Claudemberg Ferreira dos Santos
Orientador:Antônio Alfredo Ferreira Loureiro
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
{ferreira,loureiro}@dcc.ufmg.br

Uma Metodologia para Verificação Formal de Protocolos de Roteamento para Redes Móveis *Ad Hoc*

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte

6 de abril de 2004

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

Uma Metodologia para Verificação Formal de Protocolos de Roteamento para Rede Móveis Ad Hoc

CLAUDEMBERG FERREIRA DOS SANTOS

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. ANTÔNIO ALFREDO FERREIRA LOUREIRO
Departamento de Ciência da Computação - UFMG

Prof. CARLOS DE CASTRO GOULART
Departamento de Informática - UFV

Prof. CLAUDIONOR JOSÉ NUNES COELHO JÚNIOR
Departamento de Ciência da Computação - UFMG

DANIEL CÂMARA
Aluno de doutorado da Universidade de Maryland - EUA

Belo Horizonte, 07 de novembro de 2003.

Resumo

Uma rede móvel ad hoc é uma rede onde os nodos são móveis e podem se comunicar diretamente entre si. Isto significa que em cada instante do tempo a rede pode ter uma topologia diferente. Neste tipo de rede o protocolo de roteamento oferece um serviço muito importante para os protocolos das camadas superiores que é a entrega de mensagens. Atualmente, existem vários protocolos publicados na literatura para esse tipo de rede mas que não têm sido verificados através de um método formal, o que não garante efetivamente a correção.

Este trabalho propõe uma metodologia inovadora para verificar protocolos de roteamento para redes móveis ad hoc. A eficiência da solução proposta foi avaliada aplicando a metodologia a três protocolos, LAR1, LAR2 e DREAM, e identificando efetivamente problemas já conhecidos na literatura, outros ainda não relatados e restrições não mencionadas nos trabalhos publicados.

Abstract

A mobile ad hoc network (MANET) is comprised of mobile nodes that can communicate directly with each other, leading to different topologies along the time. In this kind of network, the routing protocol plays an important role since it offers the delivery of message to the upper layers. Currently there are several protocols published in the literature for MANETs that are not formally verified and, therefore, it is not possible to guarantee their correctness.

This dissertation presents a novel methodology for verifying routing protocols for MANETs. The soluting proposed is applied to three well-known protocols (LAR1, LAR2 and DREAM). As result, we were able to identify previous known errors published in the literature, new ones and restrictions not mentioned in the original papers for all three protocols.

Agradecimentos

Esta certamente é a mais difícil parte do trabalho. Evidentemente, não pelas dificuldades técnicas, mas sim pelo receio de deixar alguma estrela fora desta constelação. São tantas luzes que recebi durante esta jornada, que não posso me furtar a não mencionar alguém. Reconhecer a contribuição, mesmo que recebida de forma indireta, talvez seja um dos grandes aprendizados desta vida.

É muito interessante deixar as "coisas" passarem a nossa frente. Fluindo como uma estrela cadente, deixando seu rastro e luz por onde passa, mesmo que apenas poucos vejam tamanha beleza. O primeiro idílio é a memória. Buscar nas profundezas de nós mesmos, aquilo e aqueles poucos que se deseja citar por este ou aquele motivo. "Lembrar de fatos e pessoas que conheci, que já não se lembram mais de mim. Lembrar dos planos traçados e de seus percauços. Da nossa memória majestosa e ingrata. Lembrar de angústias e sorrisos. Medos e lágrimas. Pedras e Tijolos de ouro, ou de outro caminho. Caminhos de vivos ou de mortos, vivos em você. Memória de agradecimentos e quimeras." (Lauanda Lauar)

Para aqueles que acreditam que este plano/mundo/vida é apenas uma pequena passagem, entre os muitos caminhos percorridos, assim como eu acredito, talvez entendam que a escada é longa a ser percorrida. Vários serão os degraus que subiremos. Talvez entre aqueles do topo estejam os degraus do perdão, da gratidão, da redenção, sinceros. Por isso, este capítulo seja tão difícil. Eu ainda estou nos primeiros degraus e já quero ter as graças de quem percorreu tantos outros. Perceber que conquista não são importantes senão partilhadas. Perceber que conquistas não são puras senão doadas. Perceber que conquistas não são conquistas senão agradecidas. Este é o "meu quinhão". Esta mensagem tento passar, neste capítulo, para os vários nodos da rede.

Aos que me conhecem mais intimamente, certamente entenderão o que tento dizer. Aqueles que me conhecem superfialmente talvez achem um tanto inexperado este capítulo. Para mim, apenas espero que o capítulo seja *broadcasted* a ambos, principalmente aos últimos. Alguns talvez até se surpreendam por estarem citados. Mas aí está a beleza da gratidão. Ver o que, às vezes, pode parecer imperceptível. Agradecer, com sinceridade, a aqueles que não conseguem ver o que fizeram é ter aprendido a diferenciar um *oi* de um *olá*. Para muitos o ar é o mesmo que porcentagens de oxigênio, nitrogênio, etc. Para outros o ar é o combustível da vida. Apenas, espero, sinceramente, que algumas pessoas aqui presentes

não se preocupem com a ordem pelas quais estas linhas foram escritas. Ordenar sempre foi um problema clássico a ser resolvido em computação. E a melhor ordenação ainda hoje depende do contexto (sem querer entrar no mérito da questão). Por isso, por favor, não se apeguem ao algoritmo usado e tampouco à ordenação em si, mas sim, lembrem dos dados de entrada, a porção mais importante do processo: vocês.

Agradecer a Deus/Buda/Alá/Jesus/Tupã/Sol ou qualquer que seja a alusão feita ao Criador/Todo Poderoso é simplesmente um lugar-comum. Mas nem por isso, deve ser deixado de ser visitado. Existe um lugar-comum Maior ? Certamente, não. Por isso, agradeço a Ele por todas graças concedidas, mesmo aquelas mau interpretadas por mim, ou que até julgo não serem graças. Agradeço pelo amanhã, pelo hoje, mas principalmente pelo ontem. Por todos aqueles que cruzaram o meu caminho e me fizeram acontecer.

Sem sombra de dúvidas, a maior iluminação de toda a minha vida é a minha mãe. Sua dedicação, amor, caráter e respeito me fizeram ser a pessoa que sou. Para quem foi mãe e pai durante toda a minha vida, meu muito obrigado. Obrigado pela confiança, pelos ensinamentos, por me mostrar os caminhos, por me ensinar a ser melhor a cada dia, por sua incondicional dedicação e apoio, pelo colo, muito colo diga-se de passagem, pelos exemplos, por tudo. Muito obrigado "Mãinha". Obrigado por você ser quem é.

Este trabalho é realmente um marco muito importante para mim. Representa superação, coragem e união de muitos que conspiraram para que esta etapa fosse finalizada com sucesso e outros que nem tanto. O mais irônico disto é o fato de que atos ignóbios transformam-se em metas, inspirações, aversões, aprendizados e lutas, e que por isso mesmo engradeçeram ainda mais esta conquista.

Agradeço ao meu filho Kevin Caley e à minha esposa Lauanda. Vocês dois são o que de melhor conquistei na vida. São o meu motivo de inspiração, amor e motivação eternos. São meus pilares de apoio, alegria, amor, respeito, companherismo, conquistas, caráter, superação, compreensão, união, felicidade, luzes, paciência e muito choro, muito choro de nós três deve ser lembrado. Obrigado por compreenderem o "meu caminho", e saberem entender o quão é importante esta etapa para mim. Sem vocês realmente este trabalho não teria sido o que é. Talvez nem tivesse começado. Obrigado, obrigado, obrigado.

Aos meus orientadores e amigos Professor Antônio Alfredo Ferreira Loureiro, e Daniel Câmara, agradeço pelos valiosíssimos ensinamentos dentro e fora deste trabalho, pela confiança, pelas oportunidades, pela amizade, pelas conversas durante todo este tempo de convivência. Agradeço também às suas famílias pelos incontáveis incômodos em momentos não muito oportunos. As frases que mais me lembro dos dois é: "-As coisas mais importantes do mundo não são coisas." e "... irá acontecer *forever* , eternamente e para sempre.". Vocês dois certamente, me ensinaram muito mais do pesquisar. Obrigado.

Aos demais Membros da banca, os Professores Dr. Claudionor Nunes Coelho, Dr. Carlos Goulart, ilustres pesquisadores, que em muito enriqueceram este trabalho com im-

portantes colaborações. Obrigado.

Agradeço aos meus queridos irmãos, que não são poucos, talvez pela ausência da televisão nos idos anos, diga-se de passagem. Cada um deles me fez aprender um pouco durante este trabalho e durante minha vida. É interessante como os exemplos nos mostram o caminho de uma forma muito mais clara. E vocês me deram muitos exemplos, de amizade, de esforço, de alegria, de inocência, de humildade, de ajuda ao próximo e de perdão. Agradeço por vocês existirem e estarem sempre ao meu lado, me mostrando o caminho e as atitudes de uma forma ou de outra. Obrigado Ju, Duca, Lena, Vanê, Tau e Giza.

Aos meus sobrinhos e cunhados. Quantos são. Talvez sejam poucos ainda. Vocês me ajudaram demais do jeito de vocês. Obrigado, Lu, Bethe, Poly, Vanusa, Berguinho, Thiago, Tucs, Tutuzinho, Taynan, Alordia, Taiane, Anderson, Zu, Kimberly.

Agradeço ao Departamento de Ciência da Computação (DCC-UFMG) que me propiciou um grande oportunidade de aprendizado, crescimento profissional e pessoal. Agradeço pela enorme compreensão que todos tiveram comigo principalmente o professor Doutor Geraldo Robson de Mateus, aka, Robson, pessoa de estimável caráter, bom humor, enorme conhecimento e extremo bom-senso e paciência. Agradeço a todos os professores do Departamento que me mostraram o que é, e como aprender sobre computação, os quais foram fundamentais na minha formação, em especial ao professor Dr. Berthier Ribeiro. Agradeço a todos os laboratórios onde tive a oportunidade de trabalhar: VoD, ATM, LBD, SIAM e LAPO.

Agradeço à Instituição Banco do Brasil que sempre me propriciou, nestes últimos quatro anos, apoio, respeito, confiança, compreensão e amizade, uma grande oportunidade de aprendizado técnico, de forma que eu pudesse concluir este trabalho. Mas sobre tudo me proporcionou grande aprendizado de vida, num sentido de que uma empresa pode ser também parte de sua família. Isto foi me dito no primeiro dia de trabalho. Algo que só fui realmente entender "hoje". Muito Obrigado BB. Especialmente, eu gostaria de agradecer ao Mauro Santos Ribeiro, ao Paulo Pascotini, ao Marcelo Cavalcanti, ao Eliasibe Morais, à Cristina Maria, ao Marcelo Rovaroto, ao William, ao Renato Bracale, e ao Rodrigo. Gostaria de agradecer a todos do BB, com quem tive o prazer de trabalhar e muito me fizeram aprender, em especial ao pessoal da Infra e à equipe do Núcleo 12.

Um de meus grandes amigos sempre diz (não exatamente desta forma, talvez com algumas palavras a mais): "Amigos, aqueles de verdade, são de suprema importância e beleza. Pois, são as pessoas, a quem escolhemos como sendo os nossos "parceiros" eternos. Não nos foram impostos pela vida. Foram descobertos pelas pessoas que são." (Bicudo). Eu concordo totalmente em gênero, número, grau, cor, forma, volume, tempo, velocidade, massa, temperatura e pressão. Esse talvez seja o grande significado da palavra amigo. A escolha. O livre-arbítrio da vida. A aprendizagem e a beleza que conseguimos ver nos outros. Os amigos conseguem extrair o que há de melhor em cada um de nós. Por exemplo, os meus amigos conseguiram descobrir que eu sou capaz de beber uma lata de cerveja em no máximo 2 segundos. Não é demais! No processo deste trabalho também não foi diferente,

muito pelo contrário. Grande parte desta dissertação e de minha vida eu devo a vocês. Ordenação por apelidos:

- Bebão, vulgo Renato Salomão;
- Bicudo, vulgo Jailton Caldeira Júnior;
- Brunão, Vulgo Bruno Lauar;
- Cabeção-Tatu, Vulgo Alan Ferreira;
- Fofa, vulgo Lauanda;
- Ju, vulgo Julimar Ferreira
- Lazarento, vulgo Diogo Kurto;
- Loureiro, vulgo Antônio Alfredo
- Lu, vulgo Luciano Bertini;
- Manet, vulgo Daniel Câmara;
- Mau, vulgo Maruan;
- Rejes, Vulgo Rejane;
- Seba, Vulgo Sebastian Kummer;
- Su, Vulgo Suzane Pinto;
- Taz, vulgo Alexandre Paraguassu;
- Vi, Vulgo Viviam Bertini;

Além da brincadeira meus amigos, vocês são símbolo de muita, muita amizade, humildade, esforço e união. A vocês meus amigos, meu muito obrigado por tudo; por serem meus espelhos.

Agradeço em especial a Alan e Suzane por tudo que fizeram por mim, mas principalmente fizeram por Lau e Caleyzinho, nestes últimos meses. Sem vocês dificilmente este capítulo estaria sendo escrito. Espero que vocês recebam tudo o que merecem. Que a vida lhes dê em troca tudo o que distribuem, porque por mais que eu tente vai ser muito difícil eu conseguir retribuir a vocês. Muito obrigado.

Agradeço aos colegas da minha turma, vulgo *2000 e-mail.* Vocês me ajudaram em muito. Gostaria de agradecer especialmente ao Guilherme Rocha, vulgo Gui, e ao Foca, vulgo Rodrigo Vieira, ao Wellerson e ao Lúcio França(*Pos Mortem*). Agradeço à galera do Gedoc - Grupo de Estudos em redes móveis aD hOC -, em especial ao Frederico Mesquita. Agradeço à toda a galera-mau.

Ao meu pai, pela vida.

Normalmente agradecimentos são curtos. Espero ter conseguido dobrar minha memória em pequenos pedaços de forma a ter escrito este capítulo com serenidade e "justiça". Espero que as todas linhas escritas, não sejam em vão, não sejam jogadas ao vento, não sejam esquecidas em alguma parte longínqua ou obscura das incontáveis memórias que as lerem. Espero ter escrito algo útil pelo menos a poucos. Infelizmente não tenho certeza disto. Mas um certeza única certeza tenho, estas linhas, da primeira à última, não serão nunca esquecidas, nem tampouco abandonadas em uma caixa lacrada eternamente, por pelo menos um de nós aqui presente. Deus.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 A computação móvel

Nas últimas três décadas, houve um extraordinário crescimento nas tecnologias de computação e de comunicação sem fio [144]. Estamos vendo surgir, cada vez mais próximo, a possibilidade de, ao aliarmos todas essas tecnologias, termos um ambiente de trabalho e lazer independentemente da nossa localização. Desta forma, informações e recursos podem ser acessados e utilizados em qualquer lugar e a qualquer momento [104]. A este novo ramo de pesquisa, dá-se o nome de computação móvel.

A computação móvel é um novo paradigma computacional que tem como objetivo permitir aos usuários acessarem serviços independente de de sua localização. Dessa forma, a computação móvel amplia o conceito tradicional de computação distribuída. Isso é possível graças à comunicação sem fio que elimina a necessidade do usuário manter-se conectado à uma infra-estrutura fixa e, em geral, estática.

Hoje, sem dúvida, este está se tornando um dos novos e promissores paradigmas da computação. A mobilidade traz, além da comodidade, grandes possibilidades para a criação, diversificação e melhoria da qualidade de serviços oferecidos, pois estes antes necessitavam de dinamismo e não o possuíam. Tudo isto, agora é aliado à melhora significativa da produtividade. Porém, a mobilidade também introduz problemas e desafios que não existiam ou eram ignorados em ambientes fixos.

Agora, passando-se por uma mudança de paradigma, problemas já tidos como resolvidos em computação fixa, ou convencional, devem ser reavalidados. Muitas vezes, desde o

início, como é o caso do roteamento, que em ambientes móveis permanece praticamente em aberto. Questões como velocidade do canal, interferências do ambiente, localização da estação móvel, consumo de energia, alocação de faixas de freqência, e mesmo requisitos de QoS, são alguns dos problemas, importantes no projeto de algoritmos eficientes, que em redes móveis assumem maiores proporções do que em redes fixas.

1.2 Redes Móveis

Segundo o grupo de estudo 802.11 do IEEE (Institute of Electrical and Eletronics Engineers) [8, 52], os ambientes móveis, redes sem fio, podem ser divididos em dois grandes grupos, segundo a forma que proveêm o acesso: Estruturadas e *Ad hoc* ou não Estruturadas.

As Redes Móveis Estruturadas usam uma estação de suporte à mobilidade (ESM), que faz parte tanto da rede fixa, quanto da rede móvel, sendo que na sua interface com a rede fixa geralmente apresenta um link de alta velocidade. As estações móveis trocam mensagens com a ESM ou Estação Rádio Base (ERB), que se encarrega de entregar as mensagens aos destinos. Essa é basicamente a arquitetura utilizada em sistemas móveis celulares, como mostra a Figura 1.1[1].

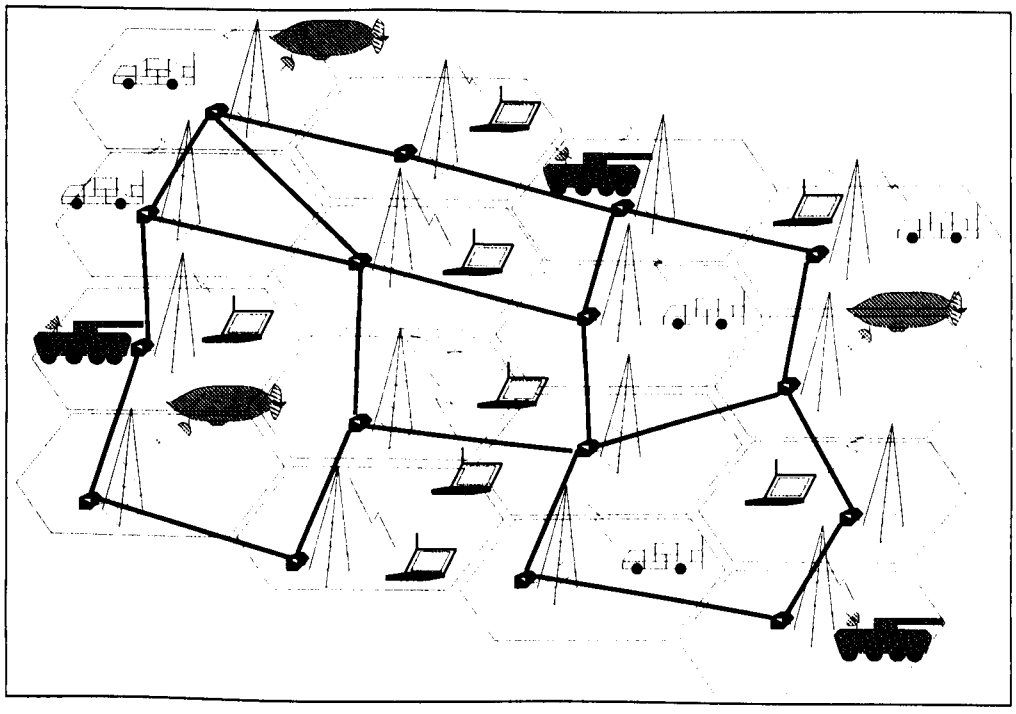

Figura 1.1: Forma de comunicação em redes móveis Estruturadas.

Como se pode ver pela Figura 1.1, nas redes estruturadas toda a comunicação entre duas entidades móveis é feita através da ESM, utilizando-se, portanto, da rede fixa existente. A ESM pode possuir uma ou mais antenas usadas na comunicação com as entidades móveis, podendo se comunicar tanto com os elementos da rede fixa quanto com as unidades móveis

[1] Algumas das figuras constantes nesta dissertação foram cedidas por Daniel Câmara [27]

em sua área de cobertura. Por outro lado, as entidades móveis só se comunicam, normalmente, com uma única ESM, a cada instante, via sinal de rádio (ondas eletromagnéticas de uma maneira geral). Como a ERB é fixa, isso faz com que a comunicação dependa diretamente da rede fixa, mesmo que os Hosts Móveis (HM) apresentem mobilidade.

O segundo tipo de rede móvel é a não estruturada, também chamada de rede móvel *Ad hoc* (MANET *Mobile Ad hoc NETwork*), onde os dispositivos computacionais são capazes de trocar informações diretamente entre si. Neste tipo de rede, todos os nodos podem se movimentar livremente e comunicar com qualquer outro nodo que esteja dentro de sua área de alcance. Como não há ERBs, os próprios nodos efetuam o trabalho de roteamento.

Em uma MANET uma rota entre dois computadores pode ser formada por vários *hops* através de um ou mais computadores na rede, como visto na Figura 1.2.

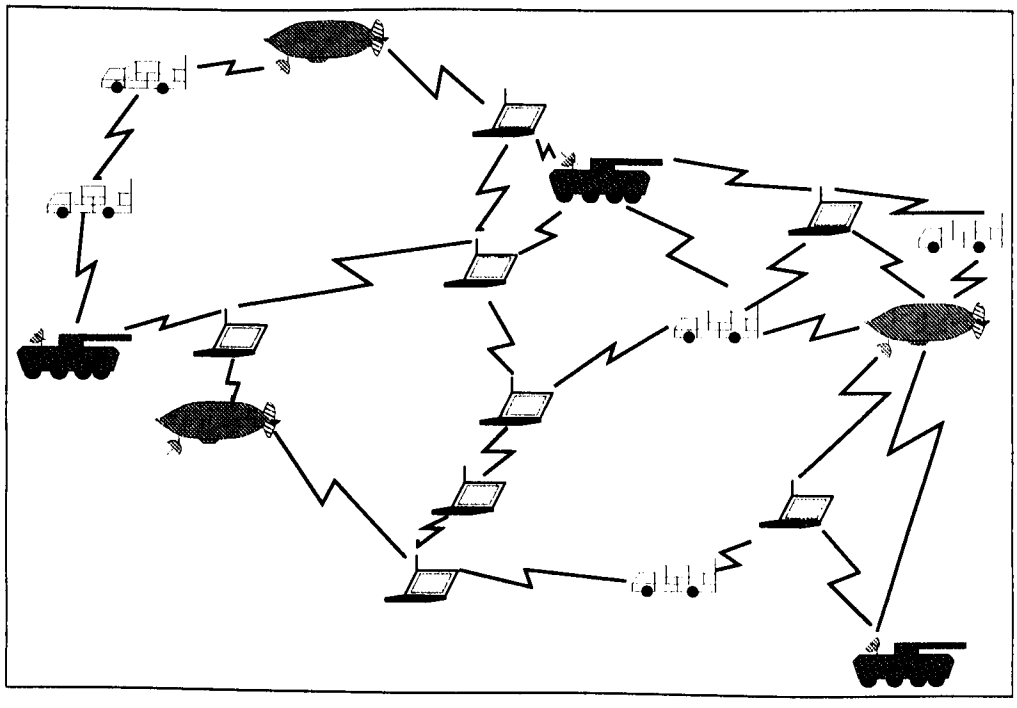

Figura 1.2: Forma de comunicação em redes móveis não Estruturadas.

Como cenários onde redes *Ad hoc* podem ser úteis podemos citar situações de desastre, como furacão, terremoto ou inundação, onde equipes de resgate precisam se coordenar, e não se tem uma rede fixa disponível. Soldados num campo de batalha trocando informações táticas, empresários compartilhando informações em reuniões ou mesmo estudantes usando laptops para participar de uma aula interativa são também exemplos de situações onde redes *Ad hoc* podem ser necessárias. Outra aplicação onde MANETs deverão ter no futuro um papel importante é na interconexão de *wearable computers* [17].

1.3 Roteamento

O desenvolvimento de procotocolos é algo bastante complexo. Quando se trata de procolos distribuídos esta complexidade aumenta ainda mai devido à presença de variáveis incontroláveis[2](Apêndice A.1) como o tempo, imcompletude de dados, entre outras. Em geral, para se obter uma caracterização confiável de um protocolo são utilizadas simuladores, prototipagem, métodos formais ou uma combinação entre eles.

Protocolos de roteamento estabelecem procedimentos para a descoberta e a manutenção de "caminhos" entre pontos distantes na rede. Esta tarefa é vital para o funcionamento da rede. Erros nos protocolos de roteamento podem gerar perdas de conexão constantemente que pode ser um processo extremamente custoso computacionalmente falando. Por este motivo, é importante verificar o protocolo, antes de que o utilizado, conhencendo, portanto, qual é o seu comportamento segundo as nossas espectativas. Esta análise normalmente é feita em uma das duas vertentes: desempenho ou corretude do algoritmo.

Simplificadamente pode-se dizer que análise de desempenho é a forma mais comum de se analisar protocolos de roteamento. Sua meta é "estressar" o protocolo numa rede real ou simulada, tentado ver qual é o seu comportamento prático levando-se em conta algumas condições iniciais que se deve assumir ou variar tais como o TTL - *Time to Live*(Apêndice A.3), o tamanho de cada PDU - *Protocol Data Unit*(Apêndice A.4), dentre outros. Análise de corretude já busca detectar os erros do algoritmo e o seu comportamento teórico em todos os casos possíveis. Portanto, uma metodologia ideal de validação de protocolos para redes móveis *ad hoc* deveria levar em conta estas duas abordagens, seja concomitantemente, seja em fases distintas do processo de análise.

Ao se pensar em roteamento, é natural termos em mente os modelos tradicionais, inerentes às redes fixas. Infelizmente, estes modelos, em geral, não se aplicam à realidade de ambientes móveis. Isso ocorre pelo simples fato de que não foram projetados levando em conta a mobilidade. Além disso, o endereço IP tradicional não pode ser usado no roteamento, haja vista que ele só pode ser associado a uma única rede [104]. Neste tipo de ambiente será assumido que cada computador possui um identificador único, mas que não referencie uma posição física do nodo assim como o IP. A idéia é apenas ter uma forma de identificação global, e sem redundância do nodo. Esse identificador do nodo não tem necessariamente nenhuma relação com a posição física do nodo. Este trabalho aborda de uma maneira específica, o roteamento em ambientes de computação móvel para MANETs.

[2]Algumas definições podem ser encontradas no Glosário.

1.3.1 Roteamento em MANETs

Um dos problemas fundamentais numa rede *Ad hoc* é determinar e manter as rotas, já que a mobilidade de um computador pode causar mudanças topologias e, consequentemente, alterar rotas entre os diversos *hosts*. Isso significa que um computador intermediário I que num determinado instante faz parte de uma rota entre dois *hosts* A e B, pode não fazer parte dessa mesma rota mais tarde. Vários algoritmos de roteamento para redes *Ad hoc* foram propostos na literatura [120]. Estes algoritmos diferem na forma em que novas rotas são determinadas e como as existentes são modificadas, quando necessário.

Do ponto de vista de arquitetura de redes do tipo *Ad hoc*, o protocolo de roteamento pode ser considerado o elemento principal. Sem um roteamento correto, os protocolos das camadas superiores não conseguirão oferecer os seus serviços efetivamente. Em geral, isso é verdade para qualquer tipo de rede, porém em redes *Ad hoc* esse é um problema crítico. Logo, é importante projetar protocolos de roteamento que sejam corretos em relação a certas propriedades como livres de *loops*.

O trabalho descrito aqui trata apenas sobre a análise de corretude para protocolos de roteamento para MANETs, o que representa um domínio específico de análise de corretude. Este domínio traz desafios específicos ao processo de modelagem e verificação tal como a mobilidade. Contudo, podemos aplicar técnicas não totalmente aplicáveis ao caso geral. Assim, alguns aspectos devem ser levados em conta:

- Todos os protocolos de roteamento são protocolos de n-nodos(*n-party protocol*), onde cada nodo executa uma cópia igual do mesmo processo. Conseqüentemente, o *state space*(Apêndice A.2) é ilimitado [16], mas simétrico em alguns pontos. Daí, as abstrações podem ser utilizadas.
- As computações de rotas envolvendo diferentes destinos não interferem entre si.
- Cada nodo tem a habilidade de se comunicar apenas com seus vizinhos imediatos.
- Cada nodo atua como roteador.

1.3.2 Análise dos Algoritmos de Roteamento

A IETF (Internet Engineering Task Force [57]) tem um grupo de trabalho chamado MANET (Mobile Ad hoc NETworks [75]) que é responsável por discutir os problemas referentes às redes *Ad hoc*. Foi publicada pelo grupo, em janeiro de 1999, a RFC 2501 Mobile

Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations [39], onde estão discutidas as principais características e problemas das redes móveis *Ad hoc*.

Segundo o grupo MANET, existem pontos que podem ser observados para avaliar, de forma quantitativa, um protocolo de roteamento para redes *Ad hoc* [39]. São eles:

- Vazão e atraso fim-a-fim dos pacotes de dados.
- Tempo para aquisição de uma rota, principalmente para algoritmos que trabalham sob demanda de rotas.
- Porcentagem de pacotes entregues fora de ordem.
- Eficiência do protocolo, que é o número de pacotes de controle necessários para que o protocolo funcione corretamente.

É importante também, no projeto do protocolo de roteamento para MANETs, que sejam levadas em consideração questões como o tamanho da rede, conectividade, capacidade do canal, mobilidade dos nodos e fração da rede que está em *doze mode* (modo pela qual o nodo fica dormindo, fazendo com que este gaste menos energia). Estes e outros pontos precisam ser observados para que se possa incrementar a escalabilidade do protocolo. Outra característica desejável, como em redes fixas, é que o protocolo tenha a capacidade de manipular e escolher a melhor rota baseado em parâmetros de QoS.

1.4 Contribuições

Projetar protocolos de comunicação para ambientes "confiáveis" normalmente não é uma tarefa difícil. Por ambiente confiável quer-se dizer um ambiente onde mensagens não são perdidas, duplicadas, alteradas ou cheguem fora de ordem. Infelizmente os ambientes onde os protocolos operam não garantem normalmente essas características. Isso faz com que o projeto de protocolos de comunicação seja uma tarefa complexa.

Para validar um protocolo podem ser utilizadas diferentes técnicas como métodos formais, simulação e testes. O objetivo dessas técnicas é identificar propriedades e/ou características presentes ou não no projeto de protocolos.

A forma ideal de provar a correção lógica da especificação de um protocolo de comunicação é aplicar algum método de verificação formal para determinar se propriedades desejáveis como *safety* e *liveness* estão presentes [76]. Este é um problema complexo e facilmente mostra-se que a verificação de uma propriedade trivial como ausência de *deadlock* (propriedade da classe *safety*) é um problema intratável computacionalmente, como será descrito na seção 2.2.1. Ao verificar a correção de protocolos de comunicação é importante que se conheça esses limites de complexidade. Isso significa que as diferentes técnicas de validação (verificação, simulação e teste) são complementares e devem ser usadas conjuntamente.

1.5 Objetivos do Trabalho

A verificação de protocolos de comunicação para redes tradicionais tem sido bastante estudada na literatura [76]. No entanto, a verificação de protocolos para redes móveis *Ad hoc* é ainda um tema de pesquisa pouco explorado. No melhor do nosso conhecimento este é o primeiro trabalho que apresenta uma metodologia para verificação formal de protocolos para redes móveis *Ad hoc*. A metodologia proposta permite verificar se algumas propriedades desejáveis em algoritmos de roteamento para MANETs estão presentes ou não. Neste trabalho, a técnica de modelagem de protocolos de roteamento para redes *Ad hoc* é aplicada aos algoritmos LAR1, LAR2 [93] e DREAM [7] usando Spin [79]. Os resultados obtidos mostrando os problemas encontrados nesses algoritmos são descritos juntamente com possíveis soluções.

Para que se possa verificar qualquer uma das propriedades listadas anteriormente, precisa-se, intuitivamente, modelar o algoritmo em uma dada topologia de rede. No entanto, se isto for feito, não se estará realmente verificando o algoritmo, já que fixamos uma topologia. Então é necessário deixar que a topologia varie arbitrariamente.

Ao deixar a topologia variar o problema torna-se intratável, pois o número de estados gerados internamente durante a verificação, transforma o tempo de execução em "infinito" computacionalmente, haja vista que à medida que se varia a topologia o número de combinações possíveis é absurdamente grande. Isto inviabiliza a verificação do sistema sem se fazer qualquer abstração. Vejamos um exemplo simplificado, para ilustrar a questão. Numa pequena rede de 5 nodos em uma região com $10X10$ posições possíveis, teríamos algo em torno de 750 Milhões de possibilidades de verificação (estados) diferentes, o que é claramente um número muito elevado de estados para um exemplo tão simples e longe do caso real. Maiores detalhes podem ser encontrados em na Sessão 2.2.1.

Desta forma, vemos claramente a inviabilidade desta abordagem, haja vista que ao se verificar problemas reais teremos certamente muito mais nodos e em um ambiente com muito mais posições. Com o intuito de sanar esse problema, foi desenvolvida a metodologia proposta no Capítulo 3.

É interessante notar que dentre os vários protocolos de roteamento para MANETs apresentados na literatura [27, 120, 129], no melhor do nosso conhecimento, nenhum deles apresenta uma verificação formal ampla do protocolo. Isso tem feito com que eventualmente outros trabalhos apontem problemas nesses algoritmos. Com a metodologia apresentada neste trabalho espera-se que novos protocolos possam ser verificados em relação a certas propriedades, aumentando assim a confiabilidade desses algoritmos.

1.6 Organização do Texto

O restante deste documento é estruturado da seguinte forma: Primeiro fazemos uma contextualização apresentando alguns trabalhos relacionados relevantes no Capítulo 2. Neste mesmo capítulo são discutidos alguns fundamentos relevantes ao trabalho. No Capítulo 3 é apresentada a metodologia proposta. Uma breve discussão sobre os protocolos utilizados como validadores da Metodologia é feita no Capítulo 4. Algumas aplicações da Metodologia sobre estes protocolos são mostradas no Capítulo 5. Finalmente, no Capítulo 6, são apresentadas as conclusões do trabalho e as perspectivas de extensão e continuação do mesmo.

Capítulo 2

Fundamentos e Trabalhos Relacionados

2.1 Fundamentos

O desenvolvimento de protocolos é algo bastante complexo. Quando se trata de protocolos distribuídos esta complexidade aumenta muito devido à presença de variáveis incontroláveis como o tempo, incompletude de dados, entre outras. Em geral, para se obter uma caracterização confiável de um protocolo são utilizados simuladores, prototipagem, métodos formais ou uma combinação entre eles.

2.1.1 Simuladores

O uso de simuladores é muito comum no desenvolvimento e validação de protocolos, uma vez que eles são ferramentas que comprimem tempo e espaço. A compressão do espaço ocorre à medida que se colocam lado-a-lado elementos que não estão fisicamente próximos. E a compressão do tempo acontece devido ao acompanhamento de um sistema por um período de tempo que normalmente é difícil de observar, como 5 ou 10 anos [161]. Esse poder de compressão do tempo nos permite saltar algumas etapas no processo de "verificação" (simulação) de um dado sistema. Infelizmente, simuladores não têm total garantia de que seus resultados são absolutamente corretos, pois há partes do sistema que não são, necessariamente, considerados durante o processo.

No entanto, antes mesmo se poder usar os simuladores o sistema necessita ser especificado segundo suas características [66]. Atualmente, especificações de um protocolo ou sistema são feitas utilizando-se linguagem natural ou algumas ferramentas como análise estruturada e diagramas de fluxos de dados. O problema com esses métodos é que não existe um método seguro de "provar" que ele traduz exatamente o que se tem em mente. Uma vez que uma especificação é algo muito subjetivo. E eles não ajudam nessa "prova" porque não possuem mecanismos que permitam ter certeza do que é representado, devido ao ser caráter um tanto informal.

Quando se escreve uma especificação de sistema, é necessário que a maneira de traduzir uma idéia seja a mais precisa possível, já que o executor desse sistema é um computador, um elemento "burro". Sem a exata tradução, ele não saberá diferenciar, por exemplo, palavras homónimas ou escritas erradamente naquele contexto, ao se utilizar a linguagem natural. Além disso, o cliente final, o implementador, pode também interpretar de forma não-correta o que se desejou representar.

2.1.2 Prototipagem

Prototipagem é uma técnica de se implementar o sistema de forma reduzida ou simplicada, eliminando-se partes desnecessárias ou inapropriadas para a primeira abordagem [115]. Busca-se validar partes do sistemas devido ao apelo comercial do futuro produto, e não garantir que estas partes funcionarão de forma correta ou eficiente. Normalmente, durante a criação do protótipo desconsidera-se importantes partes e interações entre elas, geralmente, com o intuito de mostrar uma viabilidade comercial do produto final. Portanto, esta técnica consiste da abstração do sistema ou modelo de forma prática.

Apesar de ser uma técnica interessante para fins práticos e comerciais, pois apresenta ao usuário final uma idéia de como o sistema se comportará, ou valida partes específicas do sistema junto a algum especialista, ela é bastante restrita para garantir a qualidade do trabalho final.

2.1.3 Métodos Formais

Os métodos formais possuem uma descrição sintática e semântica, proporcionando apenas uma única interpretação de qualquer "frase" escrita. Esta descrição, em linguagem matemática, nos permite implementar programas que interpretam uma especificação e dizem se ela está correta, ou não. Este processo é chamado de validação da especificação, e só é possível para elementos que tenham sintaxe e semântica formalmente definidas. Isto possibilita escrever uma especificação e "testá-la" antes de implementá-la, descobrindo os

problemas antes mesmo do fim da linha de produção do sistema [5, 31].

Métodos formais referem-se, na verdade, a uma variedade de técnicas de modelagem matemáticas que são aplicadas a projetos de sistemas de computação, tanto *hardware* quanto *software* [108]. Isto é, a técnica de métodos formais é a matemática, com cunho rigoroso, aplicada sobre sistemas de computação, e quando propriamente aplicados, podem servir em diversas áreas. Eles podem ser usados para especificar e modelar o comportamento de um dado sistema, e também para matematicamente verificar que o projeto e a implementação deste sistema satisfazem propriedades desejadas e/ou necessárias.

A especificação de um sistema consiste do uso de notações derivadas da lógica formal para descrever suposições sobre o universo no qual o sistema irá operar, requisitos necessários ao sistemas e sobretudo o projeto para que possamos encontrar tais requisitos. Quando se tem uma especificação clara e bem-definida, consegue-se, de uma forma mais simples, produzir um sistema eficiente e robusto. Mas, em geral, a sua aplicação completa e total sobre sistemas complexos e grandes é impraticável Assim, geralmente, aplicam-se os métodos formais apenas àquelas partes críticas, ganhando tempo de processamento e ao mesmo tempo aumentando a confiabilidade do sistema [159, 117, 68].

Uma parte substancial do *software* desenvolvido no mundo ainda é feito utilizando o método da tentativa e erro [40, 42, 18, 137, 43, 55]. Ou seja, senta-se na frente do micro (ou terminal) com uma vaga idéia do que se deseja fazer, escreve-se algum código e vê-se o resultado através de testes mal acabados. Evidentemente, esta forma de trabalho tem uma forte tendência ao fracasso. No entanto, como os serviços a serem prestados pelos programas são, na maioria das vezes, suficientemente simples, é possível que uma pessoa trabalhando desta forma e munida de suficiente tempo, dinheiro e paciência, acabe produzindo alguma coisa aceitável pelo usuário.

Este problema foi reconhecido há mais ou menos 30 anos. Por volta de 1970 surgiu a prova formal de programas associada à programação estruturada [122]. Os inúmeros defensores desta idéia afirmavam que os programas passariam a ser corretos por construção, tornando desnecessários os testes. Pena que, embora a proposta tenha um enorme valor, ela não resolve o problema. Programas desenvolvidos por pessoas de renome, provados estarem corretos, publicados em periódicos de renome, não resistiram a um simples teste. O interessante foi que, apesar dos frequentes fracassos e da dificuldade de se utilizar a prova formal em grandes sistemas, a comunidade acadêmica publicou vários artigos sobre artigos relatando a maravilha que são os métodos formais, usando exemplos fáceis de serem provados e raras vezes testando se os programas funcionavam de fato, o que mostra a grande dificuldade de se aplicar métodos formais à vida real [32]. Felizmente, o uso de métodos formais em sistemas dos mais variados tipos vem crescendo gradativamente, como por exemplo, em sistemas críticos [132, 100, 21, 44, 22, 20, 19, 26, 35, 63, 116, 131], e hoje em dia já há boa porcentagem de uso no desenvolvimento de sistemas diversos, podendo ser considerado uma prática comun.

Apesar da dificuldade de aplicação prática, métodos formais têm se mostrado eficientes e seguros apresentando um sucesso relativo na especificação e verificação de sistemas computacionais bastante diversificados [6, 59]. Eles podem ser aplicados a diferentes projetos de sistemas tanto de hardware, quanto de software [70, 74, 90, 109, 148, 158]. Apesar de ainda não serem amplamente utilizados, sua presença é, na maioria das vezes, maior em projetos de hardware [33, 136, 51, 160]. Métodos formais podem ser usados para especificar e modelar o comportamento de um dado sistema. Deste modo, servem para descrever o sistema de uma forma não-ambígüa, proporcionando, ou mesmo garantindo, uma maior correção do sistema modelado [92, 96]. Além disso, podem ser usados para se efetuar a verificação matemática do projeto, e se sua implementação satisfaz propriedades desejadas ou necessárias. Não obstante, não há total garantia de que o sistema modelado seja perfeito [69]. Apesar disto, métodos formais geralmente apresentam bons resultados, pois devido ao seu caracter matemático, faz-se necessário um estudo aprofundado do sistema verificado. Isto provê a habilidade de se efetuar uma descrição, objetiva e não-redundante do sistema, o que não ocorre em descrição em linguagens naturais. Mais do que isso, métodos formais podem encontrar falhas, às vezes não encontradas com as simulações, graças a sua descrição clara e precisa do mundo real. No entanto, propriedades como confiabilidade, disponibilidade e várias outras são ainda uma grande incógnita no que tange à especificação formal [160, 74, 91]. Além de tudo isso, os métodos formais podem proporcionar a longo prazo uma redução dos custos do projeto, pois eliminam grande parte dos problemas encontrados durante a manutenção, devido a incompletudes ou excessos.

2.1.3.1 Tipos de Métodos Formais

Atualmente, há um bom número de pesquisadores estudando seriamente a verificação de protocolos de comunicação. A verificação de um protocolo tipicamente trata de propriedades bem-definidas tais como *safety*(Apêndice A.7), *liveness* (Apêndice A.8), ausência de *deadlock* (Apêndice A.9) ou *starvation*(Apêndice A.10) [3]

Em geral, os dois principais ramos para a verificação de protocolos são prova de teoremas (*Theorem Proving* - Apêndice A.11) e análise de alcançabilidade (*reachabily analysis* - Apêndice A.12), também conhecido coma *model checking* [72, 33]

Provadores de teorema, tais como HOL [114], definem um conjunto de axiomas e constrõem relações entre estes axiomas. As propriedades desejadas do protocolo são então provadas matematicamente. Prova de teorema inclui um conjunto de formalismos tanto baseados em lógica, quanto no modelo adotado, o qual embasados em lógica de primeira e segunda ordem [3].

Model Checking é um ramo da verificação formal que consiste de uma família de métodos de prova completamente automatizados que começaram a ser desenvolvidos na área acadêmica na década de 80. Verificação de modelos permite depurar um sistema de

maneira muito mais rápida e abrangente do que o teste e/ou simulação. Eles estão sendo usados para verificar sistemas não triviais, como protocolos de rede, sistemas de tempo-real, componentes de *hardware*, etc., achando erros de projeto que não são encontrados usando os métodos convencionais de verificação como provadores de teoremas. Hoje, vários centros de pesquisa privados já têm seus próprios verificadores de modelos. Ferramentas comerciais já estão sendo desenvolvidas e comercializadas, criando uma necessidade de profissionais qualificados no uso destas novas técnicas. Diante dos diversos estudos e dos resultados encontrados acredita-se que a verificação simbólica de modelos é a variante da verificação de modelos que provavelmente obteve mais sucesso até hoje [135]. Neste trabalho será utilizado exatamente esta variante.

Estas duas técnicas mostram um grande compromisso entre a expressividade e automação da verificação. Uma vez que verificadores de modelos são ferramentas completamente automáticas bastando ao usuário fornecer "apenas" o modelo e o requisito a ser verificado. O verificador fará o resto e dará uma resposta "sim" ou "não" indicando se a especificação satisfaz ou não ao requisito.

Por outro lado, provadores de teoremas não têm um alto grau de automação. Os *statements* de corretude são formulados como sendo teoremas. Partes da prova deseja até podem ser feitas de forma automatizada, desde que o usuário mostre ao teorema "dicas" de como prosseguir com a prova. Isto torna o processo totalmente dependende do usuário. Contudo, a grande vantagem de provadores de teorema vem do fato de que geralmente podemos expressar e provar resultados mais gerais. Além disso, eles não têm problemas com o espaço de estados gerados. Provadores de teoremas trabalham utilizando-se as táticas *Bottom-Up*(Apêndice A.20) ou *Top-Down*(Apêndice A.19). No primeiro *approach*, novos teoremas sã construídos a partir daqueles já provados utilizando-se de ténicas de generalização. No segundo, parte-se de alguma meta ou objetivo (teorema a ser provado) e tenta-se quebrá-la em teoremas mais simples ou prová-los completamente. O grande ponto deste *approach* recai no fato de que a cada nova interação pode-se utilizar uma nova tática de quebra ou prova dos teoremas gerados. O HOL [114] utiliza este *approach.*

2.1.3.2 O Problema da Explosão de Estados no *Model Checking*

Um modelo tem internamente um conjunto de variáveis definidas diretamente (canais de comunicação, *flags*, mensagens, etc) ou indiretamente (conjunto de processos ativos, contadores de programa, valores de registradores, etc) cujos valores mudam de acordo com regras definidas tanto pelo próprio modelo quando pelo verificador. Cada combinação dos valores destas variáveis definem um **estado do sistema**. O conjunto de todos os estados alcançáveis pelo sistema é chamado de espaço de estados (*Space State*) [128, 53, 54, 112, 15, 111]

Algoritmos de análise de alcançabilidade (*reachability analisys*) [53, 54] tentam gerar

e inspecionar todos os estados do protocolo alcançáveis a partir de um dado estado inicial definido pelo modelo. Tais algoritmos sofrem do problema da explosão de estados, especialmente em se tratando de protocolos complexos [128].

As formas mais utilizadas para se tentar contornar este problema ão por em prática técnicas de busca parcial controlada e redução de estados [124, 64]. Estas técnicas focam somente em partes de todo espaço de estados e podem usar procuras probabilísticas [105], aleatória [157] ou mesmo guiadas [118].

2.1.3.3 Ferramentas de Verificação Formal

Existem atualmente várias ferramentas e linguagens de verificação que utilizam a técnica de verificação de modelos (*model checking*) tais como: Spin [78], MultiKit [34], Verus [29], VIS [113, 126], SMV [95, 154], PVS [125] ou Murphi [139, 46].

Spin [78, 79] é um pacote de software distribuído gratuitamente na Internet que suporta verificação formal de sistemas distribuídos. Foi desenvolvido pelo grupo de métodos formais da Bell Labs e é amplamente usado por pesquisadores da área de verificação formal. O Spin utiliza-se da linguagem de especificação chamada PROMELA [78, 79], que é considerada de grande flexibilidade e de boa expressividade.

MultiKit, ou *Model-Checking Kit* [34], é um conjunto de programas que permite modelar um sistema de estados finitos, usando uma grande variedade de linguagens. A ferramenta permite também e verificar o sistema usando vários *checkers*, incluindo *deadlock-checkers, reachability-checkers, e model-checkers* para lógicas temporais CTL (*Constant Temporal Logic*) e LTL (*Linear Temporal Logic*) na Sessão 3.3.1. A mais interessante característica do Kit é a independência da linguagem de descrição escolhida, considerando-se todas aquelas que o MultiKit suporta. Quase todos os checkers podem ser aplicados ao mesmo modelo. Desta forma, os contra-exemplos produzidos pelo checker são apresentados ao usuário em termos da linguagem de descrição usada para modelar o sistema [34].

Verus [29] é uma linguagem projetada com o intuito de simplificar o processo de verificação de programas de tempo real. Esta ferramenta é baseada na técnica de BDDs (*Binary Decision Diagrams*) para comprimir os grafos gerados pelos inúmeros estados do sistema. Apesar de a ferramenta ainda estar em fase de aprimoração, bons resultados têm sido reportados na literatura [29].

PVS [125] é um sistema de especificação e verificação que consiste de uma linguagem de especificação integrada com ferramentas de suporte e um provador de teoremas. PVS apresenta o estado da arte em métodos formais mecanizados e é suficiemente robusto de forma a ser usado em aplicações reais e significantes. PVS inclui um procedimento de decisão baseado em BDDs de forma que integra de forma sinérgica um provador de

teoremas e um verificador de modelos utilizando-se de lógica CTL. No entanto, o PVS é um sistema complexo que leva bastante tempo para que se tenha um conhecimento razoável sobre sua utilização o que o tornou inapropriado aos objetivos deste trabalho.

O sistema SMV [154] é uma ferramenta para se checar sistemas de estados finitos modelando-se especificações em lógica temporal CTL. A linguagem de entrada do SMV foi projetada para permitir a descrição de Máquinas de Estado Finitos (FSMs - *Finite State Machines*) que variam desde sistemas completamente síncronos até completamente assíncronos, e de uma especifição detalhada até uma bem abstrata. Normalmente a especificação é feita através de uma máquina de Mealy síncrona ou por uma "rede" de processos não-determinísticos abstratos e assíncronos. SMV utiliza-se da técnica de OBDDs (*Binary Decision Diagrams*) para comprimir os grafos gerados pelos inúmeros estados do sistema utilizando-se a lógica CTL, e das técnicas (*Multiple Parallel Asssignments*) e (*Circular Assignments*) para a mesma variável. SMV é um provadr de teoremas integrado com *Model Checing*. SMV tem uma boa aceitação na literatura e é uma ferramenta relativamente simples de se utilizar, apesar de que se leva um bom tempo para ter uma prtica razovel do mesmo. A linguagem tem boa expressividade mas tem características próprias.

VIS (*Verification Interacting with Synthesis*) [113, 126] é uma ferramenta que integra a verificação, simulação e síntese de sistemas de *hardware* de estado-finito. Ele usa VERILOG [84, 41] como *front end* e suporta checagem de modelos com CTL, checagem de nulidade de linguagens, checagem de equilência seqüêncial e combinacional, simulação baseada em cíclos e síntese hierárquica [113]. A ferramenta suporta documentação automática em HTML, gerada a partir do código-fonte.

Murphi [139, 46] é um verificador de modelos e uma linguagem de descrição. Ele foi incialmente desenvolvido para ser utilizado em pesquisas com *hardware* tais como [119, 47]. No entanto, a versão atual também é apropriada para o uso em *software* [106, 140, 107]. Sua linguagem é baseada em uma coleção de comandos executados em um *loop* infinito. Estes comandos incluem estruturas comuns como Registros, Vetores, etc, e novos tipos como o "Multiset" (conjunto de valores cuja ordem não é relevante). Estes tipos dão boa expressividade e familiaridade à linguagem. O verificador formal é baseado em enumeração explícita de estados. O verificador executa procura em profundidade ou largura (*depth or breadth search*) no grafo definido pela descrição em Murphi [139, 46]. Murphi tem várias formas de reduzir o número de estados alcançáveis e ainda sim garantindo que erros no protocolo serão encontrados.

Murphi [139, 46] utiliza Simetria[1] [85, 37, 36], Regras Reversíveis (*Reversible Rules*)[2] [86],

[1] Em alguns casos, a descrição tem elementos idênticos que podem ser permutados consistentemente sem mudar quaisquer importantes propriedades da verificação. Isto é chamado de Simetria [139].

[2] Algumas regras de condições e/ou ações podem ser executadas em sentido contrário ("*in reverse*"). Murphi usa esta propriedade para reduzir os estados pois ele elimina os estados "progenitores" através das Regras Reversas [139].

Construtores de Repetição[3] [38], para reduzir os estados gerados. Além disso, ele utiliza técnicas algorítmicas de forma a explorar o espaço de estados de uma maneira mais eficiente. Um das técnicas utilizadas é a Verificação Probabilística[4] [152, 153, 141]. Outras técnicas interessantes utilizadas são o Paralelizmo [138], e o *Caching* [142, 151]

2.1.3.4 SPIN - A Ferramenta Escolhida

Avaliações teóricas e práticas de várias ferramentas de verificação formal pesquisadas que utilizam a técnica de *model checking* nos levaram a escolher a ferramenta Spin [79, 78] por diversas razões. Das quais destacaram-se:

- O *software* é um pacote distribuído gratuitamente na Internet [79]
- O *software* suporta a verificação tanto de sistemas distribuídos quanto de sistemas não-distribuídos.
- Sua linguagem de entrada, Promela, é uma linguagem *C-Like*, que apresenta vários tipos básicos, *arrays*, estruturas, que são suficientes quase sempre para se ter uma ótima expressividade, apesar de não existirem *timers*[5].
- Promela (*PROtocol MEta LAnguage*), é uma linguagem projetada para a especificação de protocolos de comunicação, como o próprio nome sugere
- A ferramenta é amplamente usada por diversos pesquisadores na área de verificação formal
- Existem vários casos [23, 69, 150, 79, 71, 16, 13, 112, 109] em que se utilizou a ferramenta e sua linguagem de entrada para verificar sistemas de grande e pequeno porte com um alto sucesso no resultado desejado
- A ferramenta utiliza-se de lógica temporal
- A ferramenta possui documentação disponível e simples

[3] Alguns protocolos, em situações específicas, podem ser verificados ao se abstrair o número exato de processos, o que pode reduzir drasticamente o número de estados o qual é necessário ser criado na verificação [139].

[4] Verificação Probabilística aceita uma pequena probabilidade de erros não-perceptíveis (*missed error*) utilizando-se de algoritmos de compressão de tabelas *hash*. Ainda assim, a estrutura interna utilizada garante um pequeno limite superior na probabilidadede se deixar escapar um erro [139].

[5] *Timers* aumentam a expressividade uma vez que permitem modelar as intereções sincronizadas e as relações de dependêndica entre partes dos sistemas/protocolos

- A partir de Promela pode-se utilizar outros sistemas para efetuar os mesmos testes utilizando-se outras ferramentas de forma integrada tal como em [34]
- Há primitivas internas que permitem a comunicação tanto de forma síncrona como assíncrona entre os processos.
- Como sendo uma ferramenta de *Model Checking* apresenta *"Full Automation"* (Apêndice A.13)
- A ferramenta utiliza de técnicas como BDDs para otimizar o processo de verificação.
- Ele suporta ambos passagem de mensagem via *buffer* ou *rendez-vous* e ainda comunicac{cão via memória compartilhada.

A ferramenta Spin tem como dado de entrada uma descrição modelo a ser verificado em Promela e um requisito (*requirement property*) em CTL/LTL ou em Promela. O Spin executa então uma pesquisa exaustiva sobre o conjunto de estados (*state space*) do modelo, enquanto procura por violações no requisito. Ao encontrar alguma violação do requeimento, a ferramenta aborta a verificação e apresenta o contra-exemplo. Caso não encontre, ela informará ao usuário que o modelo satisfaz ao requisito.

Promela pode somente especificar sistemas com um número fixo de processos, o que seria uma grande limitação na forma de verificação de algoritmos de roteamento para MANETs, caso não considerássemos a abstração dos três nodos apresentados neste trabalho como mostra o Captulo 3. Sem esta abstração o Spin não poderia ser usado para uma Verificação Total (*Full Verification*) dos protocolos de roteamento, pois estaria executando uma Verificação de Instância ou Parcial (*Instance Verification*), ou seja, a verificação do protocolo para um cenário específico (topologia e padrão de tráfego de mensagens pré-definidos fixos).

O grande limitante do *Model Checking* e conseqüentemente da ferramenta Spin no processo de verificação, a verificação de instância, foi contornada através da abstração utilizada pela metodologia. Assim, o poder de expressão é um problema a não ser considerado quando se utiliza a metodologia, sobre uma ferramenta de model checking, uma vez que o crescimento exponencial do Espaço de Estados (*state space*) foi contornado de forma elegante e eficaz.

2.1.3.5 SPIN - Exemplos e Expressividade

Esta sessão apresenta alguns exemplos da flexibilidade do Spin e a sua implementação em Promela.

Uma boa implementação dos possíveis estados dos pacotes é usar variáveis, e sempre que possível variáveis booleanas. Por exemplo, um pacote ao chegar em um nodo intermediário pode ter apresentado algum problema de transmissão no meio físico ou não.

```
if
   :: (true) -> CorruptedMessage = true
   :: (true) -> CorruptedMessage = false
fi;
```

Neste exemplo, utilizou-se a técnica da escolha aleatória. É evidente que o verificador, utilizará os dois braços do *if* durante o processo de verificação. Mas isto será feito em diferentes caminhos, o que permitirá, por exemplo, que se teste o que aconteça caso a mensagem seja corrompida. Neste caso específico, falhas no projeto interno poderiam ser detectadas.

Algoritmos de roteamento de uma forma geral apresentam mais de um tipo de pacote [83]. Todos os tipos devem ser verificados para garantir que o protocolo está funcionando de forma correta. Supondo que o protocolo possa ter 3 tipos de mensagem, descobrimento de rota, resposta de rota e pacotes de dados, uma possível implementação disto em Promela poderia ser:

```
if
   :: (MessageType == RouteRequest) -> run HandleRRq()
   :: (MessageType == DataPacket) -> run HandleDP()
   :: (MessageType == RouteReply) -> run HandleRRp()
   :: else -> CorruptedMessage = true ; HandleCM()
fi;
```

Neste exemplo, utilizou-se a técnica de seguir o fluxo correto e ter um tratamento de "erros". Isto é útil, por exemplo, quando se deseja exprimir um comportamento esperado na modelagem. Ao mesmo tempo, deve-se guardar os outros possíveis caminhos por onde poderia seguir a mensagem. Como em Promela os guardas (condições do *if*) não precisam ser mutuamente exclusivos, você deve ter cuidado ao defini-los, de forma a evitar que seja desconsiderada na modelagem uma parte importante do sistema.

Com isto todos os pacotes vão ser testados. Do mesmo modo, o nodo ao receber o pacote a cada instante pode ser a origem, o destino ou algum nodo intermediário, como mostrado abaixo:

```
if    /* Im dest. The message was processed succesfully */
    :: (destination == myid) -> RequestComplete = true
       /*Im origin of the message. Probably a loop*/
    ::  (origin == myid) ->
      /*
       * Verify the TTL in order to know whether I've got
       * a loop or not
       */
      if   /* If the TTL was changed, it is a loop*/
        :: (ttl != TTL) -> Loop = true
        :: else -> run HandleFalseLoop()
                  /*
                   * I've read the packet
                   * I've just written on the pipe
                   */
      fi
      /*I'm a intermidiate node.  Forward the packet*/
    :: else -> ...
```

Um dos pontos mais importantes da metodologia é a capacidade de simular qualquer configuração de rede com apenas três nodos. Esta característica é fundamental, pois ao se tentar verificar formalmente todas as possíveis configurações topológicas na rede encontraria-se um custo de verificação grande o suficiente para não ser mais possível dar continuidade ao processo. Esta idéia foi traduzida para Promela através do uso de canais únicos onde os pacotes são colocados e retirados pelos nodos de forma concorrente na rede.

```
/*
 * receiving the message
 */
medium?MessageType(origin,destination,ttl,priorid);

MessageType = "Type of the Message"
        ...
/*
 * Send a Request Response
 */
medium!MessageType(origin,destination,TTL,myid);
```

O meio de leitura é único para todos os processos que representam nodos. Logo, modelados como são procedimentos concorrentes e com formas de leitura/escrita padronizados, o acesso às mensagens seria concorrente. Daí, um nodo intermediário poderia ler a mensagem, processá-la segundo o algoritmo, e reenviá-la. Quando então, um outro nodo in-

termediário qualquer poderia repetir o processo. Como esse processo é aleatório poderiam existir n nodos intermediários, dependendo, logicamente da modelagem proposta.

Outro ponto marcante da metodologia é a possibilidade da implementação de *flooding* utilizando-se apenas dois pacotes simultaneamente na rede. Um *flooding* ou inundação em redes de médio e grande porte, como o descrito normalmente em redes de computadores, é inviável de ser verificado formalmente [56], devido à enorme quantidade de casos e configurações possíveis. Uma exceção seria um *flooding* em redes de pequeno porte, diga-se com poucos nodos e com poucas configurações topológicas. No modelo proposto apenas dois pacotes garantem que todas as possíveis configurações poderão ocorrer sem perda de generalidade do algoritmo verificado porque temos apenas três nodos na rede e um deles esta enviando a mensagem.

```
     /* I'm a intermediate node. I have to forward the packet */
if   /* TTL haven't expired. So, we can forward the message   */
    :: (ttl != 0)  ->
       if     /* I've sent this message. So, it's a Loop... */
           :: (Forward == true ) -> Loop    = true
           :: (Forward == false) -> Forward = true ->
              medium!type(origin,destination,(ttl - 1),myid)->

           /*
            * Sending the message again or not
            */
            if
               :: (SendAgain == true) ->
                  medium!type(origin,destination,(ttl - 1),myid)

               /*do anything*/
               :: (SendAgain == false)->SendAgain = false
            fi ->
                  run Network()
        fi

   /* TTL have expired. We need ignore the message */
   :: else -> run IgnoreMessage()
fi
```

2.2 Trabalhos Relacionados

Na literatura existem diversos trabalhos que tratam da verificação de protocolos de comunicação. No entanto, para a área de redes móveis *ad hoc*, no melhor do nosso conhecimento, não existe ainda um trabalho que apresente uma metodologia genérica que permita fazer a verificação de protocolos para esse tipo de rede de forma sistemática. Desta forma, esta seção escreve alguns trabalhos importantes para a verificação de protocolos de roteamento para MANETs.

Corson e Macker [39] descrevem as características das redes móveis ad hoc, discutindo o projeto e a avaliação de protocolos para MANETs, com ênfase no desempenho do roteamento nessas redes. Esse trabalho apresenta algumas características qualitativas desejáveis (métricas de avaliação) para um algoritmo de roteamento, mas não discute como avaliar de forma precisa tais características.

Os trabalhos [27, 120, 129, 89, 101, 61, 60, 130, 9, 88, 28, 87] discutem e comparam diversos algoritmos de roteamento para MANETs. Em particular, os trabalhos [27, 129] identificam critérios de comparação para esses algoritmos como roteamento a partir da origem (*source routing*), roteamento baseado em posição geográfica, uso de *flooding*, roteamento sob demanda, e condições para troca de tabelas. Conhecer esses critérios é o primeiro passo para entender protocolos de roteamento para redes móveis ad hoc.

Butty, Hubaux e Capkun [25] propõe um modelo formal para protocolos comunicantes (*exchange protocols*) baseado na teoria de jogos. Este modelo é representado como sendo um conjunto de estratégias em um jogo que é jogado pelas partes do protocolo e a rede que estas partes utilizam para se comunicarem entre si. Todo o artigo é dsenvolvido considerando-se protocolos de segurança ([25]) para o ambiente sem fio como o alvo principal do trabalho. Em especial o protocolo de Syverson [143] que segundo os autores não obtêm uma troca justa por usar uma terceira parte confiável. Os autores utilizam arquiteturas multi-hop ad hoc em células ([80]) como sendo o caso "padrão" para MANETs, baseando-a em uma infra-estrutura fixa, para extender o modelo às MANETs ([82]), pois o protocolo de Syverson não seria totalmente apropriado para nodos distantes. Os autores simplificaram as MANET para que pudessem aplicar o modelo que é específico às questões de segurança dos protocolos através de troca de mensagens justas e/ou racionais (*fair and rational exchange*). Apesar de apresentarem um modelo matemático interessante, o artigo traz hipóteses restritivas.

Helmy [3, 73] apresenta em sua dissertação uma nova metodologia que objetiva o desenvolvimento sistemático e a geração de testes automáticos para protocolos *multi-hops*. Esta metodologia chamada de *STRESS - Systematic Test Synthesis for multipoint protocol design* tenta sintetizar topologias para protocolos *multi-hops* e suas seqüências de eventos que "estressam" a corretude e o desempenho do protocolo. O trabalho é extremamente interessante na medida que tenta eliminar a explosão de estados na busca de todo o grafo que compõe o protocolo através de vários *aproachs* incluindo heurísticas, busca *forward e*

backward, simulação, verificação formal, dentre outras. Mas o foco é dado para o modelo baseado em Máquinas de Estado Finito Extendidas [127] dos protocolos. No entanto, o trabalho foi feito sobre protocolos para a Internet em LANs (*Local Area Networks*). A metodologia proposta neste trabalho basicamente se divide em dois algoritmos. O primeiro efetua uma análise de alcançabilidade reduzida, utilizando as informações específicas do domínio, para evitar uma pesquisa exaustiva. O segundo algoritmo, a geração de testes orientada a falhas, utiliza *backtracking* para gerar sequências de eventos do estado final para o inicial. Utilizando-os, o autor afirma que a correção de protocolos de roteamento *MultiCast* será apontada. Seada e Helmy [134] aplicaram a metodologia proposta em [3] sobre protocolos *MultiCast* para controle de congestionamento e demostram a boa aplicabilidade da metodologia deles.

Alguns dos sistemas distribuídos verificados utilizando SPIN e Promela estão descritos em [10, 71, 79, 150]. Bhargavan, Obradovic e Gunter [10] estudam a verificação de protocolos de roteamento baseados em Vetor-Distância [45]. O trabalho tem como objetivo mostrar a viabilidade em se usar ferramentas de verificação formal, como o SPIN, conjuntamente com provadores de teoremas, como o HOL [114]. Existem dois pontos importantes entre o trabalho descrito em [10] e o proposto aqui. O primeiro é que os autores em [10] mostram que o protocolo de roteamento AODV para MANETs é livre de *loop*, sendo essa a única propriedade analisada, ao contrário deste trabalho onde outras propriedades são consideradas na modelagem e verificação formal. O segundo ponto é que o trabalho descrito em [10] não apresenta uma metodologia genérica.

Leppänen e Luukkainen [98] mostram como a técnica verificação de modelos (*model checking*) tem se mostrado eficiente e relativamente fácil na verificação de programas formalmente descritos e em especial de sistemas móveis. Os autores ressaltam que um grande ponto a ser notado ao se utilizar esta ténica é a explosão de estados. De forma a evitar esse problema, os autores mostram como a abstração é fundamental para se diminuir o número de estados do sistema analisado, utilizando-se a técnica de verificação composicional [155, 65] em sistemas de comunicação de terceira geração.

Lauschner, Macedo e Campos [97] aplicaram a técnica de *model checking* em um protocolo de roteamento para MANETs. Os resultados encontrados neste trabalho, ao se verificar formalmente o protocolo GPSAL [28] e mostrar que ele é livre de *loops*, demostram a grande aplicabilidade da verificação formal sobre os protocolos de roteamento para MANETs. Neste trabalho, os autores utilizaram a técnica de *model checking* aliada aos BDD - *Binary Decision Diagram* [81], com o intuito de diminuir o número de estados do sistemas, utilizando-se da ferramenta Verus [29].

2.2.1 O Problema da Verificação Formal de MANETs

Corson e Macker [39] descrevem algumas características que seriam bastante interessantes de serem verificadas formalmente tais como:

- Operar de forma distribuída, sendo este um requisito fundamental para qualquer algoritmo de roteamento em redes ad hoc
- Não apresentar loops de roteamento
- Operar de acordo com a demanda
- Modo de operação pró-ativo (Apêndice A.15)
- Apresentar requisitos de segurança
- é desejável também que o algoritmo de roteamento tenha suporte a links unidirecionais
- Encontrar a rota ótima, ou seja, encontrar a rota que tenha a melhor situação se comparada à métrica utilizada. Em geral, essa métrica representa o número de nodos entre o destino e a origem.
- Encontrar múltiplas rotas, ou seja, encontrar mais de uma rota ao destino
- Re-rotear caso haja enlaces rompidos (*broken-link* - Apêndice A.18)
- O algoritmo deve apresentar escalabilidade, ou seja, que o algoritmo possa se comportar bem, caso o número de nodos na rede cresce ou diminua.
- Certa independência em relação à densidade de nodos (número de nodos / largura da região)

Normalmente, o que se faz, intuitivamente ao se modelar um algoritmo é tentar recriar exatamente o mesmo ambiente no qual este procotolo está inserido de forma que se tenha um resultado mais fidedigno. No caso de verificação formal de algoritmos de roteamento para redes móveis ad hoc, é natural que se tente representar diretamente a topologia e sua dinamicidade. Entretanto, ao se fazer isso, não se tem uma garantia real de que aquele algoritmo irá se comportar da mesma forma, uma vez que no processo, foi, inequivocadamente, fixada uma topologia. Então é necessário deixar que a topologia varie ao longo do tempo. A partir disto, nota-se um grande problema.

Quando se deixa a topologia variar, o problema se torna rapidamente intratável, uma vez que o número de estados gerados internamente (*State space*) durante o processo de

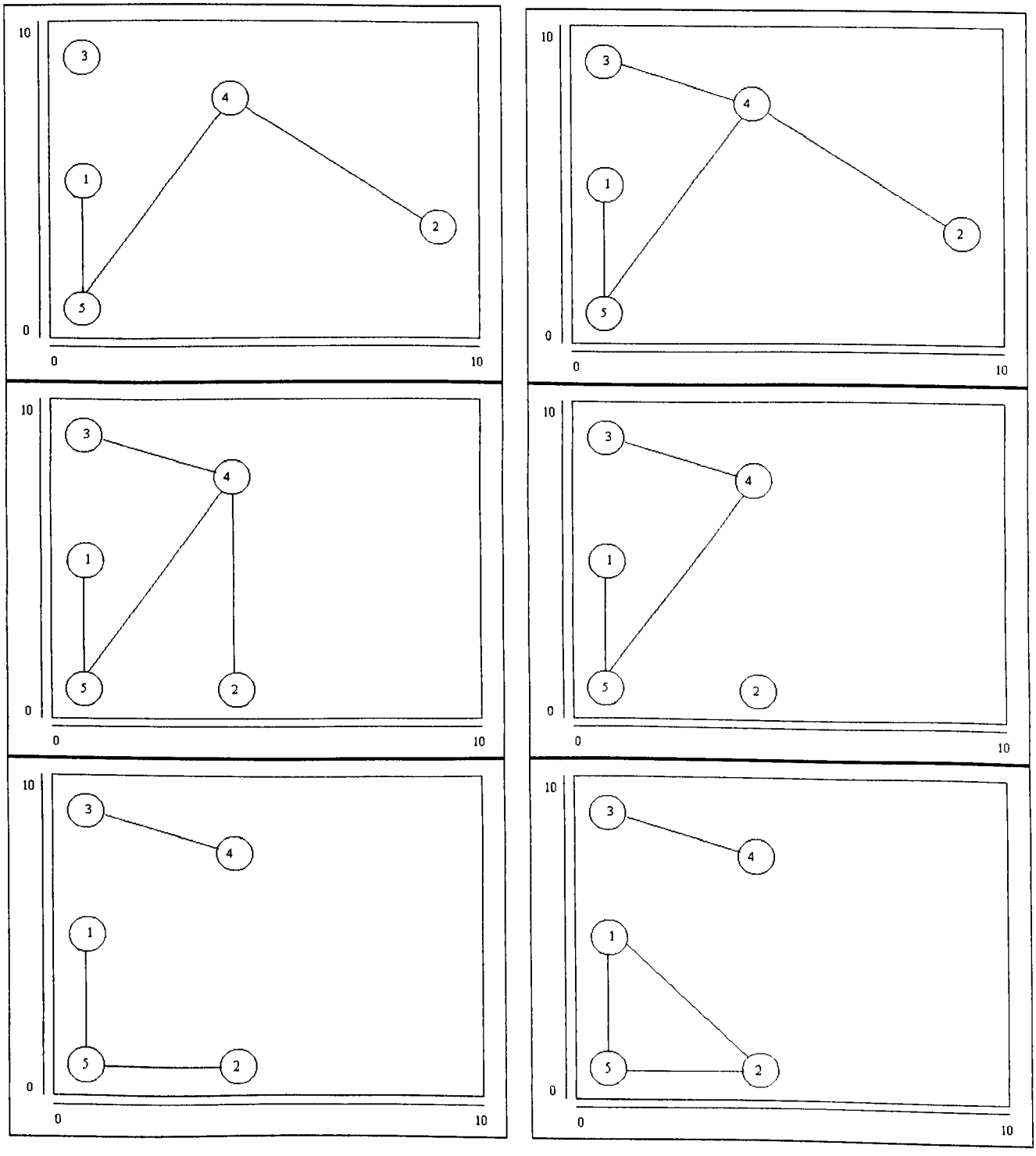

Figura 2.1: Mobilidade dos nodos $\Longrightarrow$ Explosão de Estados.

verificação, aumenta muito o tempo de execução da análise que está sendo feita. Um exemplo simplificado, para ilustrar a questão é mostrado na Figura 2.1.

Neste exemplo, devemos imaginar que tenhamos uma pequena região quadrada como mostra a Figura 2.1. Por toda a região existe um gride imaginário para que se possa identificar a posição dos nodos num dado instante. Assim, dividimos a região em um pequeno gride de $10X10$, com divisões de uma em uma unidade de forma discreta. Assim, o nodo sempre se moveria discretamente de uma unidade para outra. Esta simplificação é utilizada apenas para simplificar os cálculos e o entendimento.

Agora, devemos supor que se tenha apenas 5 nodos dentro desta região. Pode-se notar que a região contém 100 posições discretas possíveis para cada nodo estar. Como existem 5 nodos, cada um dos nodos pode estar em alguma posição em um instante, restando 99 posições para outro, e assim sucessivamente, ou seja, existirão combinações de 100 posições 5 *a* 5. Devemos agora considerar que um nodo pode ou não ter comunição com outro nodo qualquer independemente de sua localização (neste momento abstraímos questões como alcance, interferência, sentido do canal de comunicação, dentre outras), como é ilustrado na Figura 2.1 pelos nodos 1 e 3 respectivamente. Existirão, portanto, para cada posição uma combinação de 5 (nodos), 2 *a* 2 (pode ou não ter comunicação). Assim, em cada cenário apresentado, teriámos algo como 700 Milhões de combinações, conforme mostra o esquema abaixo:

$$C_{100}^{5} \times C_{5}^{2} = \frac{100!}{5!95!} \times \frac{5!}{2!3!}$$

Simplificando, temos:

$$C_{100}^{5} \times C_{5}^{2} = \frac{25 \times 33 \times 98 \times 97 \times 96}{1}$$

Calculando, temos:

$$C_{100}^{5} \times C_{5}^{2} \approx 700 \; Milhões \; de \; Possibilidades$$

Este é um problema complexo e pode-se mostrar que a verificação de uma propriedade simples como ausência de *deadlock* (Apêndice A.9), propriedade da classe *safety*(Apêndice A.7), é PSPACE-hard [128].

A partir disto, pode-se notar claramente a inviabilidade da abordagem, tendo em vista que é interessante que a rede real seja muito maior e com mais nodos, se movemendo de

forma contínua e aleatória. Desta forma, nos deparamos a intratabilidade do problema do ponto de vista de fazer a verficação convencional. Deve ser lembrado ainda que a partir de qualquer cenário anterior pode-se em ordem alcançar outro. Na prática, serão possíveis apenas algumas seqüências a partir de um conjunto limitado de restrições tais como alcançe, interência, sentido do canal de comunicação, velocidade de movimento dos nodos, tipo do nodo, tipo de comunicação, dentre outras. Porém, mesmo com várias limitações, o número de estados e combinações destes é ainda enorme, inviabilizando a modelagem, em princípio, da topologia e de sua dinamicidade.

Uma forma interessante de reduzir essa quantidade enorme de possibilidades é utilizarmos simetria, como o apresentado nos trabalhos [102, 126]. No entanto, muitas destas possibilidades não são simétricas. Mesmo que tivéssemos uma simetria de 50% das possibilidades ainda teramos um grande número a tratar. Evidentemente, simetria é uma técnica muito interessante para ser aplicada numa situação como esta, mesmo que não resolve sozinha o problema.

Com isto, surgiu a seguinte pergunta: "Como abstrair o sistema o máximo possível, de forma a diminuirmos a quantidade de estados gerados, independemnte de ser ou não possível utilizar algumas restrições, e assim ainda, representar o sistema de forma fidedigna, ou seja, como abstrair da topologia do sistema, e assim mesmo garantir as características do algoritmo, uma vez que, em geral, os algoritmos para MANETs dependem diretamente ou indiretamente da posição e da topologia para efetuarem o roteamento. Esta pergunta é respondida pelo próximo capítulo.

Capítulo 3

A Metodologia de Verificação Formal para redes móveis *Ad Hoc*

Este Capítulo apresenta toda a metodologia proposta neste trabalho, seu contexto e seu escopo, e está organizada da seguinte forma. Inicialmente, na seção 3.1 serão feitas algumas considerações relativas à metodologia e seu inter-relacionamento com os princípios apresentados no trabalho. A seção 3.2 apresenta os princípios da metodologia proposta. Na seção 3.4 são apresentadas as informações do protocolo necessárias para se efetuar a modelagem. Na seção 3.5 é apresentada a modelagem em si.

3.1 Considerações Relevantes

Há um grande variedade de estudos de protocolos de comunicação que obtiveram sucesso relativo como por exemplo [110]. Os protocolos de roteamento em especial tem os seguintes atributos que influenciam na maneira pela qual as técnicas de verificação formal podem ser aplicadas:

- Um número (essencialmente) replicado de processos simples que executam concorrentemente
- A conectividade é dinâmica

- Os processos são mini-sistemas reativos com uma interface discreta de baixa complexidade

Além do mais, a maioria dos protocolos de roteamento apresentam alguns outros atributos tais como latência do fluxo de informações (limitante para a existência de um relógio global) e a necessidade de proteção de alguns recursos da rede. Estes atributos, por vezes, tornam os algoritmos mais complexos. Desta forma, essas características são fundamentais para se escolher a abstração adequada que influenciará no processo de verificação utilizando-se da metodologia proposta.

O fato de algoritmos serem compostos por processos e serem "replicados" em cada nodo da rede, simplifica o método de estudo pois as simetrias existentes praticamente transformam cada nodo num único nodo na rede, uma vez que todos fazem as mesmas tarefas. No entanto, quando isso não é verdade o sistema deve ser modelado de forma a transpor esse problema para que se possa aplicar a metodologia de forma correta.

3.1.1 Conceitos para a Metodologia

O conjunto de técnicas utilizadas para a criação da metodologia tem o intuito de proporcionar um modelo de verificação de algoritmos. Esta verificação pode ser entendida como sendo um processo *"Top-Down"* no qual um algoritmo é verificado de modo a satisfazer um conjunto de propriedades. Este conjunto é chamado de *requerimentos* do algoritmo. Com o intuito de se efetuar esta verificação cria-se a especificação do sistema, ou seja, um modelo formal abstraído do sistema [67].

A especificação pode, às vezes, ser não-determinística. Isto significa que o sistema pode executar várias ações "simultaneamente" (num dado tempo), pois tais ações podem depender de fatores externos tais como dados de entradas, *Timeouts*, ordenações, etc. Caso isso ocorra, a representação do sistema deve ser feita de forma bastante criteriosa.

3.2 Princípios

Uma metodologia apresenta uma regra de aplicação de um conjunto de métodos independentemente de como a implementação deles é fundamentada. No entanto, toda metodologia apresenta um domínio de objetos que podem utilizar estas regras. Dá-se, portanto, bastante liberdade àqueles que pretendem usá-la, uma vez que nã é necessário se preocupar com o objeto de investigação em questão, desde que este objeto pertença à classe de aplicabilidade da metodologia.

A metodologia apresentada neste trabalho mostra uma forma de modelar protocolos de roteamento para redes móveis ad hoc e propriedades que podem ser verificadas com esse modelo. Aplicando-se esta metodologia é possível encontrar, eventualmente, situações onde propriedades de interesse podem, ou não, ser satisfeitas.

No desenvolvimento da metodologia alguns princípios importantes foram definidos:

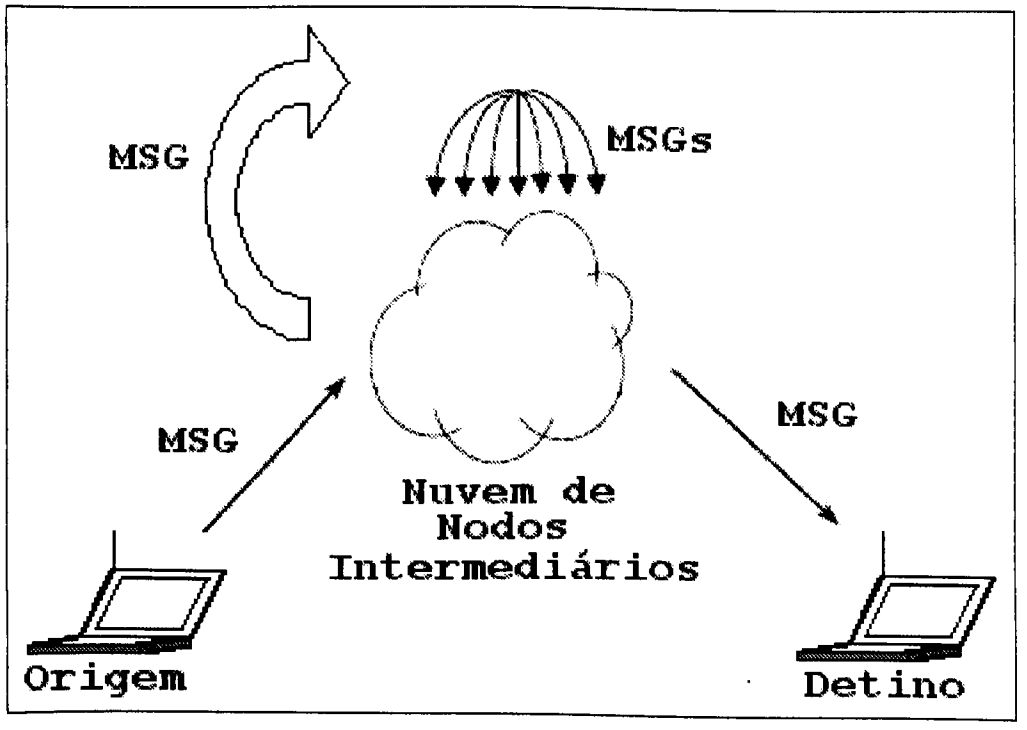

Figura 3.1: Abstração da topologia utilizando-se 3 nós.

- **Nodo intermediário e Abstração da topologia.** Um dos maiores problemas na verificação de redes móveis *ad hoc* é a modelagem da topologia que é totalmente

dinâmica. Isso faz com que seja inviável tentar verificar e/ou simular todas as combinações de topologias mesmo para um número pequeno de nodos. Um dos pontos importantes deste trabalho é a solução proposta para resolver este problema. O nodo intermediário irá modelar uma nuvem de nodos intermediários, que representa o comportamento de um nodo qualquer independente de uma topologia específica qualquer, conforme mostra a Figura 3.1.

Desta forma, consegue-se resolver o problema da topologia pois o nodo intermediário passa a representar as situações em que um nodo intermediário qualquer pode se encontrar, tais como nodo fora da área de alcance, mensagem enviada e perdida, mensagem enviada para o nodo origem, outro nodo intermediário ou destino, caminhos com diferentes quantidades de nodos intermediários, dentre outros. Com isto, a topologia passa a ser irrelevante já que a propagação de uma mensagem é feita partindo-se da origem, passando eventualmente pela nuvem de nodos intermediários, e chegando ou não até o destino.

O uso do nodo intermediário, como sendo uma nuvem de nodos intermediários, abstrai o algoritmo a ser verificado. A abstração é processo na qual se evita que a ferramenta SPIN execute todas as verificações no *Espaço de Estados* (*State Space*), verificação ilimitada (*unbounded*), quando a topologia varia com o tempo, reduzindo assim uma verificação ilimitada para uma verificação limitada ou finita [110]. Para que esse ideéia seja válida, é necessário que o sistema abstraído seja uma instância do protocolo. Nós podemos formalmente dizer que o par (M^{abs}, P^{abs}), representando um modelo e uma propriedade abstraídos, é uma abstração do par (M, P), representando um modelo e uma propriedade, se ocorrer que:

$$\boxed{M^{abs} \vDash P^{abs} \Rightarrow M \vDash P} \tag{3.1}$$

Esta implicação tem de ter equivalência, para que potencialmente não se sacrifique a integridade (*soundness*) da verificação, para preservar a completude (*completeness*).

É fundamental que seja assim para que possamos não somente garantir que a corretude da abstração implique a corretude do sistema original, mas também que uma falha na abstração implique necessariamente em uma falha do modelo. Se tivéssemos apenas:

$$\boxed{M^{abs} \vDash P^{abs} \Leftarrow M \vDash P}, \tag{3.2}$$

nada poderíamos concluir quando a abstração falhasse, pois o modelo original poderia ainda estar correto por nossa abstração fez considerações muito "fortes", o que

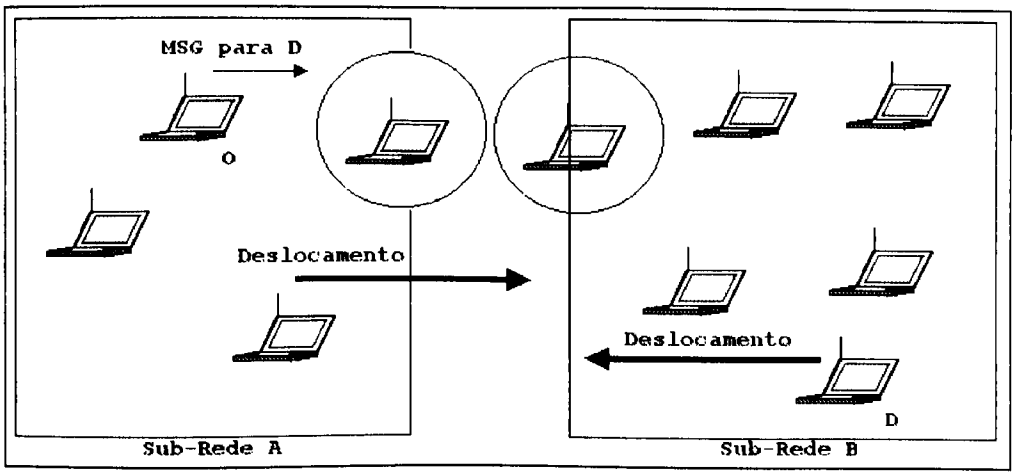

Figura 3.2: Exemplo de Abstração muito "forte" em uma rede particionada.

invalidaria o modelo abstraído. Por exemplo, suponhamos que estivéssemos modelando uma rede particionada de nodos móveis, como mostra a Figura 3.2. Veja que os nodos podem mudar de sub-rede, mesmo que a rede continuará particionada. Quando um nodo qualquer O pertencente, digamos à a sub-rede A, desejasse enviar um mensagem para um nodo qualquer D na sub-rede B, o nosso modelo abstraído simplesmente descartaria a mensagem, sem levar em consideração questões como a mobilidade, *Time-Outs*, re-envio de mensagens, *cache*, dentre outras. Isso levaria a um resultado incorreto por assumirmos considerações fortes de nossa abstração sobre o modelo desejado, pois existem nodos em movimento que irã se mudar de A para B e vice-versa que eventualmente poderiam entregar a mensagem dependendo do comportamento do algoritmo de descobrimento/manutenção de rotas.

A equivalência, no entanto, nos permitiria concluir que o sistema original é também correto neste caso, conforme mostrado em [110]. Podemos notar assim claramente que restrições podem ser vistas como uma forma de abstração. Esta técnica é espinha-dorsal da metodologia aqui descrita.

- **Posição do nodo.** Um dos critérios usados na classificação de algoritmos para MANETs é a utilização ou não da posição geográfica do nodo (por exemplo, latitude, longitude e altitude) para fazer o roteamento [39, 120, 27]. De acordo com essa classificação os algoritmos são chamados geográficos e não geográficos, respectivamente. Na metodologia proposta, mesmo para algoritmos geográficos, não é necessário definir uma posição para o nodo, já que na modelagem do algoritmo existem apenas três tipos de nodos: Origem, Destino e Intermediário, que modelam o comportamento da rede e não posições específicas. Logo, qualquer configuração topológica da rede está implícita pois o nodo intermediário representa o conjunto de todos os nodos intermediários em qualquer posição, como mostra a Figura 3.3.

  Como podemos notar pela Figura 3.3 as mensagens que eventualmente necessitam

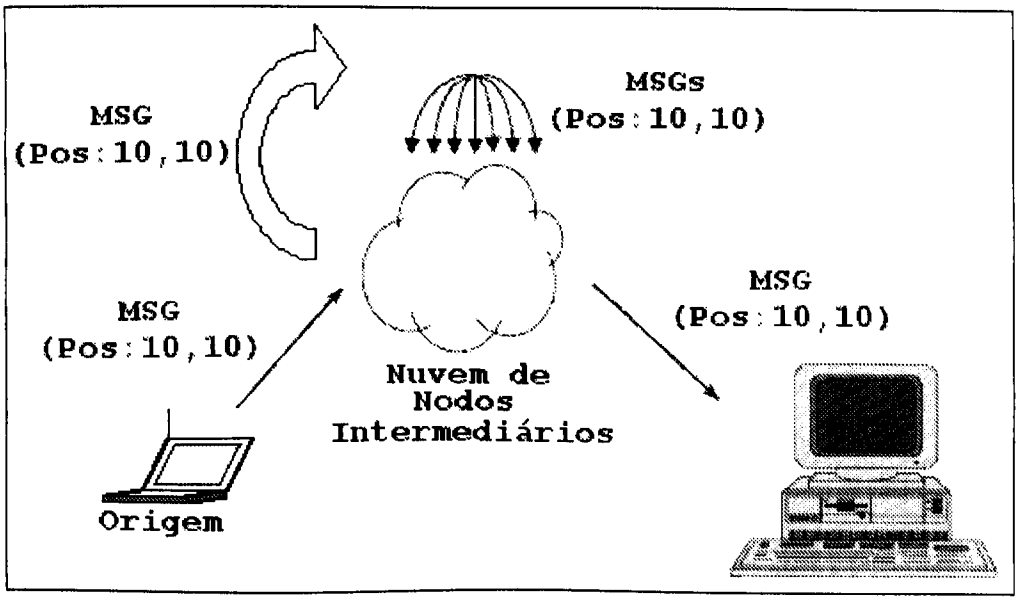

Figura 3.3: Abstração da posição utilizando-se 3 nós.

levar características de posição ou que permitam a descubramos como velocidade e direção devem ser enviadas normalmente. Assim, o nodo intermediário poderá utilizar essas informações apropriadamente para que o algoritmo decida o que fazer. No entanto, nenhuma configuração topológica da rede é diretamente considerada, devido à abstração utilizada.

- **Velocidade dos Nodos** A velocidade dos nodos é variável e grande o suficiente para permitir que os nodos mudem suas posições de forma dinâmica a cada intervalo de amostragem. No entanto, a velocidade das mensagens é infinitamente superior à velocidade dos nodos, evitando-se que os nodos e as mensagens entre em condição de corrida (*race condition*). A Figura 3.4 mostra uma condição de corrida entre o nodo B e a mensagem MSG. O nodo B consegue "fugir" da mensagem. Assumimos que esta situação nunca ocorrerá.

- **Serviço oferecido pelas camadas inferiores.** Foi necessário assumir que o serviço, ou conjunto de serviços, oferecido pelas camadas inferiores à camada de rede está disponível e é sempre executado de forma correta. Isto foi feito, para não tornar o problema ainda maior devido à necessidade de se levar em conta os protocolos de mais baixo nível. Assumiu-se, portanto, que os protocolos das camadas inferiores funcionam, mesmo que não tenham sido formalmente modelados neste trabalho.

- **Canal de comunicação.** Assumiu-se que o canal de comunicação entre os nodos, do ponto de vista da camada de enlace, além de ser confiável, é comum a todos os

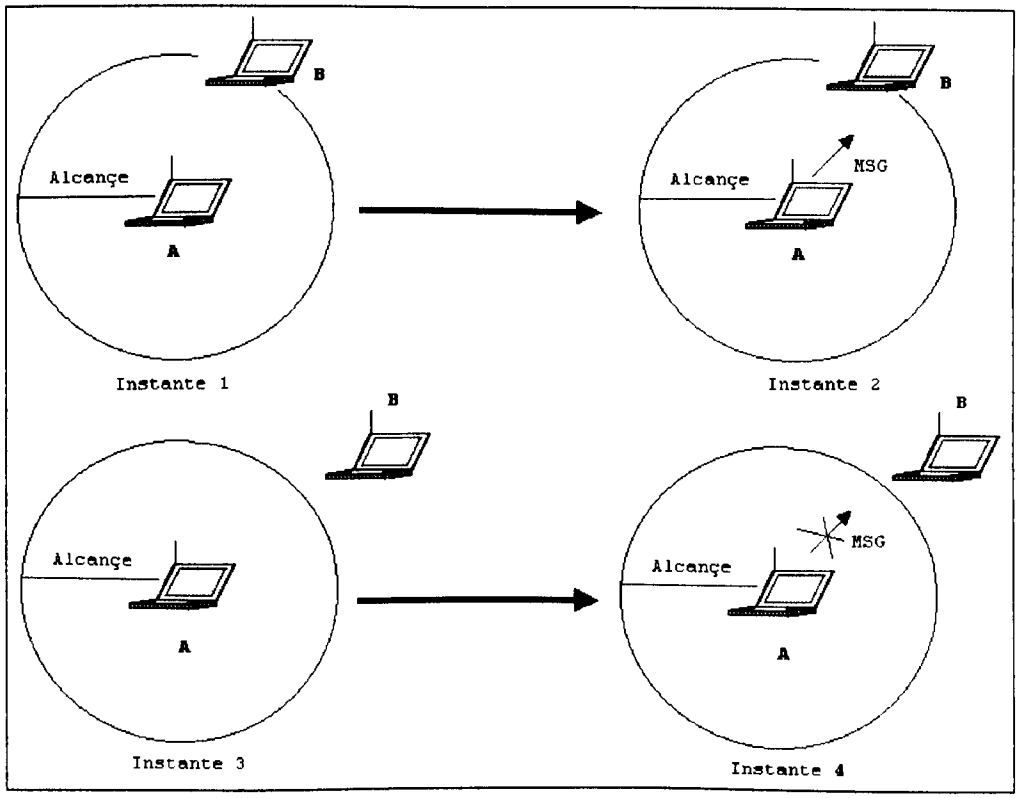

Figura 3.4: Condição de corrida entre a mensagem e um nodo qualquer.

nodos. A mensagem deve ser enviada para um único canal de comunicação e qualquer nodo pode receber a mensagem desse canal, permitindo-se, portanto, que tanto a mensagem continue a se propagar livremente pela rede (através de um nodo intermediário), quanto chegue ao destino ou mesmo retorne a origem, conforme mostra a Figura 3.5. Esta característica dá grande flexibilidade à modelagem nos quesitos posição velocidade e propagação da mensagem, na medida que quando qualquer nodo recebe a mensagem, qualquer configuração de posição, alcance, velocidade e propagação é considerada automaticamente. Não obstante, deve-se notar que é o comportamento do protocolo que determina se uma mensagem irá efetivamente alcançar o destino, retornar a origem, ou não ser entregue, e não a configuração do canal onde a mensagem trafega. Deve ser percebido pela Figura 3.5 que abstrai-se do fato de o nodo ser fixo ou móvel, o que permite que algoritmos híbridos utilizem a metodologia.

- **Pseudo-Código**. A criação do pseudo-código utilizado pela metodologia para se efetuar a análise formal deve sempre seguir a norma (*standard*) do protocolo, ou seja, a descrição padrão clara e objetiva do algoritmo. A norma de um protocolo

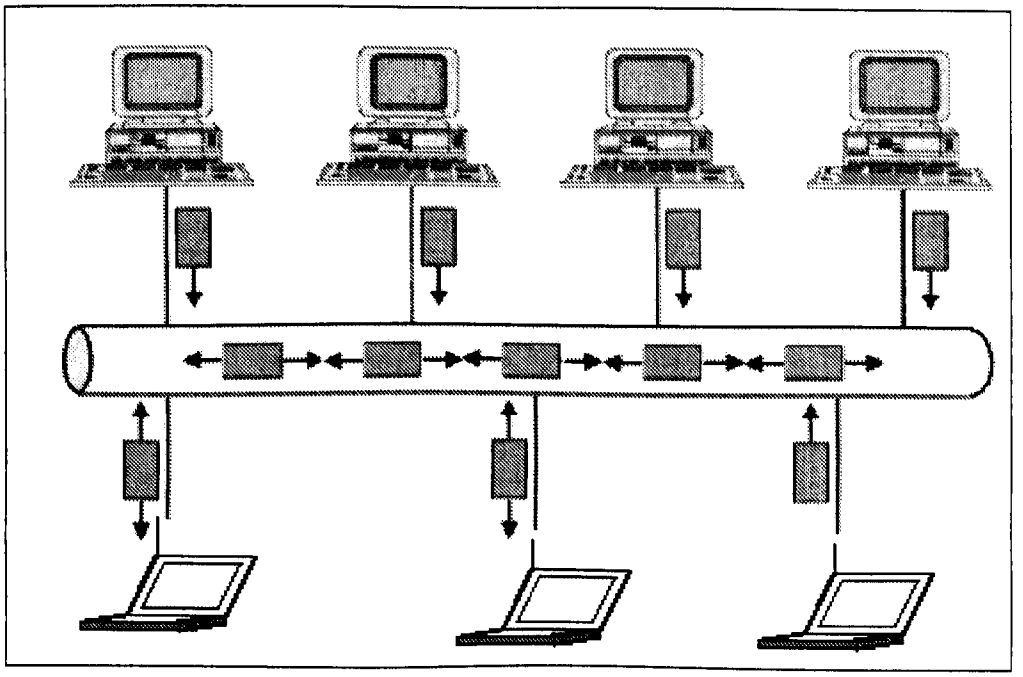

Figura 3.5: Canal de Comunicação único entre os nodos da rede.

é um conjunto de documentos que descrevem precisamente como o protocolo trabalha. Exemplos destes documentos são os *standards* do IETF ou a fase anterior deles chamados de RFCs [67]. Ao se definir esse relacionamento, está se reduzindo o problema de verificação da corretude de uma dada implementação para a verificação de sua ligação à norma. Mais além, a norma precisa ser verificada apenas uma única vez e o resultado poderá ser utilizado em várias implementações. Em alguns casos, contudo, a verificação da norma não traz o retorno esperado ou pelo fato de o protocolo estar em constante modificação ou pelo fato de a situação que está sendo verificada ser dependente da implementação, mesmo em caso qualitativos. Em [14] é apresentado um arcabouco para se determinar qual das duas situações deveria ser aplicada. No entanto, neste trabalho considera-se sempre que está se utilizando a norma do protocolo e os casos omissos deverão ser tratados pela abstração escolhida para representá-lo. Evidentemente, durante o processo de geração do pseudo-código, pode-se encontrar ambigüidades explícitas que deverão ser retiradas sem alterar o princípio básico de funcionamento do algoritmo ou estas ambigüidades podem ser formalmente verificadas e então retiradas da modelagem final de forma a evitar que usuário retire partes de forma incorreta ou mesmo altera o comportamento do protocolo.

Neste trabalho, não se pretendeu garantir que toda e qualquer propriedade poderá

ser verificada utilizando-se esta metodologia. Propriedades de tratamento estatístico, por exemplo, estão ainda fora do escopo atual.

A metodologia determina algumas características de algoritmos de roteamento para redes móveis *ad hoc* que podem ser verificadas formalmente e outras não. Além disso, essa metodologia apresenta um processo prático e consistente quando se deseja verificar formalmente, através de análise de alcançabilidade, um protocolo de roteamento para redes móveis *ad hoc*. Esta metodologia não modela propriedades como rotas e posicionamentos específicos, quantidades mínimas e/ou máximas de mensagens trocadas pela rede, consumo máximo de energia pela rede, e *timeouts*. No entanto, ela é capaz de tratar questões como detecção de *loops*, casos em que o nodo está apto ou não a entregar o pacote, problemas de comportamento do protocolo como recepção de mensagem não especificada, e casos patológicos não tratados na especificação. A vantagem em se utilizar uma ferramenta de verificação formal é que ela apresenta o cenário onde isto ocorre. De uma maneira geral, a metodologia desenvolvida tem o objetivo de caracterizar dados qualitativos e não quantitativos dos algoritmos. Alguns algoritmos foram verificados com o intuito de se fundamentar e validar a aplicabilidade e a consistência da metodologia proposta, conforme pode ser visto no Capítulo 5.

3.3 Requisitos

Com o intuito de formalmente estabelecer a corretude de um algoritmo, é necessário que sejamos extremamente hábeis para expressar os requisitos deste protocolo, criando assim o modelo a ser verificado [66]. O caso desejável ocorre quando a linguagem com a qual se especifica o algoritmo e seus requisitos têm modelos semânticos similares, pois simplifica o método de expressão e entendimento. Com isso, diminui-se a ocorrência de erros ou enganos neste processo. Entretanto, o caso comum apresenta uma linguagem de requisitos mais simples e abstrata do que a de especificação [66, 12, 11]. Neste trabalho utilizou-se a linguagem de especificação *Promela* [77, 79, 78] associada com a de requisitos LTL. Faremos uma breve discussão a este respeito.

3.3.1 LTL (*Linear Temporal Logic*)

Os formalimos lógicos para o raciocínio sobre o tempo e a temporização de eventos aparecem em diversas áreas tais como física, filosofia, ciência da computação[1], etc.

Há cerca de 25 anos atrás, Pnueli identificou a lógica temporal (LT) como sendo uma linguagem formal muito conveniente para que se pudesse declarar e raciocinar sobre as propriedades comportamentais de programas paralelos e mais especificamente de sistemas reativos [122, 103, 123].

A LTL - *Linear Temporal Logic*, Lógica Temporal Linear, foi claramente definida em [58], sendo considerada um tipo de LT aplicada em uma dimensão sendo modelada como sistemas de transição, não sendo poderosa suficiente, por exemplo, para modelar sistemas de tempo real [99].

As fórmulas *Linear Temporal Logic* - (LTL) podem ser utilizadas para expressar um grande conjunto de propriedades tais como *liveness* and *safety*. Uma fórmula LTL **f** pode conter quaisquer símbolos proposicionais **p**, combinados com operadores unários ou binários, lógicos e/ou temporais, usando a gramática mostrada abaixo ([77]):

[1] Decidibilidade e "axiomatização" são questões padrão para os lógicos; porém para os práticos, a questão importante é *Model Checking* [24]

Gramática LTL

```
f::=        p
          | true
          | false
          | (f)
          | f binop f
          | unop f

unop::=     []      (always)
          | <>      (eventually)
          | !       (logical negation)

binop::=    U       (strong until)
          | &&      (logical and)
          | ||      (logical or)
          | ->      (implication)
          | <->     (equivalence)
```

LTL é certamente lógica modal mais comum para expressar propriedades de sistemas reativos. Em [121] pode-se encontrar detalhada descrição de suas sintaxe e semântica.

LTL é simples e muito útil para expressar propriedades sobre um qualquer número fixo de nodos. Contudo, ela não é capaz de expressar propriedades sobre um número arbitrário de nodos [110]. Um exemplo disso é a propriedade liberdade de *loops*. Suponhamos que cada nodo mantenha um ponteiro de quem é o proóximo nodo vizinho para se chegar a cada destino conhecido. O nodo precisa conhecer quem são seus vizinhos. Esta propriedade, então, diz que a seqüência de ponteiros não formará ciclos. Esta propriedade não pode ser diretamente expressa em LTL, a menos que se saiba o número exato de roteadores no sistema, para um rede genérica qualquer [110]. Uma malícia a ser utilizada para que possamos expressar propriedades como esta é reduzi-las a algum *invariante local*. Por exemplo, de forma a tentar estabelecer esta propriedade, é suficiente uma estrita ordem ($\prec$) no conjunto de nodos com a seguinte propriedade [151, 107]:

$$\boxed{\forall n_1, n_2 \,.\, \Big(next(n_1) = n_2\Big) \Rightarrow \Big(n_1 \prec n_2\Big)} \tag{3.3}$$

Note que esta propriedade só depende de dois nodos e portanto pode ser expressa em LTL. Como toda a rede foi abstraída em apenas 3 nodos pela metodologia. Pode-se concluir que é possível modelar quaisquer propriedades como esta. Vardi e Wolper, e, Tauriainen e Heljanko mostraram que qualquer fórmula LTL pode ser traduzida em um autômato de Büchi [145, 156]. O autômato gerado, automaticamente [146, 79] e sob-demanda (*on-the-fly*) [62] como faz o SPIN, pela conversão da expressão LTL, formalmente *aceita* somente aquelas (infinitas, deve ser ressaltado) execuções que satisfazem à expressão LTL desejada.

Além disso, em [133] é provado que a LTL é PSPACE-completa, ou seja, garante que com a LTL é possível varrer todo o espaço de busca existente e possível.

A partir do que foi exposto, podemos concluir que o SPIN [79, 77] utilizando-se LTL é suficientemente apropriado e expressivo para se modelar, especificar e verificar formalmente, utilizando-se de verificação de modelos, algoritmos de roteamento para MANETs.

3.4 Informações para Modelagem

O processo de modelagem começa com a obtenção de informações sobre o protocolo a ser verificado. Esse é um ponto importante pois é a partir dessas informações que serão feitas as abstrações para a modelagem. Exemplos de informações a serem obtidas são:

- tipos de pacotes definidos pelo protocolo;
- cada pacote possui tempo de validade (TTL - *Time To Live*) ou não;
- a semântica de cada pacote e o seu tratamento;
- uso ou não de tabelas de roteamento e, no caso de se usar, o momento de troca dessas tabelas;
- uso ou não de informação geográfica para se fazer o roteamento;
- situações não desejáveis que podem ocorrer como rota não encontrada e *loop* de roteamento;
- o tratamento ou não de mensagens de *Hello*, dentre outras.

A organização dos dados é outro ponto-chave. Por exemplo, os pacotes devem ser separados em classes (roteamento, entrega de mensagens, etc) para que se possa entender melhor o papel de cada PDU. Idealmente, o comportamento do protocolo deve ser representado por uma máquina de estados finitos (FSM - *Finite State Machine*). Todas essas informações conjuntas que no final irão fornecer os elementos necessários à verificação do protocolo.

O passo seguinte é definir o grau de abstração escolhido para se fazer a verificação. Um cenário típico é se mensagens de *hello* são realmente necessárias durante o processo de verificação. A resposta é: depende da abstração. Se as mensagens de *hello* são essenciais para a verificação do comportamento do algoritmo ou se alteram drasticamente a forma como o protocolo atuar, então elas devem ser modeladas. Caso seja possível, o ideal seria modelá-las como sendo Recebeu ou nã *hello*. Outro caso comum é se um dado procedimento, como a atualização gratuita das tabelas, influem diretamente no processo de correção do roteamento ou é apenas uma forma de otimização que não ocasiona falha do algoritmo.

3.5 Modelagem

Como mencionado anteriormente, um dos maiores problemas na verificação de algoritmos de roteamento para redes móveis ad hoc é a modelagem da topologia, que devido à natureza do problema, é completamente dinâmica. Tentar simular combinações de topologias mesmo para um número pequeno de nodos acaba se tornando um processo totalmente inviável. Em [110] é apresentada um método de verificação de algoritmos para redes móveis *ad hoc* o qual modela a topologia como sendo uma matriz de adjacência em Promela. No entanto, esse processo é feito de forma restrita.

A metodologia pode ser vista como um modelagem do problema em módulos bem definidos conforme mostra a Figura 3.6

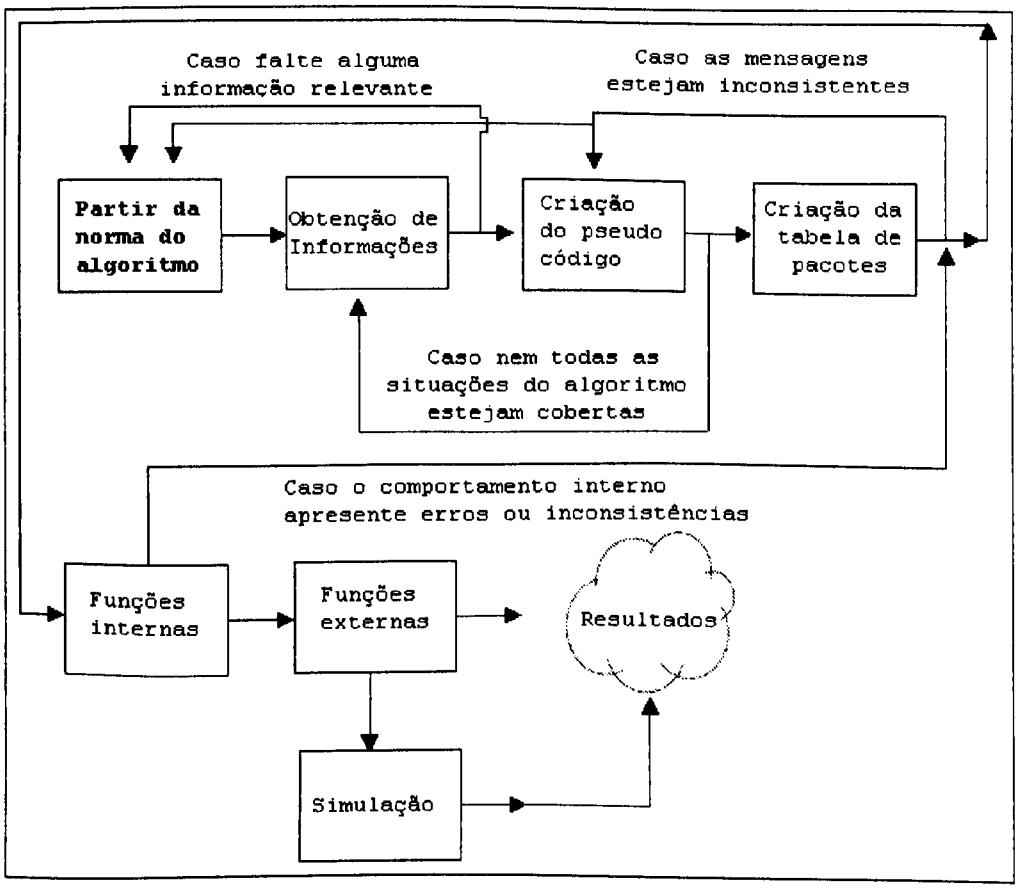

Figura 3.6: A metodologia de verificação para redes móveis *ad hoc* .

A abordagem adotada para resolver o problema consiste na divisão da modelagem do

protocolo em dois sub-problemas mais simples, devido à complexidade de se efetuar a verificação em conjunto, essa técnica é conhecida como modularização(*Modular Design*) [2, 127, 149]. A modularização pode reduzir em muito o esforço requirido durante o processo de verificação. Apesar desta técnica ser genérica, normalmente ela é aplicada ao ambiente de *hardware*.

O primeiro sub-problema é a verificação do funcionamento interno do nodo, garantindo que seus procedimentos internos não apresentam ambiguidades ou inconsitências. Isto implica na verificação do fluxo interno de dados e mensagens do protocolo. O segundo é a verificação do funcionamento do algoritmo na rede, abstraindo-se da topologia, que é o grande problema a ser tratado.

Desta forma, é possível escolher níveis diferentes de abstração tanto para o funcionamento do algoritmo na rede quando em relação às suas mensagens internas. Essa flexibilidade permite que os dois problemas sejam tratados de forma independente e no nível de detalhe desejado.

É importante observar que a modelagem não busca verificar a eficiência do protocolo e sim sua correção. Eventualmente, em algum caso específico e dependendo da abstração escolhida, a metodologia será capaz de observar a eficácia do evento.

O objetivo da metodologia é verificar se o protocolo funciona corretamente ou em quais casos ele pode falhar dadas as condições iniciais desejadas e abstração escolhida. Por exemplo, caso sejam desconsideradas as mensagens de *Hello* na abstração não se pode afirmar categoricamente que não existem *loops*. Este trabalho não pretende avaliar se o algoritmo está apto a encontrar a melhor rota e sim em que casos ele não encontra uma rota. Com esta abordagem pretende-se:

- encontrar eventualmente *loops*;
- casos em que o nodo está ou não apto a entregar um pacote;
- erros de projeto interno, como por exemplo mudanças inesperadas de comportamento;
- casos em que o protocolo não se comporta conforme o especificado, mostrando o cenário onde isto ocorre;
- casos patológicos incomuns que são, em geral, desconsiderados no projeto do algoritmo.

A modelagem não trata rotas e posicionamentos específicos dos nodos para que não se tenha o problema da topologia. No entanto, estes pontos podem ser abstraídos de forma a garantir o comportamento geral do protocolo.

A seguir, é apresentada a *Metodologia* utilizada na modelagem de algoritmos para redes móveis *ad hoc* que foi aplicada neste trabalho.

Principais tarefas:

1. Obtenção das informações necessárias à modelagem do protocolo;
2. Criação do pseudo-código detalhado baseado na descrição padrão (*standard*) do protocolo;
3. Comparar o pseudo-código gerado com a descrição e seus casos de uso. O pseudo-código deve representar fielmente todos os casos cobertos na descrição do protocolo;
4. Criar um tabela descrevendo todos os pacotes usados pelo protocolo, identificando o nodo que pode criá-los (origem, destino, intermediário) e a semântica dos pacotes para cada tipo de nodo;
5. Dividir o funcionamento do protocolo em duas partes representando seu funcionamento interno e externo:
   (a) Funcionamento Interno: descreve o fluxo da mensagem e o comportamento do protocolo internamente ao nodo;
   (b) Funcionamento Externo: descreve o comportamento do algoritmo com relação a interação entre os nodos da rede;
   (c) Consideração Interno × Externo: o funcionamento interno deve modelado de forma a poder ser tratado pelo funcionamento externo de forma abstrata, ou seja, de forma a não depender de detalhes internos ao nodo.

Funcionamento Interno:

6. A modelagem do funcionamento interno do nodo tem como objetivo validar o funcionamento do algoritmo internamente ao nodo. Deseja-se aqui verificar as ações tomadas pelo algoritmo com relação ao recebimento e envio de mensagens:
   (a) Diversas simplificações podem ser feitas em relação ao processamento das mensagens. Por exemplo, para um protocolo que trabalha com tabelas de roteamento pode não ser interessante testar o algoritmo usado para encontrar o menor caminho na tabela—tipicamente o algoritmo de Djikstra [48] ou Floyd [56], mas simplesmente saber se a rota escolhida naquele momento, baseada na informação local, é mínima ou não. Neste caso pode-se usar uma variável booleana indicando se a rota mínima foi encontrada ou não. Cada simplificação deve ser avaliada para não comprometer o funcionamento do algoritmo.

(b) Deve-se tomar cuidado na implementação da modelagem evitando inserir decisões não-aleatórias no fluxo do protocolo a ser verificado. Desta forma todos os possíveis caminhos serão verificados.

(c) O funcionamento interno do algoritmo é normalmente inicializado pelo recebimento de um pacote. Os dados deste pacote devem ser inicializados de forma aleatória considerando, por exemplo, se o nodo atual é o destino ou não, se ocorreu *timeout*, o tipo do pacote, se o pacote deve ser retirado ou não, etc.

(d) A modelagem deve cobrir todas as possibilidades de tipos de nodo, ou seja, origem, destino e nodo intermediário, sendo a escolha feita aleatoriamente no momento do recebimento do pacote.

(e) Os procedimentos auxiliares usados pelo protocolo normalmente não são incorporados à verificação. Isso é feito devido à simplicidade desejada, já que apenas o impacto dos resultados são relevantes no funcionamento do protocolo. Por exemplo, o cálculo de *Checksum* não precisa ser verificado bastando setar aleatoriamente uma variável booleana que indique se houve um erro ou não no seu cálculo.

(f) Sempre que possível deve-se abstrair procedimentos como sendo variáveis booleanas para simplificar o processamento de verificação. No entanto, como já foi discutido, isso depende de cada caso e o que se deseja verificar.

(g) As condições de erro e de exceção do protocolo devem ser mapeadas para variáveis booleanas. Desta forma pode-se facilmente encontrar a configuração em que determinado erro ocorreu.

(h) As condições externas necessárias ao funcionamento interno do protocolo devem ser mapeadas para variáveis booleanas. De forma similar ao item anterior pode-se determinar possíveis dependências do protocolo e os requisitos mínimos necessários ao seu funcionamento.

(i) As variáveis usadas na verificação devem ter um objetivo bem definido. Cada variável deve representar de forma única no protocolo um evento, condição, ação ou estado. Por exemplo: se a condição ⟨PACKETSENT⟩ é verdadeira ou falsa. Caso o protocolo possa enviar pacotes em mais de um ponto, sugere-se que seja possível indicar precisamente em que ponto o pacote foi enviado. Desta forma a identificação da condição testada será precisa.

Funcionamento Externo:

7. O funcionamento externo é basicamente a forma como o algoritmo trata a interação entre nodos. Será assumido que o passo anterior, referente ao funcionamento interno, já foi executado.

(a) Qualquer parâmetro interno ao nodo necessário à verificação externa deve ser mapeado para variáveis booleanas. Por exemplo: ⟨ RepassouPacote⟩ (sim/não), ⟨ ExisteRota)(sim/não), etc.

(b) A verificação da interação entre três nodos é suficiente para mapear todas os casos e configurações da rede. Cada nodo assume um papel, a saber: origem, destino e os possíveis nodos intermediários da rede.

(c) Mesmo que o algoritmo utilize posicionamento geográfico como é o caso do LAR [93] e DREAM [7] usados neste trabalho, os possíveis cenários que o algoritmo modela podem ser representados com os três nodos. A abstração está em que o que é modelado não é a posição geográfica do nodo e sim as possibilidades de configuração. (Um nodo está ao alcance do outro ou não, uma mensagem foi recebida pelo destino ou não, existe um caminho entre a origem e o destino ou não, etc.)

(d) O canal de comunicação é compartilhado. Todos os nodos enviam e recebem mensagens por um mesmo canal. Como os nodos estão competindo pelos mesmos pacotes, isto representa as possíveis configurações de envio e recebimento de pacotes.

(e) Caso o nodo faça envio de mensagens via *flooding* é suficiente que duas mensagens sejam colocadas na rede para cobrir os casos possíveis. Como as mensagens são colocadas no mesmo canal todas as configurações de recebimento e envio de mais de uma mensagem são também verificadas. Como se tem três nodos e duas mensagens todas as situações possíveis são testados pela ferramenta de verificação.

(f) No caso de se encontrar uma falha na verificação do protocolo, o cenário encontrado deve ser analisado identificando as condições que levaram ao problema. Deve-se verificar se as condições são válidas para o funcionamento do algoritmo. O objetivo é ter certeza que o processo de modelagem está correto e há realmente uma falha no protocolo.

(g) A mensagem deve conter apenas campos necessários à verificação do protocolo, sendo que eles devem ser inicializados aleatoriamente.

Implementação Prática:

8. Implemente o algoritmo em algum simulador apropriado e de amplo uso na comunidade científica, tal como o NS [1].

9. Teste todas as propriedades, conjunto de propriedades ou situações verificadas do algoritmo, em cenários aleatórios e específicos (normalmente aqueles cenários que a ferramenta de *Model Checking* retornou como contra-exemplo ou aqueles, conhecidos pelo usuário, que certamente deveriam validar firmemente o comportamento do

protocolo para uma determinada situação), para comparar com os resultados encontrados no processo de verificação.

(a) Compare cada situação e veja se está de acordo com o comportamento prático.
    (a) Se todas as propriedades ou situações verificadas estão de acordo com o que foi verificado, implica que o processo de verificção está coerente e correto e que sua implementação está apropriada.
    (b) Senão
        (a) Revise a implementação feita, até que se esteja seguro de sua corretude.
        (b) Revise todo o processo de modelagem efetuado.

(c) Teste os contra-exemplos gerados na prática
    (a) Se os contra-exemplos gerados se comportam, na prática, como o esperado implica que o processo de verificação está coerente e correto e que sua implementação está apropriada.
    (b) Senão
        (a) Revise a implementação feita, até que se esteja seguro de sua corretude.
        (b) Revise todo o processo de modelagem efetuado.

Capítulo 4

Protocolos Validados

Este Capítulo descreve os protocolos escolhidos para a validação da metodologia. Foram escolhidos três protocolos para redes móveis *ad hoc*: LAR1 e LAR2 [93], e DREAM [7], que são baseados em roteamento geográfico. Um aspecto interessante é que esses três protocolos apresentam comportamentos distintos em várias situações similares, o que serviu também de teste para a metodologia proposta. Os resultados da verificação, a partir da metodologia, identificaram pelo menos um problema conhecido no LAR1 e LAR2 que é o *loop* e problemas não claramente conhecidos como *loop* no DREAM.

4.1 *Location-Aided Routing* - LAR

O *Location-Aided Routing* [93] usa informações de localização obtidas através de GPS (*Global Positioning System*) [147, 49] para descobrir uma rota para um dado destino. Estas informações permitem restringir a área de busca do nodo destino, definida de acordo com a provável localização do nodo, quando deseja-se descobrir uma nova rota, como mostra a Figura 4.1. Neste exemplo, vê-se um nodo origem O que deseja encontrar um nodo destino D qualquer. De alguma forma não-determinada, pois se trata de um exemplo genérico, o nodo O envia um mensagem para um grupo pré-determinado por posições geográficas com o intuito de que a mensagem chegue ao destino D. Seguindo o mesmo algoritmo, os nodos que recebem a mensagem seguem o mesmo princípio até que a mensagem chegue a D.

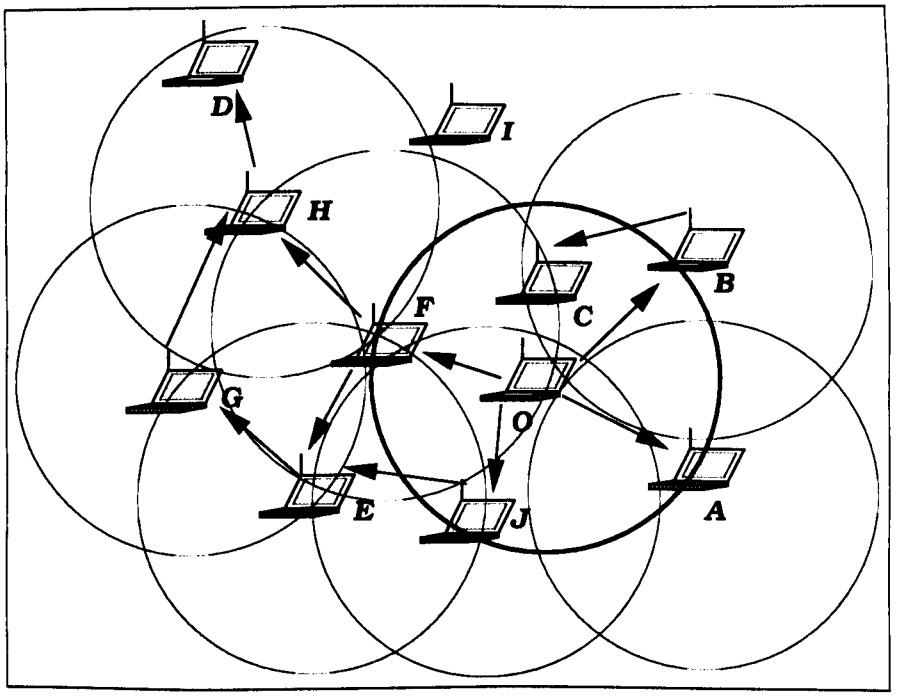

Figura 4.1: A forma de funcionamento de um algoritmo genérico que utiliza posição geográfica para encontrar o destino da mensagem.

4.1.1 Princípio de Operação

No protocolo LAR, para que os nodos-origem tenham a posição dos nodos-destino, é feita uma requisição de informações sobre a localização do nodo destino através de um

flooding puro por toda a rede. O nodo destino envia à origem do pacote a sua localização assim que recebe esta mensagem de solicitação de posicionamento. Desta forma, a tabela de posicionamento é inicializada permitindo que o processo de roteamento proposto possa ocorrer. Isto pode gerar um grande *overhead* inicial dependendo da quantidade de nodos presentes na rede.

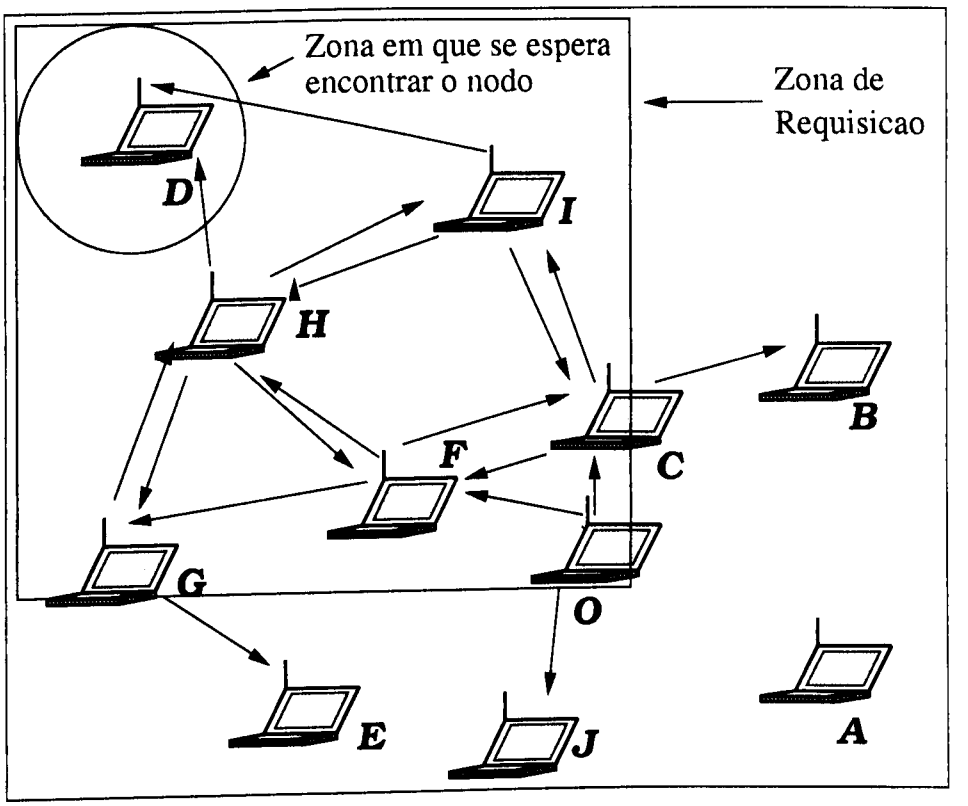

Figura 4.2: A forma de funcionamento do LAR1 na entrega de pacotes do nodo O para o nodo D.

Em [93, 94] são propostas duas variantes do LAR. A diferença básica entre as duas é a forma como o protocolo define a região (zona) provável do nodo destino se encontrar quando uma rota é solicitada. O formato da zona de requisição tem um papel importante no desempenho do protocolo como é mostrado em [93].

A primeira chamada de LAR1 usa uma zona de requisição de formato retangular que pode conter o nodo destino com alta probabilidade. O tamanho dessa área é proporcional à velocidade de movimento do nodo destino e ao tempo decorrido desde o registro da última localização do destino. Cada nodo ao receber um pacote verifica se está dentro da zona de requisição. Se estiver envia novamente a mensagem para seus vizinhos, caso contrário ignora a mensagem.

A Figura 4.2 mostra o funcionamento do LAR1 e a forma como as mensagens são entregues. Os nodos que estão dentro da zona de requisição fazem *flooding* dos pacotes de dados, até que o TTL dos pacotes expire. Os nodos de fora da região simplesmente ignoram os pacotes. Quanto maior a distância entre os nodos maior será a área de busca, podendo num caso extremo se degenerar em um flooding puro pela rede. O funciomento mais detalhado, em forma de pseudo-código, pode ser encontrado no Apêndice B.

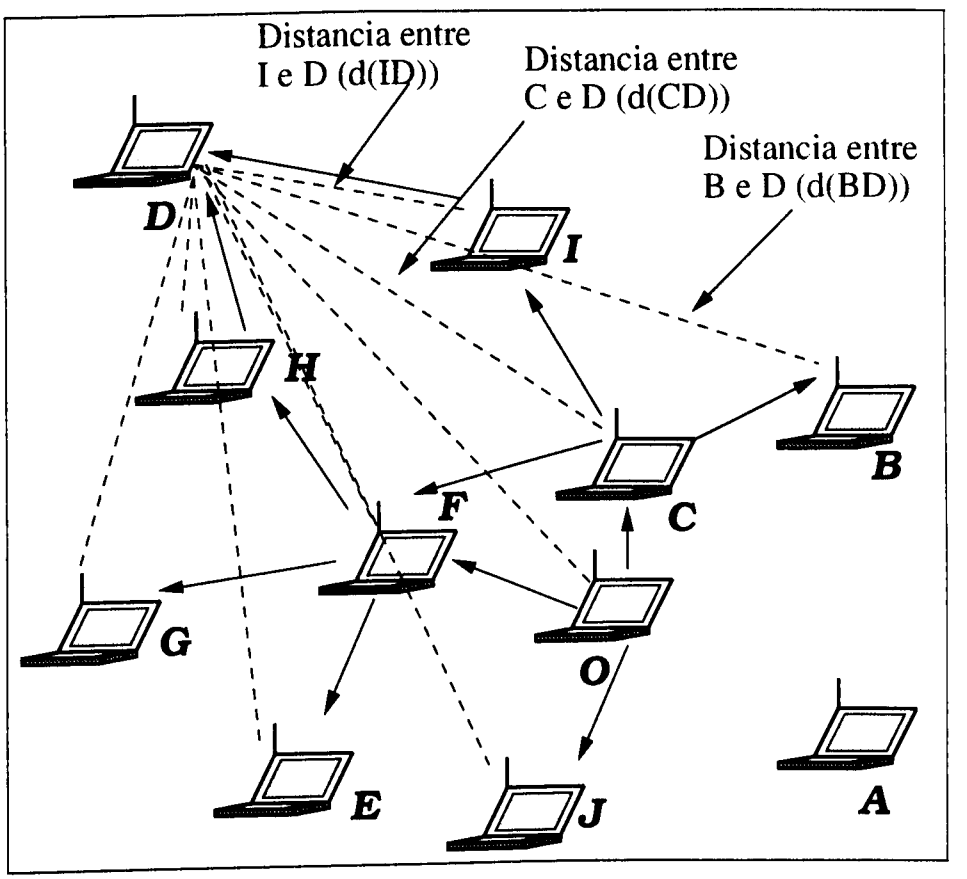

Figura 4.3: A forma de funcionamento do LAR2 na entrega de pacotes do nodo O para o nodo D.

A segunda variante chamada de LAR2 (Figura 4.3) usa uma estimativa de localização para determinar a zona mais provável onde o nodo destino pode se encontrar. Assim o LAR2 se diferencia do LAR1 porque uma mensagem possui duas informações usadas na escolha de uma zona: as coordenadas (x, y) do nodo destino e uma estimativa d de quão longe esse nodo pode estar dessas coordenadas. O valor de d é proporcional à velocidade de movimento do destino e ao tempo decorrido desde o registro da última atualização dessa posição. Essas informações são usadas em conjunto pelos nodos intermediários para determinar a zona mais provável onde o computador destino se encontra. Cada nodo ao

receber uma mensagem, nesta abordagem, verifica se está mais próximo do destino que o nodo de onde recebeu a mensagem. Se estiver, faz um *broadcast* deste pacote para seus vizinhos (envia a todos), caso contrário simplesmente ignora o pacote.

A Figura 4.3 mostra como uma comunicação se procede no LAR2. Os nodo só repassam o pacote caso estejam mais próximos do destino que o nodo que lhe enviou o pacote. No caso da comunicação entre O e D pelo caminho ⟨O, C, I, D⟩ tanto o nodo C quanto o nodo I só repassam a comunicação porque estão mais próximos do destino que o nodo anterior. A distância entre C e D (d(CB)) é menor que a distância entre O e D, e a distância entre I e D (d(IB)) por sua vez é menor que d(CB). Isso não ocorre com o nodo B, por exemplo, onde a distância entre B e D (d(BD)) é maior que d(CB). Deste modo o nodo B não repassa a informação. O funciomento mais detalhado, em forma de pseudo-código, pode ser encontrado no Apêndice C

4.1.2 Propriedades

Um grande problema com o LAR não é a exigência de que o nodo deve ter uma placa GPS, já que é um periférico relativamente barato [147], nem mesmo a falta de precisão que estes equipamentos apresentam [50], mas sim o *flooding* inicial. No entanto, existem propostas para o posicionamento sem o uso de GPS ([30]), que solucionariam esse empecilho. Para descobrir uma rota o LAR precisa de um *flooding* puro pela rede. Para redes grandes o volume de tráfego tende a crescer exponencialmente, não sendo portanto escalável.

4.2 DREAM

O protocolo *Distance Routing Effect Algorithm for Mobility* (DREAM) [7] também utiliza de informações de posicionamento geográfico, conseguidas via GPS, para efetuar o roteamento, assim com o LAR1 e LAR2. Similarmente, tais informações também têm como objetivo restringir a área de busca do nodo destino quando deseja-se descobrir uma nova rota.

4.2.1 Princípio de Operação

A tabela de posicionamento é inicializada através de uma mensagem que é difundida para toda a rede (mensagem de *flooding*) solicitando a posição do nodo. Cada destino, ao receber esta requisição, envia de volta ao nodo origem a sua posição, o que pode gerar um grande *overhead* inicial dependendo da quantidade de nodos presentes na rede. No DREAM a região de busca é triangular, onde o triângulo é formado pelos extremos da região circular (diâmetro do círculo) de maior probabilidade de se encontrar o destino, chamada de *Expected Zone* no LAR, e a posição do nodo atual. Dentro desta região de busca (*Request Zone*) não há *flooding*, pois cada nodo só repassa os dados aos seus vizinhos. Cada um dos vizinhos repete o mesmo processo, onde a sua região de busca poderá ser totalmente diferente da do nodo anterior dependendo de como estão atualizadas as suas tabelas. Esta região é caracterizada por um ângulo α de busca que é fundamental para o funcionamento do protocolo, conforme mostra a Figura 4.4. O funciomento mais detalhado, em forma de pseudo-código, pode ser encontrado no Apêndice D

4.2.2 Propriedades

O DREAM apresenta o mesmo problema notado com o LAR1 e com o LAR2, que é a não exigência de que o nodo deve ter uma placa GPS, já que é um periférico relativamente barato, nem mesmo a falta de precisão que estes equipamentos apresentam [50], mas sim o *flooding* inicial. No entanto, existem propostas para o posicionamento sem o uso de GPS ([30]), que solucionariam esse empecilho.

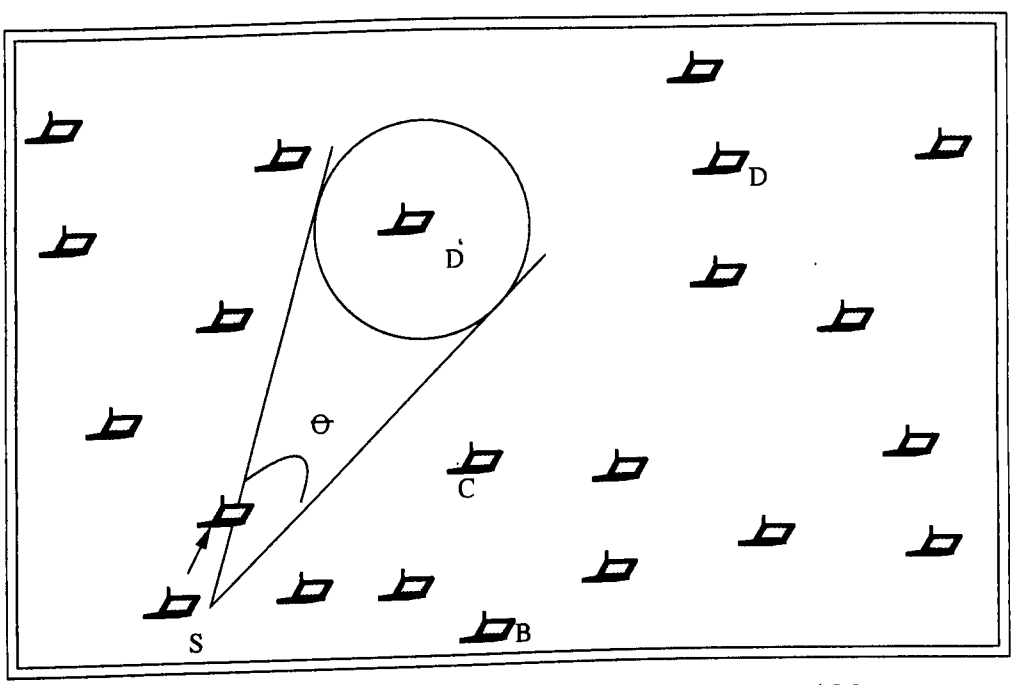

Figura 4.4: A forma de funcionamento do DREAM.

Capítulo 5

Aplicação da Metodologia

Os três protocolos descritos no Capítulo 4 foram verificados na ferramenta SPIN usando a metodologia de modelagem de protocolos para redes móveis *ad hoc*, descrita no capítulo 3. Os problemas encontrados estão relatados a seguir. Para cada problema encontrado sugere-se uma possível solução.

5.1 Ocorrência de *Loop*

Os três protocolos (LAR1, LAR2 e DREAM) podem ter *loop* no roteamento de mensagens.

5.1.1 LAR1: Interno à *Request Zone.*

O roteamento dentro da *Request Zone* é feito através de *flooding*. Nesse caso, as mensagens são retiradas da rede somente quando o seu TTL (*Time To Live*) expira. O LAR1 faz um *flooding* controlado em uma região específica. Isto significa que o pacote é descartado caso saia dessa região. Outra característica do LAR1 é que os nodos, com o objetivo de evitar *loops*, armazenam os identificadores das mensagens recebidas. Entretanto, pode haver casos em que cópias de mensagens são repassadas apenas entre os mesmos nodos internos e estas mensagens podem acabar chegando novamente em algum nodo que já a enviou. Se essa informação ("mensagem já passou por este nodo") tenha sido removida, seja por um fluxo grande mensagens na rede gerando um estouro no *buffer* que armazena estes identificadores, seja por que o tempo de permanência destas mensagens no *buffer* estourou, ou seja por qualquer outra razão de funcionamento, a mensagem será enviada novamente, gerando um *loop* de roteamento, como mostra a Figura 5.1.

5.1.1.1 *Solução*

Uma possível solução para este problema seria realizar um *flooding* controlado dentro da *Request Zone*. Uma forma simples de controlar o *flooding* seria enviar os pacotes, dentro da *Request Zone* somente para nodos em direção contrária à origem. Isto, além de inibir *loops*, poderia melhorar o desempenho do protocolo. Outra possível solução, e que não altera o funcionamento do protocolo, seria utilizar o caminho que consta na mensagem para cada nodo verificar se a mensagem já passou ou não por ele, uma vez que o trabalho em [93] omite o descarte de mensagens nesse instante.

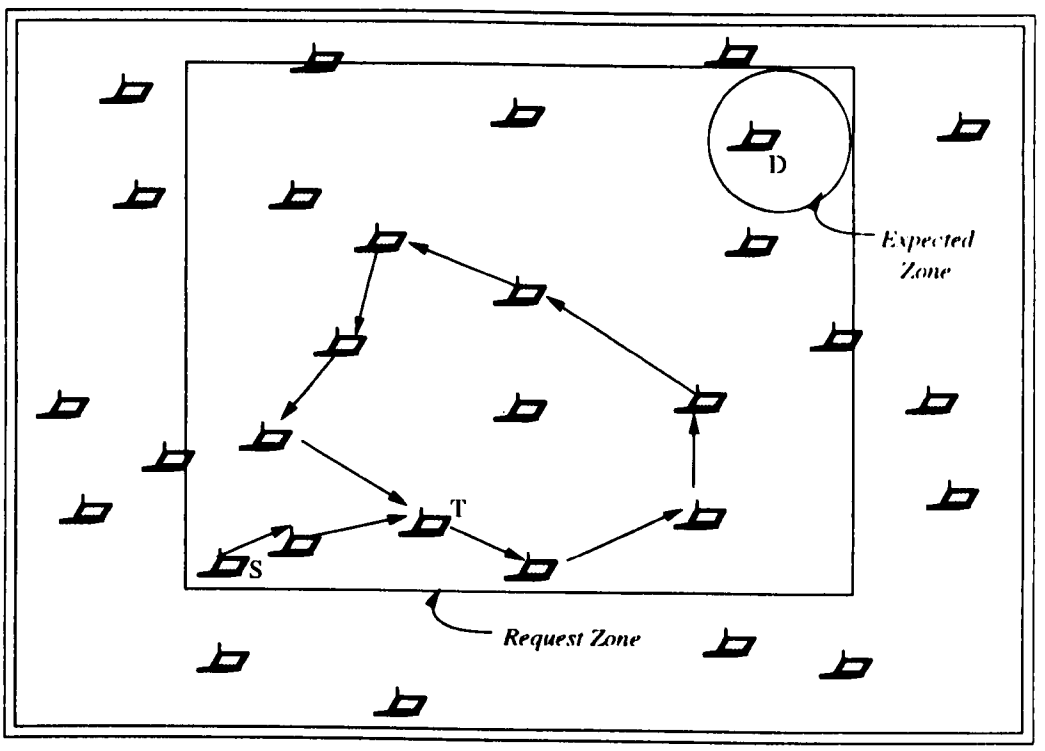

Figura 5.1: Presença de Loop no LAR1.

5.1.2 LAR2: Se nodos estão à mesma distância.

O LAR2 utiliza a distância até o destino como o parâmetro de descarte ou não de uma mensagem. Um nodo verifica se sua distancia é menor ou igual ao nodo anterior. Se for ele reenvia o pacote para seus vizinhos. Pode ocorrer que o protocolo considera que todos os nodos, devido ao fator δ, estão a uma mesma distância do destino. Isto ocorre em uma configuração onde os nodos estão posicionados em uma área circular em volta do destino. Novamente se a informação de que a mensagem já passou pelo nodo for "esquecida", por alguma situação similar àqueles mencionadas na sessão 5.1.1, pode ocorrer uma configuração tal que resultará em um *loop*. Isto é ilustrado pela figura 5.2.

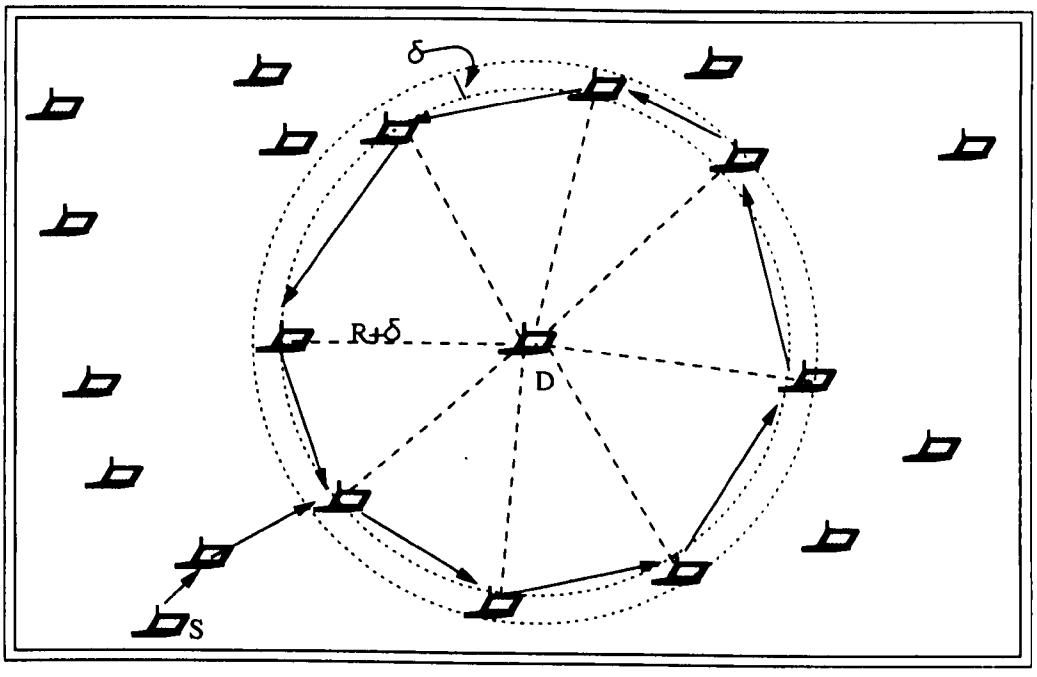

Figura 5.2: Presença de Loop no LAR2.

5.1.2.1 *Solução*

Uma solução para o problema seria utilizar a informação sobre o caminho de onde a mensagem passou para avaliar se a mensagem deve ser reenviada ou não. Evidentemente, isso geraria um *overhead* extra ao protocolo. No entanto, como o princípio de funcionamento deste algoritmo é baseado na distância entre os nodos, é possível que esse custo de se ter uma mensagem maior valha a pena pois diminuirá o número de mensagens inúteis na rede.

5.1.3 DREAM: Se ângulo de busca > 90°.

O DREAM faz uma busca em uma região triangular. Esta é a região de maior probabilidade de se encontrar o nodo, segundo um ângulo de busca. Caso este ângulo seja maior que 90° podem ocorrer *loops*. Uma vez que as tabelas de posicionamento dos nodos podem estar também desatualizadas. O *loop* pode ser simples, A → B → A, ou pode ter vários nodos intermediários, A → B → ... → A. é importante *frisar* que em [7] os autores do protocolo DREAM afirmam que o algoritmo é livre de *loops*, o que não é o caso. Através da verificação é possível caracterizar quando *loops* ocorrem no DREAM, como é mostrado na Figura 5.3.

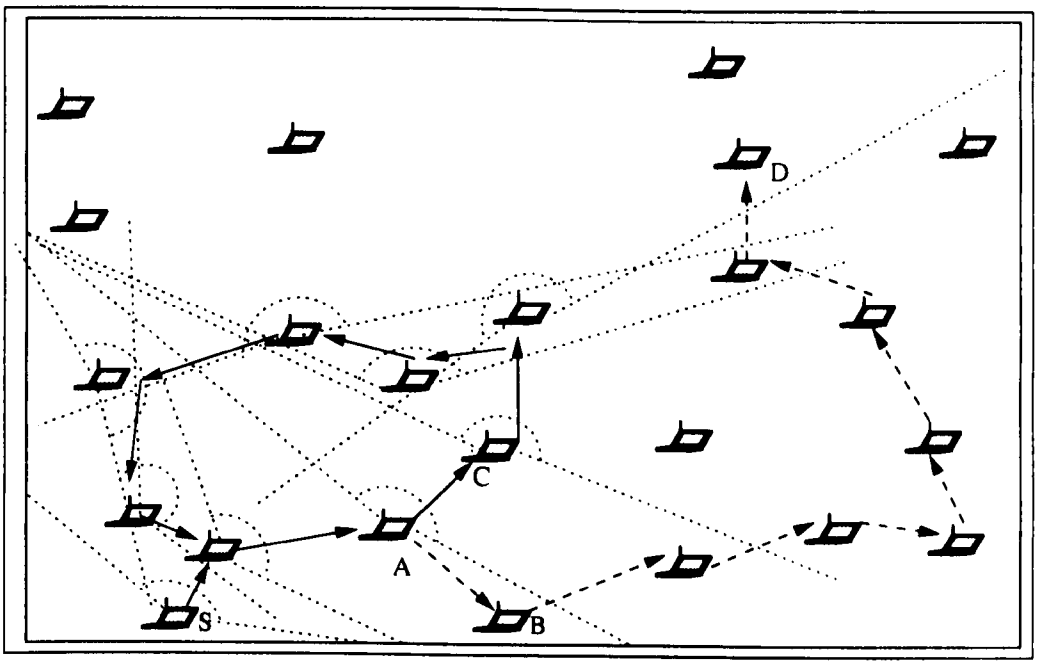

Figura 5.3: Presença de Loop no DREAM.

5.1.3.1 *Solução*

Uma solução para o problema poderia ser utilizar regiões de busca apenas com ângulos inferiores a 90°. Assim, um nodo que encontrasse uma região de busca com ângulo superior a 90° não repassaria a mensagem. Outra solução que não altera as características básicas do protocolo seria a mensagem conter em seu cabeçalho toda a rota por onde passou. Desta forma, cada nodo poderia verificar se a mensagem já passou por ele ou não. Fica claro que no segundo caso está se inserindo algum *overhead* protocolo. Como o princípio

de funcionamento deste algoritmo é baseado neste ângulo de busca e caso a rede seja por exemplo esparça isso poderia garantir que esse custo extra de se ter uma mensagem maior valha a pena pois diminuirá o número de mensagens inúteis na rede. Além disso, deve ficar claro o benefício introduzido pelo uso da metodologia (3) e que soluções mais ou menos apropriadas dependerão fundamentalmente dos objetivos específicos para os quais o protocolo foi desenvolvido.

5.1.4 Resultados do uso do SPIN para situações de *LOOP*

A tabela 5.1 apresenta as estatísticas do uso do SPIN para os casos descritos nesta sessão.

LOOP	LAR1	LAR2	DREAM
Memory Usage (MegaBytes)	1.493	1.493	1.493
Matched States	38	146	10
Stored States	91	368	28
Atomic Steps	198	644	62
Depth Reached	50	53	20
Hash Table (states)	2^{18}	2^{18}	2^{18}
State Vector (byte)	72	72	64
Hash Conflits	0	4	0

Tabela 5.1: Situações de loop de pacotes pelos três algoritmos

A primeira linha da tabela se refere à quantidade de memória utilizada para se efetuar a verificação em questão. Deve ser notado que a quantidade de memória utilizada foi pequena e a mesma para os três casos, o que indica que os modelos eram simples, similares e rapidamente se encontrou o que era desejado (violação de alguma asserção). N
ao foi necessário alocar mais espaço para o verificador construir uma árvore de verificação (Aêndice A.22) maior.

A linha seguinte nos diz o número de estados encontrados repetidos aos se ao se efetuar esta verificaão, ou seja, isto nos diz o número de estados previamente visatas na árvore de procura antes de se chegar ao objetivo (seja um erro, uma violação, etc). Vale a pena comentar que quanto maior o número de estados encontrados (*matched states*) maior foi o nível de redundância encontrada nestas condições.

A terceira linha representa o número de estados únicos gerados até que se tenha a asserção sido violada (ou alguma condição LTL).

A quarta linha nos diz o número de passos atômicos (Apêndice A.17) da verificação. Quanto maior este número menor é, geralmente, a complexidade do modelo que se está verificando [79, 77]. No entanto, pode-se, na verdade, estar abstraindo o modelo real de forma equivocada. Esse compromisso deve ser olhada com bastante critério.

A quinta linha da tabela nos mostra a profundidade alcançada pela árvore de verificações (Apêndice A.22) do verificador.

A sexta linha da tabela representa o tamanho da *Hash Table*, ou seja, representa o número máximo de estados que foi modelado o problema. Vale ressaltar que caso seja necessário mais estados, deve ser explicitamente indicar ao verificador para utilizar mais, caso necessário. No entanto, caso não seja necessário, como é o caso, devido a baixa complexidade do modelo, o verificador utilizará a menor quantidade de estados (espaço em memória) conforme possível.

A penúltima linha indica o tamanho do vetor de estados utilizado na verificação. Veja que esse vetor depende exclusivamente do modelo. Quando mais combinações de possibilidades para os valores das variáveis for possível mais estados possíveis teremos e portanto maior um vetor de estados será necessário.

A última linha mostra o número de conflitos encontrados. Quanto mais próximo de zero este número melhor. No entanto, ter esse número maior que zero não implica que o modelo está errado, apenas nos diz que o modelo tem mais de um conjunto de valores de variáveis que recaem sobre o mesmo ponto. Isso dependendo do caso, pode nos mostrar que há um ambiüidade no modelo.

5.2 Nodo Intermediário não Reenvia Mensagem

Este cenário mostra que mesmo existindo uma rota entre os nodos origem e destino a mensagem pode não ser enviada, neste caso por um nodo intermediário, devido ao princípio de funcionamento do protocolo. Com isto é possível saber limitações do uso dos protocolos, permitindo fazer uma escolha mais apropriada em função de um cenário e/ou aplicação.

5.2.1 LAR1: Nodo está fora da *Request Zone*.

No LAR1, nodos fora da zona de requisição não repassam pacotes. No entanto, um nodo fora dessa zona pode ser parte da única rota até o destino, como mostra a Figura 5.4.

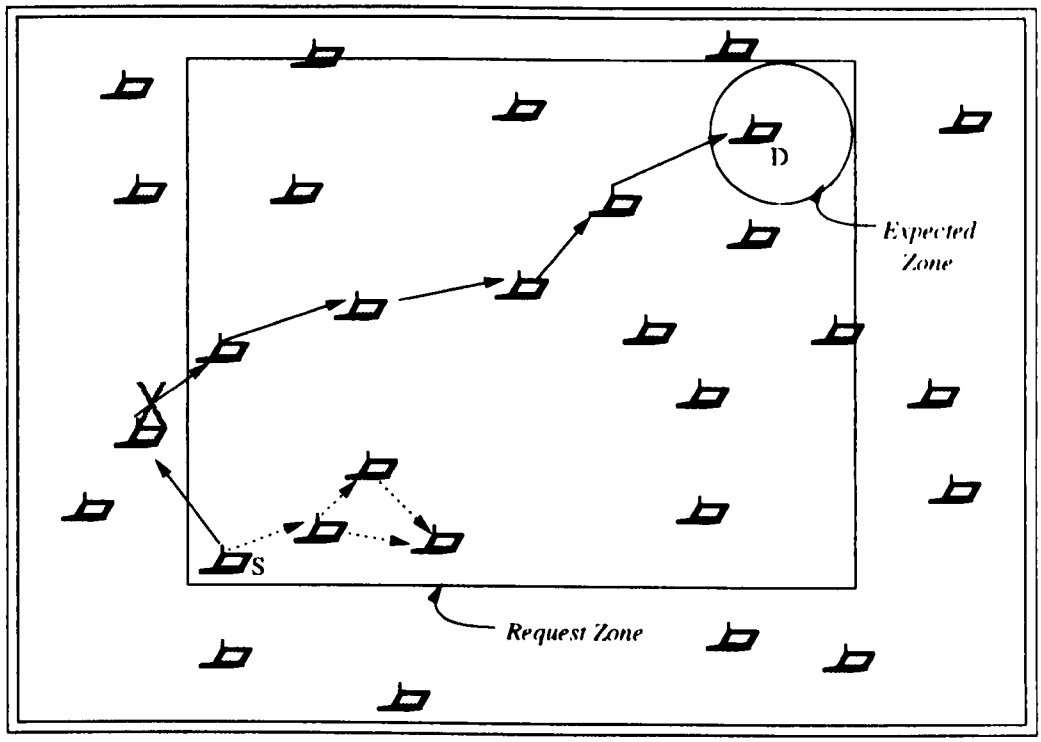

Figura 5.4: Situação de não-reenvio de pacotes no LAR1.

5.2.1.1 *Solução*

Uma solução poderia ser o nodo origem aumentar incrementalmente o tamanho da zona de requisição, até eventualmente chegar a um *flooding* puro, caso não haja resposta referente àquele pacote. Esta é uma proposta de solução adaptativa que os protocolos de roteamento para redes móveis *ad hoc* propostos na literatura não tratam.

5.2.2 LAR2: O nodo atual está mais longe do destino do que o nodo anterior.

No LAR2 quando o nodo intermediário está mais longe do destino que o nodo anterior, de acordo com os dados presentes na mensagem, o nodo intermediário descarta a mensagem. No entanto, esta pode ser a única rota existente para o destino, caso haja concavidade na rede, como mostra a Figura 5.5.

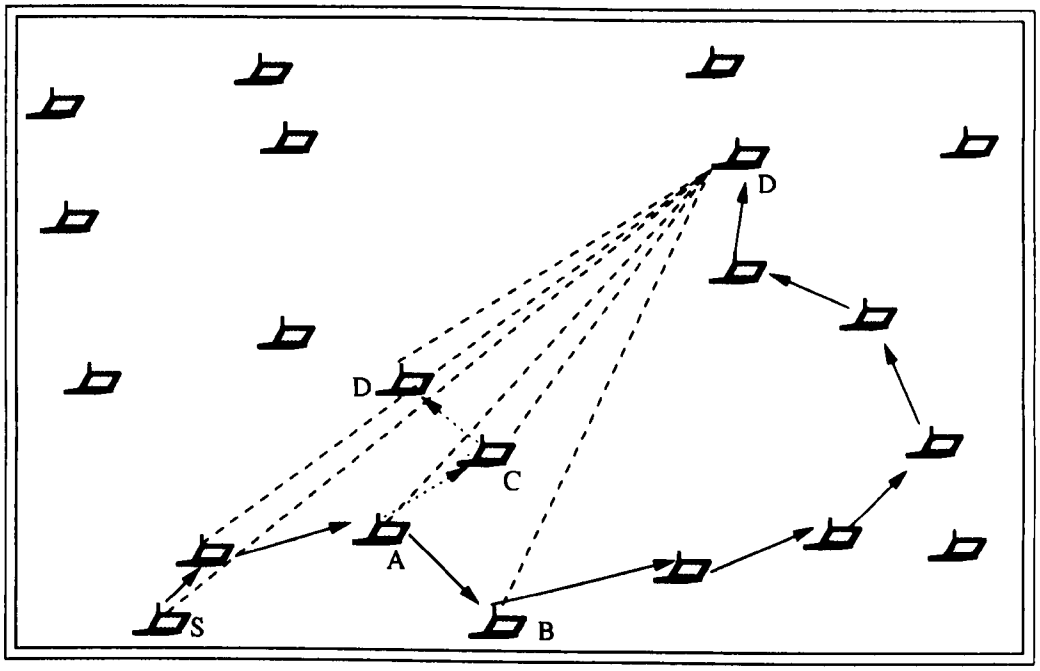

Figura 5.5: Rota Côncava no LAR2.

5.2.2.1 *Solução*

Uma solução para o problema seria o nodo origem, quando não tiver resposta para a requisição, aumentar o tamanho do parâmetro δ (lê-se *delta*) [1] associado à requisição. Outra possível solução neste caso poderia ser um *flooding* puro, só que esta é uma solução

[1] O parâmetro δ (lê-se *delta*), tal como mostrado na Figura 5.2 é um limiar de erro na variação da distância na qual cada nodo intermediário utiliza para considerar se está ou não mais distante do destino do que o nodo anterior. Assim, um nodo intermediário decidirá se esta mais distante ao destino do que o nodo anterior (cuja distância é d até o destino), se sua distância d' ao destino recair no seguinte intervalo $]d - \delta, d + \delta[$, equivalente a $d - \delta \leq d' \geq d + \delta$. Maiores detalhes podem ser encontrados em [93, 94]

extremamente custosa. Mas dependendo dos requisitos necessários ele pode compensar em alguns casos como por exemplo alguns quesitos de QoS.

5.2.3 DREAM: Não existe nodo vizinho na região triangular.

No DREAM o nodo intermediário não repassa a mensagem quando não há vizinhos (dentro de uma área de alcance) na região de busca. Porém pode haver uma rota que não foi avaliada por ser uma rota côncava por exemplo.

5.2.3.1 *Solução*

Uma solução para este problema, caso não haja um nodo vizinho no cone de pesquisa, o nodo origem aumentar o ângulo de busca gradualmente até que algum vizinho passasse a fazer parte do cone. Este procedimento pode gerar *loops* se fosse necessário aumentar esse ângulo acima de 90°. Isso implica num compromisso entre a possibilidade de ocorrer *loops* e a não entrega da mensagem. Novamente, outra possibilidade seria a origem fazer *flooding* caso não receba uma resposta para a requisição, como pode ser visto pela Figura 5.6.

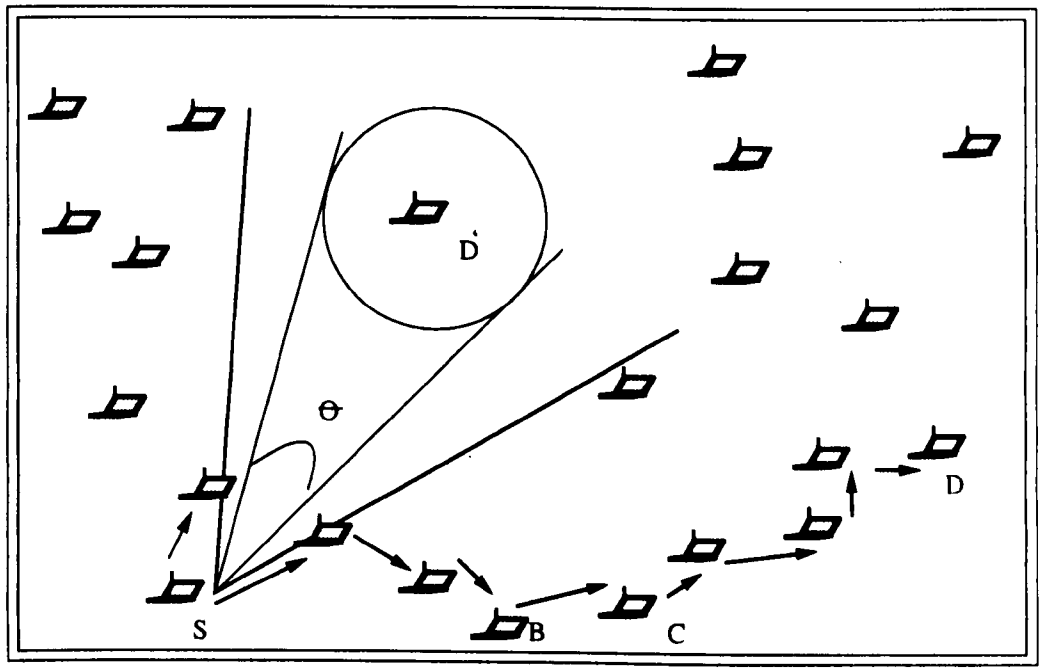

Figura 5.6: DREAM sem vizinho na área de busca.

5.2.4 Resultados do uso do SPIN para situações onde o nodo intermiário não reenvia a mensagem como deveria.

A tabela 5.2 apresenta as estatísticas do uso do SPIN para os casos descritos nesta sessão.

NÃO REENVIA	LAR1	LAR2	DREAM
Memory Usage (MegaBytes)	1.493	1.596	1.493
Matched States	240	963	1
Stored States	608	2489	4
Atomic Steps	1136	4325	10
Depth Reached	50	69	17
Hash Table (states)	2^{18}	2^{18}	2^{18}
State Vector (byte)	76	80	64
Hash Conflits	0	242	0

Tabela 5.2: Situações de não-reenvio de pacotes pelos três algoritmos

A primeira linha da tabela se refere à quantidade de memória utilizada para se efetuar a verificaão em questão. Deve ser notado que a quantidade de memória utilizada foi pequena para os três casos, o que indica que os modelos eram simples e rapidamente se encontrou o que era desejado (violação de alguma asserção). Não foi necessário alocar mais espaço para o verificador construir uma árvore de verificação (Aêndice A.22) maior.

A linha seguinte nos diz o número de estados encontrados repetidos aos se ao se efetuar esta verificaão, ou seja, isto nos diz o número de estados previamente visatas na árvore de procura antes de se chegar ao objetivo (seja um erro, uma violação, etc). Vale a pena comentar que quanto maior o número de estados encontrados (*matched states*) maior foi o nível de redundância encontrada nestas condições.

A terceira linha representa o número de estados únicos gerados até que se tenha a asserção sido violada (ou alguma condição LTL).

A quarta linha nos diz o número de passos atômicos (Apêndice A.17) da verificação. Quanto maior este número menor é, geralmente, a complexidade do modelo que se está verificando [79, 77]. No entanto, pode-se, na verdade, estar abstraindo o modelo real de forma equivocada. Esse compromisso deve ser olhada com bastante critério.

A quinta linha da tabela nos mostra a profundidade alcançada pela árvore de verificações (Apêndice A.22) do verificador.

A sexta linha da tabela representa o tamanho da *Hash Table*, ou seja, representa o número máximo de estados que foi modelado o problema. Vale ressaltar que caso seja necessário mais estados, deve ser explicitamente indicar ao verificador para utilizar mais, caso necessário. No entanto, caso não seja necessário, como é o caso, devido a baixa complexidade do modelo, o verificador utilizará a menor quantidade de estados (espaço em memória) conforme possível.

A penúltima linha indica o tamanho do vetor de estados utilizado na verificação. Veja que esse vetor depende exclusivamente do modelo. Quando mais combinações de possibilidades para os valores das variáveis for possível mais estados possíveis teremos e portanto maior um vetor de estados será necessário.

A última linha mostra o número de conflitos encontrados. Quanto mais próximo de zero este número melhor. No entanto, ter esse número maior que zero não implica que o modelo está errado, apenas nos diz que o modelo tem mais de um conjunto de valores de variáveis que recaem sobre o mesmo ponto. Isso dependendo do caso, pode nos mostrar que há um ambiüidade no modelo.

5.3 Condições de Não-entrega dos Pacotes

Este cenário mostra outras situações onde os pacotes não são entregues, mesmo existindo uma rota entre os nodos origem e destino, devido ao princípio de funcionamento do protocolo.

5.3.1 LAR1: Não Existem Nodos na *Request Zone.*

Podem existir casos, em redes tipicamente esparsas, que a *Request Zone* é pequena, e não existem nodos nessa região (veja a Figura 5.7).

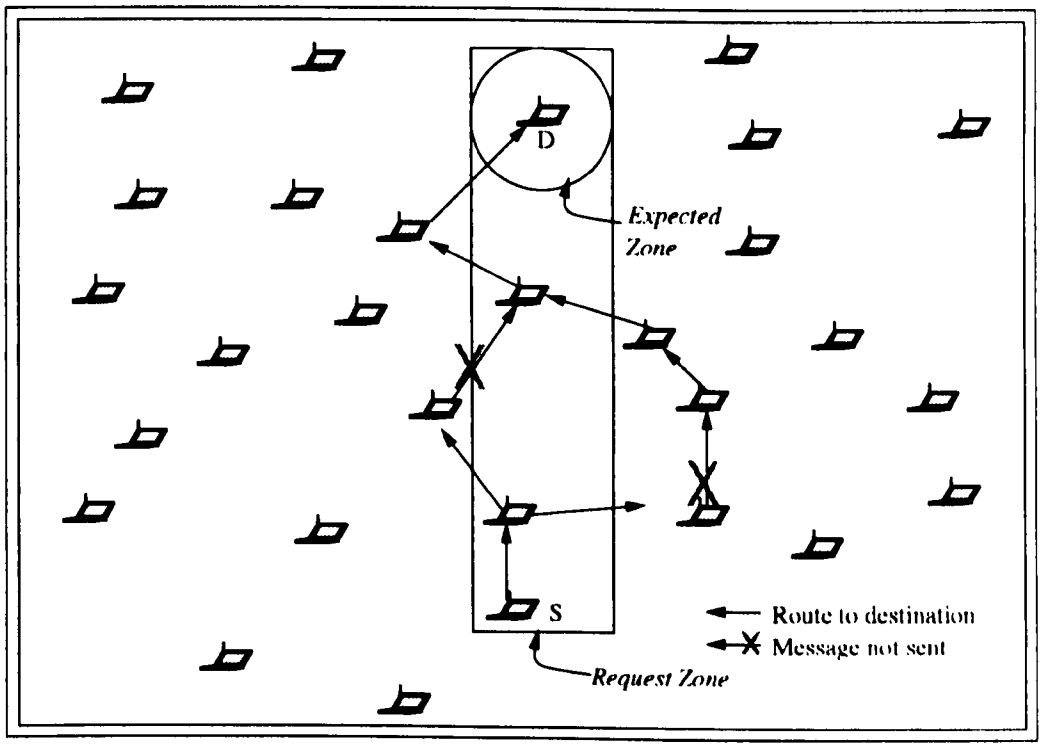

Figura 5.7: Área restrita no LAR1.

5.3.1.1 *Solução*

As soluções mais simples seriam o aumento da *Request Zone* ou, novamente, fazer um *flooding* puro. Uma outra solução que é um pouco mais interessante seria se criar uma área de busca de forma variável dependendo de como está o comportamento de movimento do nodo destino. Suponha que o nodo D, tenha se movido seguindo o padrão mostrado como pode ser visto na Figura 5.8.

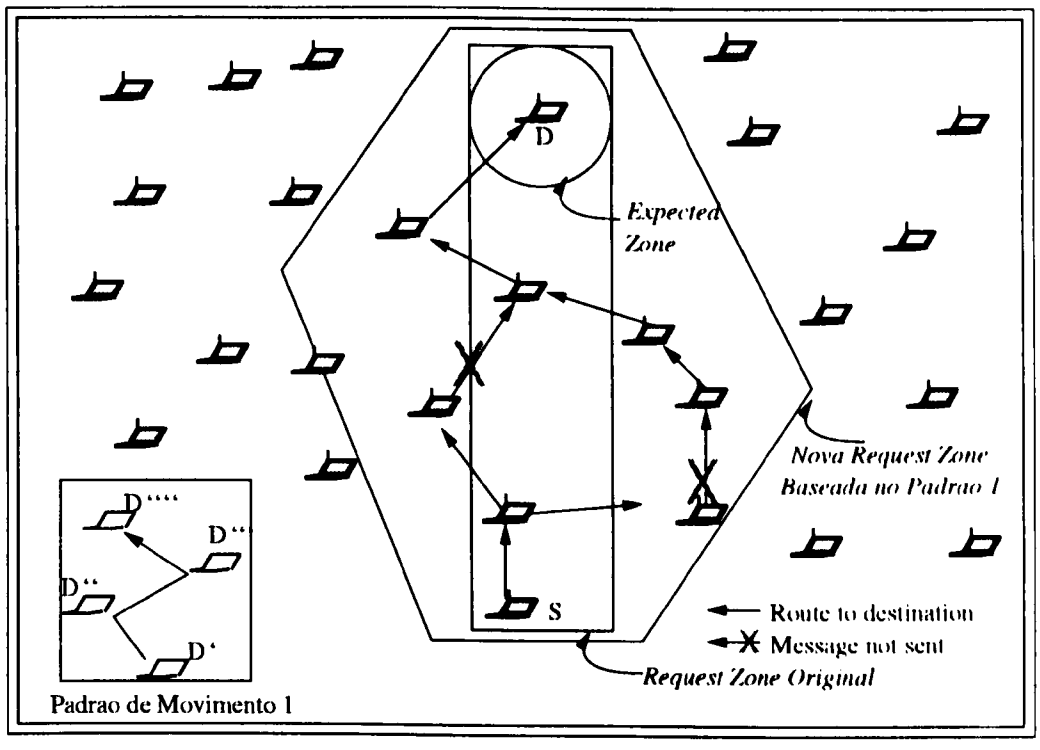

Figura 5.8: Área restrita no LAR1 - Região de busca que varia a forma, conforme os padrões de movimento.

5.3.2 LAR2: Rota Côncava.

O LAR2 não está apto a entregar os pacotes quando a única rota ao destino é côncava. Uma rota é caracterizada como côncava quando algum de seus componentes está mais distante do destino que o anterior. Alguns protocolos, como o LAR2 e o DREAM, têm a tendencia em enviar as mensagens para os nodos mais próximos do destino, mas nem sempre este princípio leva ao objetivo desejado. Este problema é análogo ao problema de busca de mínimos. Um algoritmo deve ter a capacidade de diferenciar entre mínimos locais e globais. Isto é melhor ilustrado pela Figura 5.9. Apesar de as visualizações serem similares, as situações apresentadas aqui e na sessão 5.2.2 são distintas, uma vez que a semântica delas é diferente. A situação aqui descrita é claramente um caso patológico do algoritmo enquanto a outra é um comportamento natural do protocolo.

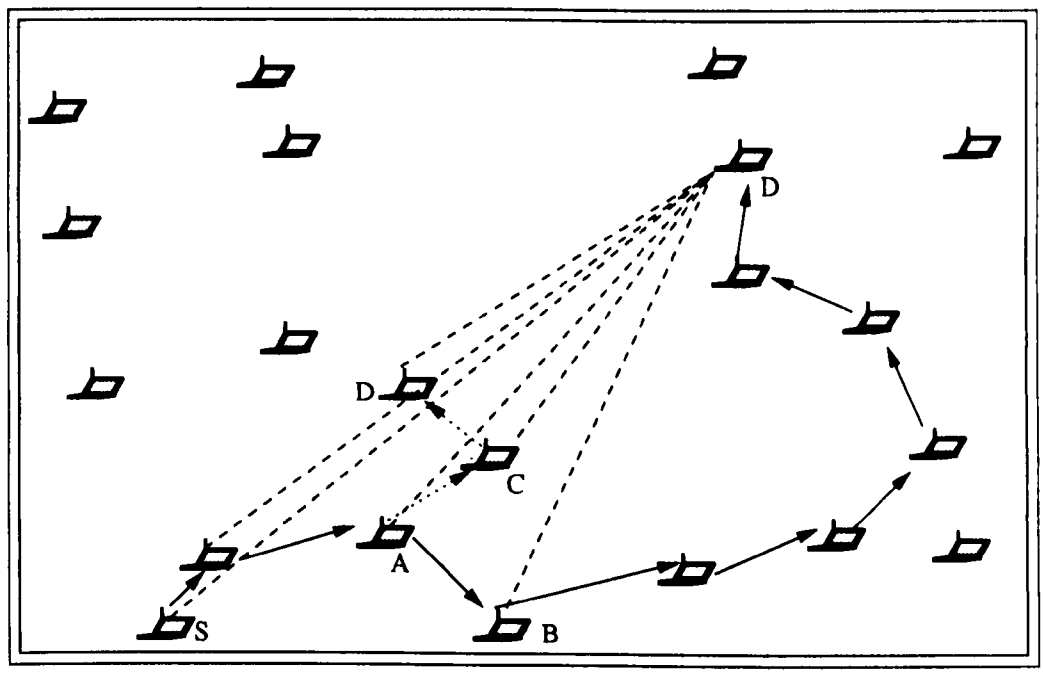

Figura 5.9: Rota Côncava no LAR2.

5.3.2.1 *Solução*

A origem, ao não receber resposta da requisição poderia aumentar o δ (delta), ou fazer *flooding* puro.

5.3.3 DREAM: Rota Côncava.

O DREAM também sofre com o problema de rotas côncavas. Porém, do ponto de vista do DREAM rotas côncavas são um caso particular do seu problema principal. Basta o nodo não estar na direção do destino que não fará parte da rota. Desta forma, diversas rotas válidas poderão não ser encontradas, como ilustrado na Figura 5.10.

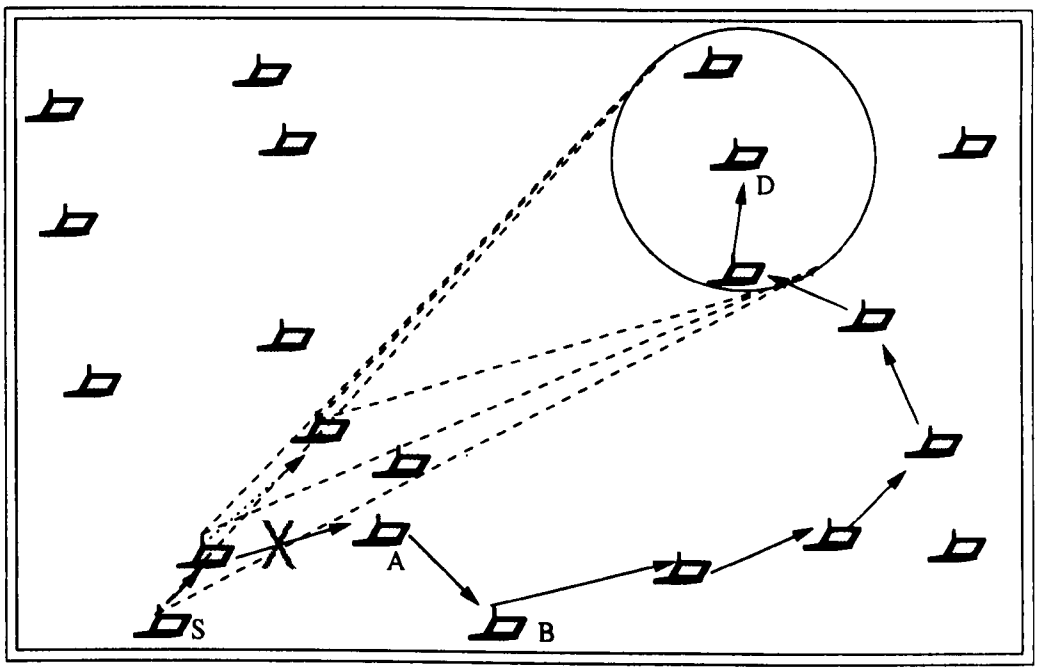

Figura 5.10: Rota Côncava no DREAM.

5.3.3.1 *Solução*

Uma solução que não altera o comportamento do protocolo seria o aumento gradual do ângulo de busca nos nodos intermediários ou na origem. Caso o aumento ocorra no nodo intermediário ele se daria quando não existissem vizinhos no cone de busca. Caso fosse na origem, ele se daria se houvesse uma temporização na requisição. A primeira solução pode apresentar um menor custo, porém podem existir casos que não serão cobertos. A segunda solução embora possa ter um custo maior, já que mais nodos serão envolvidos e todo o processo deveria ser reiniciado, cobre um número maior de possibilidades.

5.3.4 Resultados do uso do SPIN para situações não há a entrega de pacotes pelo algoritmo, como deveria.

A tabela 5.3 apresenta as estatísticas do uso do SPIN para os casos descritos nesta sessão.

NÃO ENTREGA	LAR1	LAR2	DREAM
Memory Usage (MegaBytes)	1.493	1.493	1.493
Matched States	0	0	0
Stored States	2	2	4
Atomic Steps	33	35	10
Depth Reached	35	37	17
Hash Table (states)	2^{18}	2^{18}	2^{18}
State Vector (byte)	72	72	64
Hash Conflits	0	0	0

Tabela 5.3: Situações de não-entrega de pacotes pelos três algoritmos

A primeira linha da tabela se refere à quantidade de memória utilizada para se efetuar a verificaão em questão. Deve ser notado que a quantidade de memória utilizada foi pequena para os três casos, o que indica que os modelos eram simples e rapidamente se encontrou o que era desejado (violação de alguma asserção). Não foi necessário alocar mais espaço para o verificador construir uma árvore de verificação (Aêndice A.22) maior.

A linha seguinte nos diz o número de estados encontrados repetidos aos se ao se efetuar esta verificaão, ou seja, isto nos diz o número de estados previamente visatas na árvore de procura antes de se chegar ao objetivo (seja um erro, uma violação, etc). Vale a pena comentar que quanto maior o número de estados encontrados (*matched states*) maior foi o nível de redundância encontrada nestas condições.

A terceira linha representa o número de estados únicos gerados até que se tenha a asserção sido violada (ou alguma condição LTL).

A quarta linha nos diz o número de passos atômicos (Apêndice A.17) da verificação. Quanto maior este número menor é, geralmente, a complexidade do modelo que se está verificando [79, 77]. No entanto, pode-se, na verdade, estar abstraindo o modelo real de forma equivocada. Esse compromisso deve ser olhada com bastante critério.

A quinta linha da tabela nos mostra a profundidade alcançada pela árvore de verificações (Apêndice A.22) do verificador.

A sexta linha da tabela representa o tamanho da *Hash Table*, ou seja, representa o número máximo de estados que foi modelado o problema. Vale ressaltar que caso seja necessário mais estados, deve ser explicitamente indicar ao verificador para utilizar mais, caso necessário. No entanto, caso não seja necessário, como é o caso, devido a baixa complexidade do modelo, o verificador utilizará a menor quantidade de estados (espaço em memória) conforme possível.

A penúltima linha indica o tamanho do vetor de estados utilizado na verificação. Veja que esse vetor depende exclusivamente do modelo. Quando mais combinações de possibilidades para os valores das variáveis for possível mais estados possíveis teremos e portanto maior um vetor de estados será necessário.

A última linha mostra o número de conflitos encontrados. Quanto mais próximo de zero este número melhor. No entanto, ter esse número maior que zero não implica que o modelo está errado, apenas nos diz que o modelo tem mais de um conjunto de valores de variáveis que recaem sobre o mesmo ponto. Isso dependendo do caso, pode nos mostrar que há um ambiüidade no modelo.

Capítulo 6

Conclusões e Trabalhos Futuros

A verificação formal é uma etapa muito importante no desenvolvimento de qualquer algoritmo, e em particular, de algoritmos distribuídos como protocolos de comunicação. Este trabalho propõe uma metodologia inovadora para verificar protocolos de roteamento para redes móveis *ad hoc*. O maior desafio vencido foi apresentar uma solução eficiente para o problema da modelagem de topologias dinâmicas como é o caso de MANETs.

A metodologia foi aplicada aos protocolos LAR1, LAR2 e DREAM para redes móveis ad hoc. Os resultados encontrados mostraram que a metodologia foi capaz de encontrar erros já conhecidos e erros não esperados nesses protocolos. Através da verificação formal também foi possível identificar cenários onde os protocolos não irão funcionar devido aos princípios utilizados. Isso não indica situações de erro mas limitações que devem ser conhecidas pelo projetista do protocolo e seus usuários.

A metodologia proposta simplificou efetivamente o processo de verificação ao abstrair a topologia e representá-la por três nodos. Isto foi possível modelando as interações possíveis entre os nodos, através da troca ou não de mensagens, ao invés de se preocupar com a posição física de cada unidade móvel. Desta forma a metodologia leva a resultados com características qualitativas e não quantitativas dos protocolos. Este é um ponto importante porque independente de características quantitativas (por exemplo, desempenho) o protocolo deve ser correto em relação a propriedades qualitativas (por exemplo, ausência de *loop*).

Outro ponto interessante do processo é que a verificação é feita usando no máximo duas mensagens entre os nodos. Isso evita que a árvore de verificação cresça exponencialmente tornando o problema de difícil solução ou mesmo intratável. Uma mensagem é suficiente para modelar qualquer fluxo de mensagens entre os nodos, exceto o caso de *flooding* que é

tratado por duas mensagens. Neste caso, como existem três nodos na rede e a cada instante apenas um nodo está enviando mensagens, restam apenas dois nodos para receberem tais mensagens, um representando o destino e o outro os nodos intermediários. Assim, ao se enviar duas mensagens garante-se que todos os nodos da rede estão recebendo a mesma mensagem, modelando-se o processo de *flooding*. Em protocolos de roteamento que utilizam *flooding* controlado o processo é o mesmo, mas o controle deve ser feito de acordo com o algoritmo em questão.

A separação do processo de verificação em "nodo interno" e "nodo externo" também simplificou o problema e permitiu alcançar a abstração desejada. Usando esta estratégia é possível encontrar falhas na descrição ou no processamento interno de cada nodo antes mesmo de tentar caracterizar o comportamento do protocolo na rede. Ao se passar para a interação entre os nodos, ponto fundamental de cada algoritmo, o comportamento interno do nodo é abstraído simplificando a modelagem e verificação do protocolo.

A metodologia proposta não trata de questões quantitativas como número mínimo e máximo de mensagens enviadas pela rede, tamanho de rota, consumo de energia pelo nodo, e latência da rede. Essas são questões de pesquisa que estão em aberto e que serão estudadas como continuação deste trabalho. Também pretende-se aplicar a metodologia proposta aqui a outros protocolos para redes móveis ad hoc [27].

O trabalho descrito nesta dissertação pode ser estendido em várias direções. Em relação a resultados de protocolos existem vários protocolos que seriam extremamente interessantes caso fossem verificados utilizando-se a metodologia proposta, em especial aqueles não-geográficos.

Uma tarefa interessante a ser realizada deveria ser a a formalização da metodologia em uma linguagem formal como Z, ou através de técnicas como aplicadas em HOL [114], uma vez que teríamos um melhor embasamento para se utilizar a metodologia. Esta tarefa permitiria uma maior "segurança" na utilização da metodologia, uma vez que ela também poderá ser formalmente verificada. O objetivo é garantir a ela uma semântica precisa, clara e bem definida, livre de ambiguidades e/ou eventuais inconscistências, mesmo que estes problemas não tenham sido detectados na metodologia, e os resultados encontrados mostram que a metodologia se apresenta sem problemas estruturais.

A metodologia ainda não trata totalmente de questões probabilísticas e essa seria certamente uma extensão muito interessante, pois poderiam estudar o comportamento uma grande nova gama de protocolos ainda não contemplados.

Algo bastante útil para a metodologia seria a integração com alguma forma de se efetuar uma análise de desempenho. Um solução interessante seria o desenvolvimento de uma ferramenta que integrasse um model-checker, um simulador e um provador de teorema, baseados no princípio da metodologia aqui proposta. Assim, poder-se-ia efetuar não somente a análise qualitativa ou de correção, bem como a quantitativa ou de desempenho

simultaneamente e ainda comparar os seus *tradeoffs*.

Outra ferramenta relevante que poderia ser desenvolvida seria um gerador automático de contra-exemplos a partir dos traces gerados pelo verificador de modelos (*model-checker*).

No campo da engenharia de software, poderia-se criar uma ferramenta que permitisse a criação de casos de uso (*use-cases*) reusáveis de forma que protocolos que apresentam semelhanças poderiam se beneficiar de "partes" do algoritmo já devidamente verificadas e analisadas. Isso permitiria um melhor compromisso entre os algoritmos e suas variações.

Um direção bastante apropriada seria a criação de um novo protocolo para MANETs utilizando-se da metodologia como ferramenta de auxílio no desenvolvimento.

Apêndice A

Mini-Glosário

A.1 Variáveis Incontrolveis

São variáveis externas ao protocolo no qual não há como o algoritmo modificaá-las. É apenas possível encontrar seu valor ou estado. Um bom exemplo é o tempo.

A.2 *State-space*

Entende-se como sendo o conjunto total de estados gerados pelo grafo interno que representa a propriedade que se está sendo verificada sobre o algoritmo que estamos aplicando-a. Simplificadamente, é o conjunto total de estados que representa o protocolo em questão.

A.3 *Time to Live*

Tempo na qual um dado pacote permanecerá na rede, definido normalmente em quantidade de nós. Em estrito senso, o TTL é um contador de *hops* que indica quando o pacote deverá ser descartado pela rede.

A.4 *Protocol Data Unit*

Pacote padrão contendo o cabeçalho(*header*) e parte de dados que deve ser enviado pela rede.

A.5 MANET - *Mobile Ad hoc NETworks*

É um tipo de rede móvel, não-estrutura, onde os dispositivos computacionais são capazes de se comunicarem diretamente, sem o auxílio direto de nenhum dispositivo fixo. Eventualmente, podem existir nodos fixos na rede. Porém sua existência ou não, nã implicará na existência ou não da rede.

A.6 IETF (Internet Engineering Task Force [57])

Grupo de trabalho sem fins lucrativos que visa a padronização, organização, e estudos diversos dos mais variados temos relativos a Ciência da Computação.

A.7 *Safety*

Representa uma propriedade de um programa que afirma (*assert*) que nada de mau nunca irá acontecer, a saber, isto ocorre quando o programa conseguentemente nunca entra em um mau estado. Exclusão m'utua, ausência de *deadlock*, e corretude parcial são exemplos da propriedade *safety* [4].

A.8 *Liveness*

Representa uma propriedade de um programa que afirma (*assert*) que algo bom irá conseguentemente acontecer, a saber, isto ocorre quando o programa conseguentemente alcança um bom estado. O término ou uma eventual saída de uma sessão crítica são exemplos de propriedades *liveness* [4].

A.9 *Deadlock*

É o processo pela qual uma entidade A qualquer interrompe ou nem começ o seu processamento por estar esperando algum recurso ou mensagem que deve ser disponibilizado pela entidade B. Ao mesmo tempo, a entidade B também espera por algum recurso ou mensagem que dever ser disponibilizado pela entidade A, ocasionando um travamento das duas entidades. O *Deadlock* pode acontecer em um ciclo de entidades $A \rightarrow B \rightarrow \ldots N \rightarrow A$.

A.10 *Starvation*

É um tipo de deadlock, onde uma entidade A, "morre" ou trava por não receber os dados que fica esperando.

A.11 *Theorem Proving* ou Provadores de teoreamos

Provadores de teorema so ferramentas de especificação e verificação formal baseados em teoremas. Desta forma eles definem um conjunto de axiomas e constrõe relações entre estes axiomas. As propriedades desejadas do protocolo são então provadas matematicamente. Prova de teorema inclui um conjunto de formalismos tanto baseados em lógica, quanto no modelo adotado, o qual embasados em lógica de primeira e segunda ordem [3].

A.12 *Reachability Analysis ou Model Checking*

É um ramo da verificação formal que consiste de uma família de métodos de prova completamente automatizados que começaram a ser desenvolvidos na área acadêmica na década de 80.

A.13 *Full Automation*

Full Automation ou automação total ocorre quando há uma automatização completa de algum procedimento que desejamos efetuar. No caso de verficação formal, ocorre quando o usuário não necessita interagir quase nada com o sistema. As tarefas são, portanto, realizadas de forma automatizada.

A.14 *Flooding*

É uma forma de envio de mensagem onde o cada nodo da rede envia as mensagens recebidas, que devem ser propagadas, a todos os outros nodos da rede. Isto ocasiona um grande fluxo de mensagens na rede. O número de mensagens na rede é exponencial com o número de nodos.

A.15 Modo de operação pró-ativa

Aquele modo em que o protocolo tenta se antever às possíveis situações ou problemas que o algoritmo ir enfrentar, tentando evitar erros, mal-funciomento, perda de eficiência ou desempenho.

A.16 Modo de operação reativa

Aquele modo em que o protocolo espera que ocorra algum evento, situação ou problema para que seja tomada alguma atitude.

A.17 Passos Atômicos - *Atomic Steps*

Passos atômicos são aqueles nos quais a seqüência de execução é indivisível. Assim, não há intercalação de processos, o que gera uma garantia em relação aos valores das variáveis deste passo atômico. Seqüências atômicas são um forma de diminuir a complexidade do modelo que se está verificando, principalmente quando se trata de sistemas distribuídos. Um exemplo disso seria a transformação de modelos intratáveis em tratáveis, por exemplo, ao se definir que a manipulação de todas as variáveis locais sejam atômicas [79].

A.18 *Broken link*

Ocorre quando a comunicação entre duas entidades quaiquer é interrompida.

A.19 Processo *Top-Down*

É um processo na qual se efetua alguma análise, estudo ou trabalho a partir de um nível de maior abstração até chegar em um nível de menor abstração. Por exemplo, caso estivéssemos tratando sobre o modelo de refê OSI ([144]), começaríamos o estudo pela camada de Aplicação, passando-se para a de Apresentação, Sessão, Transporte, Rede, Enlace até chegarmos a camada Física.

A.20 Processo *Bottom-Up*

É um processo na qual se efetua alguma análise, estudo ou trabalho a partir de um nível de menor abstração até chegar em um nível de maior abstração. Por exemplo, caso estivéssemos tratando sobre o modelo de refê OSI ([144]), começaríamos o estudo pela camada de Física, passando-se para a de Enlace Rede, Transporte, Sessão, Apresentação até chegarmos a camada Aplicação.

A.21 *TimeOut*

Termo que indica o fim de um determinado perído de tempo. Este termo é utilizado quando se está esperando por um evento acontecer ou nã. Quando o evento nã ocorre em um determinado período de tempo pré-estabelecido diz-se que ocorreu *Timeout*.

A.22 Árvore de Verificações

É o grafo gerado internamente pelo verificador, de forma a representar o modelo a ser verificado. Cada nó deste grafo representa um estado possível para o modelo (um conjunto válido de valores das variáveis que compõe o modelo).

Apêndice B

Pseudo-Código do LAR1

Acrônimos Usados:

Expected Zone = Dada a velocidade eh a area em que se espera que o nodo esteja considerando o tempo que se a informacao de localizacao do nodos

Requested Zone = Um quadrado que engloba a Expected Zone e onde as mensagens de Requisicao de rota (route request) vao realmente se propagarem

Flooding Puro = Mensagem enviada por inundamento para todos os nodos da rede

Broadcast do Pacote = Envia para todos os vizinhos propando o Flooding.

```
NODO LAR1
---------
Inicio
   Espera ter mensagem no buffer
   Testa se mensagem eh para envio /* a origem do pacote*/
   Se Sim
      Verifica se tem a posio do destino na tabela de roteamento
      Se Nao
         Enquanto <> de Timout ou <> de recebeu resposta faca
            Faz Flooding puro atras da informacao;
            Espera ateh resposta do Flooding ou Timout;
            Se recebeu pacote de resposta
```

```
                Atualiza informacoes na tabela de roteamento;
            Fim_enquanto;
        Fim_Se;/* se tem a posio do destino na tabela */
        Calcula "Expected_Zone";
        Calcula "Request_Zone";
        Empacota Dados;
        Insere o pacote na tabela de pacotes recentemente recebidos;
        Envia Pacote;

    Se No /*Mensagem a ser recebida*/
        Verifica se Nodo esta na "Request Zone"
        Se Nao
            Ignora pacote;
        Se Sim
            Testa se o pacote estah na tabela de pacotes recentemente
                                                          recebidos;
            Se Sim
                Ignora pacote;
            Se Nao
                Testa se eh o destino
                Se Sim
                    Atualiza a tabela com a posicao da origem;
                    Testa se eh "Rote Request"
                    Se Nao
                        Armazena Dados;
                    Se Sim /* eh um "Route Request"
                        Calcula "Expected_Zone";
                        Calcula "Request_Zone";
                        Empacota Resposta de Requisicao;
                        Insere o pacote na tabela de pacotes recentemente
                                                        recebidos;
                        Envia Pacote;
                    Fim_Se; /*se eh "Rote Request"*/
                Se_Nao /*Nao eh o destino ->  eh um nodo intermediario*/
                    Insere o pacote na tabela de recentemente recebidos;
                    Reenvia Pacote;
                Fim_Se; /*se eh o destino*/
            Fim_Se; /*se o pacote estah na tabela de recent. recebidos*/
        Fim_Se; /*se Nodo estah na "Request Zone"*/
    Fim_Se; /*se Mensagem eh para envio*/

    Finaliza processamento do pacote;
Fim.
```

Apêndice C

Pseudo-Código do LAR2

Acrônimos Usados:

Expected Zone = Dada a velocidade é a área em que se espera que o nodo esteja considerando o tempo que se a informação de localização do nodos

Requested Zone = Um quadrado que engloba a Expected Zone e onde as mensagens de Requisição de rota (route request) vao realmente se propagarem

Flooding Puro = Mensagem enviada por inundamento para todos os nodos da rede

Broadcast do Pacote = Envia para todos os vizinhos propando o Flooding.

```
NODO LAR2
---------
Inicio
   Espera ter mensagem no buffer
   Testa se mensagem eh para envio /*eh a origem do pacote*/
   Se Sim
      Verifica se tem a posicao do destino na tabela
      Se Nao
         Enquanto <> de Timout ou <> de recebeu resposta faca
            Faz Flooding puro atras da informacao;
            Espera ateh resposta do Flooding ou Timout;
            Se recebeu pacote de resposta
```

```
                Atualiza informacoes na tabela;
           Fim_enquanto;
        Fim_Se;/* se tem a posicao do destino na tabela */
        Calcula "Expected_Zone"; /* margem de erro dada pelo &(sigma)*/
        Empacota Dados;
        Insere o pacote na tabela de recentemente recebidos;
        Envia Pacote;

     Se No /*Mensagem a ser recebida*/
        Testa se o pacote estah na tabela de recentemente recebidos
        Se Sim
           Ignora pacote;
        Se Nao
  Cadastra pacote na tabela de recentemente recebidos;
  Verifica se eh o destino da mensagem
           Se Nao
     Verifica se eh "flooding puro"
              Se Sim
        Faz Broadcast do Pacote;
              Se Nao
                 Verifica se sua distancia p/ o destino eh menor do que a
                                                    do nodo anterior;
                 Se Sim
           Substitue no pacote a distancia do nodo anterior pela
                                                       sua;
           Faz Broadcast desse novo Pacote;
                 Se Nao
                 Ignora pacote;
                 Fim_se ;/*Se sua distancia p/ o destino eh menor */
             Fim_se; /* Se eh flooding puro */
           Se Sim
        Verifica se o Pacote eh de dados
              Se Sim
                 Processa Dados;
              Se Nao /*Nao eh o destino ->  Eh um nodo intermediario*/
        Cadastra endereco do destino;
              Fim_Se; /*se eh o pacote de dados */
           Fim_Se; /* Se eh o destino da MSG*/
        Fim_Se; /*Se o pacote esta na tbl dos recentemente recebidos */
     Fim_Se; /*se Mensagem eh para envio*/

     Finaliza processamento do pacote;
  Fim.
```

Apêndice D

Pseudo-Código do DREAM

```
Inicio
  Espera aparecer msg no buffer;
  Se (mesg  de envio)
     Entao {
        Se ( tenho a posicao do destino )
             Entao
                 Se ( a posicao do destino  valida?  ) - no  velha
                     Entao
                        Calcula Requested Zone ( Regiao de Busca );
                        Se ( Requested Zone est vazia )
                            Entao Faz flooding  p/ descobrir a posicao
                                                          do destino;
                            Senao
                                Se ( mesg  de dados )
                                     Entao Atualiza TTL da mesg;
                                Faz Broadcasting da mesg;
                     Senao Faz flooding  p/ descobrir a posicao do
                                                          destino;
             Senao Faz flooding  p/ descobrir a posicao do destino;
     }

     Senao {   -  de recepo
         Se (  Sou o destino da  mesg )
             Entao
                Se ( a mesg  um ack )
```

```
                Entao  Limpa/Inicializa Timer Global;
                Senao Envia um ack p/ a origem;
        Senao
           Se ( tenho a posio do destino)
                Entao
                    Se (a posio do destino  atual? No,  velha)
                       Entao
                           Calcula Requested Zone(Regiao de Busca);
                           Se ( existe algum vizinho na Requested
                                                          Zone)
                               Entao Faz Broadcasting da mesg;
                               Senao Descarta mesg;
                       Senao Descarta mesg;
                Senao Descarta mesg;

Fim. (DREAM)
```

Apêndice E

Código em Promela do LAR 1

```
/* This modeling was created by Gedoc Study Group
   Authors: {danielc,ferreira}@dcc.ufmg.br

*/

#define true 1
#define false 0
#define TTL  3
#define range 9  /* (nnodes * nnodes)*/
#define nnodes 3 /* Intermidiates + Origin + Destination */

/*==================  Assertions to test out ====================*/

#define m0 (myid == 0)
#define m1 (myid == 1)
#define m2 (myid == 2)
#define IOT (InsideOutside == true)
#define IOF (InsideOutside == false)

#define Forwarded (Forward == true)
#define origin_isme (origin == myid)
#define destination_isme (destination == myid)
#define loop_teste (Loop == true)

/* Alterated at 29/05/00 */
```

```
#define NotDellivery (ReachDestination == false)


/* Alterated */
/* Reach More Than One Message in the Destination*/
#define receivedMTO (ReceivedMoreThanOne == true)


/* ========================================================== */

bool InsideOutside;              /* Inside Route Request Region */
bool InTheRange[range];          /* Inside the node range communication*/
                                 /* we put the matrix[nnodes][nnodes] in
                                    a unidimensional array*/
bool Loop = false;               /* Reach a routing Loop */
bool Loop_int = false;           /* Reach a routing Loop in intermediate node*/
bool RequestComplete = false;    /* Complete the Request Process */
bool ReachDestination = false;   /* Arrive at the destination */
bool ReceiveData = false;        /* The data arrived at the destination*/

bool ReceivedMoreThanOne = false; /* This flag tell us if there was more than
                                     one message to the same node */

byte myid;                       /* choose in a randomicaly way the node id*/

byte  origin;        /* Origin of the message */
byte  ttl;           /* Time to Live of the message*/
byte  destination ; /* Destinatination of the message */

bool Forward = false; /*To indicate if the message has been forwarded*/

        /* packet types */
mtype = {RouteRequest, RouteReply, DataPacket};

                         /* type,  origin, destination, TTL,  Prior node Id */
chan medium = [nnodes] of {mtype, byte,   byte,        byte, byte};

bool again;       /* This Flag is used to do the choooosing of sending once
                       or twice the message */

/* This procedure is to represent the internal behavior of the node .
   Now, we aren't using it effectively, because we think that it can
   cause a lot of overhead. But maybe in future */
/* proctype node(){
```

```
bool nothing
// there can't be a void procedure in promela

} */

/* This procedure represent the network comunications among the nodes. */
proctype Network(){

 /*
  * Local variables chose randomly
  */
  byte priorid;     /* Id of the prior node */
  byte type;        /* Type of message */
  byte a;           /* Flag Variable to identify which assertion*/

atomic {


/*
 * Choosing a random value to "again" which, in a reality,chooses
 * if it sends 1 ou 2 copies of the message. When two copies
 * are sent it is done to emulate the broadcasting by choosing "again ==true"
 */

 if
   :: (true) -> again = true
   :: (true) -> again = false
 fi;


 /*
  * Choosing a random value to my ID
  */
  if
    :: (true) -> myid = 2
    :: (true) -> myid = 1
    :: (true) -> myid = 0
  fi;

  /*
   * receive the message
   */
  medium?type(origin,destination,ttl,priorid);
```

```
printf("Pkt_type:%d Origin:%d Destination:%d TTL:%d PriorID:%d Myid:%d \n",
        type,origin,destination,ttl,priorid,myid);


/*
 * If the node is in the communication range
 */
if
  :: (InTheRange[(myid* nnodes + priorid)] == true) ->
     /*
      * Verification of the message type
      */
     if
       :: (type == RouteRequest) ->

          if   /* Im origin. Probably a loop */
            :: (origin == myid) ->

                if   /* Ignore message. Ive sent a  message to myself */
                  :: (destination == myid)
                     /* Don't continue to process the message. Loop. */
                  :: else Loop = true
                fi

               /* Im dest. I will send a Route Reply */
            :: (destination == myid) ->

                if
                  :: (ReachDestination == true)  -> ReceivedMoreThanOne = true
                  :: (ReachDestination == false) -> ReachDestination = true
                fi ->

               /* Send a Request Response */
               medium!RouteReply(destination,origin,TTL,myid) ->

                /*
                 * Sending the message again or not
                 */
                if
                 :: (again == true)  ->
                       /* Send a Request Response */
                       medium!RouteReply(destination,origin,TTL,myid)
                 :: (again == false) -> again = false /* doing nothing */
                fi
```

```
      /*Im a intermediate node. I have to forward the packet */
    :: else

      if   /* TTL haven't expired. So, we can forward the message */
        :: (ttl != 0)  ->
            if
              :: (Forward ==  true) -> Loop_int = true
              :: (Forward == false) -> Forward = true ->
                    medium!type(origin,destination,(ttl - 1),myid)->
             /*
              * Sending the message again or not
              */
              if
                :: (again == true)  ->
                   medium!type(origin,destination,(ttl - 1),myid)
                :: (again == false) -> again = false /* doing nothing */
              fi ->

              run Network()

            fi
           /* TTL have expired. We need ignore the message */
        :: else -> a = 1 ;
      fi

  fi /* message type is a route request */

:: (type == RouteReply)   ->

  if    /* Im dest. The message was processed succesfully */
        /* The route request process  was completed*/
    :: (destination == myid) -> RequestComplete = true

        /*Im origin of the message. Probably a loop*/
    ::  (origin == myid) ->

      /*
       * Verify the TTL to know if a loop or not
       */
      if   /* If the TTL was changed, it is a loop*/
        :: (ttl != TTL) -> Loop = true

            /*I read the packet that I ve just put on the pipe*/
        :: else -> a = 2
      fi /* If Im the origin */
```

```
      /*Im a intermiate node.  Forward the packet*/
    :: else

      if   /*if the message is not expired and the node is
             inside of the Route Request Region, we have to
             forward the message */
        :: ((ttl != 0) && (InsideOutside == true)) ->
             if
                :: (Forward == true)  -> Loop_int = true
                :: (Forward == false) -> Forward = false ->
                     medium!type(origin,destination,(ttl - 1),myid)->

                    /*
                     * Sending the message again or not
                     */
                     if
                       :: (again == true)  ->
                          medium!type(origin,destination,(ttl - 1),myid)
                        :: (again == false) -> again = false /* doing no
                     fi ->

                     run Network()
             fi
          /* I can't forward the packet*/
        :: else -> a = 3
      fi

  fi /* if the messase is a Route Reply */

:: (type == DataPacket) ->
  if
       /*Im origin. Probably a loop*/
    :: (origin == myid) ->

      if    /*Ignore message. Ive sent a message to myself
              or I heard a message that I sent*/
         :: (ttl == TTL)

            /* A problem has occured. I have to ignore the message*/
         :: else Loop = true /*Ignore message*/
      fi /* if Im the origin */

       /*Im dest. I will receive and treat the packet*/
    :: (destination == myid) -> ReceiveData = true
```

```
            /*Im a intermediate node.  Forward the packet */
         :: else ->

            if    /*if the message is not expired and the node is
                     inside of the Route Request Region
                     I have to forward the message*/
               :: ((ttl != 0) && (InsideOutside == true)) ->
                    if
                       :: (Forward == true) -> Loop_int = true
                       :: (Forward == false)-> Forward = true->
                           medium!type(origin,destination,(ttl - 1),myid)->

                        /*
                         * Sending the message again or not
                         */
                         if
                           :: (again == true)  ->
                               medium!type(origin,destination,(ttl - 1),myid)
                           :: (again == false) -> again = false /* doing nothin
                         fi ->

                         run Network()

                    fi
                 /* A problem has occured. I have to ignore the message*/
               :: else -> a = 4 /*Ignore message*/
            fi

       fi /* If the packet is a data packet */

    fi /* Verification of the message type*/

      /* I am not in the range. So Im not hearing the message */
    :: else -> a = 5
  fi  /* the sender is in my range */
 } /* atomic */
}/* Network */

init{

  byte i;         /* counter */
  byte desl ;     /* shift the row, because we put the
                     Matrix of nodes in a unidimensional way*/
  byte Pkt_type; /* Type of the packet */
```

```
int Origin, Destination; /* Origin and Destination
                            read from channel */

atomic{

  atomic{
    /*
     * We are force the node be in his own transmission range
     */
    i=0;
    desl=0;
    do
      :: (i < range) -> InTheRange[i + desl] = true ->
                        i = i + nnodes ->
                        desl = desl + 1
      :: (i >= range) -> break
    od;

    /*
     * we are setting up randomly the others values of the range
     */
    i = 1; /* it starts in 1 because 0 is a position
              of the first node to himself */
    do
      ::  (InTheRange[i] == 0) ->
           if
             :: (true) -> InTheRange[i] = 0
             :: (true) -> InTheRange[i] = 1
           fi  ->
           if
             :: (i == 8) -> break
             :: (i != 8) -> i = i +1
           fi
      :: (InTheRange[i] == 1) ->
           if
             :: (i == 8) -> break
             :: (i != 8) -> i = i +1
           fi
    od;
  } /* atomic */

  /*
   * Choosing a random value to my Origin
   */
```

```
 if
   :: (true) -> Origin = 0
   :: (true) -> Origin = 1
   :: (true) -> Origin = 2
 fi;

/*
 * Choosing a random value to my Destination
 */
if
   :: (true) -> Destination = 0;
   :: (true) -> Destination = 1;
   :: (true) -> Destination = 2;
fi;

/*
 * Choosing a random value to the Packet Type
 */

 if
    :: (true) -> Pkt_type = 1  /* Route Request */
    :: (true) -> Pkt_type = 2  /* Route Reply   */
    :: (true) -> Pkt_type = 3  /* Data Packet   */
 fi;

/* DEBUG */
 printf("\tPkt_type:%d Origin:%d Destination:%d TTL:%d PriorID:%d \n",
          Pkt_type,   Origin,   Destination,   TTL,   Origin);

/*
 * Start the processing. First send a message to anyone
 * and continue after that. We are using only one message
 * and only one intermediate node because we think that with
 * only one i-node we can represent a lot of i-nodes.
 */
atomic {medium!Pkt_type(Origin,Destination,TTL,Origin) ->

        /*
         * Sending the message again or not
         */
         if
           :: (again == true)  ->
               medium!Pkt_type(Origin,Destination,TTL,Origin)
           :: (again == false) -> again = false /* doing nothing */
         fi ->
```

```
            run Network()}
  } /* atomic */
}/* init */
```

Apêndice F

Código em Promela do LAR 2

```
/* This modeling was created by Gedoc Study Group
   Authors: {ferreira,danielc}@dcc.ufmg.br

*/

#define true 1
#define false 0
#define TTL  3
#define range 9  /* (nnodes * nnodes)*/
#define nnodes 3 /* Intermidiates + Origin + Destination */

/*==================  Assertions to test out ====================*/

#define notDellivery (ReachDestination == false)

#define loopInt (Loop_int == true)
#define loop (Loop == true)

/* Reach More Than One Message in the Destination*/
#define receivedMTO (ReceivedMoreThanOne == true)

/* ============================================================ */

bool InTheRange[range];          /* Inside the node range communication*/
                                 /* we put the matrix[nnodes][nnodes] in
```

```
                                    a unidimensional array*/
bool Loop = false;                 /* Reach a routing Loop */
bool Loop_int = false;             /* Reach a routing Loop in intermediate node*/
bool RequestComplete = false;      /* Complete the Request Process */
bool ReachDestination = false;     /* Arrive at the destination */
bool ReceiveData = false;          /* The data arrived at the destination*/

bool ReceivedMoreThanOne = false; /* This flag tells us if there was more than
                                     one message to the same node */

byte myid;                    /* choose in a randomicaly way the node id*/

byte  origin;          /* Origin of the message */
byte  ttl;             /* Time to Live of the message*/
byte  destination ;    /* Destinatination of the message */

bool Forward = false; /*To indicate if the message has been forwarded*/

        /* packet types */
mtype = {RouteRequest, RouteReply, DataPacket};

                        /* type,  origin, destination, TTL,  Prior node Id  */
chan medium = [nnodes] of {mtype, byte,   byte,        byte, byte        };

bool again;       /* This Flag is used to do the choooosing of sending once
                     or twice the message */

bool InsertIntoRRT = false; /* Insert into the Recently Received Table */

bool SentMyself = false; /* this flag indicates if I've sent a message to myself *
bool TTLExpired = false; /* this flag indicates if TTLExpired anytime or not */

bool DistDestLTMe;  /* this flag indicates if my distance is LT the previous node
bool MesgSenttoTrash = false;  /* this flag indicates if a message was
                                  discharted or not */

bool OutOfRange = false; /* this flag indicates I'm in the range of the node who
                            just sent the message */
/* ============================ NETWORK ============================ */
/* This procedure represent the network comunications among the nodes. */
proctype Network(){

 /*
  * Local variables chose randomly
  */
```

```
  byte priorid;     /* Id of the prior node */
  byte type;        /* Type of message */
  byte a;           /* Flag Variable to identify which assertion*/

atomic {

/*
 * Choosing a random value to "again" which, in a reality,chooses
 * if it sends 1 ou 2 copies of the message. When two copies
 * are sent it is done to emulate the broadcasting by choosing "again ==true"
 */

 if
   :: (true) -> again = true
   :: (true) -> again = false
 fi;

/* Choosing if I am more distant or not to the destination */

 if
   :: (true) -> DistDestLTMe = true
   :: (true) -> DistDestLTMe = false
 fi;

 /*
  * Choosing a random value to my ID
  */
  if
    :: (true) -> myid = 2
    :: (true) -> myid = 1
    :: (true) -> myid = 0
  fi;

  /*
   * receive the message
   */
  medium?type(origin,destination,ttl,priorid);

  printf("Pkt_type:%d Origin:%d Destination:%d TTL:%d PriorID:%d Myid:%d \n",
          type,origin,destination,ttl,priorid,myid);

  /*
   * If the node is in the communication range
   */
  if
```

```
:: (InTheRange[(myid* nnodes + priorid)] == true) ->
  /* Verifies message type  */
  if
    :: (type == RouteRequest) ->
      if   /* Im origin. Probably a loop */
        :: (origin == myid) ->

            if   /* Ignore message. Ive sent a  message to myself */
              :: (destination == myid) -> SentMyself = true;
                 /* Don't continue to process the message. Loop. */
              :: else Loop = true
            fi

           /* Im dest. I will send a Route Reply */
        :: (destination == myid) ->

            if
              :: (ReachDestination == true)  -> ReceivedMoreThanOne = true
              :: (ReachDestination == false) -> ReachDestination = true
            fi ->

           /* Send a Request Response */
           medium!RouteReply(destination,origin,TTL,myid) ->

            /* Sending the message again or not */
             if
              :: (again == true)  ->
                    /* Send a Request Response */
                    medium!RouteReply(destination,origin,TTL,myid)
              :: (again == false) -> again = false /* doing nothing */
             fi

           /*Im a intermediate node. I have to forward the packet */
        :: else

          if   /* TTL haven't expired. So, we can forward the message */
            :: (ttl != 0)  ->
               /* Verify if my distance to the destination is LT or not
                  to the previous node */
               if
                 :: (DistDestLTMe == true) ->
                   if
                     :: (Forward ==  true) -> Loop_int = true
                     :: (Forward == false) -> Forward = true ->
                        medium!type(origin,destination,(ttl - 1),myid)->
```

```
                /* Sending the message again or not */
                 if
                   :: (again == true)  ->
                      medium!type(origin,destination,(ttl - 1),myid)
                   :: (again == false) -> again = false
                 fi ->
                  run Network()
              fi
               /* Message sent to the trash. Im not nearer than
                  previous node.*/
            :: else MesgSenttoTrash = true;
         fi
            /* TTL have expired. We need ignore the message */
      :: else -> TTLExpired = true ;
   fi

fi /* message type is a route request */

:: (type == RouteReply)   ->

 if    /* Im dest. The message was processed successfully */
       /* The route request process was completed*/
   :: (destination == myid) -> RequestComplete = true

       /*Im origin of the message. Probably a loop*/
   ::  (origin == myid) ->

     /*Verify the TTL to know if is a loop or not */
     if   /* If the TTL was changed, it is a loop*/
       :: (ttl != TTL) -> Loop = true
          /*I read the packet that I ve just put on the pipe*/
       :: else -> SentMyself= true;
     fi /* If Im the origin */

     /*Im a intermiate node.  Forward the packet*/
   :: else
     if   /*if the message is not expired so we have look if node is
            inside LT the preveious node so that we have to
            forward the message */
       :: (ttl != 0) ->
          if
            :: (DistDestLTMe == true) ->
               if
                  :: (Forward == true)  -> Loop_int = true
                  :: (Forward == false) -> Forward = false ->
```

```
                        medium!type(origin,destination,(ttl - 1),myid)->
                       /*Sending the message again or not*/
                        if
                          :: (again == true)  ->
                             medium!type(origin,destination,(ttl - 1),myi
                          :: (again == false) -> again = false
                        fi ->
                        run Network()
                  fi;
                 /* Message sent to the trash. Im not nearer than
                    previous node.*/
              :: else MesgSenttoTrash = true;
            fi
            /* TTL have expired. We need ignore the message */
            /* I can't forward the packet*/
         :: else -> TTLExpired = true ;
       fi
  fi /* if the messase is a Route Reply */

:: (type == DataPacket) ->
  if
        /*Im origin. Probably a loop*/
    :: (origin == myid) ->

       if    /*Ignore message. Ive sent a message to myself
               or I heard a message that I sent*/
          :: (ttl == TTL) -> SentMyself =true;

             /* A problem has occured. I have to ignore the message*/
          :: else Loop = true /*Ignore message*/
       fi /* if Im the origin */

        /*Im dest. I will receive and treat the packet*/
    :: (destination == myid) -> ReceiveData = true

         /*Im a intermediate node.  Forward the packet */
    :: else ->

      if   /*if the message is not expired so we have look if node is
             inside LT the preveious node so that we have to
             forward the message */
          ::(ttl != 0) ->
            if
              :: (DistDestLTMe == true) ->
                 if
```

```
                        :: (Forward == true) -> Loop_int = true
                        :: (Forward == false)-> Forward = true->
                           medium!type(origin,destination,(ttl - 1),myid)->
                         /*Sending the message again or not*/
                          if
                            :: (again == true)  ->
                                medium!type(origin,destination,(ttl - 1),myid)
                            :: (again == false) -> again = false
                          fi ->
                          run Network()
                      fi;
                     /* Message sent to the trash. Im not nearer than
                        previous node.*/
                   :: else MesgSenttoTrash = true;
                fi
                 /* A problem has occured. I have to ignore the message*/
                 /* TTL have expired. We need ignore the message */
             :: else -> TTLExpired = true ;
          fi

       fi /* If the packet is a data packet */

    fi /* Verification of the message type*/

      /* I am not in the range (Im not hearing the message). So I can't
         process the message */
    :: else -> OutOfRange = true;
   fi  /* the sender is in my range */
 } /* atomic */
}/* Network */


/* ========================================================================= */
init{

  byte i;         /* counter */
  byte desl ;     /* shift the row, because we put the
                     Matrix of nodes in a unidimensional way*/
  byte Pkt_type; /* Type of the packet */

  int Origin, Destination; /* Origin and Destination
                              read from channel */

  atomic{
```

```
atomic{
  /*
   * We are forcing the node be in his own transmission range
   */
  i=0;
  desl=0;
  do
    :: (i < range) -> InTheRange[i + desl] = true ->
                          i = i + nnodes ->
                          desl = desl + 1
    :: (i >= range) -> break
  od;

  /*
   * we are setting up randomly the others values of the range
   */
  i = 1; /* it starts in 1 because 0 is a position
            of the first node to himself */
  do
    ::  (InTheRange[i] == 0) ->
          if
            :: (true) -> InTheRange[i] = 0
            :: (true) -> InTheRange[i] = 1
          fi  ->
          if
            :: (i == 8) -> break
            :: (i != 8) -> i = i +1
          fi
    :: (InTheRange[i] == 1) ->
          if
            :: (i == 8) -> break
            :: (i != 8) -> i = i +1
          fi
  od;
} /* atomic */

/*
 * Choosing a random value to my Origin
 */
if
  :: (true) -> Origin = 0
  :: (true) -> Origin = 1
  :: (true) -> Origin = 2
fi;
```

```
/*
 * Choosing a random value to my Destination
 */
if
   :: (true) -> Destination = 0;
   :: (true) -> Destination = 1;
   :: (true) -> Destination = 2;
fi;

/*
 * Choosing a random value to the Packet Type
 */

 if
    :: (true) -> Pkt_type = 1  /* Route Request */
    :: (true) -> Pkt_type = 2  /* Route Reply   */
    :: (true) -> Pkt_type = 3  /* Data Packet   */
 fi;

/* DEBUG */
 printf("\tPkt_type:%d Origin:%d Destination:%d TTL:%d PriorID:%d \n",
          Pkt_type,   Origin,   Destination,   TTL,   Origin);

/*
 * Start the processing. First send a message to anyone
 * and continue after that. We are using only one message
 * and only one intermediate node because we think that with
 * only one i-node we can represent a lot of i-nodes.
 */

/*
 * Choosing whether it will be send a flooding or not
 */
if
    :: (true) -> again = true
    :: (true) -> again = false
fi;

atomic {medium!Pkt_type(Origin,Destination,TTL,Origin) ->

        /*
         * Sending the message again or not
         */
         if
           :: (again == true)  ->
```

```
                    medium!Pkt_type(Origin,Destination,TTL,Origin)
                  :: (again == false) -> again = false /* doing nothing */
                fi ->
                run Network()}

  } /* atomic */

}/* init */
```

Apêndice G

Código em Promela do DREAM

```
/*==================  Assertions to test out ====================*/

#define p  (loop==true)
#define notp  (loop==false)
#define q  (nnodes!=3)
#define t  (trash == 1)
#define sa  (sendAck == true)
#define dm  (duplicatedMessage == true)
#define ip  (IgnoredPacket == true)
#define ipna  ((IgnoredPacket == true) && (NoNeighborInArea == true))


/*=============================================================*/

/* This modeling was created by Gedoc Study Group
   Authors: {ferreira,danielc}@dcc.ufmg.br

*/

#define true   1
#define false  0
#define TTL    9
#define range  9  /* (nnodes * nnodes)*/
#define nnodes 3 /* Intermidiates + Origin + Destination */

/* ========================================================== */
```

```
/* ======================== MESSAGE FORMAT + CHAIN ======================== */


                          /* type,  origin, destination, TTL , Angle */
chan medium = [nnodes] of {mtype, byte,   byte,        byte, byte };


/* ========================= GLOBAL VARIABLES ========================= */

        /* packet types */
mtype = {AckPacket, DataPacket};


byte myid;            /* choose in a randomicaly way the node id*/

byte  origin;        /* Origin of the message */
byte  ttl;           /* Time to Live of the message*/
byte  destination ; /* Destinatination of the message */

byte destinationTimestamp; /* the age of the destination information */
byte angle  ;        /* the angle (alpha) in the region to the destination */

bool again;       /* This Flag is used to do the choooosing of sending once
                     or twice the message */
byte previousAngle; /* The flooding angle of the previous node */

bool sentAck = false ; /* This flag indicate if the ack was sent or not */

bool duplicatedMessage = false; /* This flag indicate if was sent a
                                   duplicated message */

bool HaveDestPosition; /* indicates if I have the destination position*/

bool loop = false ; /* this flag indicates that occured a loop */

bool IgnoredPacket = false ; /* indicates that the node doesnt have
                                the dest position*/

bool NoNeighborInArea = false; /* No neighbor in the flooding angle area */
```

```
bit trash = 0;
/* This procedure is to represent the internal behavior of the node .
   Now, we aren't using it effectively, because we think that it can
   cause a lot of overhead. But maybe in future */
/* proctype InternalNode(){

bool nothing

// there can't be a void procedure in promela

} */

/* ============================ NETWORK ========================== */
/* This procedure represent the network comunications among the nodes. */
proctype Network(){

 /*
  * Local variables chose randomly
  */
  byte priorid;     /* Id of the prior node */
  byte type;        /* Type of message */
/*  byte trash = 0;  */ /* necessary by if conditional structure */

atomic {

/* We will chose al parameters of the node in a random way.
- The ID
- The routing information age (destination timestamp)
- The flooding angle
   All this variables are chose in this way to guarantees that
all cases will be covered by the formal verification.

We chose as sample, to represent all  possibles angles the angles
 30, 45, 90, 135, 180.


   We split the timestamp in tree ranges. The first is assinged with
the angle 30 degrees, this represents new information. The second
range is assigned to angles between 30 and 90 degrees, that represents
old information, and the third range angles between 90 and 180
degrees, this represent the oldest information possible at the nodes.

   We made a simplification, with relationtip the angle calculation. We
will consider the age of the information directly proportional to the
flooding angle. So we are fixing the r, of the formula, and providing
```

directly the angle.

We will work with relative positions. If the previous node send the packet with the angle 30 degrees, this means that the previous node is between the actual node and the destination. The previous node is relatively before the actual node. If the degree is 90, the previous can stay at the same distance to the origin, at the same "level", no one is after the other. If the angle is greater than 90 the previous node can stay after the actual node. This means that the node could sent the message to back.

we can assume that node will be in the range because the DREAM always sends a message to its neigbours

```
 */

 /*
  * receive the message
  */

 medium?type(origin,destination,ttl,previousAngle)->

 /*  DEBUG  */
  printf("NET: Pkt_type:%d Origin:%d Destination:%d TTL:%d Myid:%d PreviousAngle:

/*
 * Choosing my ID. We put the restiction in (myid =0) to avoid loop
 *  if the angle is less than 90 degrees.As explained before.
 */
 if
   :: (previousAngle >= 90) -> myid = 0
   :: (true) -> myid = 1
   :: (true) -> myid = 2
   :: else if
           :: (true) -> myid = 1
           :: (true) -> myid = 2
         fi
 fi->


 if
   :: (myid == 0) -> loop = true
   :: else trash=1;
 fi->
```

```
if
    /* Im the destination */
  :: (myid == destination ) ->
       if
         :: (sentAck == true) -> duplicatedMessage = true
         :: (sentAck == false) -> sentAck = true
       fi

    /* Im the intermiate node */
  :: else ->

 if
   :: (HaveDestPosition == true) ->

      /*
       * Chose the information age
       */
       if
         :: (true) -> destinationTimestamp = 1
         :: (true) -> destinationTimestamp = 2
         :: (true) -> destinationTimestamp = 3
       fi->

      /*
       * Chose the angle
       */
       if
         :: (destinationTimestamp == 1) -> angle = 30
         :: (destinationTimestamp == 2) ->
           if
              :: (true) -> angle = 30
              :: (true) -> angle = 45
              :: (true) -> angle = 90
           fi
         :: (destinationTimestamp == 3) ->
           if /* one angle sample of each case */
            :: (true) -> angle = 30
              :: (true) -> angle = 90
              :: (true) -> angle = 135
              :: (true) -> angle = 180
           fi
         :: else trash = 2
        fi->
```

```
 if /* if exist any neighbor in the angle area */

      /* not found*/
   :: (true) -> NoNeighborInArea = true ->
        IgnoredPacket = true

      /* found nodes at the area */
   :: (true) -> medium!type(origin,destination,(TTL-1),angle)

 fi

      :: (HaveDestPosition == false)-> IgnoredPacket = true

    fi /* havedestposition == true or false*/

  fi/* Im the destination */

  }/* atomic */

}/* Network */

/* ============================ INIT ========================== */

init{

  byte i; /* counter */

  byte dest;  /* shift the row, because we put the
                 Matrix of nodes in a unidimensional way*/

  byte Pkt_type; /* Type of the packet */

  byte Origin, Destination; /* Origin and Destination
                               read from channel */
  byte DestinationTimestamp, Angle; /* The timestamp of the destination
       information, flooding Angle */

  atomic {

/*
 * We have fixed the Origin and the Detination to simplify
 * the model and its analisis. This doesnt represent a lost of generality.
 */
     Origin      = 0;
     Destination = 2;
```

```
/*
 * Chose the information age
 */
 if
   :: (true) -> DestinationTimestamp = 1
   :: (true) -> DestinationTimestamp = 2
   :: (true) -> DestinationTimestamp = 3
 fi->

/*
 * Chose the angle
 */
 if
   :: (DestinationTimestamp == 1) -> Angle = 30
   :: (DestinationTimestamp == 2) ->
      if
         :: (true) -> Angle = 30
         :: (true) -> Angle = 45
         :: (true) -> Angle = 90
      fi
   :: (DestinationTimestamp == 3) ->
      if /* one angle sample of each case */
 :: (true) -> Angle = 30
         :: (true) -> Angle = 90
         :: (true) -> Angle = 135
         :: (true) -> Angle = 180
      fi
   :: else trash = 2
 fi->

  /*
   * Choosing a random value to the Packet Type
   */

   if
      :: (true) -> Pkt_type = 1  /* AckPacket  */
      :: (true) -> Pkt_type = 2  /* Data Packet   */
   fi->

  /* DEBUG */
   printf("\tPkt_type:%d Origin:%d Destination:%d TTL:%d Angle:%d",
             Pkt_type,   Origin,   Destination,   TTL, Angle ) ->

   medium!Pkt_type(Origin,Destination,TTL,Angle) ->
```

```
        run Network()
 }/* atomic */
}/* init */
```

Bibliografia

[1] Ucb/lbnl/vint network simulator project. http://www-mash.cs.berkeley.edu/ns/ns.html, May 2000.

[2] Mart'in Abadi and Leslie Lamport. Composing specifications. volume 15, pages 73–132, May 1993.

[3] A. Ahmed and G. Helmy. Systematic test sysnthesis for multipoint protocol design. University of Southern California, 1999.

[4] Greg Andrews. Foundations of multithreaded, parallel, and distributed programming - glossary, Jan. 2004.

[5] Stephen Austin and Graeme Parkin. Formal methods: A survey. Technical report, National Physical Laboratory, Teddington, Middelsex, UK, March 1993.

[6] L. Barroca and J. McDermid. Formal methods: Use and relevance for the development of safety critical systems. *The Computer Journal*, 35(6), December 1992.

[7] S. Basagni et al. A distance routing effect algorithm for mobility (DREAM). In *4th ACM/IEEE MOBICOM*, pages 76 84, Dallas, Texas, USA, October 25 30 1998.

[8] Phil Belanger and Wim Diepstraten. Mac entity: Mac basic access mechanism privacy and access control. Technical report, IEEE 802.11, March 1996.

[9] Bhargav Bellur, Richard G. Ogier, and Fred L. Templin. Topology broadcast based on reverse-path forwarding (tbrpf). Internet Draft, 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-tbrpf-00.txt.

[10] K. Bhargavan, D. Obradovic, and C.A. Gunter. Formal verification of standards for distance vector routing protocols. http://www.cis.upenn.edu/ hol/papers/rip.ps, February 2000.

[11] Karthikeyan Bhargavan, Satish Chandra, Peter J. McCann, and Carl A. Gunter. What packets may come: Automata for network monitoring. In *Proceedings of the Symposium on Principles of Programming Languages (POPL'01)*, pages 206 219. ACM Press, January 2001.

[12] Karthikeyan Bhargavan, Carl A. Gunter, Elsa L. Gunter, Michael Jackson, Davor Obradovic, and Pamela Zave. The Village Telephone System: A case study in formal software engineering. In Jim Grundy and Malcolm Newey, editors, *Theorem Proving in Higher Order Logics 11th International Conference TPHOLs '98*, volume 1479 of *Lecture Notes in Computer Science*, pages 49 66, Canberra, Australia, September 1998. Springer.

[13] Karthikeyan Bhargavan, Carl A. Gunter, Moonjoo Kim, Insup Lee, Davor Obradovic, Oleg Sokolsky, and Mahesh Viswanathan. Verisim: Formal analysis of network simulations, August 2000. International Symposium on Software Testing and Analysis.

[14] Karthikeyan Bhargavan, Carl A. Gunter, and Davor Obradovic. Fault origin adjudication, August 2000. Formal Methods in Software Practice.

[15] Karthikeyan Bhargavan, Carl A. Gunter, and Davor Obradovic. RIP in SPIN/HOL, August 2000. Theorem Provers for Higher-Order Logics.

[16] Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter. Formal verification of standards for distance vector routing protocols, 2002. Journal of the ACM.

[17] Mark Billinghurst and Thad Starner. Wearable devices: New ways to manage information. *Computer*, 1(1):57–64, January 1999.

[18] Jonathan Bowen. Formal methods in safety-critical standards. In *Software Engineering Standards Symposium (SESS'93)*, pages 168 177. Brighton, UK, IEEE Computer Society Press, 30. August – 3. September 1993.

[19] Jonathan Bowen and Michael Hinchey. Formal methods and safety-critical standards. *IEEE Computer*, pages 69–71, August 1994.

[20] Jonathan Bowen and Michael Hinchey. Ten commandments of formal methods. Technical Report 350, University of Cambridge, Computer Laboratory, September 1994.

[21] Jonathan Bowen and Victoria Stavridou. Formal methods and software safety. In Heinz Frey, editor, *Safety of Computer Control Systems 1992 (SAFECOMP'92)*, pages 93–98. Pergamon Press, 28-30 October 1992.

[22] Jonathan Bowen and Victoria Stavridou. Safety-critical systems, formal methods, and standards. *IEE/BCS Software Engineering Journal*, 8(4):189 209, July 1992. Also available as Technical Report PRG-TR-5-92, Oxford University Computing Laboratory, UK.

[23] Jonathan P. Bowen and Michael G. Hinchey. Seven more myths of formal methods. *IEEE Software*, 12(3):34–41, July 1995.

[24] J.C. Bradfield and C. Stirling. *Modal Logics and mu-calculi: and introduction.* Elsevier Sciense, 2001.

[25] Levente Buttyán, Jean-Pierre Hubaux, and Srdjan Capkun. A formal model of rational exchange and its application to the analysis of syverson's protocol.

[26] James L. Caldwell. Formal methods technology-transfer: a view from nasa. In S. Gnesi and D. Latella, editors, *Proceedings of the ERCIM Workshop on Formal Methods for Industrial Critical Systems*, Oxford England, March 1996.

[27] Daniel Câmara and Antnio A.F. Loureiro. Roteamento em redes Móveis ad hoc. Minicurso da *XVII Jornada de Atualização em Informática, Congresso da SBC'99*, Julho 1999.

[28] Daniel Câmara and Antnio A.F. Loureiro. A novel routiing algorithm for ad hoc networks. 33rd Hawaii International Conference on System Sciences (HICS) , Maui, USA, January 2000.

[29] S. Campos, E. Clarke, and M. Minea. The verus tool: a quantitative approach to the formal verification of real-time systems. *Conference on Computer Aided Verification*, 1997.

[30] Srdan Capkun, Maher Hamdi, and Jean-Pierre Hubaux. GPS-free positioning in mobile ad-hoc networks. In *HICSS*, 2001.

[31] Centro de estudos avançados de sistemas. http://www.cesar.org.br/analise/$n_0$9/*metodos.html*, *June*2000.

[32] Centro de estudos avançados de sistemas. http://www.cesar.org.br/analise/$n_0$9/*arndt.html*, *June*2000.

[33] E. Clarke and J. Wing. Formal methods: State of the art and future direction. Technical Report CMU-CS-96-178, Carnegie Mellon University, 1996.

[34] Stefan Schwoon Claus Schroeter and Javier Esparza. The model checking kit. http://wwwbrauer.informatik.tu-muenchen.de/gruppen/theorie/KIT/.

[35] George Cleland and Donald MacKenzie. Inhibiting factors, market structure and the industrial uptake of formal methods. In *Workshop on Industrial-Strength Formal Specification Techniques*, pages 46–60. IEEE Computer Society, April 1995.

[36] C.N. Ip and D.L. Dill. Better verification through symmetry. In D. Agnew, L. Claesen, and R. Camposano, editors, *Computer Hardware Description Languages and their Applications*, pages 87–100, Ottawa, Canada, 1993. Elsevier Science Publishers B.V., Amsterdam, Netherland.

[37] C.N. Ip and D.L. Dill. Better verification through symmetry. In *Formal Methods in System Design*, number 1-2, pages 41–75, Hingham, MA, USA, 1996. Kluwer Academic Publishers.

[38] C.N. Ip and D.L. Dill. Verifying systems with replicated components in murphi. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 147 158, New Brunswick, NJ, USA, / 1996. Springer Verlag.

[39] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations. http://www.ietf.org/rfc/rfc2501.txt, January 1999.

[40] Dan Craigen, Susan Gerhart, and Ted Ralston. Formal Methods Reality Check: Industrial Usage. *IEEE Transactions on Software Engineering*, 21(2):90 98, February 1995.

[41] D. Abts and M. Roberts. Verifying large-scale multiprocessors using an abstract verification environment. In *Design Automation Conference (DAC)*, pages 163 168, 1999.

[42] & Ted Ralston Dan Craigen, Susan Gerhart. Formal methods reality check: Industrial usage. In J.C.P. Woodcock and P.G. Larsen, editors, *FME'93: Industrial-Strength Formal Methods*, pages 250–267. Formal Methods Europe, Springer-Verlag, April 1993. Lecture Notes in Computer Science 670.

[43] Susan Gerhardt Dan Craigen and Ted Ralston. *An International Survey of Industrial Applications of Formal Methods*, volume Volume 2 Case Studies. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD 20899, USA, March 1993.

[44] Susan Gerhardt Dan Craigen and Ted Ralston. *An International Survey of Industrial Applications of Formal Methods*, volume Volume 1 Purpose, Approach, Analysis and Conclusions. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD 20899, USA, March 1993.

[45] S. Das, C.E. Perkins, and E.M. Royer. Ad hoc on demand distance vector (aodv) routing. Internet Draft, 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-05.txt.

[46] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *International Conference on Computer Design*, pages 522 525, 1992.

[47] David L Dill, Seungjoon Park, and Andreas G. Nowatzyk. Formal specification of abstract memory models. In *Proceeding of the 1993 symposium on Research on integrated systems*, pages 38–52. MIT Press, 1993.

[48] E. W. Djikstra. A note on two problems in connexion with graphs. *Communications of the ACM*, 1(1):269 271, october 1959.

[49] G. Dommety and Raj Jain. Potential networking applications of global positioning systems (GPS). Technical report, 1996.

[50] G. Dommety and Raj Jain. Potential networking applications of global positioning systems (GPS). Technical report, 1996.

[51] Jr. Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[52] Greg Ennis. 802.11 architecture. Technical report, IEEE 802.11, March 1996. Tutorial of draft Standard IEEE 802.11.

[53] P. Chu F. Lin and M. Liu. Protocol verification using reachability analysis. volume 17. Computer Communication Review, 1987.

[54] P. Chu F. Lin and M. Liu. Protocol verification using reachability analysis: The state explosion problem and relief strategies. Proceedings of the ACM SIGCOMM 87, 1987.

[55] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[56] R. W. Floyd. ACM algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, June 1962.

[57] Internet Engineering Task Force. http://www.ietf.org.

[58] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGPLAN-SGACT symposium on Principles of programming languages*, pages 162 173, Las Vegas, Nevada, 1980. ACM Press.

[59] Marie-Claude Gaudel and Jim Woodcock, editors. *FME'96: Industrial Benefit and Advances in Formal Methods*. Springer-Verlag, March 1996.

[60] Mario Gerla, Xiaoyan Hong, Li Ma, and Guangyu Pei. Landmark routing protocol (lanmar) for large scale ad hoc networks. Internet Draft, 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-lanmar-00.txt.

[61] Mario Gerla, Guangyu Pei, Xiaoyan Hong, and Tsu-Wei Chen. Fisheye state routing protocol (fsr) for ad hoc networks. Internet Draft, 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-fsr-00.txt.

[62] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18, Warsaw, Poland, 1995. Chapman & Hall.

[63] W. Wyat Gibbs. Software's chronic crisis. *Scientific American*, pages 72 81, September 1994.

[64] P. Godefroid. Using partial orders to improve automatic verification methods. Proceedings of the 2nd Workshop on Computer-Aided Verification, Spring Verlag, New York, 1990.

[65] S. Graf and B. Steffen. Compositional minimization of finite state processes. pages 57 73. Proceedings of 2st Workshop on Computer Aided Verification, number 3 in Series in Discrete Mathematics and Theoretical Computer Science. AMS-ACM DIMACS, 1990.

[66] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37 43, May 2000.

[67] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications, June 2000. Best Paper for International Conference on Requirements Engineering.

[68] John V. Guttag, James J. Horning with S. J. Garland K. D. Jones, A. Modet, and J. M. Wing. Larch: Languages and tools for formal specification., 1993.

[69] Anthony Hall. Seven Myths of Formal Methods. *IEEE Software*, 7(5):11 19, September 1990.

[70] David Hamilton, Rick Covington, and John Kelly. Experience in Applying Formal Methods to the Analysis of Software and System Requirements. In M. Larrondo-Petrie R. France and S. Gerhart, editors, *Workshop on Industrial-Strength Formal Specification Techniques*, pages 30–43. IEEE Computer Society Press, April 1995.

[71] Pieter Hartel et al. Questions and answers about ten formal methods. In *Fourth Int. ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS'99) Proceedings of the FLoC Workshop*, volume II, pages 179 203, July 1999.

[72] A. Helmy. A survey on kernel specification and verification. Technical Report 97-654 of the Computer Sciense Department, University of Southern California, 1997.

[73] Ahmed Helmy, Sandeep K. S. Gupta, Deborah Estrin, A. Cerpa, and Y. Yu. Systematic testing of multicast routing protocols: Analisys of forward and backward search techniques. October 2000.

[74] Michael G. Hinchey and Jonathan P. Bowen, editors. *Applications of Formal Methods*. Prentice Hall, 1995. ISBN 0-13-366949-1.

[75] Mobile Ad hoc Networks (manet). http://www.ietf.org/html.charters/manet-charter.html.

[76] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall Software Series. Prentice-Hall, 1991.

[77] Gerard J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279 295, 1997.

[78] Gerard J. Holzmann. The spin model checker. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[79] Gerald Holzmman. The spin model checker. http://netlib.bell-labs.com/netlib/spin/whatispin.html, May 2002.

[80] Swades De Hongyi Wu, Chunming Qiao and Ozan Tonguz. Integrated cellular and ad hoc relaying systemas: icar. *IEEE J. Select. Areas Communications Magazine*, 19(10):2105 2215, Oct. 2001.

[81] Alan J. Hu, David L. Dill, Andreas Drexler, and C. Han Yang. Higher-level specification and verification with BDDs. pages 82 95, November 1992.

[82] J. P. Hubaux, Th. Gross, J. Y. Le Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: the Terminodes project. *IEEE Communications Magazine*, 31(1):118 124, 2001.

[83] Christian Huitema. *Routing in the Internet,*. P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1995.

[84] IEEE. Verifying logic in general.

[85] C. Norris Ip and David L. Dill. Efficient verification of symmetric concurrent systems. In *International Conference on Computer Design*, pages 230 234, 1993.

[86] C. Norris Ip and David L. Dill. State reduction using reversible rules. In *Design Automation Conference*, pages 564 567, 1996.

[87] Philippe Jacquet, Paul Muhlethaler, Amir Qayyum, Anis Laouiti, Laurent Viennot, and Thomas Clausen. Optimized link state routing protocol. Internet Draft, 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-03.txt.

[88] Jorjeta G. Jetcheva, Yih-Chun Hu, David A. Maltz, and Rice University. A simple protocol for multicast and broadcast in mobile ad hoc networks. Interned Draft, 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-simple-mbcast-00.tx.

[89] L. Ji and M.S. Corson. Differential destination multicast (ddm) specification. Internet Draft, 2001. http://www.ietf.org/internet-drafts/draft-ietf-manet-ddm-00.txt.

[90] C.B. Jones and A.M. McCauley. Formal methods selected historical references. Technical Report UMCS-92-12-2, Department of Computer Science, University of Manchester, Manchester M13 9PL, England, December 1992.

[91] Cliff B. Jones. Recent books on formal methods. Technical report, The University of Manchester, Department of Computer Science, 1995.

[92] Ralf Kneuper. Limits of formal methods. *Formal Aspects of Computing*, 9:379 394, 1997.

[93] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *4th ACM/IEEE MOBICOM*, Dallas, Texas, USA, October 25 30 1998.

[94] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 66–75, Dallas, Texas, USA, October 25 30 1998.

[95] Cadence Berkeley Laboratories. The smv model checker. http://www-cad.eecs.berkeley.edu/ kenmcmil/smv/.

[96] Peter Gorm Larsen, John Fitzgerald, and Tom Brookes. Applying Formal Specification in Industry. *IEEE Software*, 13(3):48–56, May 1996.

[97] Tanara Lauschner, Autran Macedo, and Sérgio Campos. Formal verification and analysis of a routing protocol for ad-hoc networks. http://www.dcc.ufmg.br/ tanara/art*tanara.ps*.

[98] S. Leppnen and Matti Luukkainen. Compositional verification of a third generation mobile communication protocol. Proceedings of Internation Conference on Distributed and Communication Systems(ICDCS) 2000 Workshops, IEEE, 2000.

[99] Z. Liu, A. Ravn, and X. Li. Compositional inductive verification of duration properties of real-time systems, 1997.

[100] Robyn R. Lutz and Yoko Ampo. Experience report: Using formal methods for requirements analysis of critical spacecraft software. Technical report, Jet Propulsion Laboratory, November 1994.

[101] David A. Maltz, Yih-Chun Hu, and Jorjeta G. Jetcheva. The dynamic source routing protocol for mobile ad hoc networks. Internet Draft, 2001. http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-04.txt.

[102] Gurmeet Singh Manku, Ramin Hojati, and Robert K. Brayton. Structural symmetry and model checking. In *Computer Aided Verification*, pages 159–171, 1998.

[103] Z. Manna and Amir Pnueli. The temporal framework for concurrent programs. In R.S. Boyer and J.S. Moore, editors, *Proceedings of The Correctness Problem in Computer Science*, pages 215–274. Academic Press, 1981.

[104] Geraldo Robson Mateus and Antnio A.F. Loureiro. Introdução à computação móvel. XI Escola de Computação, 1998.

[105] N. Maxemchuck and K. Sabnani. Probabilistic verification of communication protocols. Proceedings of the 7nd IFIP WG 6.1 International Workshop on Protocol Specification, Testing and Verification, North-Holland Publ., Amsterdam, 1987.

[106] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. pages 141–153.

[107] John C. Mitchell. Finite-state analysis of security protocols. In *Computer Aided Verification*, pages 71–76, 1998.

[108] NASA. Formal Methods, Specification and Verification Guidebook for Verification of Software and Computer Systems. Vol 2: A Practitioner's Companion. Technical Report NASA-GB-001-97, Washington, DC 20546, USA, May 1997. Available from http://eis.jpl.nasa.gov/quality/Formal_Methods/.

[109] M.T. Norris and S.G. Stockman. Industrilising formal methods for telecommunications. In C. Ghezzi and J.A. McDermid, editors, *ESEC'89; 2nd European Software Engineering Conference*, pages 159–175. Springer-Verlag (LNCS 387), September 1989. Coventry, UK.

[110] Davor Obradovic. Phd thesis: Formal analisys of routing protocols, December 2002.

[111] Davor Obradovic. Real-time model and convergence time of bgp. In *Proceedings of IEEE Infocom*, 2002.

[112] Davor Obradovic and Elsa Gunter. Towards the Integration of Model Checking and Theorem Proving: Embedding a Subset of Promela into HOL, May 2000. submitted for publication.

[113] University of Caifornia at Berkeley, University of Colorado at Boulder, and University of Texas at Austin. The vis (verification interacting with synthesis) group. http://www-cad.eecs.berkeley.edu/Respep/Research/vis.

[114] University of Cambridge Computer Laboratory. The hol system: Tutorial, July 1991. http://lal.cs.byu.edu/lal/holdoc/tutorial.html.

[115] P. Olveczky and S. Meldal. Specification and prototyping of network protocols in rewriting logic, 1998.

[116] Jonathan Ostroff. Formal Methods for the Specification and Design of Real-Time Safety Critical Systems. *Journal of Systems and Software*, 18(1), April 1992.

[117] Sam Owre and Natarajan Shankar. The formal semantics of pvs, 1997 - Revised 1999.

[118] J. Pageot and C. Jard. Experience in guiding simulation. Proceedings of the VIII-th nternational Workshop on Protocol Specification, Testing and Verification, North-Holland Publ., Amsterdam, 1988.

[119] Seungjoon Park and David L. Dill. An executable specification, analyzer and verifier for rmo (relaxed memory order). In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pages 34 41. ACM Press, 1995.

[120] Charles Perkins, editor. *Ad Hoc Networking.* Addison-Wesley, December 2000.

[121] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:1 20, 1981.

[122] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46 57, Providence, Rhode Island, 31 October - 2 November 1977. IEEE.

[123] Amir Pnueli. The temporal semantics of concurrent programs. In *Proceedings of Theomatica in Computer Science*, volume 13, pages 45 60, 1981.

[124] D. Probst. Using partial-order semantics to avoid the state explosion problem in asynchronous systems. Proceedings of the 2nd Workshop on Computer-Aided Verification, Spring Verlag, New York, 1990.

[125] The pvs specification and simulation system. http://pvs.csl.sri.com/, October 2003.

[126] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. -T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa. VIS: a system for verification and synthesis. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 428 432, New Brunswick, NJ, USA, / 1996. Springer Verlag.

[127] Sampath Kannan Rajeev Alur and Mihalis Yannakakis. Communicationg hierarchical state machines. volume 1644 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, 1999.

[128] John H. Reif and Scott A. Smolka. The complexity of reachability in distributed communicating processes. *Acta Informatica*, 25(3):333 354, 1988.

[129] E. Royer and C.-K. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 6(2):46 55, April 1999.

[130] Elizabeth M. Royer and Charles E. Perkins. Multicast ad hoc on-demand distance vector (maodv) routing. Internet Draft, 2000. http://www.ietf.org/internet-drafts/draft-ietf-manet-maodv-00.txt.

[131] John Rushby. Formal methods and the certification of critical systems. Technical Report CSL-93-7, Computer Science Laboratory, Menlo Park CA 94025 USA, December 1993.

[132] B. Dutertre S. Liu, V. Stavridou. The practice of formal methods in safety critical systems. *Journal of Systems and Software*, 28:77 87, 1995.

[133] Philippe Schnoebelen. The complexity of temporal logic model checking. In *Proceedings of the 4th International Workshop, Advances in Modal Logic AiML 2002*, volume 4. IEEE, September - October 2002.

[134] K. Seada and Ahmed Helmy. Fairness evaluation experiments for multicast congestion control protocols. November 2002.

[135] Viii seminfo - semana de informática da ufba - métodos formais no desenvolvimento de software e técnicas de verificação de modelos. http://www.lasid.ufba.br/eventos/seminfo/resumos/c7resumo.html, 2000.

[136] What's between simulation and formal verification, December 2000.

[137] Jonathan Bowen & Victoria Stavridou. The industrial take-up of formal methods in safety-critical and other areas: A perspective. In J.C.P. Woodcock and P.G. Larsen, editors, *FME'93: Industrial-Strength Formal Methods*, pages 183 195. Formal Methods Europe, Springer-Verlag, April 1993. Lecture Notes in Computer Science 670.

[138] U. Stern and D. Dill. Parallelizing the murphy verifier, 1997.

[139] U. Stern and D. Dill. Murphi description language and verifier, 2004.

[140] Ulrich Stern and David L. Dill. Automatic verification of the SCI cache coherence protocol. In *Conference on Correct Hardware Design and Verification Methods*, pages 21 34, 1995.

[141] Ulrich Stern and David L. Dill. A new scheme for memory-efficient probabilistic verification. In *FORTE*, pages 333 348, 1996.

[142] Ulrich Stern and David L. Dill. Using magnatic disk instead of main memory in the mur phi verifier. In *Computer Aided Verification*, pages 172 183, 1998.

[143] P. Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange, 1998.

[144] Andrew S. Tanenbaum. *Computer Networks*. Third Edition. Addison-Wesley, 1994.

[145] H. Tauriainen and K. Heljanko. Testing ltl formula translation into buchi automata, 2002.

[146] Heikki Tauriainen and Keijo Heljanko. Testing SPIN's LTL formula conversion into buchi automata with randomly generated input. In Klaus Havelund, John Penix, and Willem Visser, editors, *Proceedings of the 7th International SPIN Workshop on Model Checking of Software (SPIN'2000)*, volume 1885 of *Lecture Notes in Computer Science*, pages 54 72, Stanford University, California, USA, August 2000. Springer.

[147] The GPS Store's automated online shopping site makes buying GPS units and accessories easy and inexpensive. http://www.thegpsstore.com/site/.

[148] Martyn Thomas. The industrial use of formal methods. *Microprecessors and Microsystems*, 17(1):31 36, January 1992.

[149] Shaz Qadeer Thomas A. Henzinger and Sriram K. Rajamani. You assume, we guarantee: Methodology and case studies. 1998.

[150] J. Tretmans, Wijbrans K., and M. Chaudron. Software Engineering with Formal Methods: The Development of a Storm Surge Barrier Control System Seven Myths of Formal Methods Revisited. In *Fourth Int. ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS'99) Proceedings of the FLoC Workshop*, volume II, pages 225 237, Pisa, Italy, July 1999.

[151] Enrico Tronci, Giuseppe Della Penna, Benedetto Intrigila, and Marisa Venturini Zilli. Exploiting transition locality in automatic verification. *Lecture Notes in Computer Science*, 2144:259+, 2001.

[152] U. Stern and D.L. Dill. Improved Probabilistic Verification by Hash Compaction. In P.E. Camurati and H. Eveking, editors, *Correct Hardware Design and Verification Methods*, volume 987, pages 206 224, Stanford University, USA, 1995. Springer-Verlag.

[153] U. Stern and D.L. Dill. Combining state space caching and hash compaction. In Bernd straube and Jens Schoenherr, editors, *4. GI/ITG/GME Workshop zur Methoden des Entwurfs und der Verifikation Digitaler Systeme*, pages 81 90, Kreischa, 1996. Shaker Verlag, Aachen.

[154] Carnegie Mellon University. The smv system. http://www-2.cs.cmu.edu/ modelcheck/smv.html.

[155] A. Valmari. Application and theory of petri nets 1996. pages 29 56. 17th International Conference Compositionality in State Space Verification Methods, number 1091 in Lecture Notes in Computer Science. Springer-Verlag, 1996.

[156] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1 37, 15 1994.

[157] C. West. Protocol validation by random state exploration. Proceedings of the 6nd IFIP WG 6.1 International Workshop on Protocol Specification, Testing and Verification, North-Holland Publ., Amsterdam, 1986.

[158] M. Wirsing, editor. *ICSE-17 Workshop on Formal Methods Application in Software Engineering Practice*, 24 25 April 1995.

[159] William G. Wood. Application of formal methods to system and software specification. *ACM Software Engineering Notes*, 15(4):144 146, 1990. Proceedings of the ACM Workshop on Formal Methods and Software Development (Napa, California, December 1989).

[160] The world wide web virtual library in formal methods. http://www.comlab.ox.ac.uk/archive/formal-methods.html.

[161] Zumble. http://www.zumble.com.br/fersi.htm, July 2000.