Disc. Scientia. Série: Ciências Naturais e Tecnológicas, S. Maria, v. 7, n. 1, p. 135-145, 2006. 137

ISSN 1981-2841

ANÁLISE COMPARATIVA DE DETECTORES DE FALHAS PARA REDES MÓVEIS[1]

A COMPARISON OF FAILURE DETECTORS DESIGNED TO MOBILE NETWORKS

Fábio Lorenzi da Silva, Giuliano Lopes Ferreira, Tiago Antonio Rizzetti, Raul Ceretta Nunes e Iara Augustin

RESUMO

Realiza-se, neste artigo, um estudo comparativo entre alguns protocolos de detecção de falhas global para redes móveis. Para tanto, analisaram-se três algoritmos de detecção de falhas global (*Gossip*, *Friedman e Hutle*) e um detector de falhas local (*Sridhar*). Para realizar o comparativo, utilizou-se o simulador JiST-SWANS e as métricas de QoS definidas para detectores de falhas. Como resultado, observou-se melhor desempenho do detector de falhas local.

Palavras-chave: detector de falhas, redes móveis, análise comparativa.

ABSTRACT

This paper compares failure detectors algorithms designed to mobile network. Three global view failure detectors (Gossip, Friedman and Hutle) and one local view failure detector (Sridhar) were analyzed. The comparison looks at specific failure detector QoS metrics and it was developed on JiST-WANS simulator. As result we had observed a better performance of the local view failure detector.

Keywords: *failure detector; mobile networks; comparative analysis.*

---

[1] Trabalho apresentado no Programa de Pós-Graduação em Informática – Centro de Tecnologia – Universidade Federal de Santa Maria (UFSM),
Caixa Postal 97105-900 – Santa Maria – RS – Brasil.

INTRODUÇÃO

Redes móveis *ad-hoc* são redes de dispositivos móveis que se formam espontaneamente. Os nós móveis, geralmente, possuem recursos limitados e se comunicam com outros nós no seu raio de alcance. Para melhor utilização de recursos, os nós podem encaminhar mensagens por meio de outros nós, até atingirem um destino fora de seu raio de transmissão. Além disso, os nós cooperam entre si, pois compartilham recursos e informações.

Um aspecto importante a ser considerado, nesse tipo de rede, é a tolerância a falhas. Um detector de defeitos é um bloco básico para a construção de mecanismos de tolerância à falhas em sistemas distribuídos. A partir de sua capacidade, vários algoritmos de detecção de defeitos para redes fixas são propostos ao longo dos anos e outros são adaptados para redes móveis *ad-hoc* (AGUILERA et al., 1997; RENESSE et al., 1998; FELBER et al., 1999).

O objetivo, neste trabalho, é avaliar três algoritimos globais de detecção de defeitos bastante conhecidos, adaptados ou desenvolvidos para redes móveis, e comparar seus desempenhos. Na avaliação, também foi utilizado um algorítimo local para introduzir o projeto piloto, a fim de avaliar a possibilidade dessa classe de algorítimos obter melhor desempenho em relação aos globais.

PROTOCOLOS UTILIZADOS

De acordo com o objetivo do presente trabalho, optou-se por simular somente protocolos baseados em mensagens "*I-am-alive*", por esse algoritmo ser bastante simples e ocasionar baixo *overhead* no sistema, uma vez que protocolos baseados em outras técnicas, como *cluster* (TAI et al., 2004), ocasionariam maior *overhead*, o qual deve ser evitado nas redes em que há limitação nos recursos, como processamento e armazenamento de energia.

GOSSIP

No protocolo de *gossip* básico (RENESSE et al., 1998), um nó envia novas informações para outros nós escolhidos aleatoriamente, combinando, dessa forma, parte da eficiência da disseminação hierárquica com parte da robustez do *flooding*[2].

Cada nó mantém uma lista que indica os nós conhecidos, seu endereço e

[2] Envio de pacotes em *broadcast*, em que cada nodo que recebe o pacote retransmite-o também com *broadcast.*

um número inteiro (contador *heartbeat*) que serve para detectar a falha. A cada instante de tempo T_{gossip}, cada nó incrementa seu próprio contador e seleciona (aleatoriamente) outro nó para enviar sua lista. Ao receber uma mensagem *gossip*, o nó combina sua lista com a lista recebida e marca com uma seta o contador de cada nó para o maior contador entre o de sua lista e o da lista recebida.

Os nós também mantêm, para cada nó na lista, o *timestamp* da última vez que o contador foi incrementado. Se o contador não foi incrementado por um tempo maior que T_{fail}, o nó é considerado falho. Quando isso ocorre, ele não pode ser imediatamente esquecido, pois nem todos os nós detectarão a falha ao mesmo tempo. Então, foi definido um tempo $T_{cleanup}$ ($T_{cleanup} \geq T_{fail}$) usado para determinar quando o nó falho pode ser removido da lista.

Ocasionalmente, cada nó faz *broadcast* de sua lista para tornar-se conhecido pela primeira vez e para recuperar-se de partições na rede. Esse protocolo detecta apenas nós completamente inalcançáveis e não detecta falha no *link* entre dois nós que continuam se comunicando com um terceiro nó em comum. Esse caso é deixado para o protocolo de roteamento resolver.

PROTOCOLO DE HUTLE

O protocolo desenvolvido por Hutle (2004) é um detector de falhas eventualmente perfeito para redes particionáveis esparsamente conectadas. Ele não necessita saber quantos nós estão na rede, mas somente o *jitter* (máximo atraso) na comunicação entre vizinhos diretos.

O algoritmo utiliza *heartbeats* para determinar se um nó está na mesma partição, além disso, segundo o autor, esse protocolo é eficiente em termos de complexidade de comunicação, pois ele reduz a frequência de encaminhamentos pela distância, o que torna a informação sobre vizinhos próximos mais precisa do que informações sobre vizinhos mais distantes e mantém o tamanho das mensagens constante.

Cada nó *p* da rede conserva uma tabela contendo informações sobre seus vizinhos conhecidos. A tabela cresce dinamicamente, inserindo-se a cada novo processo que *p* conheça. Nela existem as seguintes informações: (a) um contador de *heartbeats* que contém o *heartbeat* mais recentemente recebido de um nó *q*; (b) um contador de distância que possui a distância estimada até um nó *q* e (c) um *timestamp* que tem o último *round* em que *p* recebeu um *heartbeat* de *q*.

O protocolo compreende uma tarefa periódica, em que cada nó envia parte de sua tabela para seus vizinhos, no entanto, vale salientar que a tabela local

atualiza-se quando o nó recebe novos *heartbeats*. Se um nó não recebe um novo *heartbeat* de algum de seus vizinhos conhecidos, em um período de tempo suficientemente longo, ele suspeitará desse vizinho.

PROTOCOLO DE FRIEDMAN

O protocolo, proposto por Friedman e Tcharny (2005) é um detector de falhas, baseado no protocolo Gossip, para redes móveis *ad-hoc*. Nele, periodicamente, cada nó envia uma mensagem *heartbeat* para seus vizinhos conhecidos. Se um nó não receber um novo *heartbeat* de algum vizinho, ele não suspeitará imediatamente dele, mas, invés disso, o nó aguardará a omissão de x *heartbeats* antes de o considerar suspeito. Usa-se essa técnica para dar suporte à mobilidade do nó, o qual pode sair de seu raio de alcance por alguns instantes e depois voltar.

Além disso, os *heartbeats* são distribuídos pelo protocolo *Gossip*, para que outros nós possam detectar os nós que estão se movendo. Assim, quando um nó recebe uma mensagem com um vetor de *heartbeats*, ele verifica os *heartbeats* que são mais recentes do que os que ele possui e atualiza seu vetor. Dessa forma, um nó pode receber um *heartbeat* de um vizinho que saiu de seu raio de alcance, mas que continua trabalhando na rede.

PROTOCOLO DE SRIDHAR

O protocolo desenvolvido por Sridhar (2006) é um detector de falhas local, no qual um nó só pode atestar a falha de outro nó que seja seu vizinho. Não há conhecimento do estado global dos nós, ao contrário dos protocolos apresentados anteriormente, que são detectores globais. Outra diferença está no fato desse protocolo ter a capacidade de identificar quando o nó está falho e quando ele apenas se moveu para outra parte do ambiente, saindo do alcance de seus vizinhos e entrando em outro local.

O detector pode ser implementado em duas camadas: a primeira como um detector de falhas local e a segunda como um detector de mobilidade. O detector de falhas local pode ser qualquer um, o único requisito consiste no fato de que o detector precisa manter um *timestamp* para cada nó na lista de suspeitos, que indica o instante em que o nó foi suspeitado - essa informação é usada pela camada de mobilidade. Nesse trabalho, utilizou-se uma implementação do protocolo *heartbeat*, adequada às necessidades da camada de detecção de mobilidade. Esse protocolo é brevemente apresentado na subseção seguinte.

A camada de mobilidade é executada por cada nó para compartilhar sua visão de nós falhos com os outros nós da rede e para corrigir seu conjunto de nós falhos, baseando-se na visão dos outro nós. Assim, cada nó que teve algum vizinho identificado como falho pela camada de detecção de falhas poderá verificar com os outros nós da rede se o vizinho realmente falhou ou se apenas se moveu para outro local.

PROTOCOLO HEARTBEAT

A ideia básica em torno do detector de falhas *heartbeat*, apresentada por Aguilera et al. (1997), é que cada nó mantém um vetor de contadores e um contador para cada vizinho conhecido. Esse vetor é entregue às aplicações interessadas sem nenhum processamento ou interpretação. A função do detector é somente atualizar o contador. Para isso, cada nó, periodicamente, envia uma mensagem "*I-am-alive*" (heartbeat) através de *broadcast*. Dessa forma, cada nó que recebe essa mensagem incrementa o contador correspondente.

Nota-se que esse protocolo não utiliza *timeouts* para determinar se o nó falhou ou não, ele somente conta o número total de *heartbeats* recebidos dos vizinhos e entrega o vetor de contadores. Segundo o autor, ele foi desenvolvido para resolver o problema de comunicação em sistemas assíncronos de troca de mensagens, em que os nós podem falhar ou pode haver perda do *link* de comunicação.

AMBIENTE DE SIMULAÇÃO

Para simulação e avaliação dos protocolos, foi utilizado o *framework* JiST/SWANS devido as suas características de desempenho e ao fato de permitir a implementação dos protocolos em Java, não sendo necessário o uso de uma outra linguagem de simulação.

O JiST – *Java in Simulation Time* (BARR et al., 2004) é um sistema de simulação de eventos discretos de alto desempenho que roda sobre a máquina virtual Java (JVM) (SUN, 2007), além disso, executa as simulações de forma altamente otimizada, tanto em tempo de execução quanto em consumo de memória.

A arquitetura do JiST é composta de quatro componentes: *compiler* (compilador Java padrão), *bytecode rewriter*, *simulation kernel* e *virtual machine* (JVM padrão). O *rewriter* modifica as classes compiladas, a fim de fazer as medições e o *simulation kernel* executa sobre a JVM, dando suporte às modificações feitas pelo *rewriter*.

O SWANS – *scalable wireless ad hoc network simulator* (BARR et al., 2005) é um simulador construído sobre a plataforma JiST, componentes independentes que se combinam para formar uma rede sem fio completa (todos os

níveis de protocolos) ou uma rede de sensores configurável. A rede sem fio pode formar-se por meio de combinações de diversos protocolos implementados no simulador, ou pelo usuário. Suas funcionalidades são similares as do NS2 (*THE*, 2007) e *GloMoSim* (ZENG et al., 1998), porém o SWANS pode simular redes maiores do que outros e implementa uma estrutura de dados que torna a computação da propagação de sinal mais eficiente.

O modelo de simulação foi criado no *framework* JiST/SWANS, tendo um campo de 300x300 metros. Os nodos são inicialmente dispostos num modelo em grade com cinco colunas, de forma que cada nodo esteja no raio de alcance dos outros. As configurações do rádio de cada nó são iniciadas com os valores padrões do simulador, exceto a potência de transmissão, que foi marcada para 50 dBm. Todos os nodos possuem o mesmo rádio para transmissão de dados, o que varia é o ruído recebido com o sinal.

Os nodos utilizam um modelo de mobilidade chamado *Random Waypoint*, no qual o nodo escolhe uma posição aleatória, move-se até ela e aguarda um tempo pré-determinado antes de movimentar-se novamente. O tempo de espera foi setado para 15 segundos, mantendo um grau de mobilidade mediano. A velocidade de movimentação é escolhida aleatoriamente pelo nodo, dentro dos parâmetros de velocidade máxima e mínima, que, nesse caso, foram marcados como 0 m/s e 2 m/s, respectivamente.

Como objetiva-se analisar o comportamento dos algoritmos em uma rede móvel, optou-se por não inserir um modelo de falhas. Dessa forma, as falhas ocorrerão devido à movimentação dos nodos, os quais saem da vizinhança que pertencem. Essas falhas poderão resultar em uma detecção de mobilidade (falsa suspeita) quando o nó mover-se para outra vizinhança, em uma falha por particionamento, ou quando mover-se para fora do alcance de qualquer outro.

Cada protocolo foi simulado cinco vezes durante 30 minutos e todos apresentaram resultados muito semelhantes, às vezes, até iguais. Por isso, considerou-se desnecessário repetir a simulação. Os resultados obtidos serão discutidos na seção seguinte.

Três métricas de qualidade de serviço (QoS) para protocolos de detecção de falhas propostas por Chen et al. (2002) foram usadas:

- Tempo de Detecção (*Detection Time* - T_D): mede a velocidade de detecção do detector de falhas;
- Tempo de Duração de Falsas Suspeitas (*Mistake Duration* - T_M): mede o tempo que o detector leva para corrigir uma falsa suspeita;
- Tempo de Recorrência ao Erro (*Mistake Recurrence Time* - T_{MR}): mede o tempo entre duas falsas suspeitas.

RESULTADOS

Para efeito de comparação das quatro alternativas de protocolos de detecção de falhas, citadas anteriormente, considerou-se o desempenho destes, medindo as métricas de QoS. Os resultados apresentam-se a seguir:

TEMPO DE DETECÇÃO (T_D)

Essa métrica mede o tempo de detecção de falhas, ou seja, mede a velocidade dos algoritmos na detecção de nodos falhos. Dessa forma, quanto menor for o valor, melhor será o desempenho do algoritmo detector de falhas. Na figura 1, apresenta-se o comparativo de desempenho dos algoritmos em relação ao tempo de detecção.

Apartir da análise do gráfico, na figura 1, percebe-se que o protocolo que apresenta melhor desempenho é o detector de falhas local de Sridhar e que esse tempo de detecção não é influenciado fortemente pelo número de nodos, já que, mesmo com o aumento de nodos, o tempo de detecção não sofre uma alteração considerável. Além disso, dentre os protocolos que são detectores de falhas globais, o de menor tempo é o de Hutle.

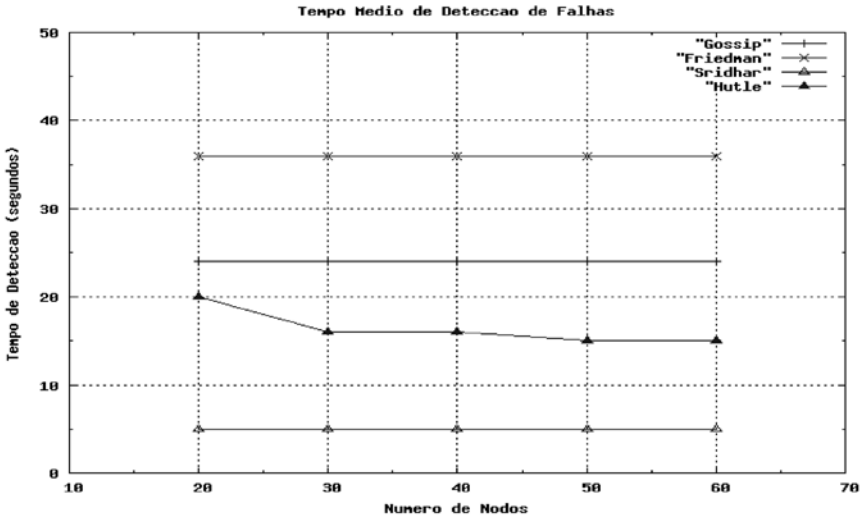

Figura 1 - Tempo médio de detecção de falhas.

TEMPO DE DURAÇÃO DE FALSAS SUSPEITAS (T_M)

Como essa métrica mede o tempo que o detector de falhas leva para corrigir um erro, quanto menor for esse valor, melhor e mais confiável é a alternativa para detecção de falhas. Na figura 2, mostra-se o comparativo de desempenho dos protocolos em relação à métrica.

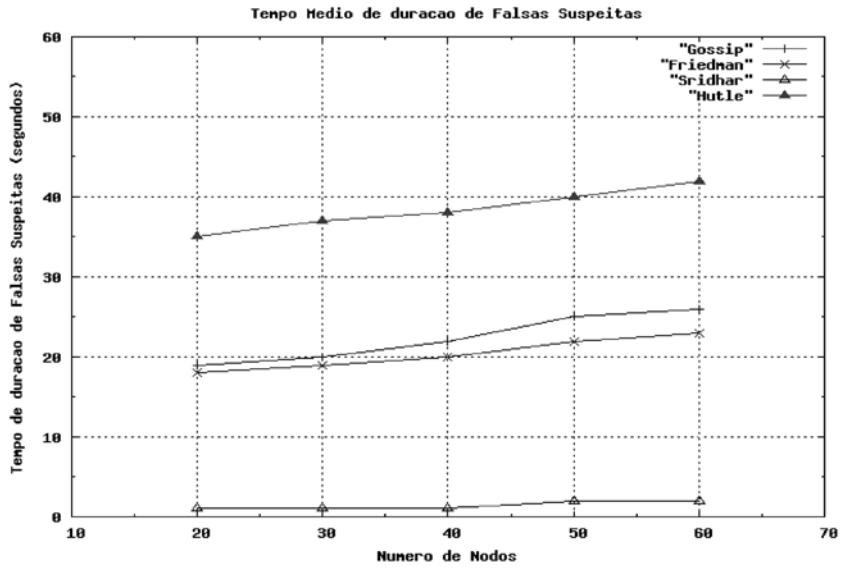

Figura 2 - Tempo médio de duração de falsas suspeitas.

Dentre os algoritmos apresentados e seus desempenhos demonstrados, conclui-se que o melhor desempenho é apresentado pelo protocolo de Sridhar e que, em todos os protocolos, o número de nodos afeta o tempo de duração de falsas suspeitas (com o aumento do número de nodos, aumenta-se o T_M). Pela figura, percebe-se também que analisar os protocolos detectores de falhas globais, excluindo o de Sridhar, o qual é local, não altera o resultado, pois quem apresentou melhor resultado foi o protocolo de Friedman.

TEMPO DE RECORRÊNCIA AO ERRO (T_{MR})

Essa métrica mede o tempo entre duas falsas suspeitas e quanto maior for esse valor mais preciso é o protocolo. Na figura 3 é apresentado o comparativo dos quatro protocolos.

A partir da análise do gráfico de desempenho, ilustrado na figura que segue, percebe-se que o protocolo detector de falhas global de Hutle é o que apresenta maior TMR, sendo, então, o mais eficiente. Ainda, avalia-se que a variação da quantidade de nodos pode afetar o tempo de recorrência ao erro, como se observa na curva do algoritmo de Sridhar.

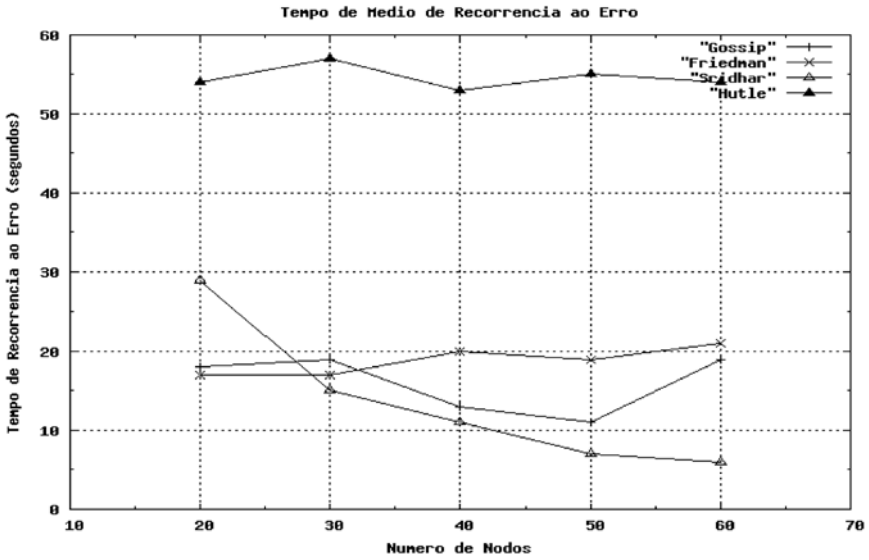

Figura 3 - Tempo médio de recorrência ao erro

CONCLUSÃO

O fato de existirem diversos algoritmos para detecção de falhas mostra a importância desse procedimento, em especial, nas redes *ad-hoc*. Pela análise comparativa realizada neste artigo, conclui-se que os algoritmos de detecção de falhas locais tendem a possuir um desempenho superior quando comparados aos algoritmos de detecção de falhas globais. Esse fato deve-se a menor necessidade de troca de informações nos algoritmos de detecção local, visto que eles mantêm apenas o estado de seus vizinhos e não uma visão geral sobre o sistema como um todo.

Dentre os algoritmos simulados e analisados, o que apresentou o melhor resultado foi o algoritmo de Sridhar, possuindo um desempenho geral significativamente melhor que os outros três algoritmos de detecção de falhas global. Como trabalho futuro, seria relevante avaliar o impacto de diversos modelos de mobilidade, variando o tempo de espera e a velocidade de movimentação. Além disso, pretende-se analisar outros protocolos locais de detecção falhas.

REFERÊNCIAS

AGUILERA, M. K.; CHEN, W.; TOUEG, S. **Heartbeat**: A timeout-free ailure detector for quiescent reliable communication. In: 11TH INTERNATIONAL WORKSHOP ON DISTRIBUTED ALGORITHMS, p. 126-140, Saarbrücken, 1997.

BARR, R.; HAAS, Z. J.; RENESSE, R. V. **Scalable wireless ad hoc network simulation.** In: HANDBOOK ON THEORETICAL AND ALGORITHMIC ASPECTS OF SENSOR, AD HOC WIRELESS, AND PEER-TO-PEER NETWORKS**,** p. 297-311, CRC Press, 2005.

BARR, R.; HASS, Z. J.; RENESSE, R. V. **JiST**: embedding simulation time into a virtual machine. In: PROCEEDINGS OF EUROSIM CONGRESS ON MODELLING AND SIMULATION. Paris, 2004.

CHEN, W.; TOUEG, S.; AGUILERA, M. K. On the quality of service of failure detectors. **IEEE Transactions on Computers**, v. 51, n. 5, p. 561-580, 2002.

FELBER, P. et al. **Failure detectors as first class objects**. In: INTERNATIONAL SYMPOSIUM ON DISTRIBUTED OBJECTS AND APPLICATIONS, p. 132-141, Edinburgh: IEEE Computer Society, 1999.

FRIEDMAN, R.; TCHARNY, G. **Evaluating failure detection in mobile ad-hoc networks.** In: INTERNATIONAL JOURNAL OF WIRELESS AND MOBILE COMPUTING, 2005.

HUTLE, M. **An efficient failure detector for sparsely connected networks.** In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING AND NETWORKS, p. 369-374, Innsbruck, 2004.

RENESSE, R. V.; MINSKY, Y.; HAYDEN, H. **A Gossip-Based Failure Detection Service**. In: PROCEEDINGS OF THE IFIP INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS AND OPEN DISTRIBUTED PROCESSING, p. 55-70, England: Springer, 1998.

SRIDHAR, N. **Decentralized Local Failure Detection in Dynamic Distributed Systems**. **In:** PROCEEDINGS OF THE 25TH IEEE SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS *(SRDS'06)*, p. 143 - 154, Washington: IEEE Computer Society, 2006.

SUN. Java Technology, 2007. Disponível em http://java.sun.com/

TAI, A. T.; TSO, K. S.; SANDERS, W. H. **Cluster-based failure detection service for large-scale ad hoc wireless network applications.** In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, p. 805 - 814, IEEE Computer Society Press, 2004.

THE Network Simulator, 2007. Disponível em: http://nsnam.isi.edu/nsnam/index.php/.

ZENG, X.; BAGRODIA, R.; GERLA, M. **GloMoSim**: a library for parallel simulation of large-scale wireless networks. In: PROCEEDINGS OF THE TWELFTH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION. p. 154 - 161, 1998.