comportamento de redes *ad hoc*.

Os gráficos exibidos a seguir mostram curvas resultantes da obtenção dos valores de: (i) taxa de entrega de pacotes de mensagens, (ii) taxa de sobrecarga de pacotes AODV, (iii) atraso médio da descoberta de rotas para uma média aritmética de 10 amostras para cada cenário testado. Os cenários foram testados à medida que aumentamos a densidade da rede. Na parte inferior dos gráficos está localizaoa o total de nós presentes na rede para o tempo de simulação de 500s para uma área constante de 500x500m^2, os demais parâmetros das simulações estão na Tabela 2 da Seção 4.4.

Nas redes testadas aqui, não foi determinada uma quantidade mínima de receptores e transmissores. Na verdade, qualquer nó pode ser um recepetor ou um transmissor. Para os modelos Uniforme e Grupo Uniforme foi utilizada a média μ=100. A quantidade de grupos (nó-referéncia) que foi utilizada para o modelo Grupo Uniforme foi 15% total de nós na rede.

5.3.1 Análise da Taxa de Entrega de Pacotes

Para ilustrar o desempenho da entrega de mensagens para cada modelo de mobilidade implementado no simulador JiST/SWANS mais o modelo RWP já presente nele, resultou o gráfico apresentado na Figura 18 . É possível perceber à medida que a quantidade de nós na rede cresce, ou seja, a partir de 128 nós, observa-se que a taxa de entrega de mensagens tende para um mesmo nível para todos os modelos. Isto pode ocorrer porque há uma grande saturação de nós se comunicando provocando um alto nível de interferência gerada pela grande quantidade de pacotes de controle que são requeridos para encontrar ou manter uma rota de comunicação.

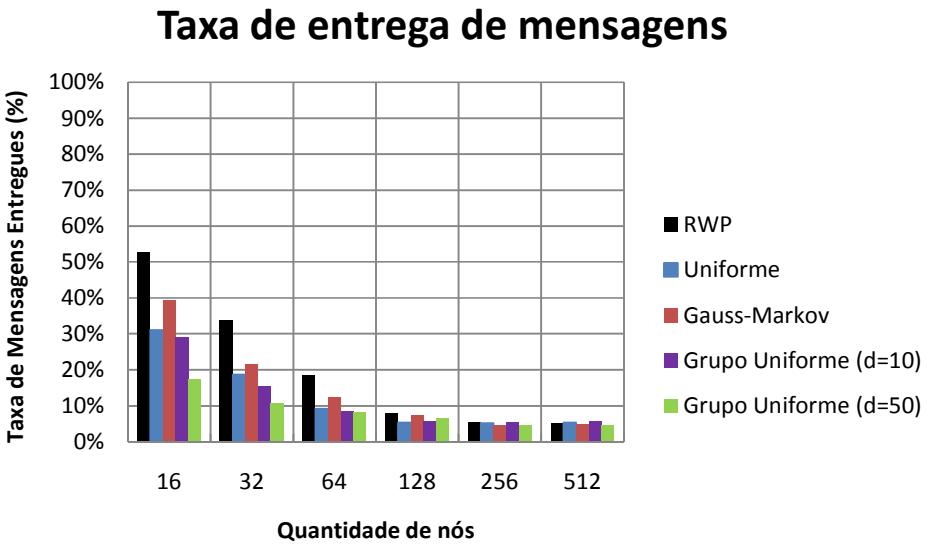

Figura 18: Gráfico da taxa de entrega de mensagens quando a densidade nós da rede aumenta.

Na figura 18, nota-se que modelo RWP possui o melhor desempenho na taxa de entrega de mensagens visto que os nós tendem a se espalhar pouco por toda a área da rede como foi visto na Figura 17(C) na Seção 5.1.1. Nesse movimento há uma maior probalidade do nó manter um enlace de comunicação por mais tempo visto que os nós sempre estão próximos uns dos outros. Para o modelo Uniforme e Grupo Uniforme (d=10) quando têm poucos nós na rede nota-se que os dois modelos têm pior desempenho em relação ao RWP. Nestes dois modelos os nós comunicantes ficam mais espalhados pela área de simulação dificultando o estabelecimento e manutenção de rotas de comunicação provocando maiores perdas nos pacotes enviados. O mesmo pode ser atribuído ao modelo Grupo Uniforme (d=10), em virtude da mínima diferença comparado ao modelo Uniforme como demonstrado na percentagem de vizinhos e na distribuiçao dos nós, abordados na Seção 5.1.5.

Os modelos RWP e Gauss-Markov têm desempenhos bastante semelhantes visto que nestes modelos há uma menor dinâmica dos nós, ao contrário dos modelos Uniforme e Grupo Uniforme(d=10), que espalham os nós por toda e área de simulação.

Já o modelo Grupo Uniforme (d=50) tem o pior desempenho de todos visto que apresenta grandes falhas de distibuição de nós, ao longo da simulação, devido à brusca

movimentação dos grupos pela área de simulação. Isso pode provocar certa instabilidade na manutenção das rotas gerando uma grande quantidade de pacotes de controle, o que pode levar a uma diminuição da taxa de entrega de mensagens, gerando perdas por estouro da pilha de pacotes mensagens.

5.3.2 Análise da Sobrecarga de Roteamento

O resultado observado em relação à taxa de sobrecarga (*overhead*) de pacotes de controle AODV é demonstrado na Figura 19. Este gráfico possui certa regularidade quando a quantidade de nós na rede tende a crescer. Isto pode ocorrer devido à enorme quantidade de nós crescentes numa mesma área, provocando muita instabilidade no desempenho da comuinicação com o protocolo AODV, o que gera muitos pacotes de controle para descoberta e manutenção da rede. Quando a quantidade de nós é pequena essa taxa varia de acordo com o modelo de mobilidade.

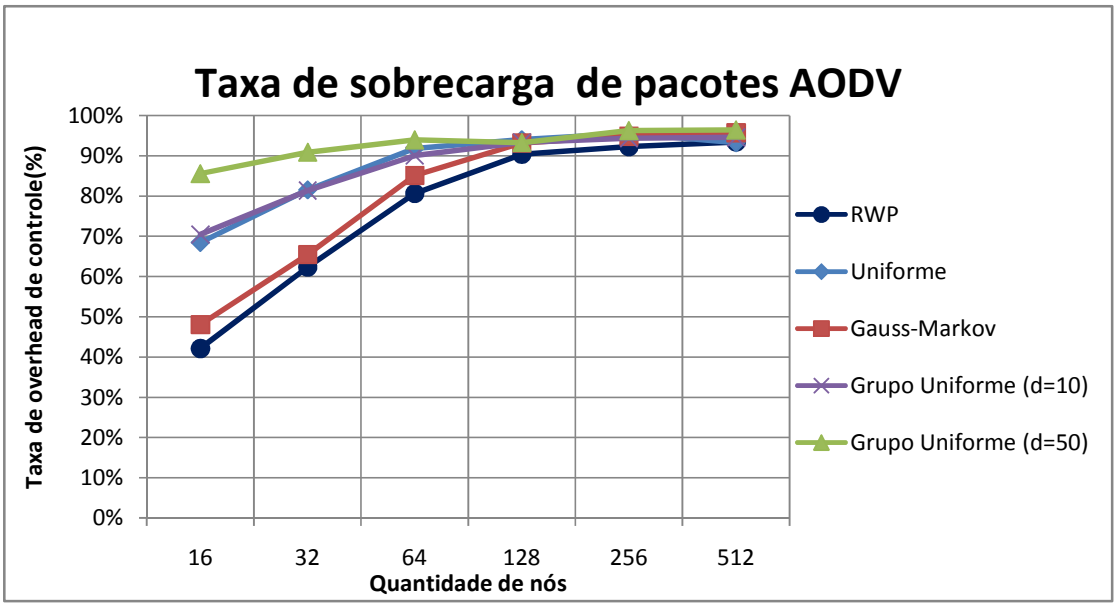

Figura 19: Gráfico da taxa de sobrecarga AODV

Os modelos RWP e Gauss-Markov apresentaram melhor desempenho em relação aos demais, uma vez que a dinâmica dos nós ao redor do centro da rede é menor em relação ao demais havendo uma concentração dos nós e uma maior estabilidade dos enlaces de comunicação.

Os modelos Uniforme e Grupo Uniforme (d=10) mostraram comportamentos semelhantes, visto que esses dois modelos têm o mesmo padrão de mobilidade, como abordado na Seção 5.1.1. Nesse sentido, tanto a distribuição quanto a percentagem de vizinhos são próximos, exceto no Grupo Uniforme, cujo padrão de mobilidade é baseado em grupos e indivíduos que se movem uniformemente tentando manter sempre uma distribuição uniforme dos nós e dos grupos na área. Essa semelhança entre Grupo Uniforme (d=10) e Uniforme pode ser explicada pelo pequeno tamanho do grupo acarretando muitos nós-externos livres para se movimentarem por toda a área. Já o Grupo Uniforme (d=50), por possuir uma variação muito grande de vizinhos, apresentou uma queda considerável no seu desempenho em relação ao Uniforme, visto que nesse modelo há uma maior probabilidade de muitos nós se moverem em grupo, podendo gerar instabilidade nos enlaces de comunicação com os demais grupos e nós.

5.3.3 Análise do Atraso Médio

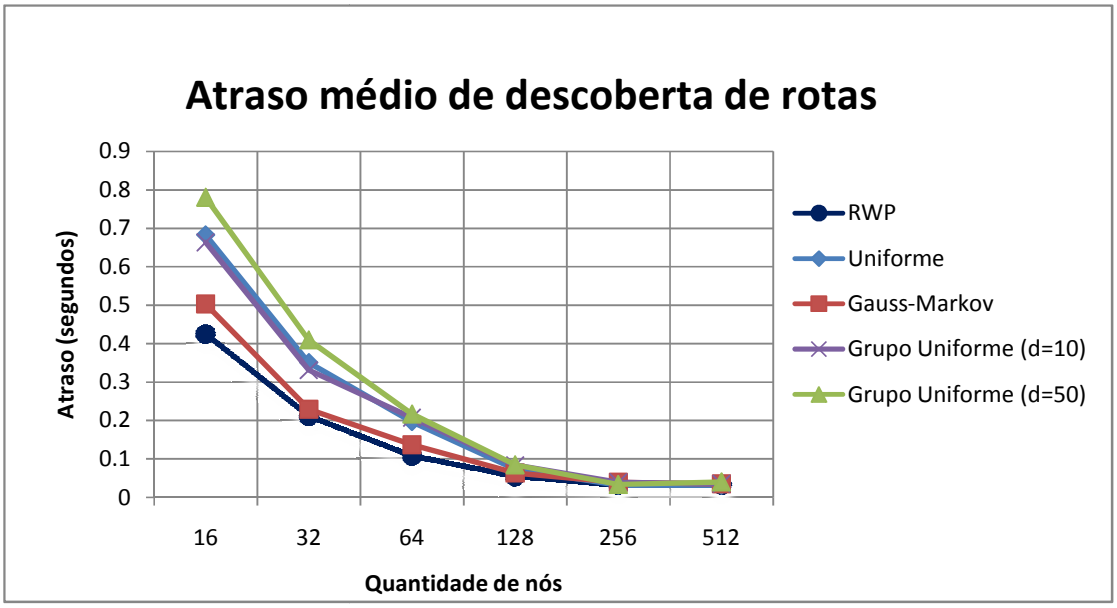

Figura 20. Gráfico do atraso médio da descoberta de rotas.

Os resultados obtidos para o atraso médio de descoberta de rotas são apresentados na Figura 20. Verifica-se que o comportamento do atraso médio é praticamente semelhante a todos os modelos de mobilidades quando a quantidade de nós é alta, isto ocorre devido à densidade na rede que tende crescer, e assim o atraso tende a dimininuir rapidamente, pois

aumenta a probalidade de encontrar um nó nas proximidades. Para uma pequena quantidade de nós modelos Unifome e Grupo Uniforme possuem um maior atraso de descoberda de rotas conseqüência dos nós mais distantes uns dos outros provocando uma maior quantidade de saltos para os pacotes chegar ao destino.

De acordo com a avaliação feita nesse trabalho, os modelos Random Waypoint e Guass-Markov possuem melhor desempenho em relação ao demais. Entretanto o ótimo desempenho ocorrido pode indicar imprecisão desses modelos visto que podem disfarçar a dinâmica real de uma rede devido à evidente concentração de nós em torno do centro da rede. Apesar do modelo Uniforme possuir pior desempenho em relação ao Random Waypoint e Gauss-Markov, o presente trabalho recomenda utilizar o modelo Uniforme ao invés do Random Waypoint para avaliações de redes *ad hoc* visto que ele pode modelar de forma mais justa ou até mesmo melhor uma realidade de movimentação dos nós. Além disso, o modelo Uniforme possui reflexão nas bordas, onde cada nó varre toda a rede ao longo do tempo, o que não é o caso do RWP e do Gauss-Markov.

Resultados bastante semelhantes foram encontrados em [32], a diferença para o nosso é que o ambiente de simulação em [32] é com densidade constante de nós, enquanto o nosso é crescente.

Capítulo 6

Conclusão e Trabalhos Futuros

Este trabalho realizou a implementação dos modelos Uniforme, Gauss-Markov, Grupo Uniforme e uma análise de desempenho do protocolo de roteamento AODV em simulações de redes a*d hoc* com o JiST/SWANS, sendo possível concluir que à medida em que a densidade de nós nas redes aumenta menor é o desempenho da rede. Também foi verificado que o modelo de mobilidade Uniforme, apesar de possuir pior desempenho que o Random Waypoint, é melhor para representar as condições da realidade em redes movéis dependedo do ambiente simulado desejado.

6.1 Contribuições e Conclusões

O estudo da mobilidade em redes a*d hoc* é de fundamental importância para futuras aplicações dessas redes em ambientes diversos, seja pela necessidade de comunicação em casos de desastres, guerras, quanto à necessidade crescente de comunicação que temos com a evolução natural da tecnologia, cujos equipamentos antes destinados a uma única finalidade, como aparelhos celulares para telecomunicação, atualmente vem abrangendo seu uso para os mais diversos fins, como acesso à Internet, TV Digital.

Para que haja uma comunicação eficiente sem a necessidade de uma infra-estrutura física, é importante que se invista em pesquisas e testes, inicialmente por meio de simulações, como as que foram realizados neste trabalho, principalmente pelo fator custo e complexidade de gerenciamento ser bem menor que a realização de testes com equipamentos reais.

Foi possível mostrar através de simulações quais modelos de mobilidade implementados causam mais impacto no desempenho de redes a*d hoc*. Além disso, foi sugeridas explicações do porque das métricas estabelecidas manterem um determinado comportamento, baseado no estudo dos protocolos e modelos utilizados para as simulações. Tambem notou-se que a densidade crescente é um fator limitante no desempenho da rede.

Nesse trabalho foi verificado que o modelo Random Waypoint possue melhor desempenho, entretando recomendamos utilizar o modelo Uniforme em avaliaçaoes de redes *ad hoc* por possuir justiça nas distribuições dos nós representando de forma mais precisa a realidade. Para que futuras pesquisas possam ser realizadas com os novos modelos de mobilidade implementados nesse trabalho será disponibilizado no seguinte endereço http://www.dsc.upe.br/~fpa/jist-swans.zip uma nova extensão do JiST/SWANS com os modelos de mobilidade acrescentados.

6.2 Trabalhos Futuros

A análise de métricas de desempenho de redes a*d hoc* ainda pode ser bastante aprofundada através da utilização de maior número de nós, utilização de outras configurações de rede disponíveis no JiST/SWANS, além de análise e implementação de outros modelos de mobilidade de nós. Existem ainda possibilidades de expandir os estudos realizados neste trabalho com a utilização de outros padrões de movimentação dos nós como Mahartan[5], Purse[5] e RGPM [5]

Outro importante aspecto que pode ser investigado é o comportamento do modelo *Random Waypoint* e Uniforme sem considerar o estado estacionário, ou seja, velocidade mínima diferente de zero. Cabe ainda como uma expansão deste trabalho avaliar o modelo Uniforme com diferentes valores da média μ usada para gerar as distâncias exponencialmente distribuídas dos trajetos realizados pelos nós, pois é possível que o desempenho do modelo Uniforme varie de acordo com a média μ.

Bibliografia

[1] J. Yoon; M. Liu; B. Noble. Random waypoint considered harmful. In: Proceedings of IEEE INFOCOM 2003.

[2] Guolong Lin, Guevara Noubir, and Rajmohan Rajaraman. Mobility Models for Ad hoc Network Simulation. In: Proceedings of IEEE INFOCOM 2004.

[3] Fan Bai et.al. IMPORTANT: a framework to systematically analyze the Impact of Mobility on Performance of RouTing protocols for Adhoc NeTworks. In: IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, v. 3, p. 825-835, 2003.

[4] David B. Johnson, Davis A. Maltz. Dynamic Source Routing in Ad Hoc Networks, Mobile Computing, T. Imielinski and H. Korth, Eds. Kulwer, p. 152- 181, 1996.

[5] V. Davies. Evaluating mobility models within an ad hoc network. Master's thesis, Colorado School of Mines, 2000.

[6] Prabhakaran, P.; Sankar, R. Impact of Realistic Mobility Models on Wireless Networks Performance. In: Wireless and Mobile Computing, Networking and Communications, 2006. (WiMob'2006). IEEE International Conference on, p. 329-334, jun. 2006.

[7] The Network Simulator - NS-2, Disponível em http://www.isi.edu/nsnam/ns/, Acesso em: 11/11/2008.

[8] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. In Workshop on Parallel and Distributed Simulation, 1998.

[9] Frank Kargl and Elmar Schoch. Simulation of MANETs: a qualitative comparison between JiST/SWANS and ns-2. In: MobiEval '07: Proceedings of the 1st international workshop on System evaluation for mobile platforms, San Juan, Puerto Rico, p. 41-46, 2007.

[10] Elmar Schoch et.al. Simulation of Ad Hoc Networks: ns-2 compared to JiST/SWANS. In: First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SimuTools), Marseille, France, 2008.

[11] SWANS Documentation, Disponível em: http://jist.ece.cornell.edu/jist-user/index.html, Acesso em: 16/11/2008.

[12] R. Barr, Z. Haas, and R. van Renesse. JiST: An efficient approach to simulation using virtual machines. In: Software Practice & Experience, 2005, v. 35(6), p. 539-576.

[13] Vanet.info homepage, Disponivel em: http://vanet.info, Acesso em: 15/11/2008.

[14] Swans++ homepage, Disponível em: http://www.aqualab.cs.northwestern.edu/projects/swans++, Acesso em: 16/11/2008.

[15] G. Brasche and B. Walke. Concepts, Services and Protocols of the new GSM Phase 2+ General Packet Radio Service, IEEE Communications Magazine, p. 94-104, 1997.

[16] Trishla Sutaria et.al. Implementation of an Energy Model for JiST/SWANS Wireless Network Simulator. In: ICN '07: Proceedings of the Sixth International Conference on Networking, p. 24-24, 2007.

[17] Tippanagoudar, V. et.al. Implementation of the Sensor-MAC Protocol for the JiST/SWANS Simulator. In: AICCSA '07: IEEE/ACS International Conference on Computer Systems and Applications, p. 225-232, 2007.

[18] N. Bansal and Z. Liu. Capacity, delay and mobility in wireless ad-hoc networks. In: Proc. of IEEE Infocom, San Francisco, California, March 2003.

[19] Tracy Camp et.al. A Survey of mobility models for mobile ad hoc network research, Wireless Communication & Mobile Computing: Special issue on mobile Ad hoc Networking: Research ,Trends and Applications, v. 2, n. 5, p. 483-502, 2002.

[20] Xiaoyan Hong et.al. A Mobility Framework for Ad hoc Wireless Networks. In: MDM '01: Proceedings of the Second International Conference on Mobile Data Management, p 185- 196, 2001.

[21] JiST Documentation, Disponivel em: http://jist.ece.cornell.edu/jist-user/index.html, Acesso em: 15/11/2008.

[22] Z. J. Haas. A new protocol for the reconfigurable wireless networks. In: IEEE International Conference on Universal Personal Communication (ICUPC), pages 562–565, October 1997.

[23] B. Liang and Z. J. Haas. Predictive distance-based mobility management for pcs networks. In Conference of the IEEE Computer and Communications Societies (INFOCOM), New York, NY, March 1999.

[24] JiST/SWANS homepage, Disponivel em: http://jist.ece.cornell.edu/, Acesso em: 15/11/2008.

[25] 7zip homepage, Disponivel em: http://www.7-zip.org/, Acesso em: 15/11/2008.

[26] Eclipse homepage, Disponível em: http://www.gnuplot.info/, Acesso em:

[27] Network Animator homepage, Disponível em: http://www.isi.edu/nsnam/nam/, Acesso em: 16/10/2008.

[28] SSJ: Simulation Stochastique en Java L'Ecuyer, P., L. Meliana and J. Vaucher. SSJ: A Framework for Stochastic Simulation in Java, In: Proceedings of the 2002 Winter Simulation Conference, ed. C. Chen, P. Sanchez, D. Ferrin and D. Morrice, pp. 234-242, San Diego, December 2002.

[29] C. E. Perkins and E.M. Royer. Ad-hoc On-Demand Distance Vector Routing. Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, pages 90–100, New Orleans, LA, February 1999.

[30] S. M. Mousavi, H. R. Rabiee, M. Moshref, A. Dabirmoghaddam. MobiSim: A Framework for Simulation of Mobility Models in Mobile Ad-Hoc Networks. In: Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2007).

[31] X. Hong, M. Gerla, G. Pei, and C. Chiang. A group mobility model for ad hoc wireless networks. In 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, pages 53–60, August 1999.

[32] Zheng Kai; Wang Neng; Liu Ai-fang, A new AODV based clustering routing protocol. In: Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on , vol.2, no., pp. 728-731, 23-26 Sept. 2005.

Apêndice A

Código da classe da implementação do modelo mobilidade Uniforme

```
import java.util.ArrayList;
import java.util.Iterator;

import jist.swans.misc.Location;
import jist.swans.misc.Util;
import jist.swans.misc.Location.Location2D;
import jist.swans.Constants;
import jist.swans.Main;

import jist.runtime.JistAPI;

class UniformeInfo implements MobilityInfo
{
  public double direcao;
  public double distancia;
  public int passos;
  public double tempoPasso;
  public double velocidade;

  public UniformeInfo(double velocityMin,double velocityMax,double mu,int
steps){
     direcao = 2*Math.PI*Constants.random.nextDouble();
     velocidade =  velocityMin + (velocityMax -
velocityMin)*Constants.random.nextDouble();
     distancia = Constants.exprnd(mu);
     this.passos = steps;
     double timeTotal = distancia/velocidade;
     this.tempoPasso =timeTotal/steps;
  }
}

public static class Uniforme implements Mobility
  {

     private double vMax,vMin,mu;
     private int passos;
     private Location.Location2D limites;
```

```
     public Uniforme(Location.Location2D bounds,String config, int n){
       this.limites = bounds;
       String ksConfigOptions [];
       ksConfigOptions= config.split(":");
       vMin = Double.parseDouble(ksConfigOptions[0]);
       vMax = Double.parseDouble(ksConfigOptions[1]);
       mu = Double.parseDouble(ksConfigOptions[2]);
       passos =Integer.parseInt(ksConfigOptions[3]);

     }

     public MobilityInfo init(FieldInterface f, Integer id, Location loc) {

       return new UniformeInfo(vMin,vMax,mu,passos);
     }

     public void next(FieldInterface f, Integer id, Location loc,
MobilityInfo info) {
       UniformeInfo uinfo = (UniformeInfo)info;
       double stepDist = uinfo.velocidade*uinfo.tempoPasso;
       double novoX = loc.getX()+ stepDist*Math.cos(uinfo.direcao);
       double novoY = loc.getY()+ stepDist*Math.sin(uinfo.direcao);
       while(novoX<0 || novoX>limites.getX() || novoY<0 ||
novoY>limites.getY()){
         double deltaXExt = novoX-loc.getX();
         double deltaYExt = novoY-loc.getY();
         double a = deltaYExt/deltaXExt;
         double b =  deltaYExt - (a*deltaXExt);
         Location2D lastPoint = null,reflexPoint=null;
         double deltaXInt, deltaYInt;

         if(novoY<0){
           lastPoint = new Location2D((float)((0 - b)/a),0);
           reflexPoint = new Location2D((float)novoX,(float)(-1*novoY));
           novoY = -1*novoY;
           deltaXInt = reflexPoint.getX() - lastPoint.getX();
           deltaYInt = reflexPoint.getY() - lastPoint.getY();

           if(uinfo.direcao>3*Math.PI/2 && uinfo.direcao<2*Math.PI)

             uinfo.direcao =  Math.atan(deltaYInt/deltaXInt);
           if(uinfo.direcao>Math.PI && uinfo.direcao<3*Math.PI/2)
             uinfo.direcao = Math.PI + Math.atan(deltaYInt/deltaXInt);

         }else if(novoY>limites.getY()){
           lastPoint = new Location2D((float)((limites.getY() -
b)/a),limites.getY());
           reflexPoint = new
Location2D((float)novoX,(float)(2*limites.getY() - novoY));
           novoY = 2*limites.getY() - novoY;
           deltaXInt = reflexPoint.getX() - lastPoint.getX();
           deltaYInt = reflexPoint.getY() - lastPoint.getY();

           if(uinfo.direcao>0 && uinfo.direcao<Math.PI/2)
             uinfo.direcao =  2*Math.PI + Math.atan(deltaYInt/deltaXInt);
           if(uinfo.direcao>Math.PI/2 && uinfo.direcao<3*Math.PI)
             uinfo.direcao = Math.PI + Math.atan(deltaYInt/deltaXInt);
```

```
          }
          else if(novoX<0){
            lastPoint = new Location2D(0,(float)(a*0 + b));
            reflexPoint = new Location2D((float)(-1*novoX),(float)novoY);
            novoX = -1*novoX;
            deltaXInt = reflexPoint.getX() - lastPoint.getX();
            deltaYInt = reflexPoint.getY() - lastPoint.getY();

            if(uinfo.direcao>Math.PI && uinfo.direcao<3*Math.PI/2)
              uinfo.direcao = 2*Math.PI + Math.atan(deltaYInt/deltaXInt);
            if(uinfo.direcao>Math.PI/2 && uinfo.direcao<Math.PI)
              uinfo.direcao = Math.atan(deltaYInt/deltaXInt);
          }else{
            lastPoint = new
Location2D(limites.getX(),(float)(a*limites.getX() + b));
            reflexPoint = new Location2D((float)(2*limites.getX() -
novoX),(float)novoY);
            novoX = 2*limites.getX() - novoX;
            deltaXInt = reflexPoint.getX() - lastPoint.getX();
            deltaYInt = reflexPoint.getY() - lastPoint.getY();

            if(uinfo.direcao>0 && uinfo.direcao<Math.PI/2)
              uinfo.direcao = Math.PI + Math.atan(deltaYInt/deltaXInt);
            if(uinfo.direcao>3*Math.PI/2 && uinfo.direcao<2*Math.PI)
              uinfo.direcao = Math.PI  + Math.atan(deltaYInt/deltaXInt);
          }
        }
        uinfo.passos--;
        Location2D newLoc = new Location2D((float)novoX,(float)novoY);
        JistAPI.sleep((long)(uinfo.tempoPasso*Constants.SECOND));
        f.moveRadio(id,newLoc);
        if(uinfo.passos<=0){
          uinfo =  new UniformeInfo(vMin,vMax,mu,passos);
        }
      }
    }
```

Apêndice B

Código da classe da implemetação do modelo Guass-Markov

```
import java.util.ArrayList;
import java.util.Iterator;

import jist.swans.misc.Location;
import jist.swans.misc.Util;
import jist.swans.misc.Location.Location2D;
import jist.swans.Constants;
import jist.swans.Main;

import jist.runtime.JistAPI;

class GaussMarkovInfo implements MobilityInfo{
  public double velocidade;
  public double velocidadeMedia;
  public double direcao;
  public double direcaoMedia;

  public GaussMarkovInfo(double speed,double speed_medio,double
direction,double direction_medio){
     this.velocidade = speed;
     this.direcao = direction;
     this.direcaoMedia = direction_medio;
     this.velocidadeMedia = speed_medio;
   }
}
public static class GaussMarkov implements Mobility{

     private double direcaoMediaInicial;
     private double velocidadeMediaInicial;
     private double borda;
     private double vMax;
     private double vMin;
     private double alpha;
     private double tempoPausa = 1;
     private Location limites;
```

```
     public GaussMarkov(Location bounds, String config){
       this.limites = bounds;
       String GaussMarkovConfigOptions [];
       GaussMarkovConfigOptions= config.split(":");
       vMin = Double.parseDouble(GaussMarkovConfigOptions[0]);
       vMax =Double.parseDouble(GaussMarkovConfigOptions[1]);
       velocidadeMediaInicial =
Double.parseDouble(GaussMarkovConfigOptions[2]);
       direcaoMediaInicial
=Double.parseDouble(GaussMarkovConfigOptions[3]);
       alpha =Double.parseDouble(GaussMarkovConfigOptions[4]);
       borda =Double.parseDouble(GaussMarkovConfigOptions[5]);

     }
     public MobilityInfo init(FieldInterface f, Integer id, Location loc) {
       double speed_inicial = vMin + (vMax-
vMin)*Constants.random.nextDouble();
       double direction_inicial = 2*Math.PI*Constants.random.nextDouble();

       return new
GaussMarkovInfo(speed_inicial,velocidadeMediaInicial,direction_inicial,dire
caoMediaInicial);
     }

     public void next(FieldInterface f, Integer id, Location loc,
MobilityInfo info) {
       double x = loc.getX();
       double y = loc.getY();
       double limiteXinf = borda*limites.getX();
       double limiteYinf = borda*limites.getY();
       double limiteXsup = limites.getX() - borda*limites.getX();
       double limiteYsup = limites.getY() - borda*limites.getY();

       GaussMarkovInfo gminfo = (GaussMarkovInfo)info;
       double direction_medio = gminfo.direcaoMedia;
       double X=limites.getX(),Y=limites.getY();
       if (x > 0 && x < limiteXinf && y > 0 && y < limiteYinf)
         direction_medio = Math.PI/4;
       else if (x > limiteXinf && x < limiteXsup && x > 0 && y <
limiteYinf)
         direction_medio = Math.PI/2;
       else if (x > limiteXsup && x < X && y > 0 && y < limiteYinf)
         direction_medio = 3*Math.PI/4;
       else if (x > limiteXsup && x < X && y > limiteYinf  && y <
limiteYsup)
         direction_medio = Math.PI;
       else if (x > limiteXsup && x < X && y > limiteYsup  && y < Y)
         direction_medio = 5*Math.PI/4;
       else if (x > limiteXinf && x < limiteXsup && y > limiteYsup  && y <
Y)
         direction_medio = 3*Math.PI/2;
       else if (x > 0 && x < limiteXinf && y > limiteYsup  && y < Y)
         direction_medio =7*Math.PI/4;
       else if (x > 0 && x < limiteXinf && y > limiteYinf  && y<
limiteYsup)
         direction_medio = 0;
       else
```

```
          direction_medio = gminfo.direcaoMedia;
       gminfo.direcaoMedia = direction_medio;
       double speed_old = gminfo.velocidade;
           double direction_old = gminfo.direcao;
           double s_ = gminfo.velocidadeMedia;
           double d_ = gminfo.direcaoMedia;
           double speed_new = alpha*speed_old + (1-alpha)*s_ +
Math.sqrt(Math.pow(1-alpha,2))*Constants.random.nextGaussian();
           double direction_new =alpha*direction_old + (1-alpha)*d_ +
Math.sqrt(Math.pow(1-direction_old,2))*Constants.random.nextGaussian();

           double x_old = loc.getX();
           double y_old = loc.getY();
           double s_old = gminfo.velocidade;
           double d_old = gminfo.direcao;
           double x_new = x_old + s_old*Math.cos(d_old);
           double y_new = y_old + s_old*Math.sin(d_old);
           gminfo.direcao = direction_new;
           gminfo.velocidade =speed_new;
           JistAPI.sleep((long)(tempoPausa*Constants.SECOND));
           Location newloc = new
Location.Location2D((float)x_new,(float)y_new);
           if(newloc.inside(new Location.Location2D(0,0) ,new
Location.Location2D(limites.getX(),limites.getY())))
             f.moveRadio(id, newloc);
     }
   }
```

Apêndice C

Código da classe de implemetação do modelo de mobilidade Grupo Uniforme

```
import java.util.ArrayList;
import java.util.Iterator;

import jist.swans.misc.Location;
import jist.swans.misc.Util;
import jist.swans.misc.Location.Location2D;
import jist.swans.Constants;
import jist.swans.Main;

import jist.runtime.JistAPI;

class GrupoUniformeInfo implements MobilityInfo
  {
    public double direcao;
    public double distancia;
    public double velocidade;
    public int passosExec;
    public double tempoPasso;
    public Location locReferencia;
    public int tipo = 3;
    public ArrayList<Integer> nosInternos=new ArrayList<Integer>();;

    public GrupoUniformeInfo(double Vmin, double Vmax,double mu,int
steps){
       this.passosExec = steps;
       direcao = 2*Math.PI*Constants.random.nextDouble();
       distancia = Constants.exprnd(mu);
       velocidade = Vmin + (Vmax - Vmin)*Constants.random.nextDouble();
       double timeTotal = distancia/velocidade;
       this.tempoPasso =timeTotal/steps;
    }
    public void renew(double Vmin, double Vmax, double mu,int steps){
       direcao = 2*Math.PI*Constants.random.nextDouble();
       distancia = Constants.exprnd(mu);
       velocidade = Vmin + (Vmax - Vmin)*Constants.random.nextDouble();
       double timeTotal = distancia/velocidade;
       this.tempoPasso =timeTotal/steps;
       this.passosExec=steps;
```

```
    }
  }
public static class GrupoUniforme implements Mobility{

    private static int qtdpointReference;
    private static int jaqtdpointReference=0;
    private GrupoUniformeInfo []nosReferencias ;
    private static double vMax,vMin;
    private int tempoPausa;
    private int passos;
    private static double mu;
    private double diagonalGrupo;
    private Location.Location2D limites;

    public GrupoUniforme(Location.Location2D bounds,String config){
      String ksConfigOptions [];
      ksConfigOptions= config.split(":");
      vMin = Double.parseDouble(ksConfigOptions[0]);
      vMax = Double.parseDouble(ksConfigOptions[1]);
      mu =Double.parseDouble(ksConfigOptions[2]);
      this.limites = bounds;
      qtdpointReference = Integer.parseInt(ksConfigOptions[3]);
      nosReferencias = new GrupoUniformeInfo[qtdpointReference];
      this.diagonalGrupo = Double.parseDouble(ksConfigOptions[4]);
      this.tempoPausa = Integer.parseInt(ksConfigOptions[5]);
      this.passos = Integer.parseInt(ksConfigOptions[6]);;
    }
    public MobilityInfo init(FieldInterface f, Integer id, Location loc) {

      GrupoUniformeInfo info;
      if(jaqtdpointReference<qtdpointReference){
        info = new GrupoUniformeInfo(vMin,vMax,mu,passos);
        info.locReferencia = loc;
        info.tipo = 1;
        nosReferencias[jaqtdpointReference++] = info;
      }
      else{
        info = new GrupoUniformeInfo(vMin,vMax,mu,passos);
        for (GrupoUniformeInfo i : nosReferencias) {
          if(loc.inside(i.locReferencia, new
Location.Location2D((float)(i.locReferencia.getX()+diagonalGrupo),(float)(i
.locReferencia.getY()+diagonalGrupo))))
          {
            i.nosInternos.add(id);
            info.tipo = 2;
          }
        }
      }
      return info;
    }

    public void next(FieldInterface f, Integer id, Location loc,
MobilityInfo info) {
      GrupoUniformeInfo noinfo = (GrupoUniformeInfo) info;
      if(id>=1 && id<=qtdpointReference){
        JistAPI.sleep(tempoPausa*Constants.SECOND);
        Location oldLoc = loc;
```

```
          Location newLoc = moveNo(f, id, loc, noinfo);
          double deltax =  newLoc.getX() - oldLoc.getX();
          double deltay = newLoc.getY() -  oldLoc.getY();
          noinfo.locReferencia = newLoc;
          for (Iterator iter = noinfo.nosInternos.iterator();
iter.hasNext();) {
            Integer nointerno = (Integer) iter.next();
            f.moveRadioOff(nointerno, new
Location.Location2D((float)deltax,(float)deltay));
          }
       }else
          moveNo(f, id, loc, noinfo);
     }
     private GrupoUniformeInfo getHeadReference(Integer id){
       for (int i = 0; i < nosReferencias.length; i++) {
         GrupoUniformeInfo element = nosReferencias[i];
         if (element.nosInternos.contains(id))
           return element;
       }
       return null;
     }
     private Location2D moveNo(FieldInterface f, Integer id, Location loc,
GrupoUniformeInfo uinfo) {
       GrupoUniformeInfo headinfo = getHeadReference(id);
       double xnode = loc.getX();
       double ynode = loc.getY();
       double X = limites.getX();
       double Y = limites.getY();
       if (uinfo.tipo == 1){
         X = limites.getX()-diagonalGrupo;
         Y = limites.getY()-diagonalGrupo;
       }
       if(X<0 || Y<0)
         throw new Error("Largura do raio do grupo deve ser menor!");
       double X_mov_rel = 0;
       double Y_mov_rel = 0;
       if(headinfo!=null){
         xnode = loc.getX()-headinfo.locReferencia.getX();
         ynode = loc.getY()-headinfo.locReferencia.getY();
         X = diagonalGrupo;
         Y = diagonalGrupo;
         X_mov_rel = headinfo.locReferencia.getX();
         Y_mov_rel = headinfo.locReferencia.getY();
       }

       double stepDist = uinfo.velocidade*uinfo.tempoPasso;
       double newX = xnode + stepDist*Math.cos(uinfo.direcao);
       double newY = ynode + stepDist*Math.sin(uinfo.direcao);
       while(newX<0 || newX>X || newY<0 || newY>Y){
         double deltaXExt = newX-xnode;
         double deltaYExt = newY-ynode;
         double a = deltaYExt/deltaXExt;
         double b =  deltaYExt - (a*deltaXExt);
         Location2D lastPoint = null,reflexPoint=null;
         double deltaXInt, deltaYInt;

         if(newY<0){
```

```
            lastPoint = new Location2D((float)((0 - b)/a),0);
            reflexPoint = new Location2D((float)newX,(float)(-1*newY));
            newY = -1*newY;
            //////new direction after reflection
            if(uinfo.direcao>3*Math.PI/2 && uinfo.direcao<2*Math.PI){

              deltaXInt = reflexPoint.getX() - lastPoint.getX();
              deltaYInt = reflexPoint.getY() - lastPoint.getY();;
              uinfo.direcao =  Math.atan(deltaYInt/deltaXInt);
            }
            if(uinfo.direcao>Math.PI && uinfo.direcao<3*Math.PI/2){

              deltaXInt = reflexPoint.getX() - lastPoint.getX();
              deltaYInt = reflexPoint.getY() - lastPoint.getY();;
              uinfo.direcao = Math.PI + Math.atan(deltaYInt/deltaXInt);
            }


          }else if(newY>Y){
            lastPoint = new Location2D((float)((Y - b)/a),(float)Y);
            reflexPoint = new Location2D((float)newX,(float)(2*Y - newY));
            newY = 2*Y - newY;
            //////new direction after reflection
            if(uinfo.direcao>0 && uinfo.direcao<Math.PI/2){

              deltaXInt = reflexPoint.getX() - lastPoint.getX();
              deltaYInt = reflexPoint.getY() - lastPoint.getY();;
              uinfo.direcao =  2*Math.PI + Math.atan(deltaYInt/deltaXInt);
            }
            if(uinfo.direcao>Math.PI/2 && uinfo.direcao<3*Math.PI){
              deltaXInt = reflexPoint.getX() - lastPoint.getX();
              deltaYInt = reflexPoint.getY() - lastPoint.getY();;
              uinfo.direcao = Math.PI + Math.atan(deltaYInt/deltaXInt);
            }
          }
          else if(newX<0){
            lastPoint = new Location2D(0,(float)(a*0 + b));
            reflexPoint = new Location2D((float)(-1*newX),(float)newY);
            newX = -1*newX;
            //////new direction after reflection
            if(uinfo.direcao>Math.PI && uinfo.direcao<3*Math.PI/2){

              deltaXInt = reflexPoint.getX() - lastPoint.getX();
              deltaYInt = reflexPoint.getY() - lastPoint.getY();;
              uinfo.direcao = 2*Math.PI + Math.atan(deltaYInt/deltaXInt);
            }
            if(uinfo.direcao>Math.PI/2 && uinfo.direcao<Math.PI){

              deltaXInt = reflexPoint.getX() - lastPoint.getX();
              deltaYInt = reflexPoint.getY() - lastPoint.getY();;
              uinfo.direcao = Math.atan(deltaYInt/deltaXInt);
            }

          }else{
            lastPoint = new Location2D((float)X,(float)(a*X + b));
            reflexPoint = new Location2D((float)(2*X - newX),(float)newY);
            newX = 2*X - newX;
```

```
            //////new direction after reflection
            if(uinfo.direcao>0 && uinfo.direcao<Math.PI/2){

              deltaXInt = reflexPoint.getX() - lastPoint.getX();
              deltaYInt = reflexPoint.getY() - lastPoint.getY();;
              uinfo.direcao = Math.PI + Math.atan(deltaYInt/deltaXInt);
            }
            if(uinfo.direcao>3*Math.PI/2 && uinfo.direcao<2*Math.PI){

              deltaXInt = reflexPoint.getX() - lastPoint.getX();
              deltaYInt = reflexPoint.getY() - lastPoint.getY();;
              uinfo.direcao = Math.PI  + Math.atan(deltaYInt/deltaXInt);
            }
          }
        }
        uinfo.passosExec--;
        Location2D newLoc = new
Location2D((float)(newX+X_mov_rel),(float)(newY+Y_mov_rel));
        JistAPI.sleep((long)(uinfo.tempoPasso*Constants.SECOND));

        f.moveRadio(id,newLoc);
        if(uinfo.passosExec<=0){
          uinfo.renew(vMin,vMax,mu,passos);
        }
        return newLoc;
      }

    }
```

Apêndice D

Código da simulação das redes testadas

```
import guiTrace.JavisTrace;
import jist.swans.field.Field;
import jist.swans.field.Mobility;
import jist.swans.field.Placement;
import jist.swans.field.Fading;
import jist.swans.field.Spatial;
import jist.swans.field.PathLoss;
import jist.swans.radio.RadioNoise;
import jist.swans.radio.RadioNoiseImprovedIndep;
import jist.swans.radio.RadioInfo;
import jist.swans.mac.Mac802_11;
import jist.swans.mac.MacAddress;
import jist.swans.net.NetAddress;
import jist.swans.net.NetMessage;
import jist.swans.net.NetIp;
import jist.swans.net.PacketLoss;
import jist.swans.trans.TransUdp;
import jist.swans.route.RouteInterface;
import jist.swans.route.RouteAodv;
import jist.swans.misc.MessageBytes;
import jist.swans.misc.Util;
import jist.swans.misc.Mapper;
import jist.swans.misc.Location;
import jist.swans.misc.Message;
import jist.swans.Constants;

import jist.runtime.JistAPI;
import java.util.Vector;

public class aodv
{
  private static final int PORT = 3001;
  private static int protocol = Constants.NET_PROTOCOL_AODV;
  private static int nodes = 20;
  private static Location.Location2D field = new Location.Location2D(1500,
300);
  private static String mobilityString="waypoint";
  private static int mobilityModel = Constants.MOBILITY_WAYPOINT;
  private static String mobilityOpts = "0:1:1:20";
  private static String lossOpts = "0.2";
  private static double sendRate = 4*60.0;
  private static int startTime = 10;
```

```
  private static int duration = 900;
  private static int resolutionTime = 10;
  private static double limiteSNR = 10;

  public static void addNode( int i, Vector routers,
       Field field, Placement place, RadioInfo.RadioInfoShared radioInfo,
Mapper protMap,
       PacketLoss inLoss, PacketLoss outLoss)
  {
    RadioNoiseImprovedIndep r = new RadioNoiseImprovedIndep(i, radioInfo);
    r.setThresholdSNR(limiteSNR);
    RadioNoise radio =r;

    Mac802_11 mac = new Mac802_11(new MacAddress(i),
radio.getRadioInfo());

    final NetAddress address = new NetAddress(i);
    NetIp net = new NetIp(address, protMap, inLoss, outLoss,
field.getTrace());

    RouteInterface route = null;
    switch(protocol)
    {
    case Constants.NET_PROTOCOL_AODV:
      RouteAodv aodv = new RouteAodv(address);
      aodv.setNetEntity(net.getProxy());
      aodv.getProxy().start();
      route = aodv.getProxy();
      routers.add(aodv);
      break;
    default:
      throw new RuntimeException("invalid routing protocol");
    }

    TransUdp udp = new TransUdp();

    Location location = place.getNextLocation();
    field.addRadio(radio.getRadioInfo(), radio.getProxy(), location);
    field.startMobility(radio.getRadioInfo().getUnique().getID());

    radio.setFieldEntity(field.getProxy());
    radio.setMacEntity(mac.getProxy());
    byte intId = net.addInterface(mac.getProxy());
    net.setRouting(route);
    mac.setRadioEntity(radio.getProxy());
    mac.setNetEntity(net.getProxy(), intId);
    udp.setNetEntity(net.getProxy());
    net.setProtocolHandler(Constants.NET_PROTOCOL_UDP, udp.getProxy());
    net.setProtocolHandler(protocol, route);
  }

  private static Field buildField(final Vector routers)
  {
    Mobility mobility = null;
    switch(mobilityModel)
    {
    case Constants.MOBILITY_WAYPOINT:
```

```
        mobility = new Mobility.RandomWaypoint(field, mobilityOpts);
        break;
      case Constants.MOBILITY_UNIFORM_CIRCLE:
        mobility = new Mobility.UniformCircular(field, mobilityOpts,nodes);
        break;
      case Constants.MOBILITY_UNIFORM_RECT:
        mobility = new Mobility.Uniforme(field, mobilityOpts,nodes);
        break;
      case Constants.MOBILITY_UNIFORM_RECT_NOMANDE:
        mobility = new Mobility.GrupoUniforme(field, mobilityOpts);
        break;
      default:
        throw new RuntimeException("unknown node mobility model");
      }
      Spatial spatial = null;
      spatial = new Spatial.LinearList(field);


      Field field = new Field(spatial, new Fading.Rayleigh(), new
PathLoss.TwoRay(),
          mobility, Constants.PROPAGATION_LIMIT_DEFAULT);
      RadioInfo.RadioInfoShared radioInfo = RadioInfo.createShared(
          Constants.FREQUENCY_DEFAULT,
          Constants.BANDWIDTH_DEFAULT,
          Constants.TRANSMIT_DEFAULT,
          Constants.GAIN_DEFAULT,
          Util.fromDB(Constants.SENSITIVITY_DEFAULT),
          Util.fromDB(Constants.THRESHOLD_DEFAULT),
          Constants.TEMPERATURE_DEFAULT,
          Constants.TEMPERATURE_FACTOR_DEFAULT,
          Constants.AMBIENT_NOISE_DEFAULT);

      Mapper protMap = new Mapper(new int[] { Constants.NET_PROTOCOL_UDP,
protocol, });
      PacketLoss outLoss = new
PacketLoss.Uniform(Double.parseDouble(lossOpts));
      PacketLoss inLoss = new
PacketLoss.Uniform(Double.parseDouble(lossOpts));
      Placement place =  new Placement.Random(aodv.field);


   JavisTrace.createTraceSetTrace(field,"aodvsim_"+nodes+"_"+mobilityString+
"Snr_"+limiteSNR+"_NodeSim");
      for (int i=1; i<=nodes; i++)
      {
        addNode( i, routers,  field, place, radioInfo, protMap, inLoss,
outLoss);
      }

      JavisTrace.drawGuiTrace(field);

      JistAPI.sleep(startTime*Constants.SECOND);
      int numTotalMessages = (int)Math.floor(((double)sendRate/60) * nodes *
duration);
      long delayInterval = (long)Math.ceil((double)duration *
(double)Constants.SECOND / (double)numTotalMessages);
      for(int i=0; i<numTotalMessages; i++)
```

```
    {
      int srcIdx = Constants.random.nextInt(routers.size());
      int destIdx;
      do
      {
        destIdx = Constants.random.nextInt(routers.size());
      } while (destIdx == srcIdx);
      RouteAodv srcAodv = (RouteAodv)routers.elementAt(srcIdx);
      RouteAodv destAodv = (RouteAodv)routers.elementAt(destIdx);

      Message m = new MessageBytes(new byte[64]);

      TransUdp.UdpMessage udpMsg = new TransUdp.UdpMessage(PORT, PORT, m);

      NetMessage msg = new NetMessage.Ip(udpMsg, srcAodv.getLocalAddr(),
destAodv.getLocalAddr(),
          Constants.NET_PROTOCOL_UDP, Constants.NET_PRIORITY_NORMAL,
Constants.TTL_DEFAULT);
      srcAodv.getProxy().send(msg);
      JistAPI.sleep(delayInterval);
    }

    return field;
  }

  public static void main(String[] args)
  {

     long endTime = startTime+duration+resolutionTime;
       if(endTime>0)
       {
         JistAPI.endAt(endTime*Constants.SECOND);
       }
    final Vector routers = new Vector();
    buildField(routers);

    JistAPI.runAt(new Runnable()
    {
      public void run()
      {
        long filewait = 15;
        try {
          Thread.sleep(filewait);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }

      }
    }, JistAPI.END);
  }
}
```

Apêndice E

Programa para extração das métricas

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <sstream>
#include <cstring>
#include <vector>
#include <iomanip>
#include <time.h>
using namespace std;
const string namefile = "aodvsim_output_jist_swans.nam";

double AVG_Lantencia_Aquisicao_Rota(){

    vector<double>  resultados;
    ifstream file;
    file.open(namefile.c_str());
    char str[2000];
    for(int i=0;i<4;i++)
        file.getline(str,2000);

    while(!file.eof())
    {
        file.getline(str,2000);
        double timeStart,timeEnd,delay,srcSend,dstSend,srcRec,dstRec;
        char * tokens1,*tokens2,*pktSend,*pktRec;
        tokens1 = strtok (str," ");
        if(tokens1[0]=='+'){

            tokens1 = strtok (NULL, " "); // -t
            tokens1 = strtok (NULL, " "); // <time>
            timeStart = atof(tokens1);
            tokens1 = strtok (NULL, " "); // -s
            tokens1 = strtok (NULL, " "); // <src>
            srcSend = atof(tokens1);
            tokens1 = strtok (NULL, " "); // -d
            tokens1 = strtok (NULL, " "); // <dst>
            dstSend = atof(tokens1);
            tokens1 = strtok (NULL, " "); // -p
            tokens1 = strtok (NULL, " "); // <packet_type>
            pktSend = tokens1;
```

```
            if (dstSend==-1 && strcmp (pktSend,"aodv") == 0){ //é route
request? pois -1 é broadcast.



                while(!file.eof())
                {
                    file.getline(str,2000);
                    tokens2 = strtok (str," ");
                    if(tokens2[0]=='r') // é um evento de receive?
                    {

                        tokens2 = strtok (NULL, " ");  // -t
                        tokens2 = strtok (NULL, " "); // <time>
                        timeEnd = atof(tokens2);
                        tokens2 = strtok (NULL, " "); // -s
                        tokens2 = strtok (NULL, " "); // <src>
                        srcRec = atof(tokens2);
                        tokens2 = strtok (NULL, " "); // -d
                        tokens2 = strtok (NULL, " "); // <dst>
                        dstRec = atof(tokens2);
                        tokens2 = strtok (NULL, " "); // -p
                        tokens2 = strtok (NULL, " "); // <packet_type>
                        pktRec = tokens2;
                        if (srcSend==dstRec && strcmp (pktRec,"aodv") ==
0){ // é o route replie corresponete a requisao?
                            delay = timeEnd-timeStart;
                            resultados.push_back(delay);
                            break;
                        }
                    }
                }

            }

        }
    }
  file.close();

    vector<double>::iterator iterator_vector;
    double sum_delay=0;
    double n = resultados.size();
    for(int i = 0;i<n;i++)
        sum_delay+= resultados[i];

    return sum_delay/n;
}

double Total_AODV_Packets_Send(){

    double  sum=0;
    ifstream file;
    file.open(namefile.c_str());
    char str[2000];
    for(int i=0;i<4;i++)
        file.getline(str,2000);
```

```
    while(!file.eof())
    {
        file.getline(str,2000);
        double timeStart,timeEnd,delay,srcSend,dstSend,srcRec,dstRec;
        char * tokens1,*tokens2,*pktSend,*pktRec;
        tokens1 = strtok (str," ");
        if(tokens1[0]=='+'){

            tokens1 = strtok (NULL, " "); // -t
            tokens1 = strtok (NULL, " "); // <time>
            timeStart = atof(tokens1);
            tokens1 = strtok (NULL, " "); // -s
            tokens1 = strtok (NULL, " "); // <src>
            srcSend = atof(tokens1);
            tokens1 = strtok (NULL, " "); // -d
            tokens1 = strtok (NULL, " "); // <dst>
            dstSend = atof(tokens1);
            tokens1 = strtok (NULL, " "); // -p
            tokens1 = strtok (NULL, " "); // <packet_type>
            pktSend = tokens1;


            if (strcmp (pktSend,"aodv") == 0) //é aodv packet.
                sum++;
        }
    }
    file.close();

    return sum;
}

double Total_Mensagens_Recebidas(){
double  sum=0;
    ifstream file;
    file.open(namefile.c_str());
    char str[2000];
    for(int i=0;i<4;i++)
        file.getline(str,2000);

    while(!file.eof())
    {
        file.getline(str,2000);
        double timeStart,timeEnd,delay,srcSend,dstSend,srcRec,dstRec;
        char * tokens1,*tokens2,*pktSend,*pktRec;
        tokens1 = strtok (str," ");
        if(tokens1[0]=='r'){

            tokens1 = strtok (NULL, " "); // -t
            tokens1 = strtok (NULL, " "); // <time>
            timeStart = atof(tokens1);
            tokens1 = strtok (NULL, " "); // -s
            tokens1 = strtok (NULL, " "); // <src>
            srcSend = atof(tokens1);
            tokens1 = strtok (NULL, " "); // -d
            tokens1 = strtok (NULL, " "); // <dst>
            dstSend = atof(tokens1);
            tokens1 = strtok (NULL, " "); // -p
```

```
            tokens1 = strtok (NULL, " "); // <packet_type>
            pktSend = tokens1;


            if (strcmp (pktSend,"udp") == 0) //é aodv packet.
                sum++;
        }
    }
    file.close();

    return sum;

}

int main(int argc, const char *argv[])
{
    ofstream output;
    output.open("output");
    double atraso=AVG_Lantencia_Aquisicao_Rota();
    double aodv =Total_AODV_Packets_Send();
    double mensagens=Total_Mensagens_Recebidas();
    cout << "Atraso médio de Descoberta de Rota: " << atraso << endl;
    output << atraso<< endl;
    cout << "Total de Pacotes AODV gerados: " <<  aodv << endl;
    output<< aodv<< endl;
    cout << "Total de Mensagens Entregues: " << mensagens << endl;
    output << mensagens<< endl;
    output.close();


  return 0;
}
```