PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

USO DE REDES DE AUTÔMATOS ESTOCÁSTICOS – SAN NA MODELAGEM E AVALIAÇÃO DO PROTOCOLO DSR EM REDES *WIRELESS AD HOC*

EVERTON RICARDO DO NASCIMENTO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Paulo Henrique Lemelle Fernandes

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

N244u	Nascimento, Everton Ricardo do Uso de redes de autômatos estocásticos – SAN na modelagem e avaliação do protocolo DSR em redes wireless as / Everton Ricardo do Nascimento. – Porto Alegre, 2009. 172 f. Diss. (Mestrado) – Fac. de Informática, PUCRS. Orientador: Prof. Dr. Paulo Henrique Lemelle Fernades. 1. Informática. 2. Redes de Computadores. 3. Redes de Autômatos Estocásticos. 4. Protocolos de Aplicação Sem Fio (Protocolos de Rede de Computação). 5. Roteamento – Redes de Computadores. I. Fernandes, Paulo Henrique Lemelle. II. Título. CDD 004.6

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "O Uso de Redes de Autômatos Estocásticos – SAN na Análise e Avaliação do Protocolo DSR em Redes Wireless Ad Hoc", apresentada por Everton Ricardo do Nascimento, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 17/12/09 pela Comissão Examinadora:

Prof. Dr. Paulo Henrique Lemelle Fernandes – Orientador — PPGCC/PUCRS

Prof. Dr. Fernando Luís Dotti – PPGCC/PUCRS

Dra. Thais Christina Webber dos Santos — Pesquisadora FACIN/PUCRS

Homologada em 07/06/2010, conforme Ata No. 09/2010 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Antes de mais nada, dedico este trabalho à Deus, por me permitir dar mais esse passo na minha carreira acadêmica., e por me fazer um instrumento concentrador, gerador e transmissor de conhecimento, já que este mestrado, além de ser para mim uma enorme realização, é também a concretização de um sonho, sendo também para meus alunos uma grande fonte de novas possibilidades.

Dedico ao meu amor, meu porto seguro, minha Heluiza, por ser paciente e carinhosa em todos os momentos da nossa vida juntos, por saber compreender a minha ausência, a minha completa falta de paciência quando tudo sumia debaixo dos meus pés, você sempre me resgatou da minha tristeza, e por tudo isso e muito mais que eu te amo....

Dedico este trabalho à meus pais, mesmo sabendo que meu querido pai não está mais nessa dimensão, mas eu sei que ele está em boa companhia, perto do Grande Arquiteto do Universo, olhando por mim, e me guiando em meus passos e à minha mãe, rocha fundamental da minha família.

Ao meu irmão, por tudo que passamos juntos, e indiferentemente que qualquer coisa, nada nem ninguém pode mudar aquilo que somos, e que sempre, sempre, o sangue é mais forte!

Ao meu sogro e minha sogra, por serem a minha segunda família, e por sempre me apoiarem incondicionalmente como um pai e uma mãe apoia o filho, muito obrigado.

AGRADECIMENTOS

Elencar nomes quando se pretende agradecer à várias pessoas é um grande pecado, mas devido à grandiosidade desta conquista, me permito cometer este pecado sem remorso.

Ao meu orientador, Prof. Dr. Paulo, pelo conhecimento, pela atenção, pela paciência, ambos sabemos que este mestrado foi um desafio feito à todos, professores e alunos, e, ao término deste desafio, me sinto imensamente agradecido por tamanha contribuição que me destes!

À todos os meus amigos, que torceram por mim nessa caminhada, que de uma forma ou outra contribuíram para que eu não desanimasse, que eu continuasse essa luta, meu muito obrigado!

Aos meus companheiros de MINTER, cada um à sua forma, mas em especial à galera que caminhou (literalmente) junto: Fernando, Rodrigo, Fernandes, André, Uelinton, Maurão, Luciano e Toni, e claro, à 100% das mulheres do MINTER, Estelinha e Ju, e a todos os outros guerreiros que chegaram até aqui, que se tornaram amizades que se fortaleceram e que com certeza durarão por muito e muito tempo, obrigado por toda colaboração, por todas as noites mal dormidas que passamos fazendo trabalhos e estudando, pelos finais de semana de estudo, e por todos os happy hour's que merecidamente fizemos!

À todos os professores que participaram deste mestrado e que contribuíram para a nossa formação, obrigado pela paciência e pela humildade de compartilhar conosco uma parcela do vosso conhecimento, e também pelos momentos de lazer, afinal, futebol também é cultura!

Ao pessoal do PEG, em especial à Thais e ao Ricardo, que humildemente e pacientemente nos ajudaram, sempre da melhor maneira possível, compartilhando saberes e nos mostrando caminhos, muito obrigado!

Me perdoem aqueles que esqueci ou que não elenquei, mas saiba que todos são lembrados nas minhas orações, e que a minha gratidão para com vocês é imensa, o meu muito obrigado!

USO DE REDES DE AUTÔMATOS ESTOCÁSTICOS NA MODELAGEM E AVALIAÇÃO DO PROTOCOLO DSR EM REDES *WIRELESS AD HOC*

RESUMO

A avaliação de desempenho dos protocolos de roteamento de redes *Ad hoc* é realizada, em sua maioria, através do uso de técnicas e ferramentas de simulação. Através do uso das Redes de Autômatos Estocásticos, surge uma nova alternativa para realizar esta avaliação, já que este formalismo provê grande quantidade de espaços de estados, que era um dos grandes problemas de outros formalismos analíticos. Os protocolos de roteamento para redes sem fio *Ad hoc* são um fator determinante no sucesso da tarefa de transmissão de dados entre os nodos. Sendo assim, foi avaliado neste trabalho o protocolo DSR - *Dynamic Source Routing*, que tem como características principais as técnicas de descoberta e manutenção de rota, a fim de garantir a menor - ou melhor - rota dentro de uma rede. A primeira contribuição presente neste trabalho trata diretamente da avaliação deste processo de descoberta de rota, bem como da possibilidade de falhas de transmissão ocorrerem e o processo de manutenção de rota se fazer necessário. Estes resultados estão diretamente ligados aos resultados obtidos avaliando-se o *workload* dos nodos da rede, que reflete diretamente na utilização das rotas. A segunda contribuição é a análise do comportamento de redes quando estas utilizam uma particularidade do protocolo DSR que é a escuta promíscua *(promiscuous listening*). Busca-se avaliar se o uso de tal característica auxilia no funcionamento da rede. São apresentados também resultados de avaliação de métricas como vazão com diferentes tamanhos de pacotes e tempos de pausa dos nodos durante a transmissão dentro de uma rede que utiliza o protocolo DSR para o roteamento das informações. Adicionalmente busca-se demonstrar a eficiência das Redes de Autômatos Estocásticos na obtenção destas métricas e assim apresentar este formalismo como uma alternativa para avaliação deste tipo de ambiente, dada a similaridade dos resultados obtidos, se comparados àqueles apresentados na literatura.

Palavras-Chave: Protocolos de Roteamento, Redes *Wireless*, Avaliação de Desempenho, DSR.

USING STOCHASTIC AUTOMATA NETWORKS IN MODELING AND EVALUATION OF THE PROTOCOL DSR IN *AD HOC WIRELESS* NETWORKS

ABSTRACT

The performance evaluation of routing protocols for *Ad hoc* networks is usually made through simulation techniques and tools. The use of Stochastic Automata Networks is a new alternative to evaluate such protocols, since this formalism can handle very large spaces of states, which it was one of the major problems of other analytical formalisms. The routing protocols for *Ad hoc* networks are a key factor in the success of the data transmissions task between the nodes, so the DSR - Dynamic Source Routing, protocol was evaluated at this work. DSR protocol main characteristics are the techniques of discovery and maintenance of routes, in order to assure the smaller - or better - route inside a network. The first contribution of this work is the straight evaluation of route discovery process, as well as the possibility of transmission failures and the need of route maintenance. These results are directly connected with the *workload* of the network, which is directly related to route utilization results. The second contribution is the analysis of networks with a DSR protocol with *promiscuous listening* in order to evaluate if the use of such characteristic helps the networks behavior. Are also presented evaluation results of metrics like throughput with different packets sizes and nodes pause time during the transmission inside a network that uses the DSR protocol to information routing. Additionally, this work demonstrate the efficiency of the Stochastic Automata Networks formalism to obtain these metrics as an alternative to evaluate this type of environment while similar results were obtained in literature.

Keywords: Routing Protocols, *Wireless* Networks, Performance Evaluation, DSR.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACK – Acknowledgment
AODV – *Ad hoc* On Demand Vector
AUP – Agile Unified Process
Bps – Bytes per second
bps – bits per second
CDMA – Code Division Multiple Access
CSMA – Carrier Sense Multiple Access with Collision Avoidance
DoD – Department of Defense
DSR – Dynamic Source Routing
GloMo –Global Mobile Informations System
IEEE – Institute of Electrical and Electronics Engineers
IETF – Internet Engineering Task Force
IP – Internet Protocol
LAN – Local Area Network
MAC – Medium Access Control
MANET – Mobile *Ad hoc* Networks
PEPS – Performance Evaluation of Parallel Systems
pps – packets per second
PRNET – Packet Radio Networks
RREP – Route Reply
RREQ – Route Request
RTS/CTS – Request to Send/ Clear to Send
RWP – Random Waypoint
SAN – Stochastic Automata Networks
SURAN – Survivable Adaptive Radio Network
TDMA – Time Division Multiple Access
TTL – Time –to-live

SUMÁRIO

1 INTRODUÇÃO

Com a crescente demanda por novas formas de trocar informações, as redes sem fio – ou *wireless* – surgem como uma nova alternativa que vem ganhando espaço significativo em todos os setores da sociedade, sejam eles científicos, comerciais ou acadêmicos.

Conforme Forouzan em [Fou06], as redes sem fio podem possuir duas topologias bem distintas: a infra-estruturada e a *Ad hoc*. A principal diferença entre elas deve-se ao fato que, nas redes infra-estruturadas, é utilizado um concentrador, chamado de *access point*, para distribuir e direcionar as informações, o que não acontece com as redes *Ad hoc*, sendo que cada nodo presente na rede atua como transmissor (ou roteador) e receptor (ou cliente).

Devido ao fato que as redes *Ad hoc* não possuírem um concentrador e dependerem diretamente dos nodos presentes nessa rede para realizem todo o processo de transmissão de dados, este tipo de redes tem sido objeto de diversos estudos, como afirma Ramanathan *et al.* em [Ram02]. E por conseqüência desse crescente interesse em redes *Ad hoc*, cada vez mais se foi fazendo necessário melhorar a interoperabilidade e a comunicação entre os nodos presentes em uma rede deste tipo. Nesse sentido, as formas como os dispositivos ou nodos presentes nestas redes descobrem como se comunicar e passar ou repassar as informações entre fonte e destino é um objeto de estudo de grande importância e que tem sido dispensada cada vez mais atenção, no que abrange as redes sem fio, seja através de novos algoritmos de roteamento para otimizar essas rotas, seja através de melhorias quanto à segurança ou consumo eficiente de energia.

Um dos principais esforços para essa comunicação ser eficiente são os protocolos de roteamento. Os protocolos de roteamento foram criados para facilitar a comunicação dentro da rede, descobrindo rotas entre os nodos, de acordo com Royer *et al.* em [Roy99]. Ainda segundo a mesma autora, o objetivo principal dos protocolos de roteamento é o correto e eficiente estabelecimento de rotas entre um par de nodos para que as mensagens possam ser entregues de maneira oportuna [Roy99].

Existem hoje, basicamente, duas categorias de protocolos, os reativos (ou sob demanda) e os pró-ativos, onde os reativos têm como principal característica um custo menor de transmissão haja vista que estes protocolos somente iniciam suas atividades de descoberta de rota quando uma transmissão necessita ser realizada, e as finaliza quando o destino é encontrado, fazendo com que a atividade de rede seja menor, gerando um consumo de banda, energia e tráfego menor do que em outros tipos de protocolo.

Por sua vez, os protocolos pró-ativos, tem como característica o fato de todos os nodos presentes na rede possuem uma tabela para todas as rotas conhecidas para todos os destinos possíveis [Fee99]. Ainda segundo Feeney em [Fee99], os nodos trocam informações periodicamente, para diminuir o tempo de requisição de rota no envio de um dado, característica essa que pode ser considerada uma vantagem, porém, isso faz aumentar consideravelmente o custo dos recursos da rede.

Tais protocolos, quando avaliados, comumente têm suas tarefas e características avaliadas via simulação, onde usa-se dados semelhantes a informações do mundo real em modelos que possam determinar índices e resultados, levando em consideração uma escala adequada de tempo, segundo Pid em [Pid92].

Os métodos analíticos, por sua vez, descrevem também situações do mundo real, com um nível de abstração maior que a simulação, gerando modelos puramente matemáticos [Cha05]. Neste tipo de modelo, o funcionamento do sistema real é reduzido a relações matemáticas. Desenvolver modelos analíticos normalmente exige maior abstração de aspectos da realidade, se comparado a modelos de simulação [Cha05]. Logo, poucos são os estudos voltados a protocolos de roteamento para redes sem fio *Ad hoc* que se vale de modelos analíticos para determinar seu desempenho, até porque essas avaliações feitas a partir de simulações geravam um número maior de estados e componentes do modelo, o que se configura como uma barreira para modelos analíticos, como por exemplo, Cadeias de Markov [Ste94].

O surgimento do formalismo de Redes de Autômatos Estocásticos (*Stochastic Automata Networks*) – SAN [Pla85] – surge como uma alternativa em relação a essa barreira imposta em outros formalismos analíticos, uma vez que este pode atender a grandes espaços de estados, através de um alto nível de abstração do que se pretende modelar, usando de primitivas como modularização, eventos sincronizantes e taxas funcionais, entre outras, e com isso apresentar resultados satisfatórios de análise e avaliação de desempenho dos mais variados tipos de sistemas, sendo possível chegar a resultados com um nível maior de abstração e com menos tempo de processamento e custo de memória computacional.

1.1 Objetivos do trabalho

O uso de autômatos estocásticos enquanto formalismo analítico na avaliação de protocolos de roteamento se configura como uma nova forma de mensurar tais protocolos, uma vez que, em sua maioria, estes são estudados utilizando técnicas de simulação, para apresentarem resultados de avaliação.

Este trabalho tem por objetivo a proposta de modelar, utilizando o formalismo de autômatos estocásticos, o protocolo de roteamento de redes *wireless Ad hoc* DSR, por se tratar de um dos protocolos mais utilizados em redes *Ad hoc*, bem como ser um dos protocolos mais avaliados na literatura, a fim de determinar as probabilidades de descoberta de rotas entre os nós da rede e suas características, e com isso gerar resultados passíveis de comparação com outros métodos de avaliação de desempenho.

Para melhor contextualizar o objetivo deste trabalho, são elencados aqui os objetivos específicos que auxiliarão na compreensão do problema:

• Aprofundar o estudo sobre o formalismo de redes de autômatos estocásticos e também sobre o protocolos de roteamento para redes *wireless Ad hoc* DSR;

• Identificar quais são as principais funcionalidades do protocolo de roteamento DSR e quais índices serão avaliados;

• Propor os modelos SAN para descrever o protocolo DSR e suas particularidades;

• Utilizar a ferramenta PEPS [Ben03] para obter os resultados dos modelos SAN criados;

• Apresentar tais resultados a fim de verificar a usabilidade do formalismo SAN como uma nova alternativa de avaliação em protocolos de roteamento e redes de computadores *wireless Ad hoc*.

1.2 Método e estrutura de pesquisa

A revisão bibliográfica foi a primeira etapa na construção deste trabalho, onde foram feitos levantamentos bibliográficos acerca dos assuntos pertinentes à esta pesquisa. Inicialmente foram

revisados os conceitos sobre redes de computadores *wireless* e sobre os protocolos de roteamento para as mesmas e em especial o protocolo DSR.

Na segunda etapa foram realizados estudos em relação ao formalismo de autômatos estocásticos e sobre o funcionamento da ferramenta PEPS e de que forma esta pode auxiliar no processo de modelagem e avaliação do referido protocolo de roteamento.

Na terceira etapa, foram gerados os primeiros modelos de redes de autômatos estocásticos para modelagem e avaliação do protocolo de roteamento DSR e suas características básicas, a fim de identificar possíveis problemas na construção dos referidos modelos, bem como testar tais características.

Na quarta etapa, foram feitas as alterações nos modelos gerados para sanar os problemas identificados, bem como a geração de modelos mais complexos que representassem as características que anteriormente foram estudadas à nível teórico. A Figura 1.1 mostra as etapas desenvolvidas para este trabalho.

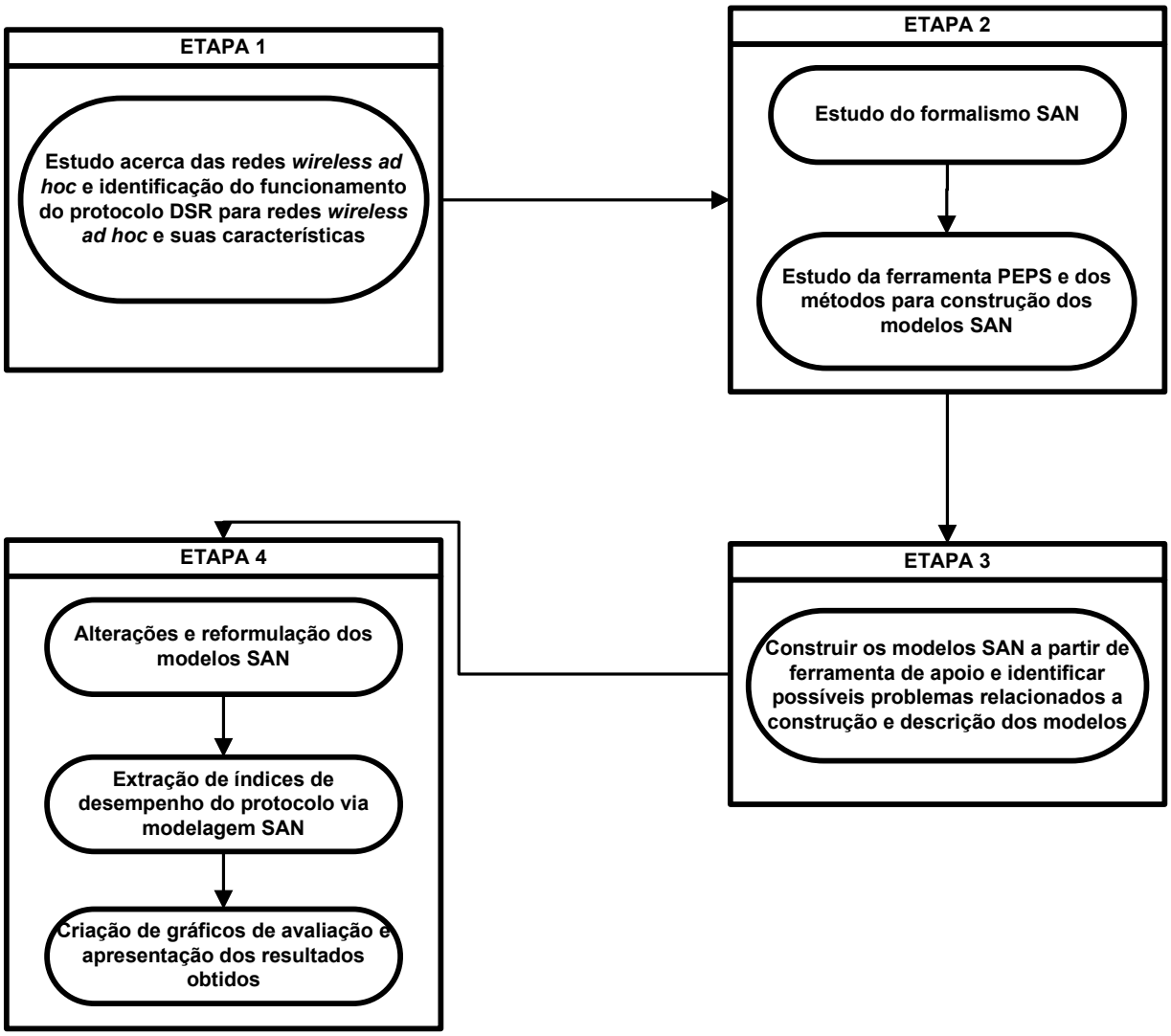


Figura 1.1: Etapas da pesquisa

1.3 Estrutura do trabalho

Este trabalho está estruturado em sete capítulos. O primeiro capítulo se destina a introdução, apresentando os objetivos deste trabalho, a metodologia de pesquisa utilizada e a estrutura do trabalho, bem como os trabalhos relacionados a esta pesquisa, onde são discutidos os principais trabalhos acerca da área de avaliação de protocolos de roteamento para redes *wireless Ad hoc*, em especial ao protocolo DSR, e que serviram tanto como fonte de informação quanto parâmetro de comparação para as avaliações realizadas com o formalismo SAN.

O segundo capítulo é voltado para as redes de autômatos estocásticos, onde são apresentadas as principais características deste formalismo, sua contextualização informal e também um exemplo de modelagem para melhor representar estes conceitos.

No terceiro capítulo são apresentados os conceitos referentes às redes de computadores *wireless Ad hoc*, onde as principais características deste tipo de redes e suas funcionalidades são discutidas. Serão demonstrados também os conceitos acerca dos protocolos de roteamento para redes *Ad hoc*, onde será discorrida a taxonomia destes protocolos.

O quarto capítulo trata em específico do protocolo de roteamento DSR, onde será mostrado o seu funcionamento, bem como as características mais relevantes, e a forma como este protocolo realiza a troca de dados entre os nodos presentes em uma rede.

O quinto capítulo apresenta os modelos gerados para realizar a avaliação do protocolo de roteamento DSR e de suas características, onde são apresentados vários modelos gerados, de acordo com a complexidade das características do protocolo DSR.

Já no sexto capítulo, os resultados obtidos com a modelagem em SAN são apresentados, bem como o comparativo destes resultados com outros formalismos para demonstrar a confiabilidade de SAN na avaliação de desempenho proposta.

O sétimo e último capítulo destina-se às conclusões e propostas de trabalhos futuros referentes ao tema.

1.4 Trabalhos relacionados

Quando da realização de avaliação de desempenho de protocolos de roteamento para redes *wireless Ad hoc*, é possível encontrar uma gama diversificada de trabalhos relacionados na literatura. Na maioria das vezes, quase em sua totalidade, esses trabalhos utilizam-se do formalismo de simulação para realizar tal avaliação, devido ao fato que, segundo Pid [Pid92], a simulação costuma consumir menos tempo para que os índices sejam calculados, permitindo que sejam feitos quantos experimentos forem necessários, porém, com uma quantidade maior de dados sendo extraídos para interpretação, o nível de precisão eventualmente não é satisfatório com um número restrito de experimentos.

Outro motivo do uso de tal formalismo deve-se ao fato que os aplicativos utilizados para realizar a simulação se aproximarem cada vez mais da realidade, dada a quantidade de variáveis que podem ser assumidas, e também fornecerem ferramentas que abrangem todos os pontos de avaliação de uma rede, desde a possibilidade de se simular um dispositivo de hardware, como um roteador e sua capacidade de alcance de sinal, assim como os protocolos de roteamento e a forma como estes exercem seu papel dentro de uma rede.

Na construção deste trabalho, que trata da avaliação de desempenho do protocolo de roteamento DSR, bem como seus processos de descoberta de rotas e suas principais características, utilizando um formalismo analítico - redes de autômatos estocásticos - buscou-se utilizar ambientes passíveis de comparação com àqueles que foram usados nos trabalhos com simulação - e algumas vezes com uma abstração maior que estes - para realizar as avaliações necessárias. Nos capítulos 5 e 6 encontram-se todas as informações referentes aos modelos gerados bem como os ambientes usados na avaliação do protocolo.

Das *et al.* em [Das98], tem como contribuição de maior relevância em seu trabalho, a comparação de vários protocolos de roteamento, através de diversas métricas (que serão discutidas na apresentação da geração dos modelos para a avaliação de desempenho). Nos modelos de simulação gerados, os autores focaram apenas detalhes baseados na camada de rede, deixando de lado detalhes de camada de enlace, como protocolo MAC, interferência de múltiplos acessos ou erros de conexão, bem como descartaram detalhes de camada física. A principal preocupação em distinguir os protocolos de roteamento foram os pacotes de roteamento (que agem diretamente na descoberta de rota) e a manutenção da rota.

Para a manutenção de rota, os pacotes de roteamento são separados dos pacotes de dados, o que favorece na obtenção de informações sobre conexões defeituosas ou falhas. Os ambientes criados para determinar o *workload* da rede levam em conta um nodo fonte que deve transmitir dados para um destino, baseado em taxas constantes, sem que exista nenhum controle de congestionamento nem de fluxo dentro da rede.

Em [Per01], Perkins *et al.* discute algumas características peculiares dos protocolos DSR e AODV [Per99], para melhor fundamentar sua avaliação, como o fato de que o protocolo DSR tem acesso à uma quantidade significantemente maior de informações de roteamento que outros protocolos.

Quando o autor compara e avalia o protocolo DSR com o protocolo AODV [Per99], este apresenta características relevantes quanto a escolha do protocolo DSR para ser usado como alternativa em redes *wireless Ad hoc*, como por exemplo, o processo em que o protocolo DSR realiza um ciclo de descoberta de rota, com pedidos de requisição e resposta de rota, o nodo fonte pode aprender rotas para cada nodo intermediário na rota adicionalmente ao destino requerido.

Outro fator importante que os autores elencam deve-se ao fato de que o protocolo DSR responde a todos os pedidos que atingem o destino a partir de um único ciclo de requisições. Assim, a fonte aprende várias rotas alternativas para o destino, no caso da rota de menor número de saltos ou a rota mais eficiente venha a falhar.

Utilizando-se das mesmas métricas de Das *et al.* em [Das98], agora levando em consideração a carga da camada MAC, Perkins *et al.* em [Per01] conduz a avaliação também para a camada de enlace, expandindo o trabalho dos autores anteriormente citados.

Já em Broch *et al.* [Bro98], o interesse dos autores deu-se a partir da simulação do funcionamento dos protocolos de roteamento reativos e pró-ativos para avaliar seu comportamento quanto às características de múltiplos saltos na transmissão de pacotes. Sendo que este estudo serviu de base para muitos outros estudos, devido ao fato de ter sido o primeiro trabalho a prover uma análise comparativa, quantitativa e realística do desempenho de diversos protocolos de roteamento para redes *wireless Ad hoc*.

Nas simulações que foram realizadas em [Bro98], foram levadas em consideração características referentes à mobilidade dos nodos, à camada física, sendo consideradas também a interface de rádio da rede e a forma de acesso ao meio pela camada MAC contida dentro do protocolo IEEE 802.11.

Outro trabalho relevante e muito utilizado e citado quando se trata de avaliação de protocolos de roteamento é o trabalho de Boukerche em [Bou04], que se preocupou em simular cenários com várias topologias de diferentes tamanhos e cargas de transmissão. Para comparar tais protocolos, o autor ateve-se à algumas características dos protocolos, como o número de descobertas de rotas; as mensagens de "*hello*" (em protocolos pró-ativos), que determinam se os nodos e seus respectivos vizinhos estão presentes na rede; o roteamento de fonte e; a transmissão das requisições de rota.

O autor apresenta em suas conclusões, informações sobre a eficiência do protocolo DSR sobre os outros avaliados em relação ao *overhead* de roteamento que é relevantemente menor em

relação aos demais, e também à sua vazão, que é muito maior, apresentando o protocolo DSR como um protocolo eficiente e que respondeu bem às avaliações a ele impostas.

No trabalho de Oliveira *et al.* em [Oli05], os protocolos foram avaliados no uso de aplicações par-a-par (*peer-to-peer*), avaliando o funcionamento dos protocolos de roteamento em cenários de aplicações P2P, fazendo uso de métricas sob novas condições. Este trabalho tem a relevância de apresentar os protocolos trabalhando em um ambiente diferente de outros trabalhos, mas com a vantagem de se aplicar as mesmas métricas.

No trabalho apresentado por Ahmed e Alam [Ahm06], a avaliação de desempenho dos protocolos é feita baseada em algumas características em nível de pacotes, como simulação de tráfego de envio e recebimento, vazão e utilização entre outros. Neste trabalho, os autores tiveram a preocupação de limitar o tamanho da rede a ser simulada, para que pudessem assumir várias condições de rede que otimizariam seus modelos. Essa limitação também foi assumida neste trabalho, a fim de aproximar os ambientes utilizados nas avaliações.

A partir das contribuições adquiridas com os trabalhos relacionados, podem-se elencar variáveis que contribuíram na construção tanto dos modelos quanto dos ambientes gerados para a realização deste trabalho, e com isso, verificar a obtenção de tais variáveis via literatura e assim transportá-las para este trabalho.

2 REDES DE AUTÔMATOS ESTOCÁSTICOS

Várias formas de se avaliar e modelar protocolos de roteamento foram criadas a fim de obter resultados que auxiliassem no processo de análise do funcionamento e análise de desempenho de tais protocolos para redes *wireless Ad hoc*, contudo, o que será utilizado neste estudo será a modelagem através do formalismo de Redes de Autômatos Estocásticos – SAN, por ser um método analítico que permite uma maior abstração e que pode ter grandes espaços de estados sem que isso comprometa seus cálculos [Fer98].

O formalismo SAN é um formalismo de modelagem que se aplica para sistemas com grandes espaços de estados [Pla85], proposto primeiramente por Plateau, em [Pla85], tendo como idéia principal a modelagem de sistemas que teriam vários subsistemas que podem ser entendidos como módulos, com comportamentos diferenciados, de acordo com Fernandes *et al.* [Fer98].

2.1 Autômatos Estocásticos

Um autômato estocástico pode ser definido como um autômato onde as transições são modeladas por processos estocásticos de tempo contínuo ou discreto [Fer98]. Outra forma como se pode definir autômatos estocásticos foi usada em Benoit *et al.* em [Ben03], que diz que um autômato estocástico pode ser definido como um modelo matemático de um sistema que possui entradas e saídas discretas.

Esse autômato pode conter todos os estados possíveis do sistema, ou então, pode trabalhar em paralelo com outros autômatos, sendo estes concorrentes ou não no disparo das transições através dos eventos que acontecem dentro do sistema. Sendo assim, podemos dizer que um autômato estocástico é um conjunto finito de estados e um conjunto finito de transições entre esses estados. A denominação de estocásticos atribuída a esses autômatos dá-se pela razão do tempo ser tratado como uma variável aleatória, a qual obedece a uma distribuição exponencial na escala de tempo contínua e geométrica no caso de escala de tempo discreta [Fer98].

2.2 Estados

Atribui-se o nome de estado global ao estado do modelo SAN, onde são combinados todos os estados locais dos autômatos presentes dentro deste modelo [Fer98, Pla85]. Estado local é o nome dado a cada estado individual de cada um dos autômatos presentes dentro do modelo. Toda vez que um estado local é alterado, este afeta diretamente o estado global do modelo. Toda mudança de um estado local para outro é feita através de transições [Fer98].

Estas transições só podem ocorrer se existir pelo menos um evento diretamente associado a ela, fazendo com que ela seja realizada dentro do autômato. Evento é a entidade do modelo responsável pela ocorrência de uma transição, a qual muda o estado global do modelo [Pla85].

Os eventos que podem ocorrer dentro dos autômatos podem ser definidos de duas formas, como locais ou sincronizantes, definidos nos tópicos seguintes.

2.2.1 *Eventos locais*

Eventos locais são eventos que alteram o estado do autômato em que eles ocorrem, sem que este evento dispare qualquer outra transição em qualquer outro autômato presente dentro do modelo [Ste94, Pla85].

Esse tipo de evento é particularmente interessante, pois permite que vários autômatos tenham um comportamento paralelo, trabalhando independentemente sem que haja interação entre eles, sendo que, mesmo estes agindo sobre um único autômato, podem ter taxas ou probabilidades funcionais, os quais serão abordados no decorrer do trabalho. É possível observar eventos locais na Figura 2.1, onde o autômato $A^{(1)}$ possui os eventos locais e_1 e e_2 enquanto o autômato $A^{(2)}$ possui os eventos e_3 e e_4. Estes eventos apresentam taxas associadas que podem ser constantes e funcionais, taxas estas que serão posteriormente explicadas na seção 2.3.

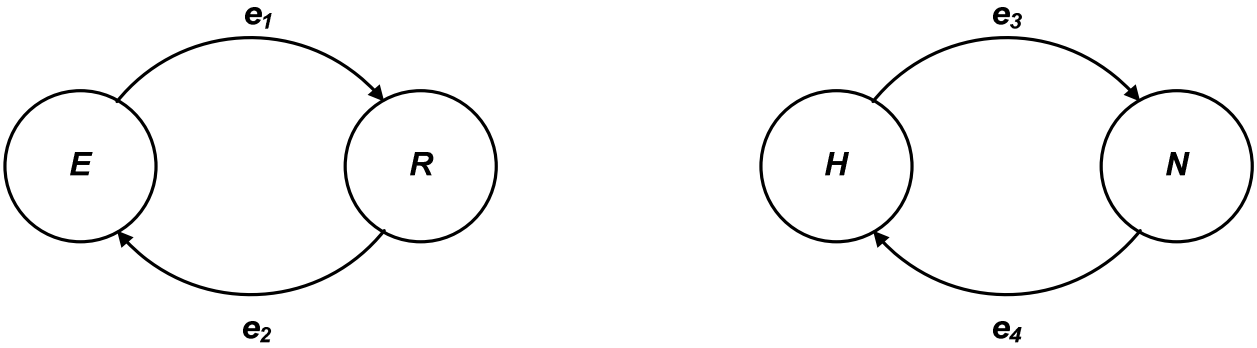
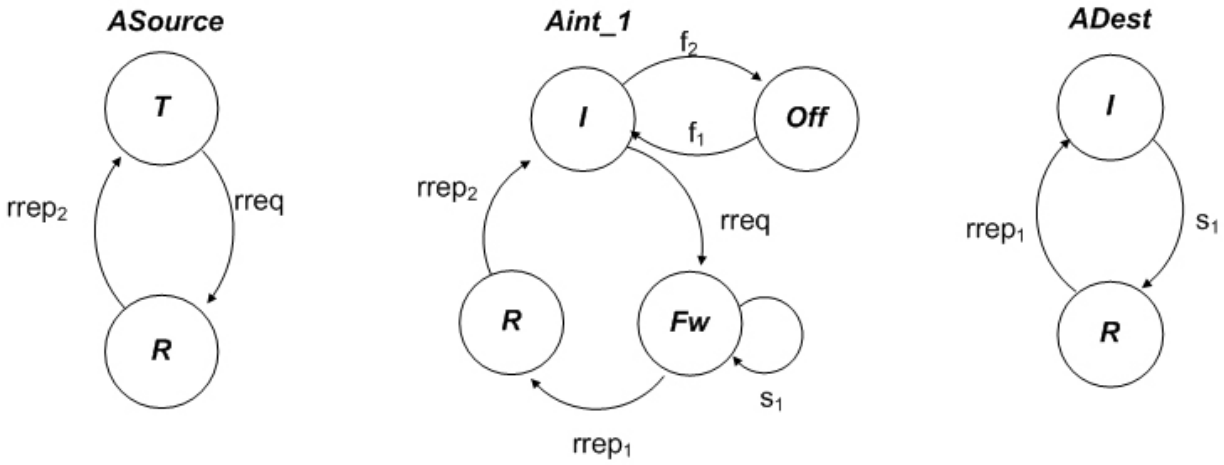


Figura 2.1: Autômatos com seus respectivos eventos locais

2.2.2 Eventos sincronizantes

Evento sincronizante é todo aquele que quando ocorre, afeta mais de um autômato simultaneamente, e o mesmo depende de taxas de ocorrência e de probabilidades associadas a ele para que este possa ser disparado em diversos autômatos ao mesmo tempo.

Essas taxas e probabilidades têm valores associados a elas que podem ser tanto valores constantes ou valores funcionais, sendo que as taxas e probabilidades funcionais assumem valores diferentes conforme os estados dos outros autômatos do modelo, e a interação entre autômatos dá-se através do sincronismo no disparo das transições [Ben03].



Tipo	Evento	Taxa
loc	f_1	0,5
loc	f_2	0,5
syn	rreq	1
syn	s_1	0,333
syn	$rrep_1$	0,777
syn	$rrep_2$	0,4444

Figura 2.2: Exemplo de modelo SAN com eventos locais e sincronizantes

Como se pode observar na Figura 2.2, existem dentro deste modelo SAN três autômatos (*ASource*, *Aint_1* e *ADest*), os quais possuem eventos locais e sincronizantes, cada um com sua respectiva taxa. A tabela presente no modelo é uma representação de como se identificam os eventos dentro do modelo.

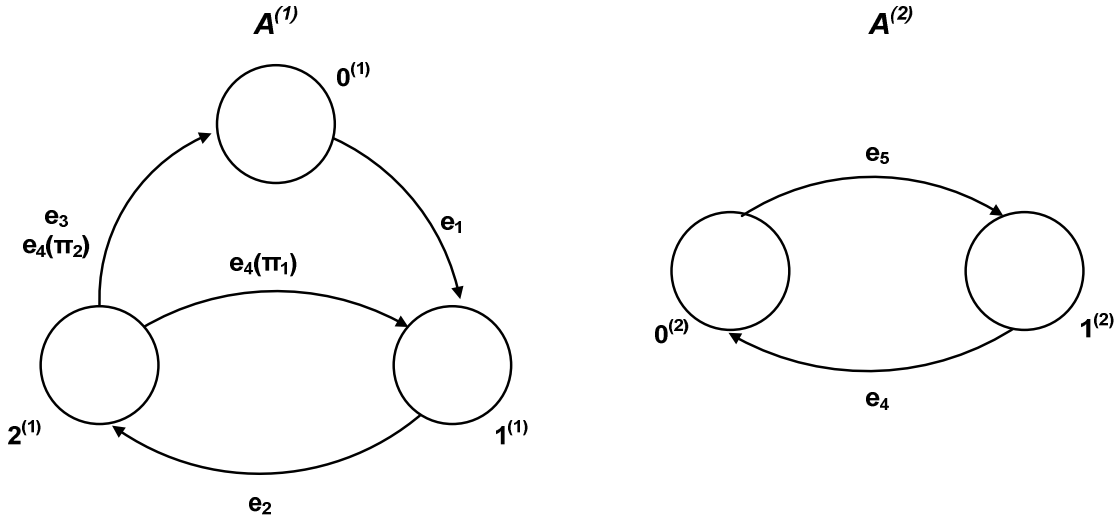
Conforme a tabela presente no modelo, os eventos f_1 e f_2 são eventos locais, particulares do autômato *Aint_1*, enquanto os eventos *rreq*, s_1, $rrep_1$ e $rrep_2$ são classificados como eventos sincronizantes, uma vez que seu disparo afeta mais de um autômato. Como dito anteriormente, cada evento local ou sincronizante pode possuir taxas e probabilidades funcionais, as quais são descritas a seguir.

2.3 Taxas e probabilidades funcionais

As taxas e probabilidades funcionais constituem a segunda possibilidade de interação entre os autômatos de um modelo [Ben03], podendo ou não associar valores diferentes a um mesmo evento, de acordo com o estado global do modelo SAN.

Através da Figura 2.3, é possível identificar que o evento e_4 é um evento sincronizante onde existem probabilidades de transição π_1 e π_2 no autômato $A^{(1)}$, quando da sua mudança de estados tanto dentro do autômato em que este está presente quanto na sincronização com outro autômato.

Para o evento local e_5 foi atribuída uma função f_1 associando taxas de ocorrência dependentes de determinados estados do autômato $A^{(1)}$ [Ben03].



Tipo	Evento	Taxa
loc	e_1	τ_1
loc	e_2	τ_2
loc	e_3	τ_3
syn	e_4	τ_4
loc	e_5	f_1

Figura 2.3: Modelo SAN com taxa funcional
Fonte: adaptado de [Ben03]

No autômato apresentado na Figura 2.3, o evento e_5 por se tratar de um evento com taxas funcionais, faz com que a seguinte configuração de transições ocorra dentro do modelo:

$$f_1 = \begin{cases} \lambda_1 \text{ se autômato } A^{(1)} \text{ está no estado } 0^{(1)} \\ 0 \text{ se autômato } A^{(1)} \text{ está no estado } 1^{(1)} \\ \lambda_2 \text{ se autômato } A^{(1)} \text{ está no estado } 2^{(1)} \end{cases}$$

O que implica dizer que, a taxa de ocorrência na transição do estado $0^{(2)}$ para o estado $1^{(2)}$ ocorre com uma taxa de ocorrência λ_1. Se o autômato $A^{(1)}$ estiver no estado $0^{(1)}$, ocorrerá uma taxa λ_2, ou então, se o autômato $A^{(1)}$ estiver no estado $1^{(1)}$, não ocorrerá transição [Ben03].

Existem outras funções que podem ser utilizadas dentro do contexto de SAN [Ben03], que são as funções de atingibilidade e as funções de integração, explicadas a seguir.

2.4 Função de atingibilidade

A função de atingibilidade é dada a partir da definição dos estados atingíveis [Ben03] e usa as mesmas regras que são seguidas para definir as taxas e probabilidades funcionais através de uma função booleana.

Para que seja possível compreender melhor como funciona a função de atingibilidade, será usado como exemplo um modelo simples de compartilhamento de recursos adaptado de [Ben03] para a realidade de um estacionamento, onde *V* vagas finitas são disponibilizadas para uma quantidade (*C*) de carros não mensuráveis. Cada um desses processos alterna entre os estados *Aguardando* e em uso (*Estacionado*).

Quando um processo deseja se mover do estado *Aguardando* para o estado *Estacionado*, ou seja, quando um carro deseja ocupar uma determinada vaga, e encontram *C* carros já utilizando os recursos, ou seja, ocupando a vaga, este processo então falha ao acessar o recurso e volta ao estado Aguardando. Quando $C = 1$, o modelo acaba reduzido ao problema usual de exclusão mútua. Analogamente, quando $C = V$, todos os processos são independentes e não existem restrições quanto a vagas.

É atribuída então uma taxa que será chamada de γ_i designada para quando um carro deseja sair do estado de Aguardando para poder ocupar a vaga, ou seja, indo para o estado Estacionado, e

para a taxa onde o processo que o carro deixa a vaga, para que outro possa requisitá-la, será atribuído um valor η_i.

Desta maneira, para que as transições entre os estados aconteçam, os seguintes eventos devem ser disparados:

• E_i para todo i-ésimo processo que ocupa a vaga, com a taxa γi; e

• D_i para todo i-ésimo processo que desocupa a vaga, com a taxa η_i.

Partindo do pressuposto que existem mais carros *C* do que vagas *V*, ou seja, todos os carros ocupando vagas se torna um estado inatingível, já que o número de carros é maior que o número de vagas oferecidas, o que faz com que a função de atingibilidade elimine esses estados considerados inatingíveis, levando em consideração que não é possível atender a todos os carros.

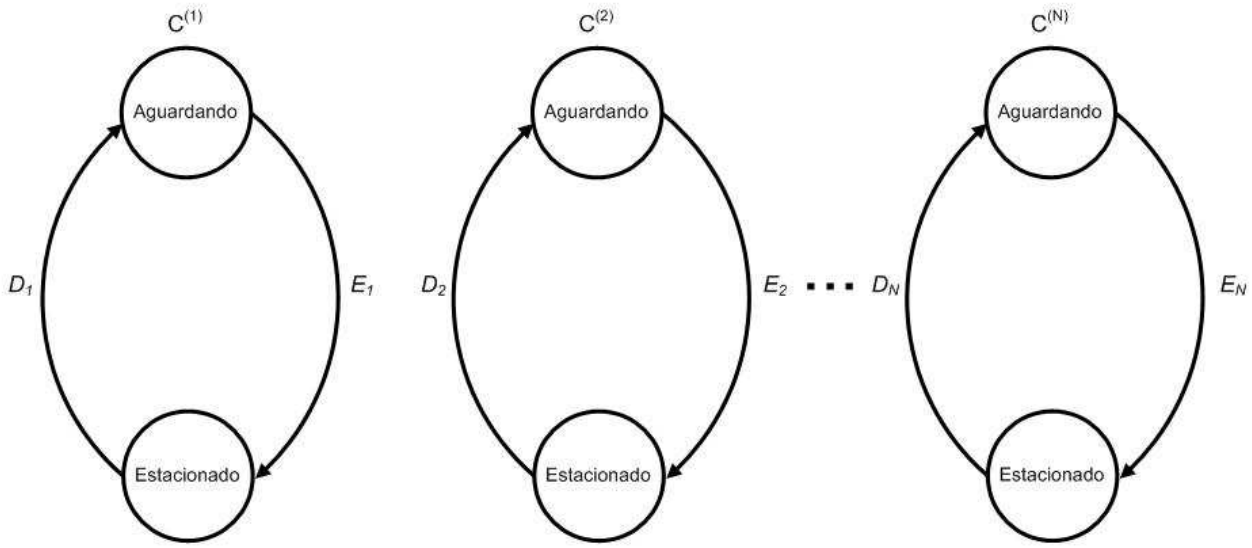


Figura 2.4: Modelo proposto para exemplificação do estado de atingibilidade

A fórmula para a função de atingibilidade, que é chamada *reachability* quando utilizada a ferramenta PEPS para que seja possível modelar a SAN correspondente, pode ser descrita da seguinte forma:

$$reachability = (st\ (C^{(i)}) == Estacionado) \leq V)$$

Outra possibilidade de utilização da função de atingibilidade, específica para o uso na ferramenta PEPS, é a atingibilidade parcial, ou *partial reachability*, onde a atingibilidade é restrita a um subconjunto de estados atingíveis dentro do conjunto possível. Esta função é particularmente especial nos modelos gerados neste trabalho, haja vista a necessidade da realização de um mapeamento das redes, que foi possível graças a essa particularidade, e que será explicitada na apresentação dos modelos no Capítulo 5.

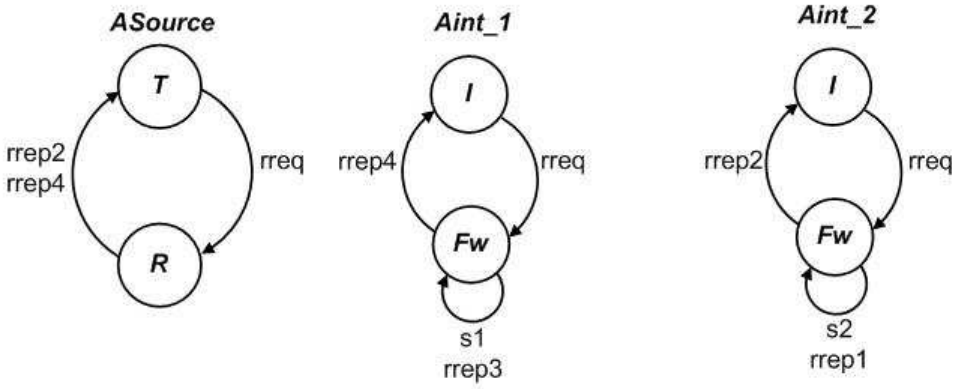
2.5 Funções de Integração

Embora não façam parte do formalismo SAN, As funções de integração, assim com as taxas e probabilidades funcionais e a função de atingibilidade, servem para obter resultados sobre um modelo SAN, apresentando qual a probabilidade desse modelo encontrar-se em um determinado estado ou ponderações sobre a probabilidade de estados, por exemplo, recompensas [Fer98].

Quando um dado modelo se encontrar em um conjunto de estados definidos, índices de desempenho e confiabilidade podem ser elencados a partir do uso de funções de integração, partindo de um vetor de probabilidade que contém a probabilidade do modelo se encontrar em cada um dos estados pertencentes a ele [Ben03]. Esse tipo de função é uma implementação da ferramenta PEPS [Ben03] na obtenção de resultados a partir dos modelos gerados.

2.6 Exemplo de descrição textual

A partir do modelo apresentado na Figura 2.5, será mostrada a descrição textual da referida SAN. Esta descrição textual serve de entrada no processo de avaliação utilizando-se da ferramenta PEPS [Ben03].



Tipo	Evento	Taxa
syn	rreq	0,5
syn	s_1	0,25
syn	s_2	1
syn	s_3	0,2
syn	$rrep_1$	0,2
syn	$rrep_2$	0,2
syn	$rrep_3$	0,2
syn	$rrep_4$	0,2

Figura 2.5: Modelo SAN de exemplificação para descrição textual

A SAN apresentada na Figura 2.5 corresponde a um processo simples de descoberta de rota em uma rede com 5 nodos, sendo um nodo fonte, três nodos intermediários e um nodo destino, onde para que a transmissão seja realizada, o nodo fonte deve estar transmitindo e o nodo destino deve estar ocioso. Será apresentada a descrição textual referente a este ambiente, usada a título de ilustração da sintaxe, uma vez que não foi feita neste trabalho uma descrição formal deste formalismo. É apresentada no texto que se segue a descrição textual da SAN apresentada.

```
identifiers  // Onde os parâmetros são descritos, tanto valores
numéricos
quanto funções

lambda = 0.5;   // taxa de ocorrência do evento rreq
tau    = 0.25; // taxa de ocorrência do evento s1
gama   = 1;   // taxa de ocorrência do evento s2
psi    = 0.2; // taxa de ocorrência do evento s3
alfa   = 0.2; // taxa de ocorrência do evento rrep1
beta   = 0.2; // taxa de ocorrência do evento rrep2
delta  = 0.2; // taxa de ocorrência do evento rrep3
omega  = 0.2; // taxa de ocorrência do evento rrep4
```

```
events   // Onde são apresentadas as descrições dos eventos (tipo,
nome
e taxa)
   syn rreq lambda;
   syn s1 tau;
   syn s2 gama;
   syn s3 psi;
   syn rrep1 alfa;
   syn rrep2 beta;
   syn rrep3 delta;
   syn rrep4 omega;

partial reachability = (st ASource == T) && (st ADest == I);   //
Onde são
definidos os estados atingíveis
dentro do modelo

network Adsr (continuous) //define o nome do modelo e qual o tipo
de
escala de tempo utilizado

aut ASource    //descrição do autômato ASource e das transições
entre os
estados e os seus respectivos eventos
    stt I to (Fw) rreq
    stt Fw to (I) rrep2 rrep4

aut AInt_1
    stt I  to (Fw) rreq
    stt Fw to (Fw) s1 rrep3
           to (I) rrep4

aut AInt_2
    stt I to (Fw) rreq
    stt Fw to (Fw) s2 rrep1
           to (I) rrep2

aut AInt_3
    stt I  to (Fw) s1
    stt Fw to (Fw) s3 rrep1
          to (I) rrep3

aut ADest
    stt I to (R) s2 s3
    stt R to (I) rrep1

results   // Descritas as funções usadas para calcular os índices
de
desempenho dos modelos
t_AInt_2 = st AInt_2 == Fw;   // Função de utilização do nodo 2
(autômato AInt_2)
```

```
t_AInt_3 = st AInt_3 == Fw;   // Função de utilização do nodo 3
(autômato AInt_3)
t_Asource_Idle = st ASource == I;   // Função que verifica a
ociosidade do
nodo ASource
t_ADest_receiving_AInt_2 = (st AInt_2 == Fw) && (st ADest == R);
// Função que define a probabilidade de rota através do nodo 2
t_ADest_receiving_AInt_3 = (st AInt_3 == Fw) && (st ADest == R);
// Função que define a probabilidade de rota através do nodo 3
```

3 REDES *WIRELESS AD HOC* E PROTOCOLOS DE ROTEAMENTO

Pode-se definir uma rede de computadores como uma rede formada por um conjunto de dispositivos ou equipamentos conectados de alguma forma para troca de informações, ou então, como um conjunto de dispositivos conectados por links de comunicação (denominados frequentemente de nós), conforme Fourozan em [Fou06].

Segundo Tanenbaum em [Tan03], as redes de computadores têm entre as mais variadas funções, a de compartilhar recursos. Tais recursos podem vir em forma de uma impressora, um servidor de dados ou até mesmo informações comerciais. A relevância disso para este estudo é a melhor forma de compartilhar esses recursos a fim de garantir o acesso de qualquer nó da rede, independente da sua posição dentre de um espaço físico predeterminado.

E nesta busca por compartilhamento de recursos, ou pelo simples fato da necessidade de interligar dispositivos, novas formas de tornar isso possível foram sendo geradas. A redução ou total extinção de cabeamento na tarefa de conectar nodos dentro de uma rede hoje é passível de ser realizada, graças às tecnologias *wireless* existentes, onde redes sem fio podem ser criadas para os mais diversos fins, desde redes comerciais até redes de uso militar que não podem concentrar informações.

3.1 Redes *wireless Ad hoc*

Redes *wireless*, ou sem fio, basicamente têm os mesmos princípios das LAN's, sendo que seu diferencial mais relevante diz respeito à mobilidade por transmitir seus dados sem o uso de fios, e com taxas de transmissão maiores que algumas que são oferecidas com cabeamento [Tan03].

Tais redes operam transmitindo informações de duas formas topologicamente distintas: de maneira infra-estruturada, onde existe um concentrador (também chamado de *access point*) para gerenciar o envio e recebimento das informações que trafegam pela rede, e de forma *Ad hoc*, onde os dispositivos presentes na rede atuam tanto como clientes (receptores) quando roteadores (transmissores), não havendo uma estrutura fixa ou concentrador de distribuição de informação [Fou06].

Essa configuração de rede tem diversas aplicações, desde o uso militar, passando por redes em ambiente de catástrofe, até redes montadas para interligar participantes em um congresso

[Ram02], atuando sempre dentro de um espaço físico restrito, restrição essa dada pelo alcance de transmissão do sinal. Devido a essa independência física, é possível dentro de uma rede que os nós se movimentem, alterando a configuração física da mesma, e com isso, uma vez que esses nós são transmissores e receptores dos pacotes que pela rede trafegam, é necessário estabelecer regras de roteamento dessas informações.

As pesquisas sobre redes *wireless Ad hoc* se iniciaram na década de 70, quando o DoD (Department of Defense) deu início ao projeto PRNET (*Packet Radio Network*), onde este explorava o uso de redes de pacotes via rádio para ambientes táticos para realizar a comunicação de dados, segundo Ramanathan *et al.* em [Ram02]. Este projeto deu origem ao programa SURAN (*Survivable Adaptive Radio Network*), no início dos anos 80, que tinha como objetivo prover a troca de pacotes em uma rede para elementos móveis em campo de batalha e ambientes hostis sem a menor infra-estrutura, atendendo a soldados, tanques, entre outros, que seriam os nodos da rede.

Em sequência a esses projetos, o DoD criou um programa de pesquisa chamado GloMo, que vinha ao encontro dos anseios de ambientes corporativos com padrão Ethernet que tinham necessidade de conectividade multimídia a qualquer hora e em qualquer lugar, através de dispositivos móveis [Ram02].

As redes *Ad hoc* podem ser definidas como redes que são geradas por um período de tempo determinado, com o fim de atender alguma necessidade ou algum tipo de situação que faça uso de uma estrutura não fixa ou altamente mutável [Ram02]. Neste tipo de redes, todos os nodos presentes movem-se arbitrariamente sem que estejam atrelados a algum tipo de estrutura fixa de comunicação. Contudo, um nodo de rede *Ad hoc* sem fio está limitado a transmitir informações apenas com outros nodos que estejam em seu raio de alcance de transmissão, logo, se um nodo que deseja transmitir para outro fora do seu raio de alcance, dependerá de outros nodos presentes na rede, ditos nodos intermediários, e que farão o papel de roteadores dos pacotes até o seu nodo destino.

Os terminais (ou nós) presentes dentro de uma rede *wireless Ad hoc* podem se comunicar de duas formas distintas: através de comunicação direta, onde todos estão dentro da área (raio) de alcance de todos, ou através de múltiplos saltos, onde, como dito anteriormente, os nós presentes na rede servem de roteadores (intermediários) entre os nós de origem dos pacotes de dados e de destino, uma vez que estes não se encontrem no raio de alcance de um para outro. A Figura 3.1 apresenta graficamente estas formas de comunicação anteriormente citadas, onde a área pontilhada representa o alcance de sinal de cada nodo, sendo na comunicação direta todos os nodos dentro do

alcance dos outros nodos na rede, e na rede de múltiplos saltos, serem necessários o uso de protocolos de roteamento para realizar o repasse e a troca de informações.

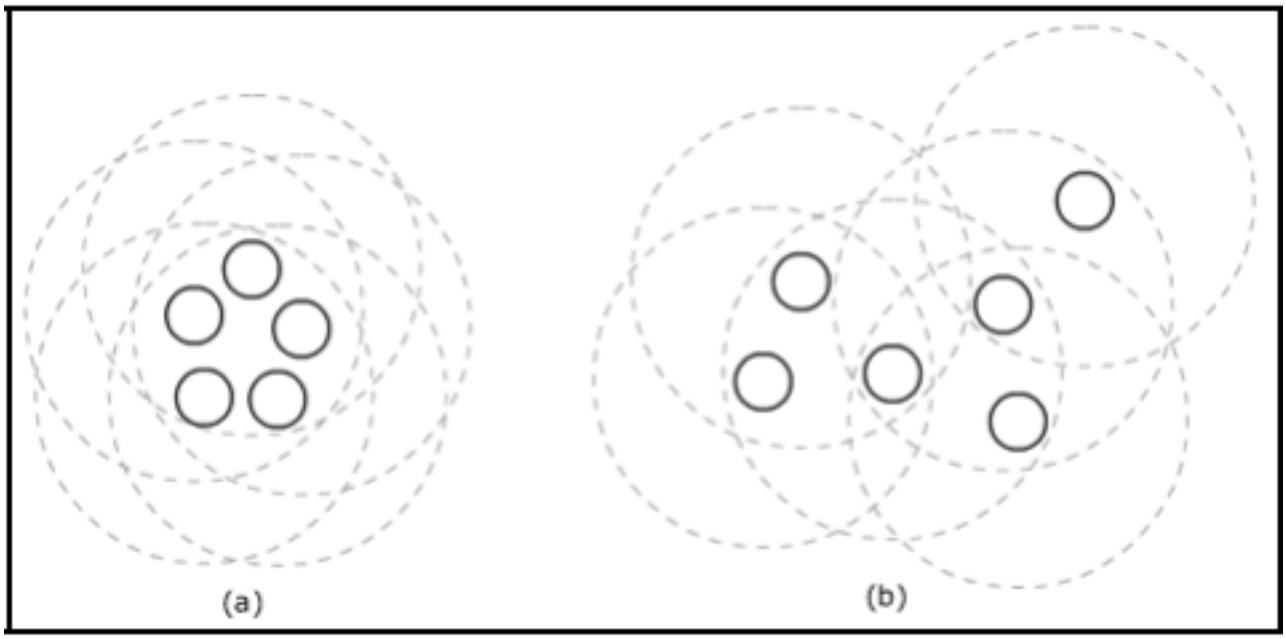


Figura 3.1: Dois tipos de redes *Ad hoc* (a) comunicação direta; (b) múltiplos saltos.
Fonte: [CAM99]

Quando comparadas às redes infra-estruturadas, as redes *Ad hoc* se destacam pelas seguintes vantagens [Cam99]:

• Rápida instalação: as redes podem ser instaladas rapidamente em locais sem nenhuma infra-estrutura prévia pelo fato de que não necessitam de bases fixas para rotear as mensagens;

• Tolerância a falhas: o mal funcionamento ou desligamento de uma estação pode ser resolvido através de uma reconfiguração dinâmica da rede. Em uma rede fixa, ao contrário, quando ocorre uma falha em um roteador, o redirecionamento do tráfego é uma operação complexa, quando possível, ou seja, quando há caminhos suficientes para isto. Falhas em redes infra-estruturadas são mais graves, principalmente se ocorrerem na estação de suporte de mobilidade, pois se trata de uma entidade centralizadora, isto é, todas as comunicações dos nós dependem dela;

• Conectividade: se duas estações estão dentro da área de alcance das ondas de rádio, elas tem um canal de comunicação. Em uma rede fixa, mesmo que duas estações estejam uma ao lado da outra, é necessário que as estações estejam ligadas por um meio guiado, para que troquem informações. Em uma rede infra-estruturada, é necessária a comunicação do host com a estação base, e, desta, para o outro host;

• Mobilidade: em contraposição à falta de mobilidade dos computadores fixos.

Para o correto estabelecimento da troca de pacotes dentro de uma rede *Ad hoc*, foram criados protocolos de roteamento para esta tarefa. Estes serão abordados no tópico que se segue.

3.2 Protocolos de roteamento

Os protocolos de roteamento foram criados para facilitar a comunicação dentro da rede, descobrindo rotas entre os nodos, de acordo com Royer *et al.* em [Roy99]. Ainda segundo a mesma autora, o objetivo principal dos protocolos de roteamento é o correto e eficiente estabelecimento de rotas entre um par de nodos para que as mensagens possam ser entregues de maneira oportuna [Roy99].

O IETF - *Internet Engineering Task Force* - em seu documento que trata diretamente das redes *Ad hoc* [Man99] afirmam que os protocolos de roteamento para redes *Ad hoc* devem atentar às seguintes características:

• Topologia dinâmica: os nós são livres para se moverem arbitrariamente, ou seja, a topologia de rede muda frequentemente e os links podem ser bidirecionais ou unidirecionais;

• Tamanho de banda restrito: links sem fio continuam a ter capacidade significativamente menor do que links das redes fixas;

• Operação para economia de energia: alguns ou todos os nós numa MANET utilizam baterias para fornecimento de energia; neste sentido um dos critérios mais importantes de desenvolvimento do sistema deve ser a conservação de energia;

• Segurança física limitada: as redes móveis sem fio são mais vulneráveis às ameaças à segurança do que as redes fixas; o aumento da possibilidade de escuta, invasão e ataques devem ser cuidadosamente considerados; em contrapartida, por terem um controle descentralizado, possuem como benefício terem maior robustez em relação às redes centralizadas.

Vários esforços têm sido criados no intuito de desenvolver melhorias no funcionamento dos protocolos de roteamento para que estes atendam às características necessárias ao seu propósito, inclusive às acima anteriormente elencadas. Esses protocolos, segundo a IETF [Iet99], devem apresentar algumas propriedades, como as que se seguem:

• Operação distribuída: propriedade essencial para o roteamento em uma rede *Ad hoc* para evitar a centralização que leva à vulnerabilidade;

• Protocolos livres de loops: para que os pacotes não fiquem trafegando durante um período de tempo relativamente grande na rede, pode ser usada como solução uma variável do tipo TTL (*time-to-live*), mas uma abordagem melhor estruturada é mais indicada;

• Operação baseada na demanda: o algoritmo de roteamento deve ser adaptável às condições de tráfego; se isto for feito de forma inteligente, os recursos de energia e largura de banda serão utilizados de forma mais eficiente;

• Segurança: se as camadas de rede e de enlace não garantirem segurança, os protocolos de roteamento MANET estarão vulneráveis a muitas formas de ataque; é necessário que haja mecanismos para inibir modificações nas operações dos protocolos;

• Operação nos períodos de “sonolência” do nó: como resultado da conservação de energia ou de alguma outra inatividade, os nós devem parar de transmitir e/ou receber pacotes, por um período arbitrário de tempo, sem que isto resulte em maiores consequências;

• Suporte a links unidirecionais: tipicamente, uma rede MANET assume links bidirecionais e muitos algoritmos são incapazes de funcionar corretamente sobre links unidirecionais. Entretanto, links unidirecionais podem ocorrer em redes sem fio.

O uso dos protocolos de roteamento remetem à uma série de recomendações, bem como de diversos aspectos que merecem a devida atenção, para que estes venham a atender todas as necessidades de uma rede no momento da troca de informações.

3.3 Taxonomia dos protocolos de roteamento

Para uma melhor compreensão dos protocolos de roteamento para redes *wireless Ad hoc*, foram realizados diversos estudos que focaram diferentes aspectos de tais protocolos, entretanto, Feeney em [Fee99], reuniu todas as características pertinentes aos protocolos de roteamento para redes *Ad hoc*, e apresenta uma taxonomia dos protocolos onde se analisa quatro pontos de extrema relevância para a construção desta:

• Modelo de comunicação: qual é o modelo de comunicação sem fio utilizado?

• Estrutura: todos os nodos são tratados uniformemente? Como são distinguidos os nodos selecionados?

• Estado da informação: a informação sobre a dimensão da topologia da rede é obtida a cada nodo?

• Sincronização: a informação de roteamento é continuamente mantida para cada destino?

3.3.1 Modelos de comunicação

No que tange o modelo de comunicação dos protocolos de roteamento, existem dois tipos de modelo para canal de comunicação, que são os modelos para um único canal e para múltiplos canais.

3.3.1.1 Modelo de Comunicação de múltiplos canais

Segundo Feeney em [Fee99], os protocolos para múltiplos canais são protocolos de roteamento de baixo nível que combinam as tarefas do canal e as funcionalidades de roteamento.

Esses protocolos são geralmente usados em redes baseadas nos padrões TDMA (*Time Division Multiple Access*) e CDMA (*Code Division Multiple Access*), sendo usados também em comunicação de clusters. A maioria dos protocolos, entretanto assumem que os nodos se comunicam através de um único canal lógico sem fio.

3.3.1.2 Modelos de comunicação de único canal

Nos protocolos que utilizam o modelo de comunicação de um único canal, mesmo sendo geralmente orientados pela técnica CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*), esses protocolos variam na extensão com a qual estes confiam nos comportamentos específicos da camada de conexão. Neste estudo, procurou-se abordar protocolos de roteamento que são baseados em propriedades mais específicas da camada de conexão, como a sequência de

controle RTS/CTS (*Request to Send*/*Clear to Send*) usado no padrão IEEE 802.11, na camada MAC, para a prevenção de colisões, sendo que estas características estão presentes em um grande número de trabalhos relacionados.

3.3.2 Estrutura

Quanto à estrutura dos protocolos de roteamento, estes podem ser classificados de duas formas, como uniformes e não-uniformes [Fee99].

3.3.2.1 Protocolos Uniformes

Os protocolos uniformes são assim classificados devido ao fato que nenhum nodo tem um papel distinto no esquema de roteamento, sendo que cada um dos nodos envia e recebe (ou responde) as mensagens de controle de roteamento da mesma forma, não existindo assim uma estrutura hierárquica imposta na rede.

Embora tais protocolos evitem os custos de recursos que são necessários na manutenção de uma estrutura de alto nível, a escalabilidade pode se tornar uma questão complicadora em redes de grande tamanho [Fee99].

3.3.2.2 Protocolos não-uniformes

Os protocolos não uniformes tentam limitar a complexidade de roteamento através da redução dos nodos participantes na transmissão dos dados [Fee99]. Esta abordagem pode prover escalabilidade e reduzir a sobrecarga de comunicação, suportando o uso de algoritmos de grande complexidade computacional ou de comunicação, voltados para redes *Ad hoc*.

Os protocolos não uniformes atuam de duas maneiras: baseado em atividades de roteamento entre os vizinhos do nodo e em particionamento de topologia.

Nos protocolos que operam baseados em atividades de roteamento entre os vizinhos (ou seleção de vizinhos), cada nodo seleciona algum subconjunto entre os seus vizinhos para distinguí-los no tráfego computacional e/ou tráfego de repasse de informações no seu nome [Fee99]. Cada

nodo faz suas seleções independentemente, não há processo de negociação, fazendo com que os nodos realizem um consenso, assim, nenhuma seleção de nodos é afetada por mudanças de topologia que não são ligadas a estas seleções.

Já nos protocolos que trabalham baseados em particionamento de topologia, os nodos negociam um particionamento de topologia da rede, sendo que esta é uma operação distribuída, não havendo uma gerência de topologia central.

Geralmente, os nodos são particionados em grupos, nos quais os membros mudam de acordo com a mudança de conectividade da rede. Alguns nodos têm um papel distinto no processo de roteamento, possivelmente atuando como nodo de negociação (*cluster-head*) entre dois grupos [Fee99].

3.3.3 Estado da Informação

Os protocolos podem ser descritos em termos do estado da informação obtida a cada nodo e/ou troca entre os nodos [Fee99], podendo ser divididos em protocolos baseados em topologia e baseados em destino.

3.3.3.1 Protocolos baseados em topologia

Nestes protocolos, os nodos participantes mantém informações sobre a topologia em larga escala. Os protocolos mais conhecidos desta categoria são os protocolos *link-state*, onde nesses protocolos cada nodo anuncia sua conectividade com cada um dos seus vizinhos diretos a todos os outros nodos da rede. Nesta categoria de protocolos, os caminhos entre os nodos são mantidos (e gerados) através do uso de uma tabela de roteamento presente em todos os nodos [Fee99].

Cada nodo tem uma completa informação da topologia da rede em que está presente o que facilita na escolha do menor caminho para a transmissão das informações, porém, como agravante, isso aumenta consideravelmente o tamanho dos cabeçalhos de transmissão de pacotes e também o espaço de *buffer*.

3.3.3.2 Protocolos baseados no destino

Os nodos que participam em redes que são baseadas em protocolos desta categoria não mantém uma informação de larga escala sobre a topologia da rede, como em protocolos baseados em topologia, e sim mantém informações sobre a topologia local, como por exemplo, vizinhos de um ou dois saltos [Fee99].

A categoria de protocolos mais conhecida entre os protocolos baseados em destino são os protocolos *distance-vector* que mantém uma medida de distância, como um contador de saltos, e um vetor com o próximo salto, para atingir o destino. Cada nodo presente na rede troca suas estimativas de distância para todos os outros nodos da rede com os quais são seus vizinhos diretos. Esta categoria de protocolos tem como vantagem o fato de ser livre de loops de roteamento, uma vez que os nodos possuem informações de topologia de seus vizinhos mais próximos, logo, não repassam a informação para diversas rotas, e sim apenas para aquelas à sua volta.

3.3.4 Sincronização

A sincronização é a última classificação proposta por Feeney [Fee99] em seu estudo da taxonomia dos protocolos de roteamento e também estudada e apresentada por [Bou04] e [Das98], que classificam a sincronização como reativas e pró-ativas.

3.3.4.1 Protocolos pró-ativos

Protocolos pró-ativos ou "*table-driven*" são protocolos que tentam manter informações de roteamento para todos os destinos conhecidos em todas as fontes [Bou04, Fee99], que são todos os nodos que desejam enviar informações. Nestes protocolos, os nodos trocam informações sobre rotas periodicamente de acordo com a mudança da topologia da rede.

Isto pode ser considerado como uma vantagem, uma vez que reduz o atraso na obtenção da rota quando o tráfego para o destino é iniciado, pois rapidamente o caminho entre fonte e destino é determinado. Entretanto, este processo pode consumir recursos significativos de rede. Além disso, os recursos usados para estabelecimento e restabelecimento de rotas não usadas é inteiramente desperdiçado [Fee99].

3.3.4.2 Protocolos sob demanda ou reativos

Em ambientes de redes *Ad hoc*, o custo de manutenção de informação desnecessária de roteamento é muito mais grave do que em redes fixas. Em virtude disso, protocolos que descobrem rotas apenas quando solicitados - ou sob demanda - foram propostos [Fee99].

Estes protocolos consistem de processos de descoberta de rota (*route discovery*) e de manutenção de rota (*route maintenance*) para estabelecer e manter as rotas quando estas são requisitadas [Bou04, Das98].

O processo de descoberta de rotas é iniciado quando um nodo fonte deseja uma rota para um nodo destino, pelo qual este transmite uma requisição de rota. Cada nodo intermediário que recebe a requisição armazena o link e substitui o último que foi recebido, e o retransmite (ignorando silenciosamente rotas duplicadas). Quando uma requisição atinge o nodo destino, este envia uma resposta de rota de volta ao nodo fonte, indicando a rota pelos nodos intermediários por onde a informação deve percorrer. Pode ocorrer de o nodo destino receber diversas requisições de rota e pode selecionar uma delas baseado em alguma métrica. Assim que a resposta atinge o nodo fonte, o tráfego dos dados pode ser iniciado para o nodo destino.

Os protocolos sob demanda têm a vantagem de não gastar recursos computacionais mantendo rotas desnecessárias, pois somente iniciam o processo quando um nodo fonte solicita uma rota. O processo de transmissão de requisições de rota dá-se comumente pelo processo de *flooding* de rede, ou seja, ocorre uma “inundação” de mensagens na rede, já que uma grande quantidade de mensagens é enviada em um mesmo momento. Uma vez que este processo pode causar redundância de informações de rota, técnicas de contenção ou de prevenção de colisão são muito utilizadas [Fee99].

O outro processo presente em protocolos sob demanda é o processo de manutenção de rota. Quando é detectada uma mudança na topologia da rede, ou uma conexão falha, como por exemplo, um nodo se retira da rede, reinicia-se o processo de descoberta de rota para determinar novas rotas ou então utilizar rotas “sub-ótimas” para o nodo destino [Bou04, Fee99].

O protocolo que será apresentado no capítulo seguinte, DSR [Joh96a], faz parte dos protocolos reativos ou sob demanda. Na Figura 3.2 foi realizada a taxonomia do protocolo DSR, de acordo com o que fora discorrido neste capítulo.

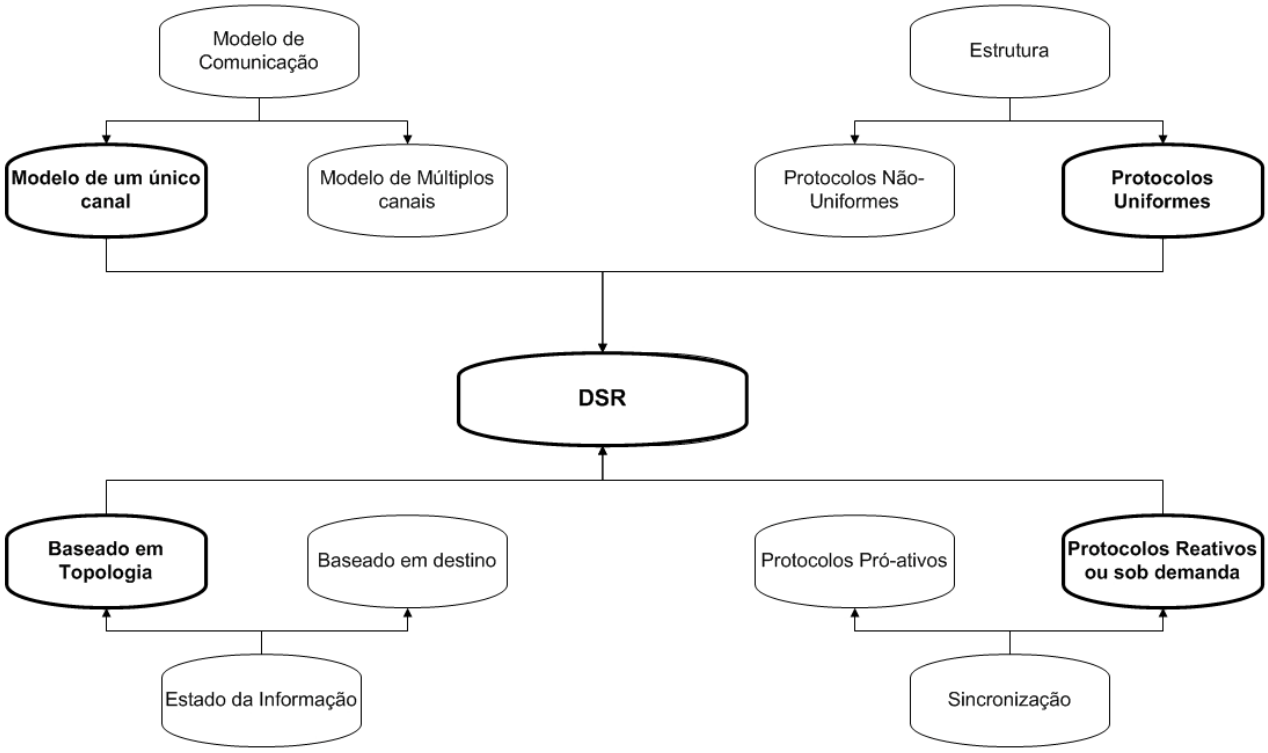


Figura 3.2: Taxonomia do protocolo DSR

4 PROTOCOLO DSR – *DYNAMIC SOURCE ROUTING*

Conforme Johnson em [Joh96a], o protocolo DSR é um protocolo de roteamento criado especificamente para ser utilizado em redes de nodos móveis *wireless Ad hoc* de múltiplos saltos. Este protocolo permite que a rede seja auto-configurável e se auto-organize, sem a necessidade de qualquer administração de rede ou infra-estrutura.

Dentre os protocolos utilizados em redes *wireless Ad hoc*, este é um dos mais aplicados, devido à sua simplicidade em relação à troca de informações, e também ao custo dessa operação, uma vez que os nodos presentes em uma rede que utiliza o protocolo DSR não armazenam informações sobre a topologia de rede referente a todos os nodos presentes na rede, mas sim possuem um *cache* de possíveis rotas que possam ser utilizadas a partir de determinado nodo fonte.

Como um dos objetivos deste trabalho remete à utilização de redes de autômatos estocásticos como alternativa na modelagem e avaliação de redes de computadores, em especial redes *wireless Ad hoc* mais especificamente em relação a algum protocolos de roteamento, optou-se então por modelar o protocolo DSR, dentre uma gama de protocolos, por este possuir características passíveis de serem modeladas, tais como os processos de descoberta e manutenção de rota, bem como pelo fato deste protocolo estar presente na maioria das avaliações de protocolos de roteamento para redes *wireless Ad hoc* na literatura.

O funcionamento do protocolo DSR, bem como suas características serão discutidas nos tópicos que se seguem, para que seja possível fundamentar a modelagem do mesmo, e através dos trabalhos relacionados, obter índices para a construção das variáveis que auxiliaram no processo de avaliação do protocolo utilizando redes de autômatos estocásticos.

4.1 Formatos dos cabeçalhos do protocolo DSR

Os esquemas de cabeçalho para o protocolo DSR quando aplicados ao formato IPv4 consiste de quatro cabeçalhos individuais principais, conforme citam [Adi08, Joh07], que são: o cabeçalho DSR de pacotes de dados, o cabeçalho DSR fixo (de opções), o cabeçalho DSR RREQ (para a requisição de rota) e o cabeçalho DSR RREP (para a resposta de rota).

O cabeçalho responsável pelos dados relacionados aos pacotes de dados é composto do cabeçalho IP, que ocupa a primeira parte, seguido do cabeçalho fixo do DSR (cabeçalho de opções). Logo após, tem-se o cabeçalho das rotas de fonte DSR, com os endereços e a requisição de ACK de DSR e por fim, o bloco dedicado aos dados. Na Figura 4.1, é apresentada a estrutura do cabeçalho DSR de pacotes de dados.

Figura 4.1: Cabeçalho DSR de pacote de dados
Fonte [Adi08]

O cabeçalho fixo DSR, também conhecido como cabeçalho de opções, que vem após o cabeçalho IP, contém uma ou mais das seguintes opções [Adi08]: RREQ, RREP, erro de rota, *acknowledge*, requisição de *acknowledge* e rota de fonte DSR. Estas opções podem variar conforme a necessidade da rede e de transmissão de cada rede. A Figura 4.2 apresenta o formato do cabeçalho fixo DSR.

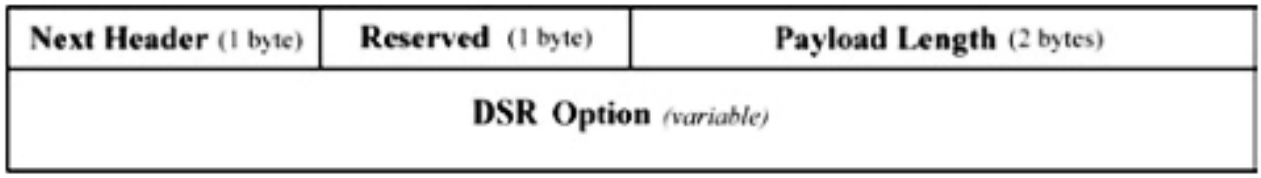


Figura 4.2: Cabeçalho DSR fixo (cabeçalho de opções)
Fonte: [Adi08]

O formato de cabeçalho RREQ do DSR contém os seguintes campos: *option type* (tipo de opção), que é um campo opcional. O campo *option length* (comprimento da opção), que é um byte inteiro não marcado, representa o tamanho da opção. O campo *Identification* (Identificação) é um único número anexado a qualquer par de mensagens RREQ-RREP [Adi08].

O campo *Destination Address* contém o endereço do destino que a informação irá percorrer. O índice IN representa a interface de índice que indica que um nodo identificado pelo endereço (*i*) recebeu o pacote RREQ, e o índice OUT é a interface de índice pelo qual o pacote RREQ foi

retransmitido. Os endereços de saltos (*hop addresses*) são onde os saltos por onde a mensagem RREQ trafegou são gravados. A Figura 4.3 ilustra melhor a composição deste cabeçalho.

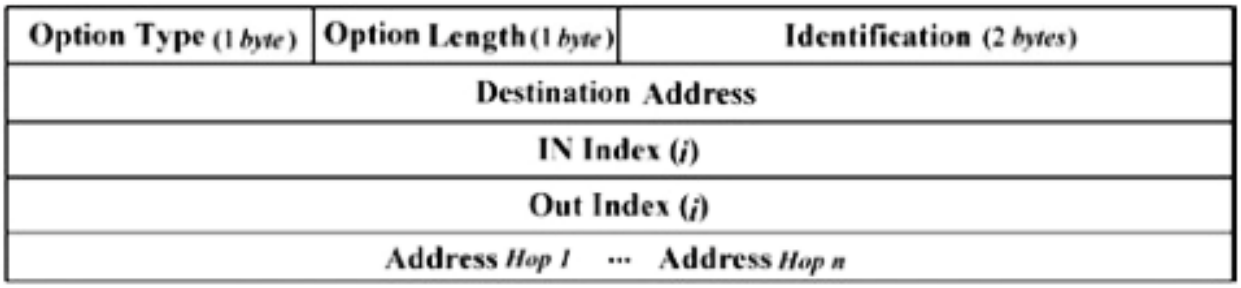


Figura 4.3: Cabeçalho RREQ (para requisição de rotas)
Fonte: [Adi08]

o cabeçalho RREP é similar ao formato do cabeçalho RREQ, exceto pelo byte *reserved*, o qual é definido para zero, ou será ignorado pelo destinatário. Na Figura 4.4, o formato do cabeçalho RREP é apresentado com seus respectivos campos.

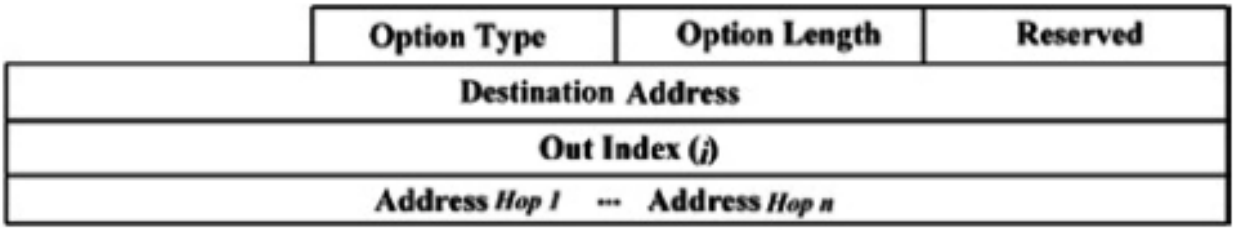


Figura 4.4: Cabeçalho RREP (para resposta de rotas)
Fonte: [Adi08]

4.2 Funcionamento

O protocolo DSR utiliza duas técnicas para a transmissão de dados dentro de uma rede *wireless Ad hoc*:

• descoberta de rota, que é quando um nodo fonte deseja enviar um pacote para um nodo destino, este nodo fonte obtém uma rota para o nodo destino, porém, quando este nodo fonte não possui uma rota armazenada no seu *cache*, este inicia o mecanismo de descoberta de rotas para que

através da passagem da informação de rota entre os nodos intermediários seja possível chegar ao nodo destino; e

• manutenção de rota, que é um mecanismo onde o nodo fonte, está apto a detectar, enquanto usa uma determinada rota para o nodo destino, se a topologia de rede mudou tal que não possa mais usar a rota corrente para a transmissão até o nodo destino. Quando a manutenção de rota detecta que a rota possui uma quebra ou um erro de transmissão, esta automaticamente inicia um novo processo de descoberta de rota entre o nodo fonte e o nodo destino [Joh96b]. Estes mecanismos serão melhor exemplificados nos próximos tópicos deste capítulo.

O protocolo DSR permite que nodos possam acessar ou deixar a rede de forma muito dinâmica, ou então que características de transmissão das redes *wireless* - como, por exemplo, o alcance - venham a mudar, e diante disso, todo o roteamento é automaticamente determinado e mantido, sem que haja perdas ou problemas de conexão entre os nodos [Joh96a]. Uma vez que o número ou a sequência de nodos intermediários necessários para atingir qualquer nodo destino pode mudar a qualquer momento, a topologia de rede pode mudar rapidamente para adequar a transmissão dos dados.

O protocolo também permite que os nodos presentes na rede descubram rotas de fonte (*source route*) [Joh96a] através de múltiplos saltos na rede para qualquer destino na rede *Ad hoc*.

Cada pacote de dados carrega em seu cabeçalho uma completa e ordenada lista dos nodos através do qual os pacotes devem passar, permitindo que o roteamento dos pacotes seja livre de loops e evitando a necessidade de atualizações de informações de roteamento nos nodos intermediários pelo qual o pacote está sendo repassado.

4.2.1 Descoberta de Rotas – *Route Discovery*

O protocolo DSR utiliza o mecanismo de descoberta de rotas sempre que um nodo fonte deseja enviar um pacote para um nodo destino e este nodo fonte não tem possui nenhuma rota armazenada em seu *cache* para o nodo destino. Este armazenamento dá-se na memória de rotas (*Route Cache*) de cada nodo, que é previamente gravado sempre que um nodo fonte deseja enviar um pacote de dados para algum nodo destino.

Na Figura 4.5, o nodo fonte (*Source*) deseja trocar informações com o nodo destino (*Destination*), porém, como não possui uma rota armazenada em seu *Route Cache*, este inunda a

rede com uma mensagem chamada RREQ (*Route Request*) contida em um único pacote de transmissão, que será possivelmente recebido por todos os nodos ativos presentes dentro da rede no alcance do nodo destino (*Source*).

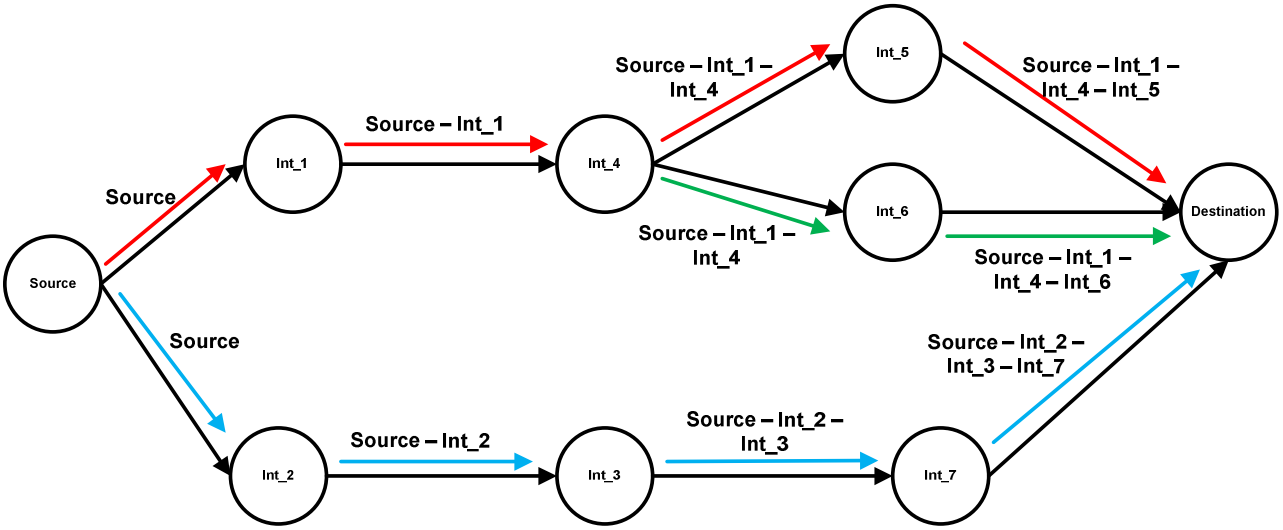


Figura 4.5: Processo de descoberta de rotas do protocolo DSR dentro de uma rede *wireless Ad hoc*
Fonte: O autor

Cada mensagem de requisição de rota identifica o nodo fonte e o nodo destino da descoberta de rota, e grava ao longo do processo de descoberta de rota, juntamente com o fonte e o destino o endereço de cada nodo intermediário através do qual essa requisição foi repassada. Esse processo de gravação de endereços é inicializado em uma lista vazia no nodo fonte que solicitou o processo de descoberta de rota [Joh96a].

Quando um nodo recebe a requisição de rota e este é o nodo destino da descoberta de rota, o mesmo retorna uma mensagem de RREP (*Route Reply*) para o nodo fonte, onde está contida a rota acumulada percorrida gravada da requisição de rota. Assim que o nodo fonte recebe a resposta de rota, este grava em seu *cache* esta rota, para ser utilizada em futuros envios para aquele determinado destino.

É possível que um nodo receba mais de uma requisição de rota partindo do mesmo nodo fonte, logo, este descarta a requisição, uma vez que esta já foi feita. Por outro lado, este nodo acrescenta seu próprio endereço à lista de endereços da rota da mensagem de requisição de rota e o propaga para que a transmissão continue até o destino [Joh96a].

Quando a mensagem de resposta de rota retorna para o nodo que gerou a descoberta de rota, tal como apresentado na Figura 4.6 onde o nodo destino (*Destination*) retorna a rota para o nodo fonte (*Source*), o nodo destino examina seu próprio *cache* de rotas para localizar uma rota de volta para o nodo fonte, e se encontra, irá utilizá-lo na rota para o fonte para entregar o pacote de resposta de rota.

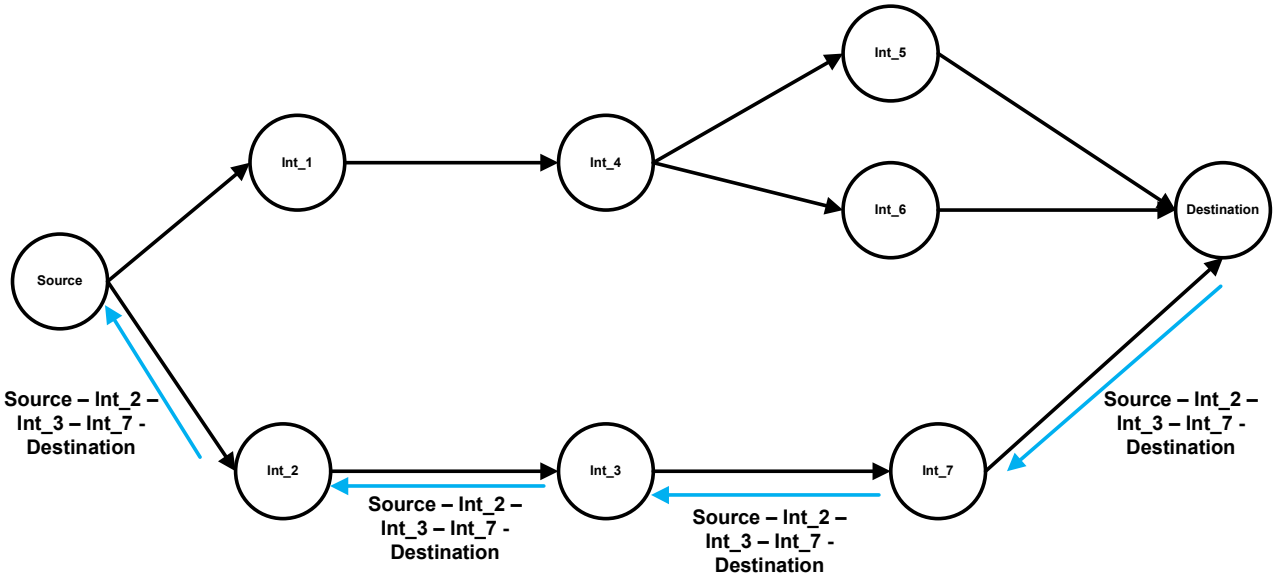


Figura 4.6: Processo de resposta de rota dentro de uma rede *wireless Ad hoc*
Fonte: O autor

É possível que o nodo destino gere seu próprio processo de descoberta de rota para o nodo fonte, o que poderia gerar *loops* infinitos de resposta de rota. Esse problema é prevenido utilizando-se de processos de *piggybacking* da mensagem de resposta de rota a partir da mensagem de requisição de rota.

Em especial para redes *wireless*, que utilizam o protocolo MAC nos padrões 802.11 [Iee07], é utilizado o processo de rota reversa (ou seja, a rota que foi utilizada para a requisição de rota é a mesma para a resposta de rota) para prevenir um segundo processo de descoberta de rota e com isso gastos desnecessários de recursos da rede [Joh96a].

Quando um processo de descoberta de rota é iniciado, o nodo fonte inicia um processo de *buffer* para a lista de endereços de nodos intermediários até chegar ao nodo destino. Este processo é chamado de *Send Buffer* [Joh96a, Joh96b], e contém uma cópia de cada pacote que não pode ser transmitido por este nodo por não ter uma rota para o destino. Técnicas de FIFO ou outra técnica de

substituição pode ser utilizada para prevenir a sobrecarga desse *buffer* ou então para desalocar pacotes antes destes expirarem e não serem transmitidos.

4.2.2 Manutenção de rotas – *Route Maintenance*

Comumente, os protocolos de roteamento utilizam de algum mecanismo de monitoramento de rotas, como o envio de pacotes de atualização a fim de manter as rotas sempre conhecidas. No protocolo DSR, porém, outro método é utilizado para manter suas rotas ativas sem despender recursos computacionais, que é o mecanismo de manutenção de rotas.

Tal mecanismo funciona da seguinte forma: quando um pacote é gerado ou repassado usando uma rota a partir do nodo fonte, cada nodo que transmite tal pacote é responsável pela confirmação que o pacote foi recebido pelo próximo nodo ao longo da rota, utilizando um determinado número de tentativas de entrega, até que o pacote seja entregue ao nodo destino [Joh96a, Joh96b].

Quando um nodo esgota o número de tentativas de entrega do pacote para o nodo seguinte na rota, este gera uma mensagem de Erro de rotas (*Route Error*) para o nodo que gerou o pacote, para identificar em que ponto da rota a conexão foi quebrada ou interrompida, e com isso, o nodo fonte retira esta rota do seu *cache* de rotas, gerando um novo processo de descoberta de rotas, para que o pacote seja entregue para o nodo destino de forma correta [Joh96a, Joh96b].

4.3 Características

4.3.1 Prevenção de “tempestades” de respostas de rotas

O protocolo DSR utiliza como dito anteriormente, dois processos para a descoberta de rotas quando estas não são conhecidas pelo nodo que deseja enviar informações, que são os processos de *Route Request* e *Route Reply*, sendo o primeiro para solicitar a rota entre os nodos intermediários até o nodo destino e o segundo traz as informações armazenadas de rota, confirmando assim a rota a ser seguida e estabelecendo contato com o nodo fonte para a transmissão de dados.

Toda vez que um nodo recebe a mensagem de *Route Request*, este verifica em seu *cache* de rotas se possui uma rota conhecida, e então confirma a rota e repassa a informação. Este processo pode vir a causar colisões dentro da rede, uma vez que o nodo fonte envia sua *Route Request*, esta inunda a rede alcançando todos os nodos ativos presentes na rede e em seu raio de alcance. Como estes nodos podem vir a receber simultaneamente a requisição de rota, um processo de *delay* aleatório é gerado pelo próprio protocolo, para que o nodo que esteja enviando a resposta de rota possa atrasar o envio desta, de forma a evitar colisões na rede.

Este processo é particularmente possível graças ao modo promíscuo, que será tratado neste capítulo, pois como os nodos podem "ouvir" a transmissão dos pacotes de *Route Reply*, este envia a sua própria resposta de rota randomicamente, durante um período de tempo determinado pela seguinte fórmula [Joh96a]:

$$D = H * (h - 1 + r)$$

onde h é o comprimento em número de saltos na rede para a rota retornar a resposta de rota no nodo, r é um valor randômico entre 0 e 1, e H é um pequeno *delay* constante (pelo menos duas vezes o *delay* máximo de propagação de conexão sem fio) a ser introduzido por salto [Joh96a].

Este *delay* randomiza o tempo pelo qual cada nodo envia sua resposta de rota, com todos os nodos enviando as respostas de rota fornecendo rotas de comprimento menores que h, enviando suas respostas antes deste nodo, e todos os nodos enviando as respostas de rota fornecendo rotas de comprimento maior que h enviando suas respostas depois deste nodo.

Dentro deste período de *delay*, este nodo recebe promiscuamente todos os pacotes, observando os pacotes de dados do nodo fonte que iniciou a transmissão destinada ao nodo destino, e, se o pacote de dados recebido pelo nodo durante o período de *delay* conter uma rota de comprimento menor ou igual a h, este nodo pode inferir que o nodo fonte já tenha recebido uma resposta de rota, ofertando uma rota igualmente satisfatória ou melhor que esta. Neste caso, este nodo cancela seu timer de atraso e não envia sua resposta de rota para esta descoberta de rota.

Tendo como objetivo a prevenção de colisões durante a transmissão das mensagens de comunicação entre os nodos, esta característica só é passível de ser utilizada quando a escuta promíscua na rede for possível, fazendo com que os nodos tenham acesso às mensagens de requisição e resposta de rota. Assim, é necessário conceituar o processo de escuta promíscua na rede, citado no tópico que se segue.

4.3.2 Escuta Promíscua – *Promiscuous Listening*

Assim como os demais protocolos, o protocolo DSR possui diversas características e otimizações para melhor realizar as suas atividades. Dentre as diversas características, optou-se por apresentar e avaliar neste trabalho o mecanismo de escuta promíscua, uma vez que este mecanismo tem um comportamento passível de avaliação com o uso de SAN, por se tratar de um processo de transmissão com comportamento paralelo, semelhante ao apresentado em [Dot05], onde, através de duas cadeias de nodos, modelou-se a interferência de um determinado nodo de uma cadeia de nodos na transmissão da outra cadeia. Para o modelo proposto neste trabalho, a título de comparação, o processo de escuta promíscua atua semelhantemente a interferência modelada em [Dot05].

O mecanismo de escuta promíscua ou modo promíscuo permite que os caminhos entre os nodos sejam reduzidos, desde que seja possível para um nodo “ouvir” a transmissão do pacote, e este tenha uma rota para o nodo fonte que possa otimizar o caminho, tornando-o mais curto ou mais rápido até o seu destino.

Um nodo pode escutar a transmissão de um pacote de dados ou de requisição de rota e adicionar essas rotas para seu *cache*. Assim, quando uma requisição ou um pacote estiver sendo transmitido dentro do raio de alcance de transmissão deste nodo, e este possuir uma rota em seu *cache* para o nodo destino da rota que vem sendo transmitida, pode oferecer uma rota alternativa ou otimizada para o destino, sem que isso gere problemas na entrega dos pacotes [Rao04, Joh96b].

Na Figura 4.7, onde a grade representa a área de atuação da rede *wireless Ad hoc* modelada, estão sendo geradas duas requisições de rotas, através dos nodos *Source1* e *Source2*. O nodo *Int_21* está no raio de alcance do nodo *Int_13*, que estará transmitindo a requisição de rota, e assumindo que este nodo tem armazenado em seu *cache* uma rota conhecida para ambos os nodos destino que irão receber os pacotes. Logo, a rota definida pelas setas vermelhas mostram que a rota em que tanto o nodo *Int_21* e *Int_25* podem oferecer para o nodo fonte Source1 através do nodo *Int_13* um caminho dentro das redes presentes na topologia para chegar ao nodo destino Destination1.

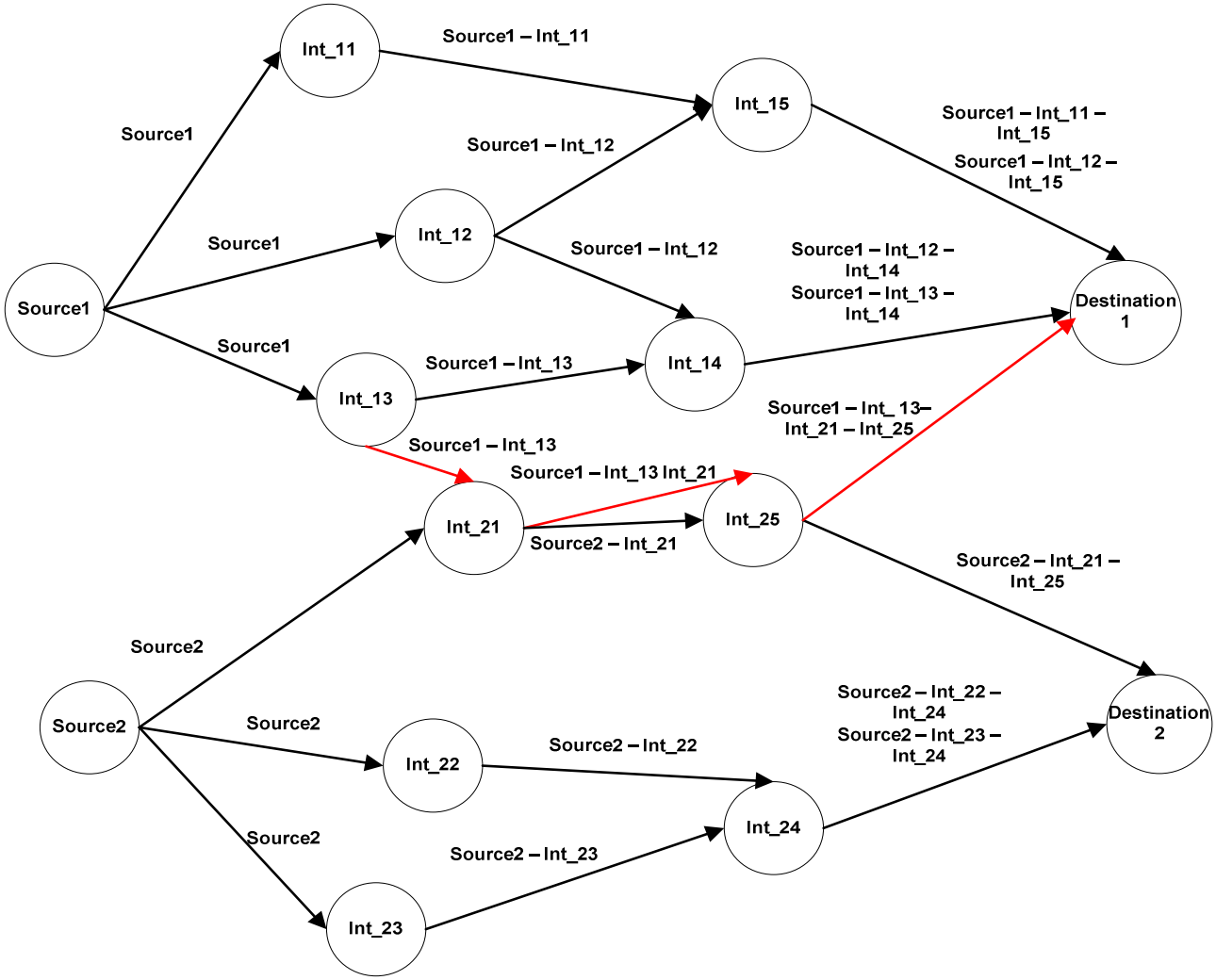


Figura 4.7: Exemplo de escuta promíscua

Mesmo não participando diretamente do processo de descobertas de rota do nodo fonte Source1, os nodos *Int_21* e *Int_25*, como assumido anteriormente, podem ter em seu *cache* de rotas um caminho até o nodo *Destination1*. O modo de escuta promíscuo não garante efetivamente a melhor rota para um determinado nodo destino, mas seu uso é comumente aplicado a rotas mais curtas entre os nodos. Na Figura 4.8, é apresentada a resposta de rota gerada pelo nodo destino levando em consideração a rota utilizada no modo promíscuo.

A Figura 4.8 mostra o processo de resposta de rota, citado anteriormente, agora com a diferença de que esta resposta passa por nodos que “ouviram” a requisição de rota e realizaram o processo de repasse da requisição de rota. Como o nodo *Int_21* estava ao alcance do nodo *Source1*, e tinha armazenado em seu *cache* um caminho para o nodo *Destination1*, este também realizou o

repasse da requisição de rotas para o nodo *Int_25*, que por não ser o nodo destino, mas ter uma rota para o mesmo fez o repasse da mensagem.

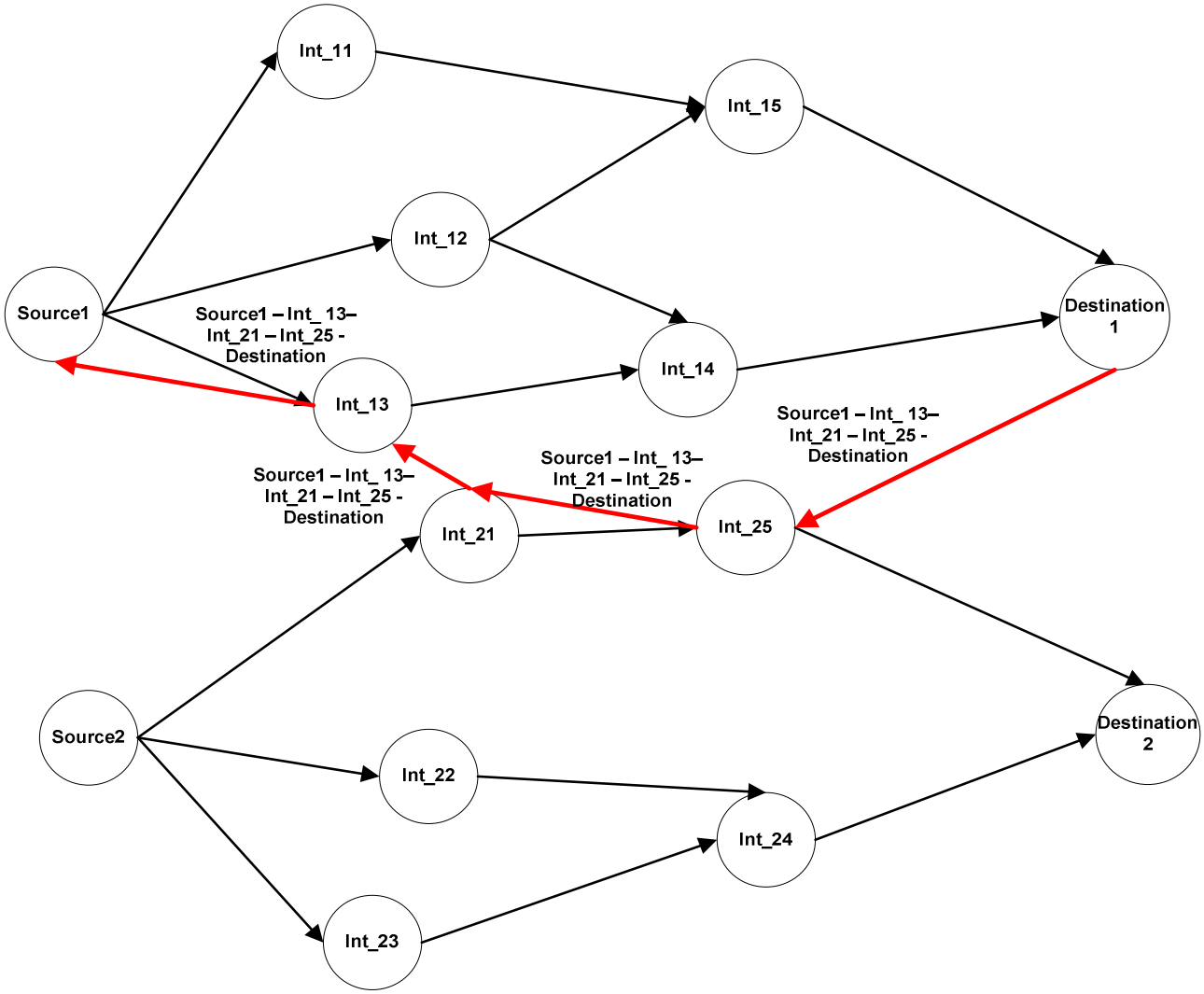


Figura 4.8: Resposta de rota gerado pelo modo de escuta promíscua

Neste processo, por ter se mostrado mais eficiente em relação a outras rotas, sendo este desempenho medido através de alguma métrica associada ao processo, o nodo *Destination1* retornará a resposta de rota através destes nodos, que mesmo não participando efetivamente da rede no qual tanto fonte e destino participam, possuíam informações de rota previamente armazenadas em seu *cache* e puderam contribuir para o estabelecimento da conexão entre os nodos.

Existem diversas implementações pertencentes ao protocolo DSR, mas como o intuito deste trabalho é focado no processo de descoberta de rotas e avaliação do protocolo a partir do uso do formalismo SAN, optou-se por descrever somente a escuta promíscua, uma vez que esta apresenta

particularidades passíveis de avaliação com o formalismo SAN, como processamento de rotas baseada em princípios de paralelismo, como nodos transmitindo e recebendo informações, eu uma rede ativa, sem que estas atividades interfiram na transmissão ou ociosidade de outra rede presente dentro do mesmo modelo, ou então duas redes transmitindo informações particulares de cada rede, e ainda assim, nodos poderem compartilhar de informações das duas redes.

5 MODELOS GERADOS

Neste estudo, para a geração dos modelos que representam as características do protocolo DSR, foram assumidas algumas situações de comportamento da rede que fora criada para esta modelagem.

Assim como nos trabalhos relacionados anteriormente [Bou04, Bro98, Das98, Joh96a, Joh96b, Oli05, Per01, Rao04], algumas situações de ambientação foram criadas para a modelagem proposta para que fosse possível gerar dados que se assemelhassem à um ambiente real.

Dentre as situações assumidas para a construção dos modelos, podem-se elencar como principais as seguintes:

• A rede tem inicialmente, um número limitado de nodos, com a seguinte composição: um ou dois nodos fonte; um ou dois nodos destino e entre sete e quatorze nodos intermediários interagindo entre si na transmissão de mensagens;

• Todas as medições realizadas levaram em consideração determinados momento da rede e de sua topologia, através do uso de tempo de pausa, onde os nodos permanecem estáticos em um determinado momento e em uma determinada posição dentro da malha da rede, quando da necessidade de se avaliar índices que dependessem desse tipo de situação entre uma transmissão e outra;

• O padrão de mobilidade *Random Waypoint* (RWP) foi admitido para os modelos deste trabalho, sendo este o mesmo utilizado em [Joh96a] na avaliação do protocolo DSR, através de uma taxa de movimentação constante;

• Foi assumido, assim como em outros trabalhos encontrados na literatura que existe uma frequência constante de requisições de rota partindo do nodo fonte, fazendo com que a rede tenha uma atividade contínua;

• Vários tamanhos de pacotes foram testados, para poder criar um comparativo de desempenho de transmissão, descartando perdas, em um primeiro momento, e então consideramos possibilidades de falhas na transmissão com a possibilidade de nodos indisponíveis na rede;

• Alguns valores dos cabeçalhos dos pacotes de dados foram aproximados, levando em consideração redes com taxas constantes, para adequar nosso modelo nos moldes dos encontrados na literatura utilizados para simulação [Bro98, Bou04].

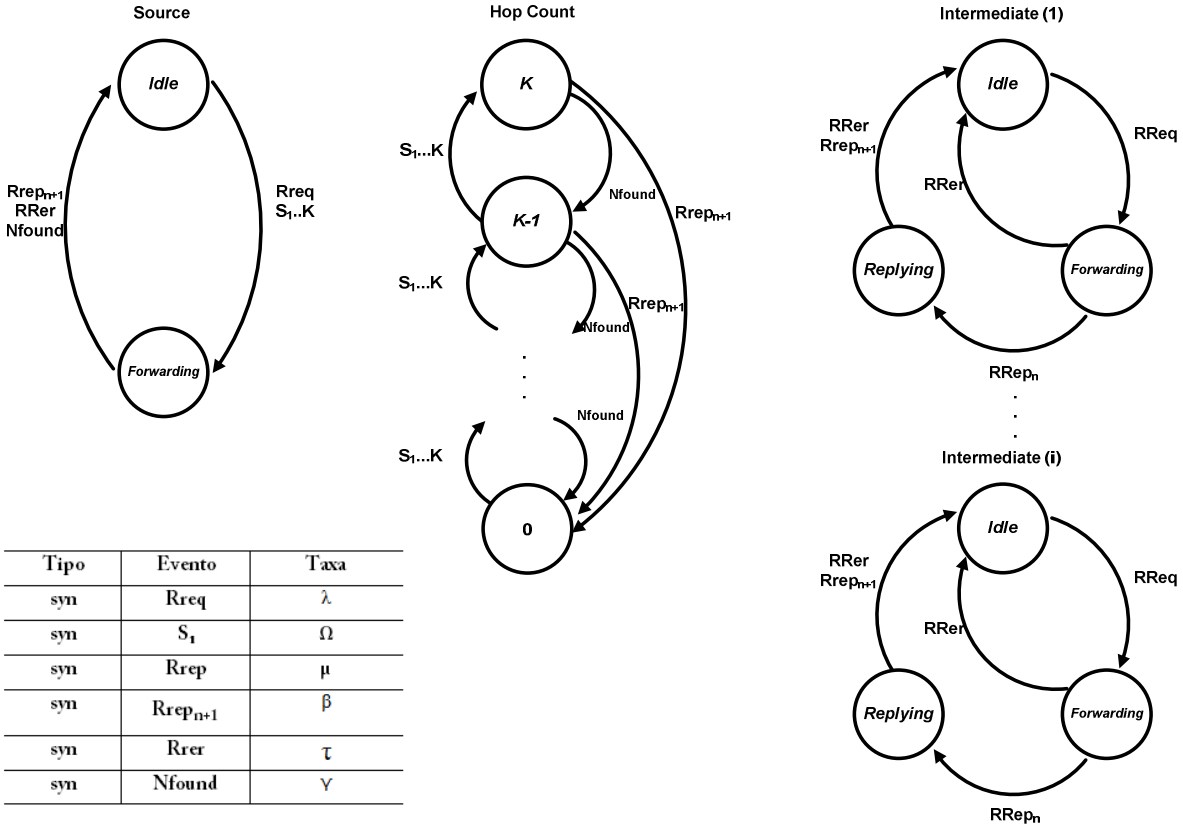
Para o processo de descoberta de rotas, usou-se a mesma definição de Johnson *et al.* em [Joh96a]: quando um nodo fonte deseja enviar um pacote para um destino conhecido, e este não tem em seu *cache* de rotas o caminho mais curto e otimizado (com o menor número de saltos) para o destino, este então inicia o processo de descoberta de rota, chamado de *rreq* nos modelos deste trabalho, sendo este um evento sincronizante.

Este processo se dá por inundação da rede (*flooding*), ou seja, todos os nodos que estão ao alcance do sinal do nodo fonte recebem a requisição de rota, e estes nodos intermediários então fazem a verificação se algum deles é o destino que o nodo fonte está procurando ou se tem uma rota conhecida para este destino entre seus nodos vizinhos, e o processo continua da mesma forma, por inundação, até que o destino seja atingido.

Nesse processo os nodos vão armazenando o caminho que percorrem até encontrar o nodo destino, e quando este é atingido, gera uma resposta de rota para determinar o caminho que foi percorrido para atingi-lo e assim, enviando esta resposta de rota (sendo um evento sincronizante chamado de *rrep* nos modelos gerados), faz com o que nodo fonte conheça a rota a ser seguida para o envio das mensagens.

Na primeira modelagem gerada, foi gerado um autômato fonte (Source), que possuía um evento sincronizante *Rreq* com uma taxa constante λ, que representa a requisição de rotas, e o evento sincronizante S_n que realizava a contagem de saltos no autômato *Hop Count*, que é o autômato responsável pela contagem de saltos dentro da rede e que controlava a chegada ao nodo destino, onde também era disparado o evento sincronizante *Nfound*, onde, através de uma taxa γ, que representava uma ocorrência de erro de transmissão, o autômato *Source* voltava ao estado *Idle*, para reiniciar a transmissão.

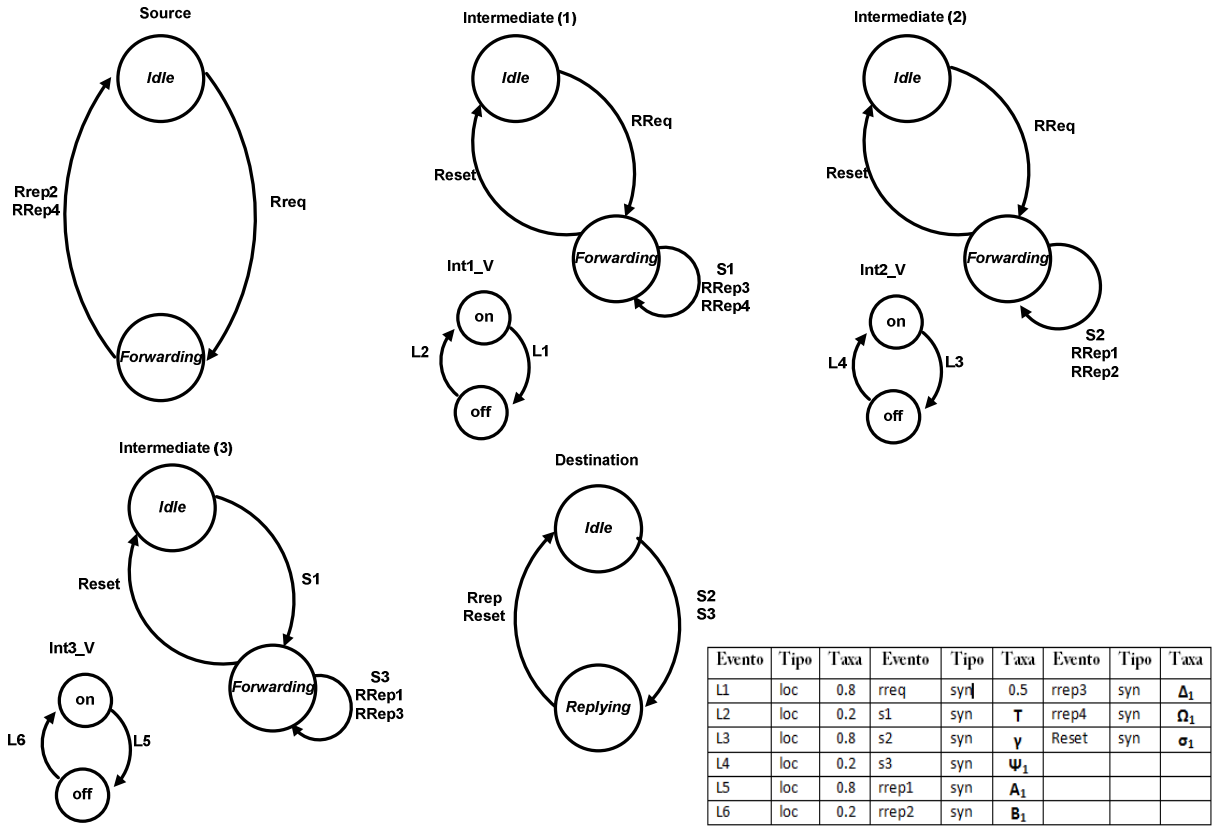
Neste modelo também existiam n autômatos chamados de *Intermediate(n)*, que representavam os nodos intermediários da rede, e que recebiam os eventos sincronizantes *Rreq*, para a requisição de rotas, *Rrep* para a resposta de rota e o evento *Rrer*, com uma taxa τ, que representava a mensagem de erro na transmissão, gerando o processo de manutenção de rota, e que era sincronizado com o autômato *ASource*, para que assim, o processo de descoberta de rotas fosse iniciado novamente. Este modelo foi gerado com o intuito de primar pela contagem dos saltos predeterminados, para modelar o comportamento da rede em relação ao desempenho do tamanho da mesma, como pode ser visto na Figura 5.1.



Tipo	Evento	Taxa
syn	Rreq	λ
syn	S_1	Ω
syn	Rrep	μ
syn	$Rrep_{n+1}$	β
syn	Rrer	τ
syn	Nfound	γ

Figura 5.1: Primeiro modelo SAN gerado

Neste primeiro modelo, as características de descoberta e manutenção de rotas já foram inseridas para atuarem em conjunto (através dos eventos *Rreq* e *Rrer*), sendo que esta foi uma das situações complicadoras do modelo, pois se baseava em taxas admitidas para que os nodos se tornassem indisponíveis na rede, refletindo diretamente na contagem de saltos. Outra situação deu-se pelo fato que o modelo levava em consideração o número de saltos, e o nodo destino era determinado pelo estado *K* do autômato *HopCount*, já que os autômatos que representavam os nodos presentes na rede não eram diferenciados, exceto pelo nodo fonte. Com isso, gerou-se um problema de determinação de desempenho do menor caminho entre nodo fonte e nodo destino. Assim, uma vez que este modelo não contemplava a realidade do protocolo, foi gerado um novo modelo, que é mostrado na Figura 5.2.



Evento	Tipo	Taxa	Evento	Tipo	Taxa	Evento	Tipo	Taxa
L1	loc	0.8	rreq	syn	0.5	rrep3	syn	Δ_1
L2	loc	0.2	s1	syn	T	rrep4	syn	Ω_1
L3	loc	0.8	s2	syn	γ	Reset	syn	σ_1
L4	loc	0.2	s3	syn	Ψ_1			
L5	loc	0.8	rrep1	syn	A_1			
L6	loc	0.2	rrep2	syn	B_1			

Figura 5.2: Segundo modelo SAN gerado

Neste modelo, foi possível determinar o processo de descoberta de rota, através do evento sincronizante *Rreq*, levando em consideração uma topologia em um determinado momento da rede, onde se buscava atingir a rota entre fonte e destino através da menor e melhor rota entre os nodos intermediários.

Com esta modelagem, as rotas entre os nodos da rede puderam ser atingidas, porém, muitos estados que não eram necessários na modelagem estavam sendo atingidos, mesmo com uma função de atingibilidade bem definida, uma vez que eram encontradas várias situações que não eram passíveis de acontecer, como por exemplo, o autômato *Destination*, representando o nodo destino, estar respondendo sem receber requisições de rota.

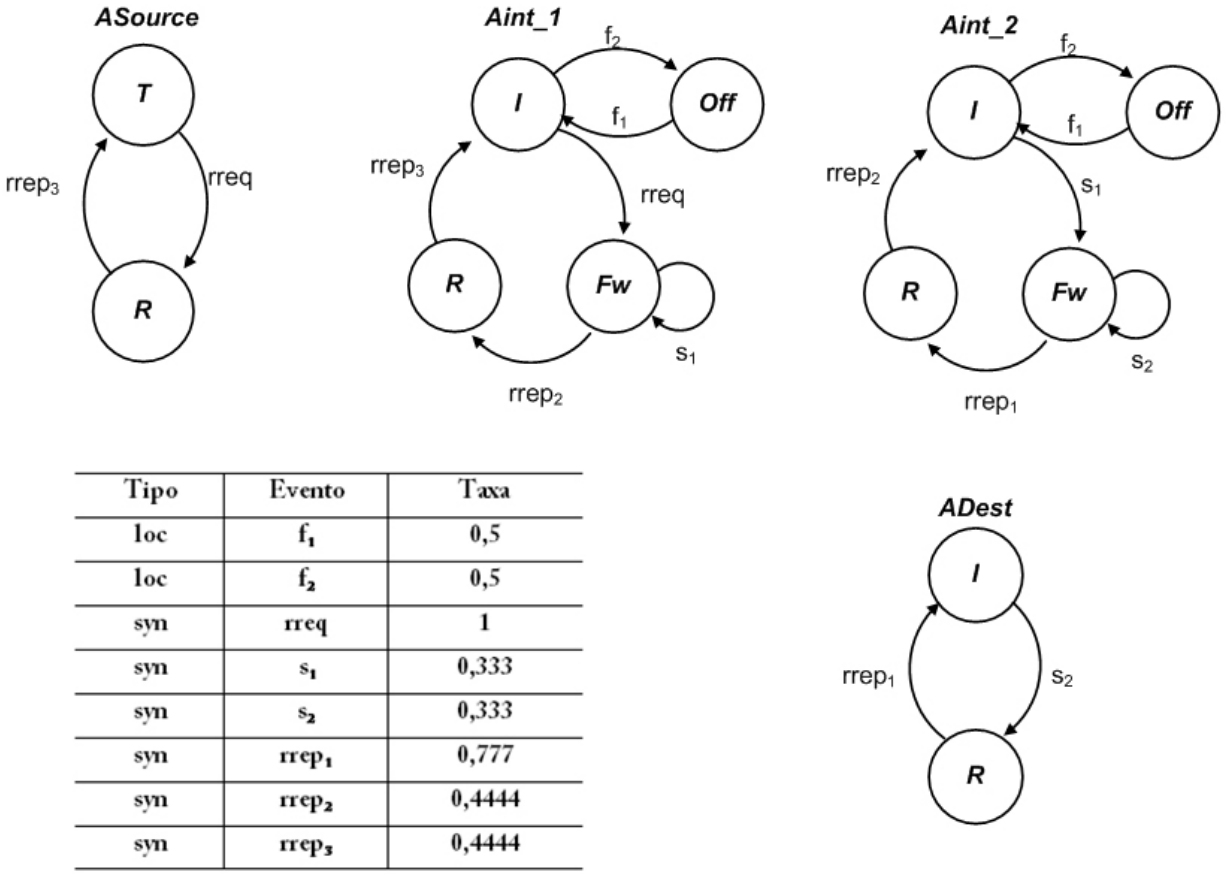
Outra situação que fez com que este modelo não contemplasse a realidade que se buscava atingir deve-se ao fato de muitas transições de estados ocorrendo em paralelo, como por exemplo, os autômatos que representavam os estados de disponibilidade dos nodos na rede não eram integrados aos autômatos representativos dos nodos, aumentando consideravelmente cada ciclo de descoberta de rotas para cada autômato referente à disponibilidade dos nodos, aumentando assim o tempo de iteração dos modelos para se extrair resultados.

Para resolver o problema de estados atingíveis incoerentes com a realidade que se estava buscando avaliar, foi feito uma análise deste segundo modelo, no qual se constatou que, o fator que determinava o grande número de estados atingíveis era resultante de que tanto as requisições de rota quanto as respostas de rota tinham suas transições disparadas junto ao estado *Forwarding*.

Tal estado estava sempre recebendo algum evento, fosse este disparado por um evento para descoberta ou resposta de rota, fazendo com que o mesmo estivesse sempre em uso, e com isso, dificultando a determinação de estados atingíveis.

Na análise do modelo, outro ponto complicador devia-se ao fato de que a probabilidade dos nodos intermediários estarem disponíveis na rede ou não era analisada levando-se em consideração transições em autômatos independentes com eventos sincronizantes que ativavam - ou não - os autômatos representativos destes nodos intermediários.

Diante do problema acima relatado nessa modelagem, foi criado um novo modelo, que agora trazia em sua configuração uma nova representação dos autômatos responsáveis pelos nodos intermediários, como pode ser observado na Figura 5.3.



Tipo	Evento	Taxa
loc	f_1	0,5
loc	f_2	0,5
syn	rreq	1
syn	s_1	0,333
syn	s_2	0,333
syn	$rrep_1$	0,777
syn	$rrep_2$	0,4444
syn	$rrep_3$	0,4444

Figura 5.3: Terceiro modelo SAN gerado

Agora, os autômatos que representam os nodos intermediários possuem o estado *I* (*Idle*), o estado *Fw* (*Forwarding*) onde ocorrem as transições de requisições de rota através dos eventos *rreq* e s_n, e o estado *R* (*replying*) para as transições de resposta de rota, utilizando os eventos $rrep_n$. Dessa forma, foi possível determinar – e definir – os estados atingíveis para a função de atingibilidade no processo de descoberta de rotas do protocolo DSR, já que antes estes eventos ocorriam em um mesmo estado, dificultando esse processo de avaliação. Outra modificação no modelo é que agora a disponibilidade ou não do autômato na rede, necessária para realizar o processo de manutenção de rotas é um estado do autômato com eventos locais f_n, onde o estado *Off* representa a indisponibilidade deste nodo no momento da transmissão dos pacotes.

Tendo agora um modelo que realiza satisfatoriamente os processos de descoberta, manutenção e resposta de rota, fez-se necessário elencar alguns pontos para avaliar o protocolo DSR utilizando esta modelagem. Para isso foram identificados nos trabalhos relacionados os principais índices de avaliação utilizados pelos autores em seus estudos de forma a gerar uma base

compreensível de informações e que tivessem um nível de abstração passível de ser usado com o formalismo SAN.

Em [Bro98], os autores utilizaram como métrica principal a proporção de pacotes entregues (vazão), que se dá pela relação entre o número de pacotes gerados pelos nodos fontes e o número de pacotes recebidos pelo nodo destino ao final da transmissão. Outra métrica importante encontrada em [Bro98] diz respeito ao *overhead* de roteamento, que é o numero total de pacotes de roteamento transmitidos durante a avaliação realizada.

Em [Bou04], o autor buscou avaliar os protocolos de roteamento a partir de métricas como a vazão, já elencada anteriormente; o atraso médio fim-a-fim dos pacotes de dados, que analisa todos os possíveis atrasos que podem ocorrer na rede, e também o *overhead* de roteamento. Em suas simulações, o autor utilizou de várias configurações de rede, com diferentes números de nodos presentes na rede.

Das *et al.* em [Das98] e [Das00] buscou avaliar em seus trabalhos, dentro de diversas configurações de rede, a fração de pacotes entregues, o atraso fim-a-fim e a carga de roteamento, que é dada pelo número de bytes dos pacotes de roteamento transmitidos pelos bytes de pacotes de dados transmitidos, esta métrica em especial, segundo os autores, serve para dar a idéia da banda de rede consumida por pacotes de roteamento em detrimento aos pacotes de dados úteis.

Layuan *et al.* em [Lay07], avalia os protocolos do roteamento levando em conta a dinâmica da topologia da rede, diferentemente dos trabalhos anteriormente citados que traziam tamanhos de rede, banda e velocidades constantes. Neste trabalho, os autores utilizaram como métricas índices de *QoS* como atraso médio, vazão média fim-a-fim e proporção de perda, além da carga de roteamento, índice esse utilizado em outros trabalhos estudados.

Como nos outros trabalhos já citados, Perkins *et al.* [Per01] utilizou métricas que avaliavam a proporção de entrega de pacotes, o atraso médio fim-a-fim de pacotes de dados, a carga de roteamento e também a carga de acesso ao meio – MAC – que avalia o número de pacotes de roteamento, endereçamento e controle transmitidos pela camada MAC para cada pacote entregue.

O impacto dos padrões de tráfego nos protocolos de roteamento foi avaliado por Pucha *et al.* em [Puc07], e para isso, foram utilizadas as métricas de *overhead* de roteamento, a proporção de pacotes entregues e o atraso médio dos protocolos.

Vários outros trabalhos abordam outras métricas para diferentes fins de avaliação, bem como muitos outros apenas reapresentam os resultados obtidos nos trabalhos que são citados aqui

para afirmar a eficiência dos protocolos de roteamento na transmissão de informações dentro de uma rede *wireless Ad hoc*.

Como o intuito deste trabalho diz respeito diretamente à modelagem e análise do protocolo de roteamento DSR e do seu processo de descoberta de rota utilizando o formalismo SAN, e com isso poder extrair resultados que contemplassem informações para métricas de avaliação, optou-se então por manter a mesma linha dos autores citados utilizando alguns dos índices para possíveis comparações de resultados utilizando agora uma metodologia analítica para obtenção dos índices de desempenho, avaliando assim vazão tanto para os pacotes de dados quanto para vazão associada a tempo de pausa, onde é possível determinar em um momento estático da rede a capacidade de transmissão entre os nodos, além do *workload* dos nodos e o processo de descoberta de rota (utilização das rotas) do protocolo.

5.1 SAN para redes simples

Na construção dos modelos, como dito anteriormente, foram observadas as condições assumidas para que se tenha um nível de abstração próximo de um ambiente real e que também se assemelhe aos ambientes criados nos trabalhos relacionados.

Vários modelos com vários caminhos a serem percorridos do nodo fonte para o nodo destino foram gerados durante o processo de construção do ambiente. Com o intuito de avaliar o processo de descoberta de rotas do protocolo DSR, algumas características do ambiente são aqui apresentadas:

• Para o modelo simples, têm-se um nodo fonte, sete nodos intermediários e um nodo destino;

• O nodo fonte envia sua requisição de rotas com uma taxa de ocorrência de 500ms (milissegundos) entre uma requisição e outra;

• Foi considerado para o ambiente uma área onde se assumiu que todos os equipamentos presentes na rede têm a mesma capacidade de transmissão e o mesmo alcance de sinal, tendo uma configuração de múltiplos saltos, havendo a necessidade direta dos nodos intermediários para realizar o repasse das informações;

• Não existem obstáculos entre os nodos que possam interferir na transmissão e no sinal;

• A avaliação leva em consideração um determinado momento da rede, onde os nodos estão em posições aleatórias dentro da malha de rede, e para a avaliação de possíveis alterações de topologia, é inserido tempo de pausa nos modelos, como dito anteriormente.

A Figura 5.4 apresenta a topologia gerada para o modelo simples de redes criado.

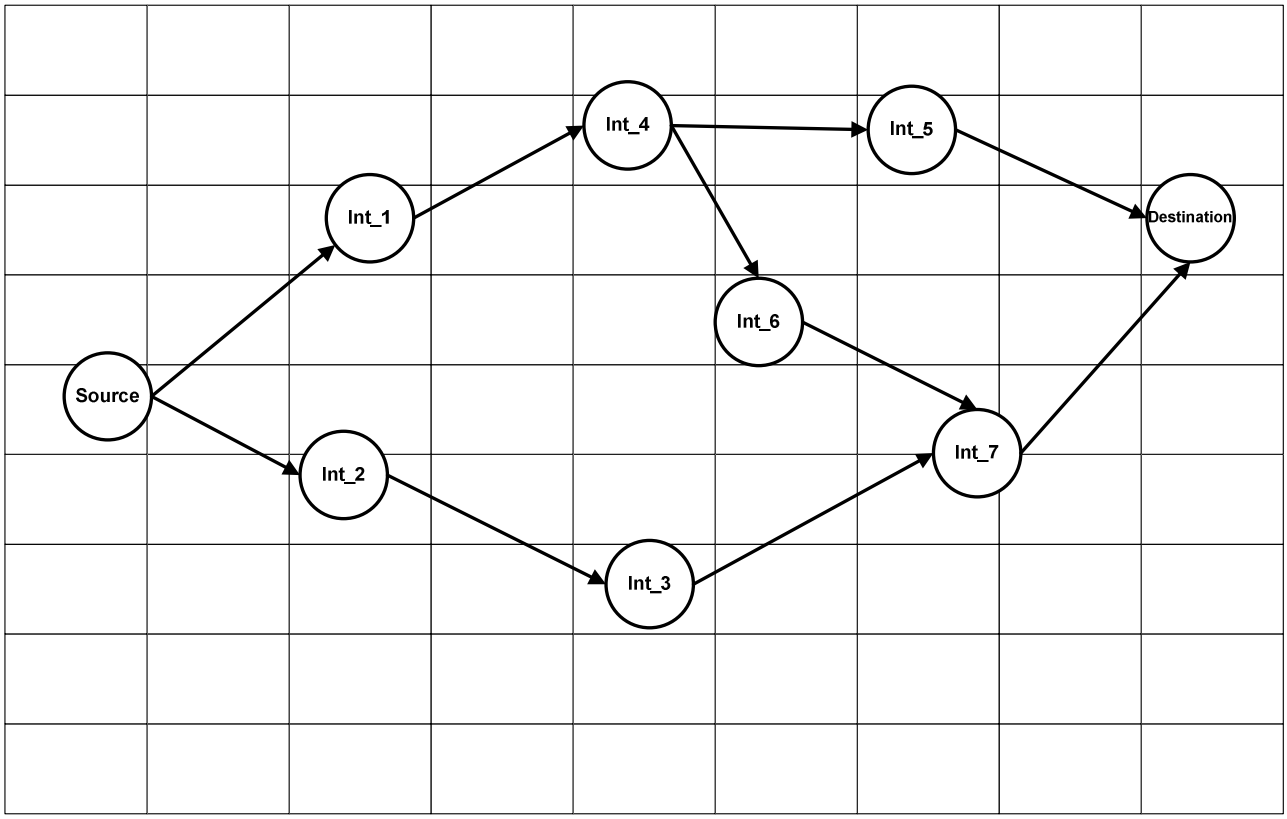


Figura 5.4: Topologia de rede para o modelo SAN simples

A partir da topologia de rede apresentada na figura 5.4, gerou-se então o modelo SAN correspondente, que traz três tipos de autômatos: o autômato *ASource* representa o nodo fonte, e possui os estados *T* (*transmiting*) e o estado R (*receiving*).

A transição do estado *T* para o estado *R* se dá pelo disparo do evento *rreq* (*route request*) que sincroniza com os nodos intermediários que representam os nodos da rede ao alcance do nodo fonte, e assim, se inicia o processo de descoberta de rota, sendo que este evento também faz a transmissão dos pacotes de dados (quando estes são adicionados ao cabeçalho do protocolo), e o processo de resposta de rota é dado pelo disparo do evento sincronizante $rrep_n$, que pode ocorrer mais de uma vez, dado o fato que mais de uma rota seja passível de responder à requisição do processo de descoberta de rotas. A Figura 5.5 apresenta o autômato referente ao nodo fonte da rede modelada.

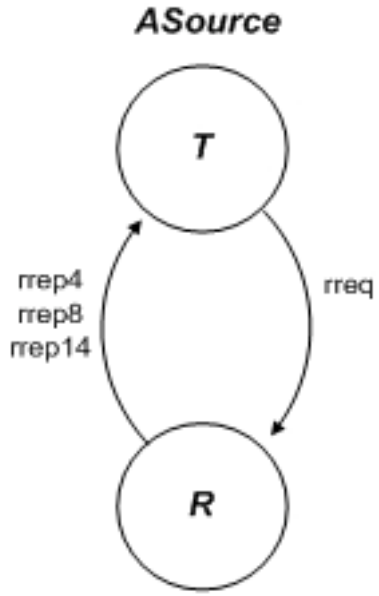


Figura 5.5: Autômato referente ao nodo fonte

Os autômatos que representam os nodos intermediários recebem a denominação de *AInt_n* ou *AInt_mn*, que podem variar de acordo com o número de nodos presentes na rede, por exemplo *AInt_1*, ou então *AInt_23*, no caso de existirem duas redes, onde *m* representa a identificação da rede e *n* o número do nodo na rede correspondente.

Os estados presentes nos autômatos intermediários são os seguintes: *I* (*Idle*), que é o estado que o autômato se encontra quando está ocioso aguardando para transmitir ou quando recebe a resposta de rota e volta a ficar ocioso; *Off* (*Off*) que representa a disponibilidade ou não do nodo na rede; *Fw* (*Forwarding*), que representa o estado do nodo transmitindo as solicitações de rota na rede ou os pacotes de dados; e por fim o estado *R* (*Replying*), que realiza a tarefa de retornar as transições representativas das mensagens de resposta de rota da rede, bem como de confirmação de pacote entregue e liberação da rede.

Todos os eventos que ocorrem dentro destes autômatos são sincronizantes, com exceção dos eventos disparados para o estado *Off*, que são locais e representam, como dito anteriormente, a disponibilidade ou não daquele autômato em particular na rede. A Figura 5.6 traz o autômato correspondente ao nodo intermediário da rede.

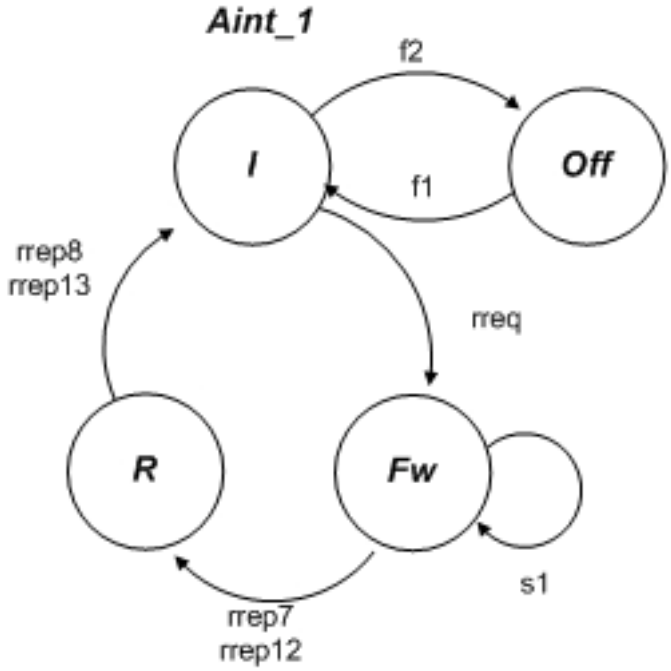


Figura 5.6: Autômato referente à um nodo intermediário da rede

Os nodos fonte e destino, e mais a quantidade de nodos intermediários na rede é o que determina a quantidade de autômatos presentes na SAN correspondente.

O autômato referente ao nodo destino é denominado *ADest*, tendo dois estados: *I* (*Idle*), que é o estado em que o autômato encontra-se aguardando uma requisição de rota ou a entrega de um pacote; e o estado *R* (*Replying*), onde o autômato recebeu a requisição de rota e estará gerando uma resposta de rota para o nodo fonte, ou então enviando a confirmação do pacote entregue, confirmando assim o caminho a ser seguido de volta ao nodo fonte.

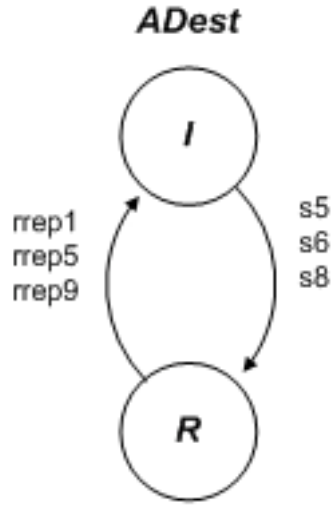


Figura 5.7: Autômato referente ao nodo destino

A Figura 5.7 apresenta o autômato representativo ao nodo destino, e suas transições de estados.

As transições entre os estados são realizadas pelos seguintes eventos sincronizantes:

• *rreq*: evento que inicia a transmissão de uma requisição de rota através da rede e também dos pacotes de dados;

• s_n: evento gerado pelo evento rreq que gera o repasse da informação de requisição de rota para os nodos vizinhos;

• *rrep*: evento que representa a resposta de rota sendo enviada do nodo destino para o nodo fonte, passando pelos nodos intermediários e confirmando a rota a ser seguida.

Salientando ainda a existência dos eventos f_n que representam os eventos locais que correspondem à ocorrência da disponibilidade dos nodos intermediários na rede.

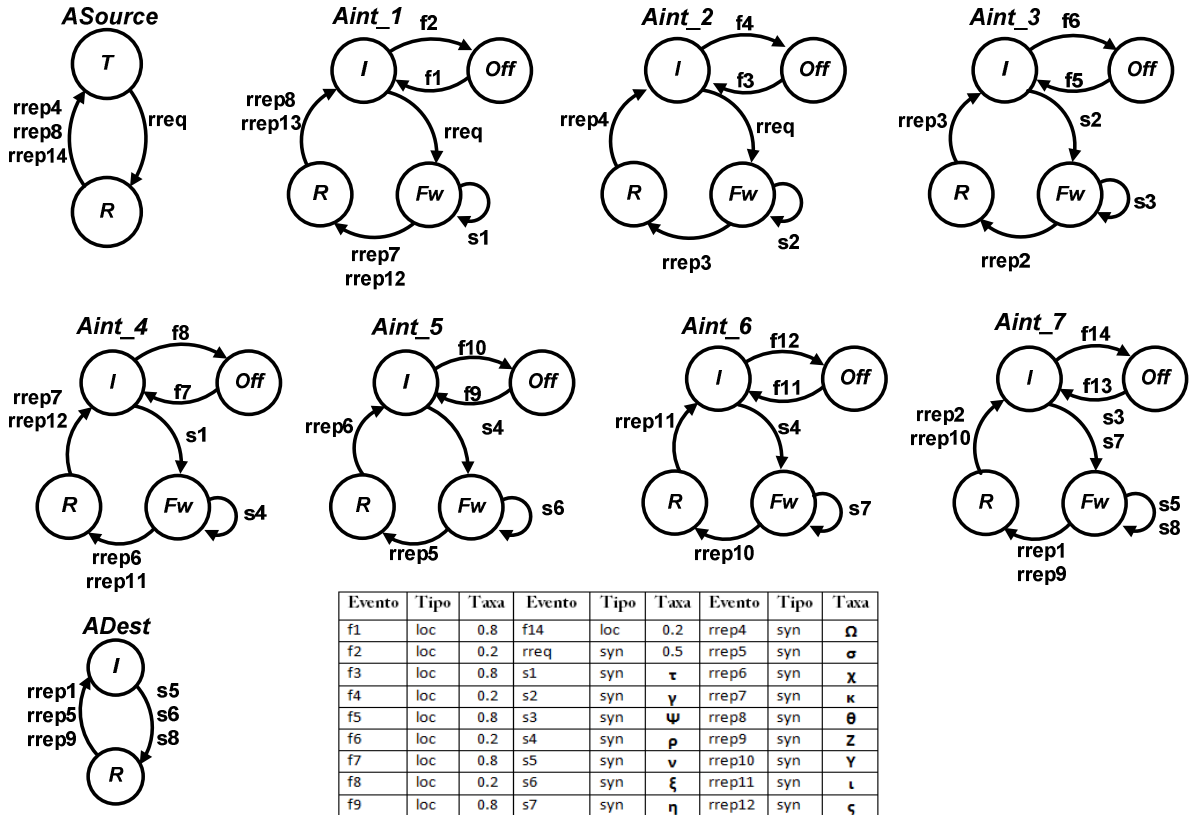
Para melhor ilustrar os eventos que ocorrem dentro dos modelos, serão apresentadas as sintaxes utilizadas na ferramenta PEPS, com os devidos comentários para cada evento inseridos após o uso de "//".

```
identifiers
f1       = (st AInt_1 != Off) * 0.8; // taxa admitida para que o
nodo esteja disponível na rede no momento da transmissão
f2       = (st AInt_1 == Off) * 0.2; // taxa admitida para que o
nodo NÃO esteja disponível na rede no momento da transmissão,
fazendo com que o processo de manutenção de rota ocorra
```

```
ocur_rate = 500; // ocorrência da taxa de requisição de rotas em
milissegundos
lambda = 0.5;    // ocorrência da taxa de requisição de rotas em
segundos sendo (1 segundo = 1000 milissegundos)
tau     = (st ASource == T) && ((st AInt_1 == Fw) && (st AInt_1 !=
Off)) && ((st AInt_4 == I) && (AInt_4 != Off) && (AInt_4 != R)) &&
((AInt_3 != Fw) || (AInt_3 != R)) && (ADest == I) && ((AInt_2 !=
Fw) && (AInt_5 != Fw) && (AInt_6 != Fw) && (AInt_7 != Fw)); //
nesta taxa em específico representa a mensagem sendo enviada do
nodo fonte para o nodo 1, que necessita que o nodo 4 esteja
disponível e não esteja transmitindo para recebê-la, bem como os
outros nodos na rede não estejam transmitindo
alfa    = (st ADest == R) && ((st AInt_7 == Fw) && (st AInt_7 !=
Off) && (st AInt_7 != R)) && (st ASource == R) && ((st AInt_6 !=
R) && (st AInt_5 != R) && (st AInt_4 != R) && (st AInt_2 != R) &&
(st AInt_3 != R) && (st AInt_1 != R)); // nesta taxa, que
representa a resposta de rota, o nodo destino está retornando a
mensagem para o nodo que a enviou, no caso o nodo 7, e este
considera que os outros nodos não estejam enviando a mesma
mensagem para o nodo fonte

events

loc on1 f1;
loc off1 f2;

syn rreq   lambda;
syn s1     tau;
syn rrep1  alfa;
```

Estes autômatos são usados em todos os modelos gerados, o que os diferencia é a complexidade de suas taxas funcionais, a quantidade de eventos que cada autômato terá, e por consequência do aumento de autômatos, a quantidade de estados atingíveis.

A Figura 5.8 apresenta a SAN correspondente à topologia criada para o modelo simples.



Evento	Tipo	Taxa	Evento	Tipo	Taxa	Evento	Tipo	Taxa
f1	loc	0.8	f14	loc	0.2	rrep4	syn	Ω
f2	loc	0.2	rreq	syn	0.5	rrep5	syn	σ
f3	loc	0.8	s1	syn	τ	rrep6	syn	χ
f4	loc	0.2	s2	syn	γ	rrep7	syn	κ
f5	loc	0.8	s3	syn	ψ	rrep8	syn	θ
f6	loc	0.2	s4	syn	ρ	rrep9	syn	Ζ
f7	loc	0.8	s5	syn	ν	rrep10	syn	Υ
f8	loc	0.2	s6	syn	ξ	rrep11	syn	ι
f9	loc	0.8	s7	syn	η	rrep12	syn	ς
f10	loc	0.2	s8	syn	ε	rrep13	syn	Ǫ
f11	loc	0.8	rrep1	syn	α			
f12	loc	0.2	rrep2	syn	β			
f13	loc	0.8	rrep3	syn	δ			

Figura 5.8: SAN correspondente à topologia do modelo simples

Neste modelo SAN, a quantidade de estados globais gerados é de 65.536 estados e 1.264 estados atingíveis, se levados em consideração as ausências de nodos na rede e a necessidade de manutenção de rotas, sendo estes números relativamente pequenos se comparados aos outros modelos gerados neste trabalho, e também passível de solução utilizando outras alternativas de modelagem analítica, como Cadeias de Markov, porém, é importante ressaltar que o uso de SAN para estas modelagens facilita modelar os componentes da rede, haja vista a dificuldade em gerar estados globais para grandes cadeias de Markov.

Outro fator determinante na escolha de SAN para esta modelagem deve-se ao fato que com o aumento da complexidade dos modelos, de seus espaços de estados e suas transições, será possível identificar o crescimento exponencial de estados presentes em cada modelo.

A função de atingibilidade para se obter o processo de descoberta de rotas é dada pela seguinte sintaxe:

```
    partial reachability = ((st ASource == T) && (st ADest == I)
&& ((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 == Fw))
```

```
&& ((st AInt_2 != R) && (st AInt_2 != Fw) && (st AInt_2 != Off))
&& ((st AInt_3 != R) && (st AInt_3 != Fw) && (st AInt_3 != Off))
&& ((st AInt_6 != R) && (st AInt_6 != Fw) && (st AInt_6 != Off))
&& ((st AInt_7 != R) && (st AInt_7 != Fw) && (st AInt_7 != Off)))
||
((st ASource == T) && (st ADest == I) && ((st AInt_2 == Fw) && (st
AInt_3 == Fw) && (st AInt_7 == Fw)) && ((st AInt_1 != R) && (st
AInt_1 != Fw) && (st AInt_1 != Off)) && ((st AInt_4 != R) && (st
AInt_4 != Fw) && (st AInt_4 != Off)) && ((st AInt_5 != R) && (st
AInt_5 != Fw) && (st AInt_5 != Off)) && ((st AInt_6 != R) && (st
AInt_6 != Fw) && (st AInt_6 != Off)))
||
((st ASource == T) && (st ADest == I) && ((st AInt_1 == Fw) && (st
AInt_4 == Fw) && (st AInt_6 == Fw) && (st AInt_7 == Fw)) && ((st
AInt_2 != R) && (st AInt_2 != Fw) && (st AInt_2 != Off)) && ((st
AInt_3 != R) && (st AInt_3 != Fw) && (st AInt_3 != Off)) && ((st
AInt_5 != R) && (st AInt_5 != Fw) && (st AInt_5 != Off)));
```

Esta sintaxe restringe a atingibilidade às três rotas possíveis do modelo, quando do uso da função *partial*, que restringe o subconjunto de estados representativos às rotas do modelo dentro do conjunto de estados atingíveis, sendo que esta situação é obtida quando se têm o nodo *ASource* no estado *T*, disparando o evento rreq, que sincroniza com os autômatos *AInt_1* e *AInt_2*, que alteram seus estados para *Fw*, e vão disparando os eventos s_n, até que estes atinjam o autômato *ADest*, fazendo com que o processo de descoberta de rota se complete.

Outro fator determinante para que o processo de descoberta de rotas ocorra com o uso desta sintaxe dá-se pelo fato de que em cada rota definida, como por exemplo:

```
((st ASource == T) && (st ADest == I) && ((st AInt_1 == Fw) && (st
AInt_4 == Fw) && (st AInt_5 == Fw)) && ((st AInt_2 != R) && (st
AInt_2 != Fw) && (st AInt_2 != Off)) && ((st AInt_3 != R) && (st
AInt_3 != Fw) && (st AInt_3 != Off)) && ((st AInt_6 != R) && (st
AInt_6 != Fw) && (st AInt_6 != Off)) && ((st AInt_7 != R) && (st
AInt_7 != Fw) && (st AInt_7 != Off)))
```

O trecho

```
((st ASource == T) && (st ADest == I) && ((st AInt_1 == Fw) &&
(st AInt_4 == Fw) && (st AInt_5 == Fw)))
```

é onde a rota é definida, e o restante do código limita a atividade de transmissão para os outros nodos da rede, fazendo com que a mensagem seja transmitida ou por uma rota ou por outra (através do uso do conector ‘||’, presente na primeira sintaxe apresentada), sem que haja interferência entre os nodos.

Como o protocolo DSR usa a técnica de *piggybacking* para que possa realizar o processo de resposta de rota, retornando a mensagem pelos mesmos nodos por onde a mensagem passou, identificados pelos id's contidos nos cabeçalhos de dados, considerando que todos os nodos de cada rota estejam disponíveis no momento do processo de descoberta de rotas, a mensagem retorna pela rota por onde esta veio, sendo assumido que os nodos estejam sempre disponíveis e ativos.

Para realizar a modelagem do processo de descoberta de rota considerando a manutenção de rotas, onde, através de uma probabilidade de ocorrência estabelecida junto aos eventos locais f_n de cada autômato intermediário, o nodo pode ou não estar ativo na rede durante a transmissão das mensagens, a seguinte sintaxe foi utilizada:

```
partial reachability = ((st ASource == T) && (st ADest == I) &&
((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 == Fw)))
||
((st ASource == T) && (st ADest == I) && ((st AInt_2 == Fw) &&
(st AInt_3 == Fw) && (st AInt_7 == Fw)))
||
((st ASource == T) && (st ADest == I) && ((st AInt_1 == Fw) &&
(st AInt_4 == Fw) && (st AInt_6 == Fw) && (st AInt_7 == Fw)));
```

É possível perceber que agora somente as rotas estão sendo determinadas para se atingir o nodo fonte, sendo que os nodos intermediários podem estar ou não presentes na rede, fazendo com que seja possível realizar o processo de manutenção de rotas, uma vez que um nodo pode não estar disponível no momento do processo de descoberta de rotas. Os resultados serão apresentados no Capítulo 6, juntamente com os resultados de outros modelos gerados neste trabalho e a comparação com alguns resultados de outros formalismos.

5.2 SAN para redes múltiplas

Em redes múltiplas, o cenário gerado para a avaliação levou em consideração algumas das situações assumidas para modelos simples, como o fato que nodo fonte tem uma frequência constante de envio de requisições de rota; outro fator assumido é o de que não existem obstáculos entre os nodos que possam interferir na transmissão e no sinal; e por fim o de que a avaliação leva em consideração um determinado momento da rede, onde os nodos estão em posições aleatórias dentro da malha de rede, já que não estão sendo analisados padrões de mobilidade nos modelos presentes neste trabalho.

As diferenças para este modelo devem-se aos seguintes fatos:

• Existem dois nodos fontes transmitindo, cinco nodos intermediários para cada rede e dois nodos destino, onde a transmissão de uma rede não interfere na outra rede dentro da malha física;

• As redes são independentes em um primeiro momento, sem que haja qualquer tipo de interação entre elas, tal interação será modelada nas redes com escuta promíscua, que é o próximo modelo presente neste capítulo;

A Figura 5.9 apresenta a topologia gerada para redes múltiplas.

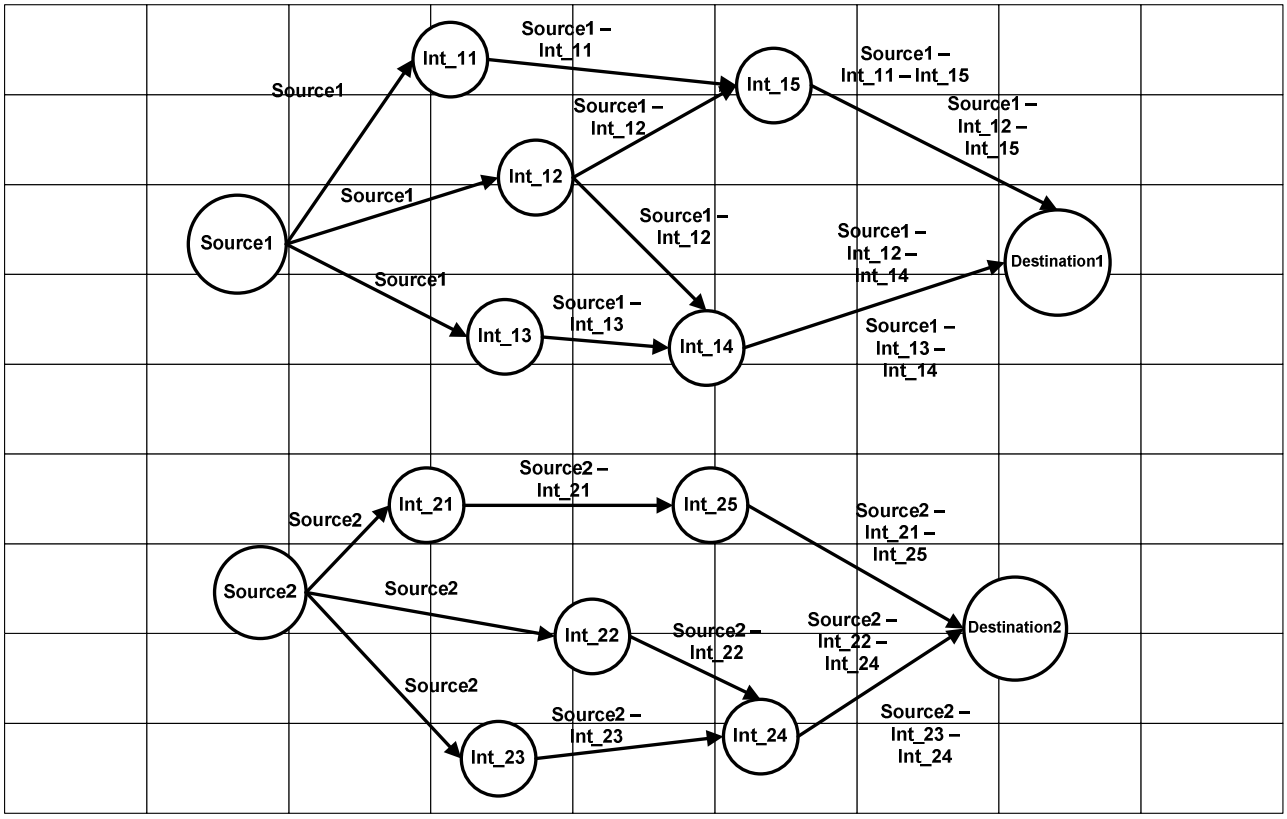


Figura 5.9: Topologia de rede para o modelo SAN de redes múltiplas

As estruturas dos autômatos presentes no modelo para redes múltiplas permanecem as mesmas dos modelos gerados para redes simples, havendo a necessidade de construir mais autômatos para representar os nodos que foram acrescentados na rede e assim, atender a todas as transições entre os autômatos.

Neste modelo SAN, a quantidade de estados atingíveis é muito maior que no modelo para redes simples, uma vez que são duas redes transmitindo informações em paralelo, e o número máximo de estados gerados é de 4.194.304 estados e 371.608 estados atingíveis (se levados em

consideração, como no modelo simples, a possibilidade dos nodos não estarem presentes na rede e o processo de manutenção de rota), enquanto que nos modelos SAN para redes simples é de 65.536 estados.

A função de atingibilidade para se obter o processo de descoberta de rotas direto aos caminhos atingíveis, descartando perdas e manutenção de rotas é dada pela seguinte sintaxe:

```
partial reachability = (((st ASource1 == T) && (st ADest1 == I) &&
((st AInt_11 == Fw) && (st AInt_15 == Fw)) && ((st AInt_12 != R)
&& (st AInt_12 != Fw) && (st AInt_12 != Off)) && ((st AInt_13 !=
R) && (st AInt_13 != Fw) && (st AInt_13 != Off)) && ((st AInt_14
!= R) && (st AInt_14 != Fw) && (st AInt_14 != Off)))
||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_14 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) &&
(st AInt_11 != Off)) && ((st AInt_13 != R) && (st AInt_13 != Fw)
&& (st AInt_13 != Off)) && ((st AInt_15 != R) && (st AInt_15 !=
Fw) && (st AInt_15 != Off)))
||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_15 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) &&
(st AInt_11 != Off)) && ((st AInt_13 != R) && (st AInt_13 != Fw)
&& (st AInt_13 != Off)) && ((st AInt_14 != R) && (st AInt_14 !=
Fw) && (st AInt_14 != Off)))
||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) &&
(st AInt_14 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) &&
(st AInt_11 != Off)) && ((st AInt_12 != R) && (st AInt_12 != Fw)
&& (st AInt_12 != Off)) && ((st AInt_15 != R) && (st AInt_15 !=
Fw) && (st AInt_15 != Off)))
&&
(((st ASource2 == T) && (st ADest2 == I) && ((st AInt_21 == Fw) &&
(st AInt_25 == Fw)) && ((st AInt_22 != R) && (st AInt_22 != Fw) &&
(st AInt_22 != Off)) && ((st AInt_23 != R) && (st AInt_23 != Fw)
&& (st AInt_23 != Off)) && ((st AInt_24 != R) && (st AInt_24 !=
Fw) && (st AInt_24 != Off)))
||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) &&
(st AInt_24 == Fw)) && ((st AInt_21 != R) && (st AInt_21 != Fw) &&
(st AInt_21 != Off)) && ((st AInt_23 != R) && (st AInt_23 != Fw)
&& (st AInt_23 != Off)) && ((st AInt_25 != R) && (st AInt_25 !=
Fw) && (st AInt_25 != Off)))
||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) &&
(st AInt_24 == Fw)) && ((st AInt_22 != R) && (st AInt_22 != Fw) &&
(st AInt_22 != Off)) && ((st AInt_21 != R) && (st AInt_21 != Fw)
&& (st AInt_21 != Off)) && ((st AInt_25 != R) && (st AInt_25 !=
Fw) && (st AInt_25 != Off))));
```

Os estados atingíveis uma vez que se aplicam a manutenção de rotas, e novos processos de descoberta de rotas são dados pela seguinte sintaxe:

```
partial reachability = (((st ASource1 == T) && (st ADest1 == I) &&
((st AInt_11 == Fw) && (st AInt_15 == Fw)) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw)&&
(st AInt_14 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_15 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) &&
(st AInt_14 == Fw))) && (((st ASource2 == T)
&&
(st ADest2 == I) && ((st AInt_21 == Fw) && (st AInt_25 == Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) &&
(st AInt_24 == Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) &&
(st AInt_24 == Fw)));
```

Assim é possível avaliar a rede como um todo, existindo a possibilidade de nodos não estarem presentes na rede durante o processo de descoberta de rotas e também durante a transmissão de pacotes.

5.3 SAN para *Promiscuous Listening*

Para este modelo, foi utilizada a topologia criada para redes múltiplas, entretanto, agora se aplicando a característica de escuta promíscua presente no protocolo DSR, onde é possível um nodo que tenha em seu *cache* um caminho conhecido para o destino que este “ouviu” a transmissão, pode solicitar para si a requisição de rota ou os pacotes que estão sendo transmitidos, e oferecer um caminho mais curto ou otimizado para o nodo destino.

A Figura 5.10 mostra a topologia gerada para o processo de escuta promíscua.

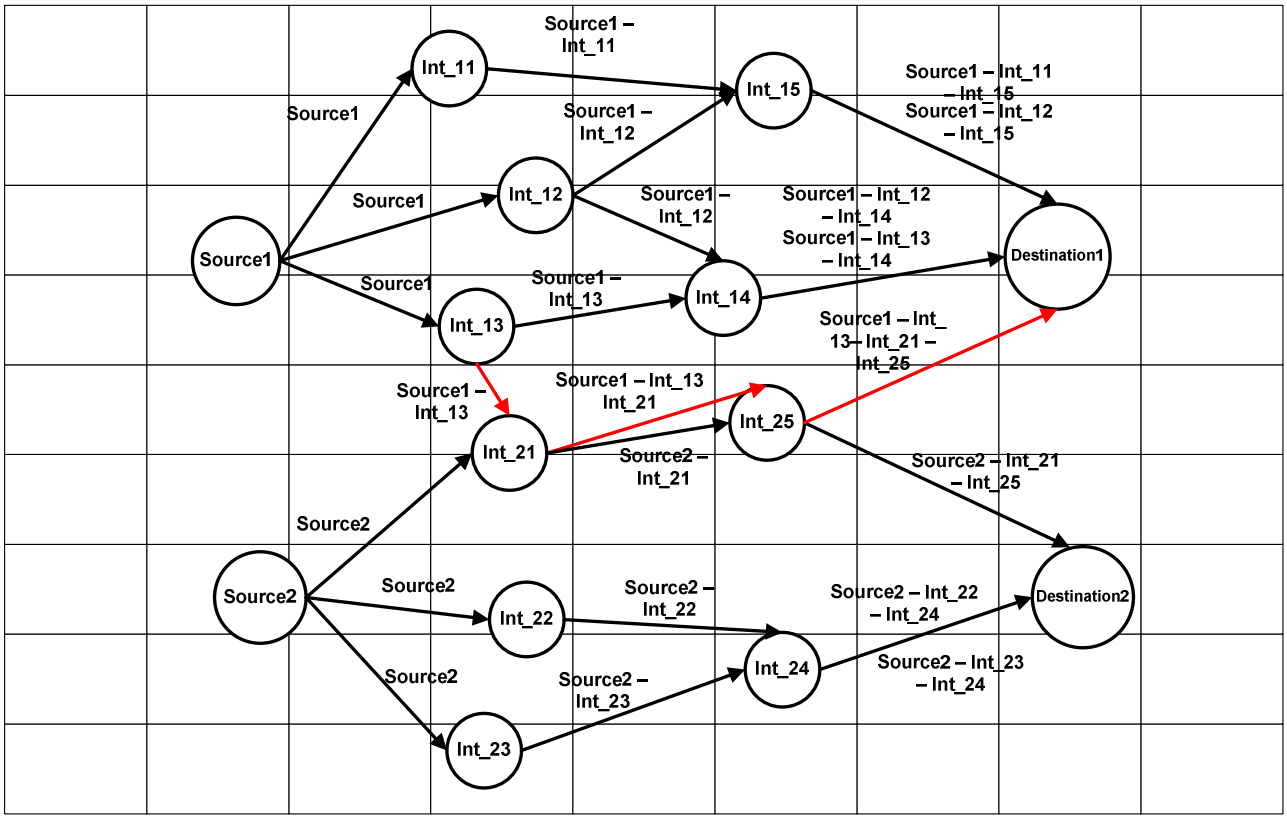


Figura 5.10: Topologia de rede para o modelo SAN do processo de escuta promíscua

É possível observar que, embora não esteja diretamente ligado à rede 1*n*, o nodo *Int_21* "escuta" a transmissão e indica um caminho por entre os nodos participantes da sua rede, e retransmite a informação para o nodo *Int_25*, fazendo com que a mensagem seja entregue ao nodo destino por uma rota baseada na segunda rede presente no ambiente de transmissão. A sintaxe para o processo de descoberta de rotas com estados atingíveis que descartam falhas na transmissão e manutenção de rotas é dada por:

```
partial reachability = (((st ASource1 == T) && (st ADest1 == I) &&
((st AInt_11 == Fw) && (st AInt_15 == Fw)) && ((st AInt_12 != R)
&& (st AInt_12 != Fw) && (st AInt_12 != Off)) && ((st AInt_13 !=
R) && (st AInt_13 != Fw) && (st AInt_13 != Off)) && ((st AInt_14
!= R) && (st AInt_14 != Fw) && (st AInt_14 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_14 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) &&
(st AInt_11 != Off)) && ((st AInt_13 != R) && (st AInt_13 != Fw)
&& (st AInt_13 != Off)) && ((st AInt_15 != R) && (st AInt_15 !=
Fw) && (st AInt_15 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_15 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) &&
(st AInt_11 != Off)) && ((st AInt_13 != R) && (st AInt_13 != Fw)
&& (st AInt_13 != Off)) && ((st AInt_14 != R) && (st AInt_14 !=
Fw) && (st AInt_14 != Off))) ||
```

```
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) &&
(st AInt_14 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) &&
(st AInt_11 != Off)) && ((st AInt_12 != R) && (st AInt_12 != Fw)
&& (st AInt_12 != Off)) && ((st AInt_15 != R) && (st AInt_15 !=
Fw) && (st AInt_15 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) &&
(st AInt_21 == Fw) && (st AInt_21 == Fw)) && ((st AInt_11 != R) &&
(st AInt_11 != Fw) && (st AInt_11 != Off)) && ((st AInt_12 != R)
&& (st AInt_12 != Fw) && (st AInt_12 != Off)) && ((st AInt_14 !=
R) && (st AInt_14 != Fw) && (st AInt_14 != Off)) && ((st AInt_15
=! R) && (st AInt_15 != Fw) && (st AInt_15 != Off)))
&&
(((st ASource2 == T) && (st ADest2 == I) && ((st AInt_21 == Fw) &&
(st AInt_25 == Fw)) && ((st AInt_22 != R) && (st AInt_22 != Fw) &&
(st AInt_22 != Off)) && ((st AInt_23 != R) && (st AInt_23 != Fw)
&& (st AInt_23 != Off)) && ((st AInt_24 != R) && (st AInt_24 !=
Fw) && (st AInt_24 != Off))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) &&
(st AInt_24 == Fw)) && ((st AInt_21 != R) && (st AInt_21 != Fw) &&
(st AInt_21 != Off)) && ((st AInt_23 != R) && (st AInt_23 != Fw)
&& (st AInt_23 != Off)) && ((st AInt_25 != R) && (st AInt_25 !=
Fw) && (st AInt_25 != Off))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) &&
(st AInt_24 == Fw)) && ((st AInt_22 != R) && (st AInt_22 != Fw) &&
(st AInt_22 != Off)) && ((st AInt_21 != R) && (st AInt_21 != Fw)
&& (st AInt_21 != Off)) && ((st AInt_25 != R) && (st AInt_25 !=
Fw) && (st AInt_25 != Off))));
```

Para que o processo de escuta promíscua ocorra, a seguinte expressão foi inserida na sintaxe da função de atingibilidade:

```
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) &&
(st AInt_21 == Fw) && (st AInt_21 == Fw)) && ((st AInt_11 != R) &&
(st AInt_11 != Fw) && (st AInt_11 != Off)) && ((st AInt_12 != R)
&& (st AInt_12 != Fw) && (st AInt_12 != Off)) && ((st AInt_14 !=
R) && (st AInt_14 != Fw) && (st AInt_14 != Off)) && ((st AInt_15
=! R) && (st AInt_15 != Fw) && (st AInt_15 != Off)))
```

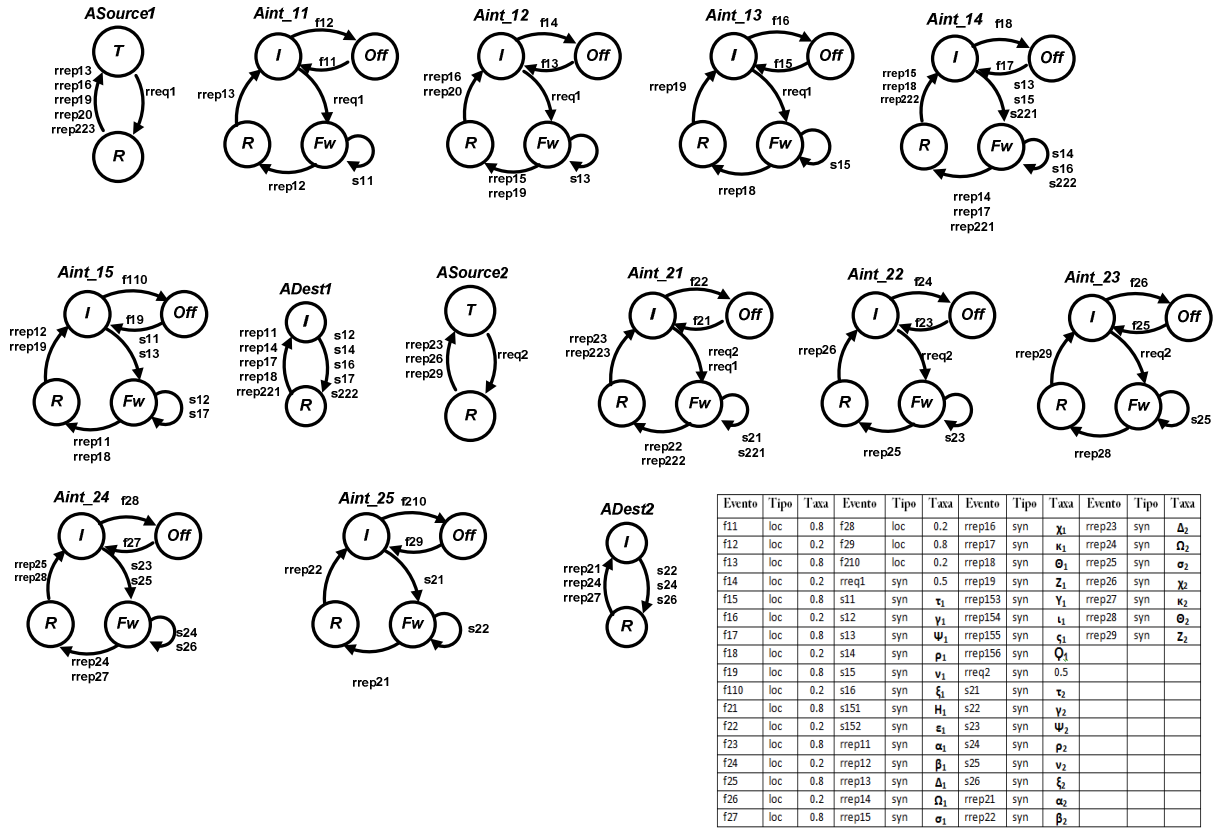
Utilizando-se dessa sintaxe, os estados atingíveis ficam restritos à oito estados possíveis, ou seja, as oito rotas possíveis dentro da rede, num escopo de 16.777.216 estados gerados. Se considerada a possibilidade de nodos ausentes na rede, e a necessidade de manutenção de rotas, o número de estados atingíveis é de 1.486.432 estados. Para que isso seja possível, utilizou-se da seguinte sintaxe:

```
partial reachability = (((st ASource1 == T) && (st ADest1 == I) &&
((st AInt_11 == Fw) && (st AInt_15 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_14 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
```

```
(st AInt_15 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) &&
(st AInt_14 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_21 == Fw) &&
(st AInt_14 == Fw))))
&&
(((st ASource2 == T) && (st ADest2 == I) && ((st AInt_21 == Fw) &&
(st AInt_25 == Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) &&
(st AInt_24 == Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) &&
(st AInt_24 == Fw))));
```

Agora, as transições entre os estados dos autômatos ocorrem de forma que os nodos possam realizar o processo de manutenção de rotas, caso um dos nodos presentes nas rotas admitidas para a rede não esteja ativo, devido ao disparo de um evento f_n que faça com que este nodo esteja indisponível para transmitir.

O modelo SAN gerado para este processo de escuta promíscua é dado na Figura 5.11.



Evento	Tipo	Taxa	Evento	Tipo	Taxa	Evento	Tipo	Taxa	Evento	Tipo	Taxa
f11	loc	0.8	f28	loc	0.2	rrep16	syn	χ_1	rrep23	syn	Δ_2
f12	loc	0.2	f29	loc	0.8	rrep17	syn	κ_1	rrep24	syn	Ω_2
f13	loc	0.8	f210	loc	0.2	rrep18	syn	Θ_1	rrep25	syn	σ_2
f14	loc	0.2	rreq1	syn	0.5	rrep19	syn	Z_1	rrep26	syn	χ_2
f15	loc	0.8	s11	syn	τ_1	rrep153	syn	Y_1	rrep27	syn	κ_2
f16	loc	0.2	s12	syn	γ_1	rrep154	syn	ι_1	rrep28	syn	Θ_2
f17	loc	0.8	s13	syn	Ψ_1	rrep155	syn	ς_1	rrep29	syn	Z_2
f18	loc	0.2	s14	syn	ρ_1	rrep156	syn	Q_1			
f19	loc	0.8	s15	syn	ν_1	rreq2	syn	0.5			
f110	loc	0.2	s16	syn	ξ_1	s21	syn	τ_2			
f21	loc	0.8	s151	syn	H_1	s22	syn	γ_2			
f22	loc	0.2	s152	syn	ε_1	s23	syn	Ψ_2			
f23	loc	0.8	rrep11	syn	α_1	s24	syn	ρ_2			
f24	loc	0.2	rrep12	syn	β_1	s25	syn	ν_2			
f25	loc	0.8	rrep13	syn	Δ_1	s26	syn	ξ_2			
f26	loc	0.2	rrep14	syn	Ω_1	rrep21	syn	α_2			
f27	loc	0.8	rrep15	syn	σ_1	rrep22	syn	β_2			

Figura 5.11: SAN correspondente à topologia do modelo de escuta promíscua

5.4 SAN para agregação de serviços

Baseado no modelo de escuta promíscua e semelhante à característica de *cluster-head* presente em alguns protocolos de estrutura não-uniforme para redes *Ad hoc*, foi gerado um modelo que utilizasse de agregação que desempenha este papel, atuando nas duas redes ao mesmo tempo, fazendo o repasse de informações em ambas as redes, atuando como se estivesse concentrando as propriedades de roteamento de dois nodos, e assim, participando das duas redes sem a necessidade de usar de escuta promíscua para realizar a transmissão, e assim, contribuir com uma forma diferente de transmissão que não se encontra presente na literatura ou em técnicas pertencentes ao protocolo DSR.

Foi então construída uma topologia onde um nodo (*AInt_1321*) agregasse a tarefa de transmissão representando o que antes seria realizado quando da presença de mais dois nodos (*AInt_13* e *AInt_21*) na rede.

A topologia para esta nova rede é apresentada na Figura 5.12.

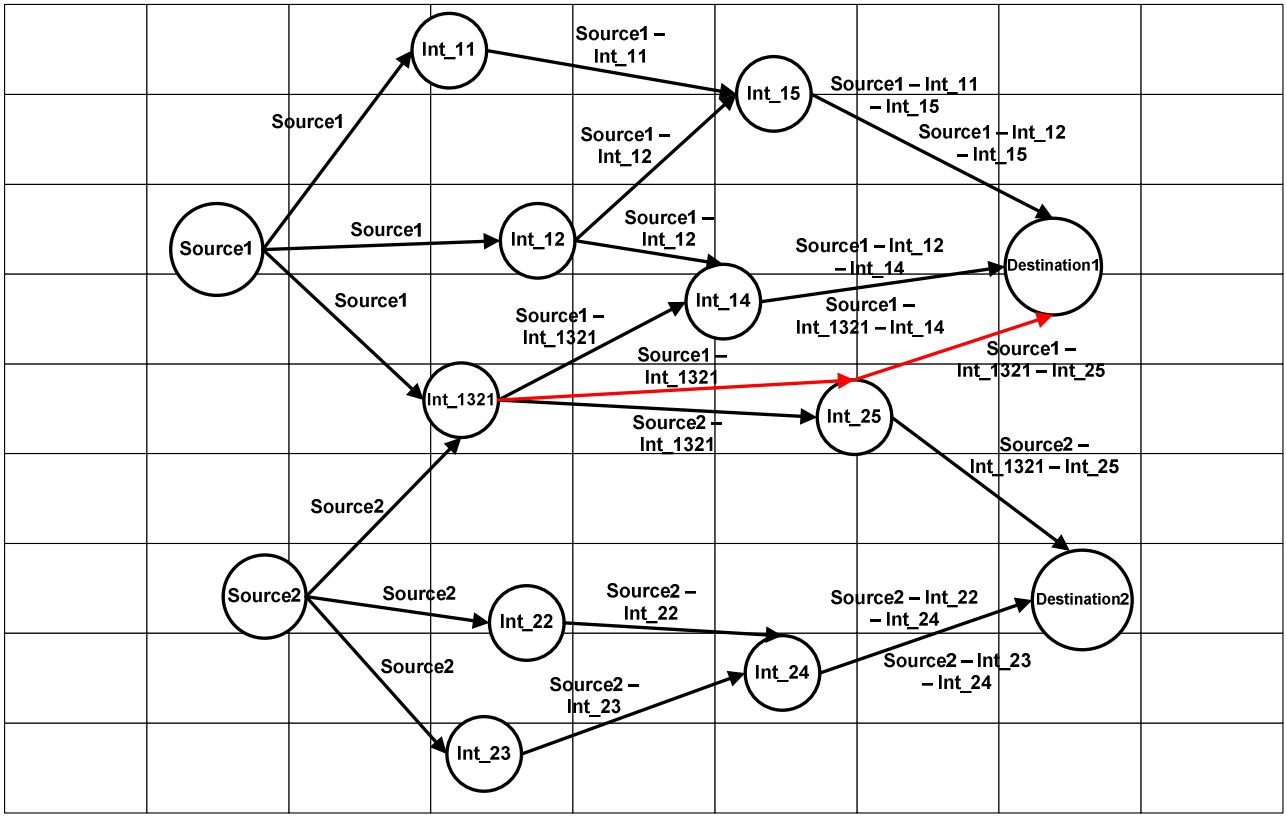


Figura 5.12: Topologia de rede para o modelo SAN utilizando de agregação de serviços de transmissão

Baseado nesta nova topologia houve uma redução considerável na quantidade de estados gerados (de 16.777.216 para 4.194.304 estados) bem como em relação aos estados atingíveis (quando se utiliza o processo de agregação, reduz-se para 371.608 os estados atingíveis). Já, quando se reduz os estados atingíveis através da função de atingibilidade, os estados atingíveis permanecem com a mesma quantidade do modelo anterior (oito estados), determinados pela função de atingibilidade parcial.

A sintaxe para o modelo com agregação que descarta a probabilidade de ausência ou falha de nodos na rede e o processo de manutenção de rotas é apresentada a seguir:

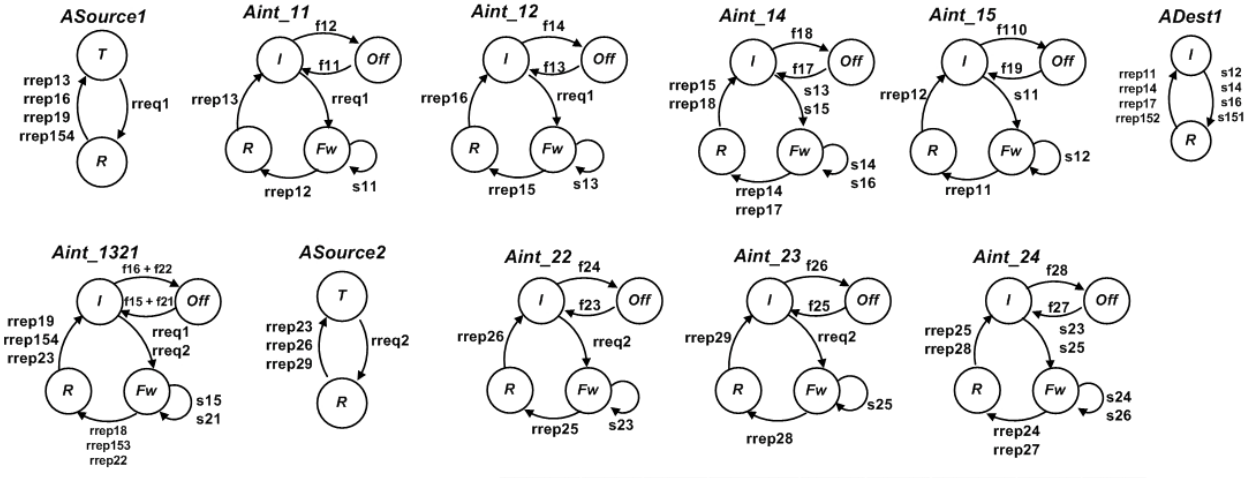
```
partial reachability = (((st ASource1 == T) && (st ADest1 == I) &&
((st AInt_11 == Fw) && (st AInt_15 == Fw)) && ((st AInt_12 != R)
&& (st AInt_12 != Fw) && (st AInt_12 != Off)) && ((st AInt_1321 !=
R) && (st AInt_1321 != Fw) && (st AInt_1321 != Off)) && ((st
AInt_14 != R) && (st AInt_14 != Fw) && (st AInt_14 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_14 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) &&
(st AInt_11 != Off)) && ((st AInt_1321 != R) && (st AInt_1321 !=
Fw) && (st AInt_1321 != Off)) && ((st AInt_15 != R) && (st AInt_15
!= Fw) && (st AInt_15 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_15 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) &&
(st AInt_11 != Off)) && ((st AInt_1321 != R) && (st AInt_1321 !=
Fw) && (st AInt_1321 != Off)) && ((st AInt_14 != R) && (st AInt_14
!= Fw) && (st AInt_14 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_1321 == Fw)
&& (st AInt_14 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw)
&& (st AInt_11 != Off)) && ((st AInt_12 != R) && (st AInt_12 !=
Fw) && (st AInt_12 != Off)) && ((st AInt_15 != R) && (st AInt_15
!= Fw) && (st AInt_15 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_1321 == Fw)
&& (st AInt_25 == Fw)) && ((st AInt_14 != R) && (st AInt_14 =! Fw)
&& (st AInt_15 != Off)) && ((st AInt_11 != R) && (st AInt_11 !=
Fw) && (st AInt_11 != Off)) && ((st AInt_12 != R) && (st AInt_12
!= Fw) && (st AInt_12 != Off))&& ((st AInt_15 =! R) && (st AInt_15
!= Fw) && (st AInt_15 != Off)))
&&
(((st ASource2 == T) && (st ADest2 == I) && ((st AInt_1321 == Fw)
&& (st AInt_25 == Fw)) && ((st AInt_22 != R) && (st AInt_22 != Fw)
&& (st AInt_22 != Off)) && ((st AInt_23 != R) && (st AInt_23 !=
Fw) && (st AInt_23 != Off)) && ((st AInt_24 != R) && (st AInt_24
!= Fw) && (st AInt_24 != Off))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) &&
(st AInt_24 == Fw)) && ((st AInt_1321 != R) && (st AInt_1321 !=
Fw) && (st AInt_1321 != Off)) && ((st AInt_23 != R) && (st AInt_23
!= Fw) && (st AInt_23 != Off)) && ((st AInt_25 != R) && (st
AInt_25 != Fw) && (st AInt_25 != Off))) ||
```

```
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) &&
(st AInt_24 == Fw)) && ((st AInt_22 != R) && (st AInt_22 != Fw) &&
(st AInt_22 != Off)) && ((st AInt_1321 != R) && (st AInt_1321 !=
Fw) && (st AInt_1321 != Off)) && ((st AInt_25 != R) && (st AInt_25
!= Fw) && (st AInt_25 != Off))));
```

A sintaxe para obter as rotas atraves do processo de descoberta de rotas, considerando manutenção de rotas e possíveis falhas na transmissão é a seguinte:

```
partial reachability = (((st ASource1 == T) && (st ADest1 == I) &&
((st AInt_11 == Fw) && (st AInt_15 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_14 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) &&
(st AInt_15 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_1321 == Fw)
&& (st AInt_14 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_1321 == Fw)
&& (st AInt_25 == Fw))))
&&
(((st ASource2 == T) && (st ADest2 == I) && ((st AInt_1321 == Fw)
&& (st AInt_25 == Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) &&
(st AInt_24 == Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) &&
(st AInt_24 == Fw))));
```

Para este processo de agregação, foi construído o seguinte modelo SAN, apresentado na figura 5.13.



Evento	Tipo	Taxa	Evento	Tipo	Taxa	Evento	Tipo	Taxa
f11	loc	0.8	rreq1	syn	0.5	rrep153	syn	ι_1
f12	loc	0.2	s11	syn	τ_1	rrep154	syn	ς_1
f13	loc	0.8	s12	syn	γ_1	rreq2	syn	0.5
f14	loc	0.2	s13	syn	Ψ_1	s21	syn	τ_2
f15	loc	0.8	s14	syn	ρ_1	s22	syn	γ_2
f16	loc	0.2	s15	syn	ν_1	s23	syn	Ψ_2
f17	loc	0.8	s16	syn	ξ_1	s24	syn	ρ_2
f18	loc	0.2	s151	syn	H_1	s25	syn	ν_2
f19	loc	0.8	rrep11	syn	α_1	s26	syn	ξ_2
f110	loc	0.2	rrep12	syn	β_1	rrep21	syn	α_2
f23	loc	0.8	rrep13	syn	Δ_1	rrep22	syn	β_2
f24	loc	0.2	rrep14	syn	Ω_1	rrep23	syn	Δ_2
f25	loc	0.8	rrep15	syn	σ_1	rrep24	syn	Ω_2
f26	loc	0.2	rrep16	syn	χ_1	rrep25	syn	σ_2
f27	loc	0.8	rrep17	syn	κ_1	rrep26	syn	χ_2
f28	loc	0.2	rrep18	syn	Θ_1	rrep27	syn	κ_2
f29	loc	0.8	rrep19	syn	Z_1	rrep28	syn	Θ_2
f210	loc	0.2	rrep152	syn	Y_1	rrep29	syn	Z_2

Figura 5.13: SAN correspondente à topologia do modelo de agregação de serviços de transmissão

No modelo SAN apresentado, é possível observar que o autômato *AInt_1321* recebe tanto a mensagem de requisição de rota da primeira rede (através do evento *rreq1*) quanto da segunda rede (evento *rreq2*), disparando eventos de repasse de mensagem de requisição de rota (*s15* para a primeira rede e *s21* para a segunda rede) e também os eventos que fazem a resposta de rota das redes (eventos $rrep_n$).

Com esta modelagem, comparando-se com o modelo de escuta promíscua, foi observada uma redução de estados globais, em função do autômato assumir uma posição de concentrador de informações de roteamento para ambas as redes, atuando tanto em uma rede quanto em outra e ainda assim garantir a transmissão dos pacotes.

O objetivo de agregar serviços de roteamento em um determinado nodo, quando modelado em SAN, trouxe, além da redução do número de autômatos presentes na rede, mostrou-se também como um esforço na busca de redução de estados possíveis para os modelos SAN. Uma vez implementadas novas possibilidades de redução de estados globais do modelo, a preocupação na geração dos modelos é direcionada para a otimização de tempo e custo computacional.

6 RESULTADOS NUMÉRICOS

Nesta seção serão apresentados os resultados gerados dos índices avaliados do protocolo DSR, entre os vários modelos construídos e avaliados na ferramenta PEPS, bem como uma comparação com alguns resultados apresentados em estudos que utilizaram a simulação como formalismo de avaliação. Nos trabalhos relacionados elencados no Capítulo 5, constatou-se uma preocupação referente à vazão tanto de pacotes de transmissão do próprio protocolo DSR quanto de pacotes de dados a serem transmitidos, bem como com a inserção de tempo de pausa entre as transmissões. [Bou04, Bro98, Per01, Lay07].

Outra métrica avaliada utilizando formalismo SAN foi o de *workload* da rede, que auxiliou na extração de informações sobre as melhores rotas no processo de descoberta de rotas, sendo avaliado junto com este processo.

Uma vez que este trabalho visa além de avaliar o protocolo DSR, ter como contribuição a obtenção de dados referentes ao seu processo de descoberta de rotas serão apresentados inicialmente os resultados obtidos para este processo de todos os modelos gerados, e subsequentemente a apresentação dos índices obtidos e algumas comparações com aqueles encontrados na literatura com o uso de simulação.

6.1 Processo de descoberta de rotas

O processo de descoberta de rotas do protocolo DSR apresentado e avaliado neste trabalho é uma técnica que não é avaliada na literatura que utiliza simulação como formalismo de avaliação de desempenho, sendo assim, buscou-se identificar qual a rota mais eficiente em termos de entrega da mensagem de requisição de rota, para que assim, o envio de pacotes fosse feito através desta.

Alguns aspectos foram assumidos no momento de iniciar o processo:

• O nodo fonte sempre terá uma frequência constante de pacotes a serem enviados, com um intervalo assumido de 500ms;

• Como o nodo fonte necessita iniciar o processo de descoberta de rotas, é assumido que este não tem nenhuma rota para o nodo destino armazenado em seu *cache* de rotas;

Os primeiros índices apresentados dizem respeito ao processo de descoberta de rota, apresentados a seguir na Figura 6.1, dizem respeito ao modelo de redes simples, sem que houvesse falhas na transmissão, ou seja, é admitido que todos os nodos estejam disponíveis na rede, descartando assim os eventos locais f_n que podem indisponibilizar o nodo na rede.

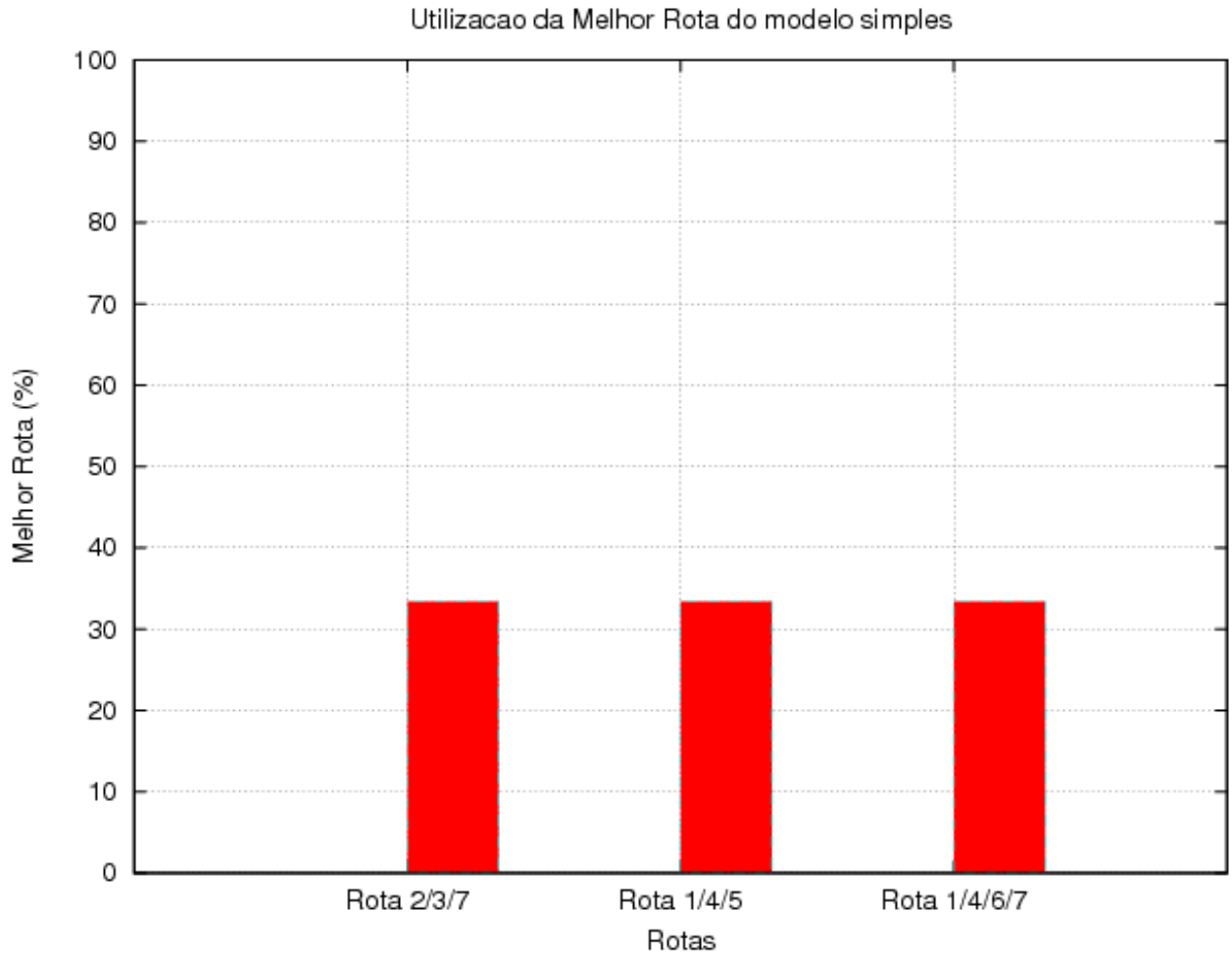


Figura 6.1: Gráfico de desempenho de rotas do modelo simples

Este gráfico mostra que no modelo de rotas simples, todas as rotas possuem desempenho semelhante em relação à utilização das rotas para o processo de descoberta das mesmas e de repasse das informações de requisição de rota, mesmo a rota com maior número de nodos. Nesta avaliação, foram levados em consideração apenas os processos de descoberta de rota, com todos os nodos disponíveis e descartando falhas.

Para que estes resultados fossem obtidos, foram usadas funções de integração que levavam em consideração as probabilidades dos autômatos que representam os nodos intermediários estarem

no estado *Fw*, ou seja, transmitindo a mensagem de requisição de rota (*rreq*). A sintaxe para tal função é assim descrita:

```
t_route_237 = ((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st
AInt_7 == Fw));
t_route_145 = ((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st
AInt_5 == Fw));
t_route_1467 = ((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st
AInt_6 == Fw) && (st AInt_7 == Fw));
```

A título de comparação, quando realizado o mesmo processo de descoberta de rotas considerando falhas de transmissão, obtêm-se valores diferentes dos apresentados na Figura 6.1, como pode ser visto na Figura 6.2.

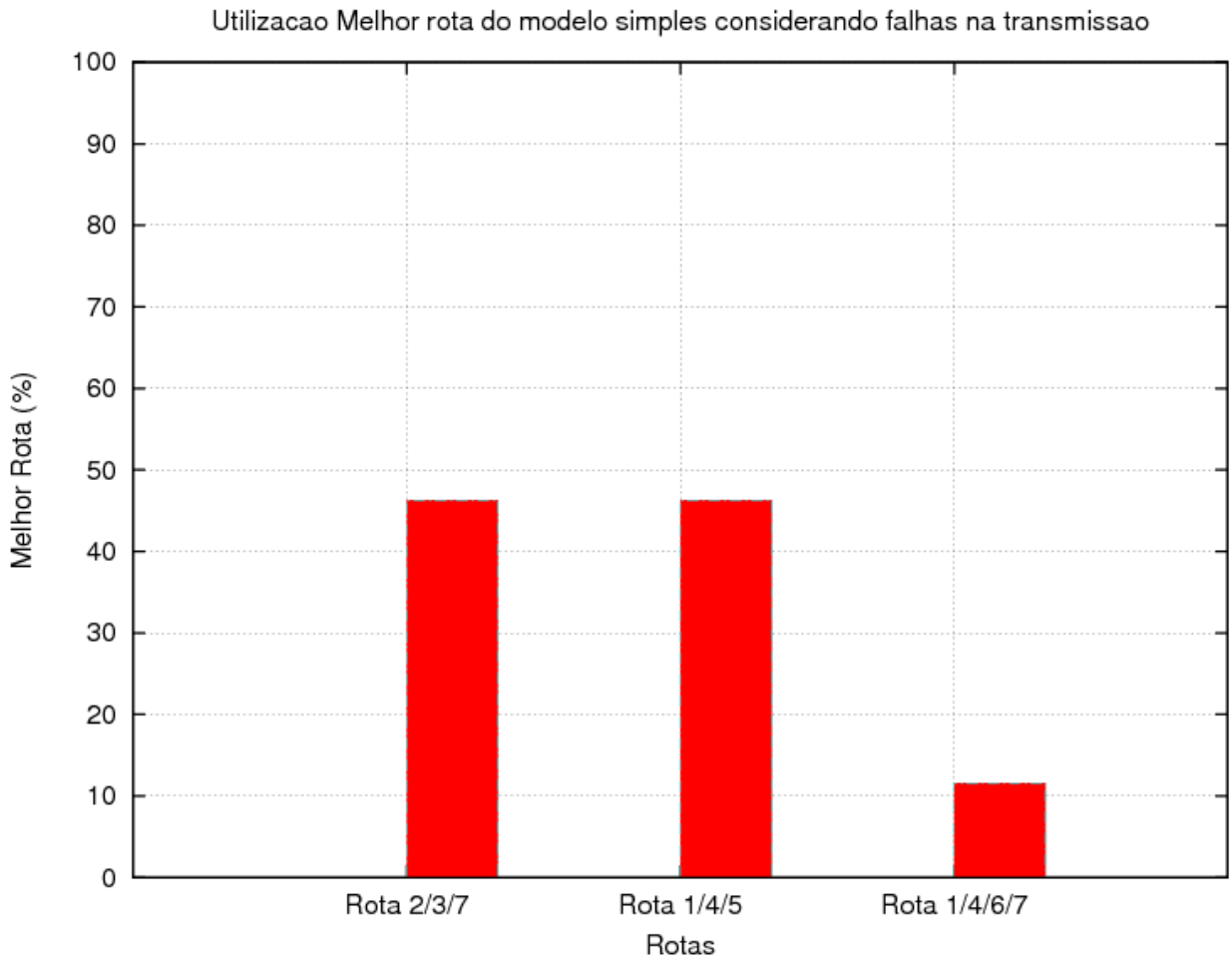


Figura 6.2: Gráfico de desempenho de rotas do modelo simples considerando falhas de transmissão

Quando existe a probabilidade de falha na transmissão, como mostrado na sintaxe no capítulo anterior, e os autômatos que representam os nodos intermediários podem estar indisponíveis na rede, quando os eventos locais f_n são admitidos, com taxas assumidas de 80% (0.8)

de que o nodo esteja disponível e 20% (0.2) para que este não esteja disponível no momento de disparo do evento *rreq*, a rota que possui mais nodos (1/4/6/7) se mostrou menos utilizada do que as outras rotas, que tiveram uma utilização ainda maior que no modelo que ignorava as possíveis falhas de transmissão.

A Tabela 6.1 apresenta os resultados numéricos da avaliação do processo de descoberta de rotas para os modelos simples, sendo que a coluna *Sem probabilidades de falhas* diz respeito à probabilidade de uso da rota no modelo que desconsidera falhas de transmissão e processo de manutenção de rota, e a coluna *Com probabilidades de falhas* mostra a probabilidade de uso das rotas do modelo, levando em consideração as possíveis falhas de transmissão e consequentemente o processo de manutenção de rotas.

Tabela 6.1: Comparação dos resultados de utilização das rotas no processo de descoberta de rota para o modelo simples com e sem a possibilidade de falhas na transmissão

Rota	Sem probabilidade de falhas – Utilização (%)	Com probabilidade de falhas – Utilização (%)
2/3/7	33.3333	46.2929
1/4/5	33.3333	46.2929
1/4/6/7	33.3333	11.5732

Na avaliação do *workload* dos nodos na rede, onde se buscou mensurar a carga de roteamento de cada nodo presente na rede, avaliou-se cada nodo individualmente, de forma a apresentar a carga de trabalho no momento da transmissão dos pacotes de roteamento do protocolo DSR, para isso, para cada nodo, foi construída uma função de integração que levava em consideração a probabilidade do nodo encontrar-se transmitindo mensagens de requisição de rota, ou seja, no estado *Fw*.

A sintaxe para a obtenção de tais índices é apresentada a seguir, e tem o mesmo formato para todos os outros modelos, buscando sempre obter a probabilidade dos autômatos intermediários se encontrarem no estado *Fw*.

```
t_AInt_1_Fw = (st AInt_1 == Fw);   //Apresenta a probabilidade do
nodo 1 estar transmitindo mensagens de requisição de rota ou
repassando tal mensagem
t_AInt_1_Fw = (st AInt_2 == Fw);
t_AInt_1_Fw = (st AInt_3 == Fw);
t_AInt_1_Fw = (st AInt_4 == Fw);
t_AInt_1_Fw = (st AInt_5 == Fw);
t_AInt_1_Fw = (st AInt_6 == Fw);
t_AInt_1_Fw = (st AInt_7 == Fw);
```

Para o modelo que descarta as probabilidades de falhas de transmissão, a Figura 6.3 mostra o *workload* dos nodos.

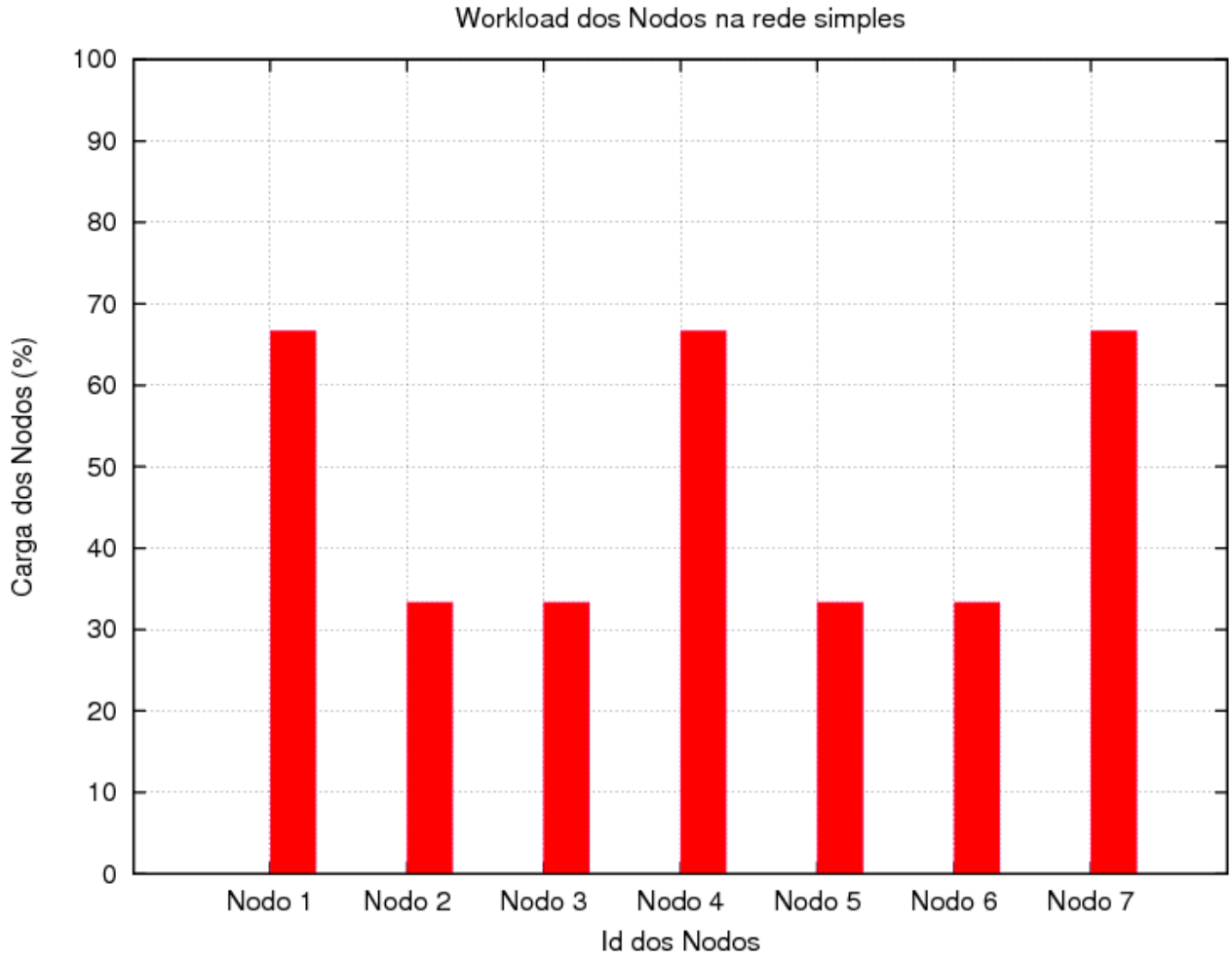


Figura 6.3: *Workload* dos nodos no processo de descoberta de rotas do modelo simples

O que pode ser analisado em relação ao desempenho do uso dos nodos na rede é diretamente ligado à quantidade de rotas em que cada nodo participa no momento da transmissão. Os nodos 1, 4 e 7 participam cada um de duas rotas de transmissão, enquanto os nodos 2, 3 e 5 só são requisitados em uma rota cada um. O resultado da avaliação do *workload* dos nodos reflete diretamente na avaliação de utilização das rotas no processo de descoberta de rotas, pois é perceptível que nas rotas que possuem maior utilização, são as mesmas que tem os nodos com maior carga de trabalho de roteamento.

No modelo em que existe a probabilidade de falhas na transmissão, os nodos agora apresentaram um *workload* maior que no modelo anterior, mesmo havendo as falhas, pois, devido a

elas, as atividades de roteamento são mais intensas dentro da rede. Os resultados são apresentados na Figura 6.4.

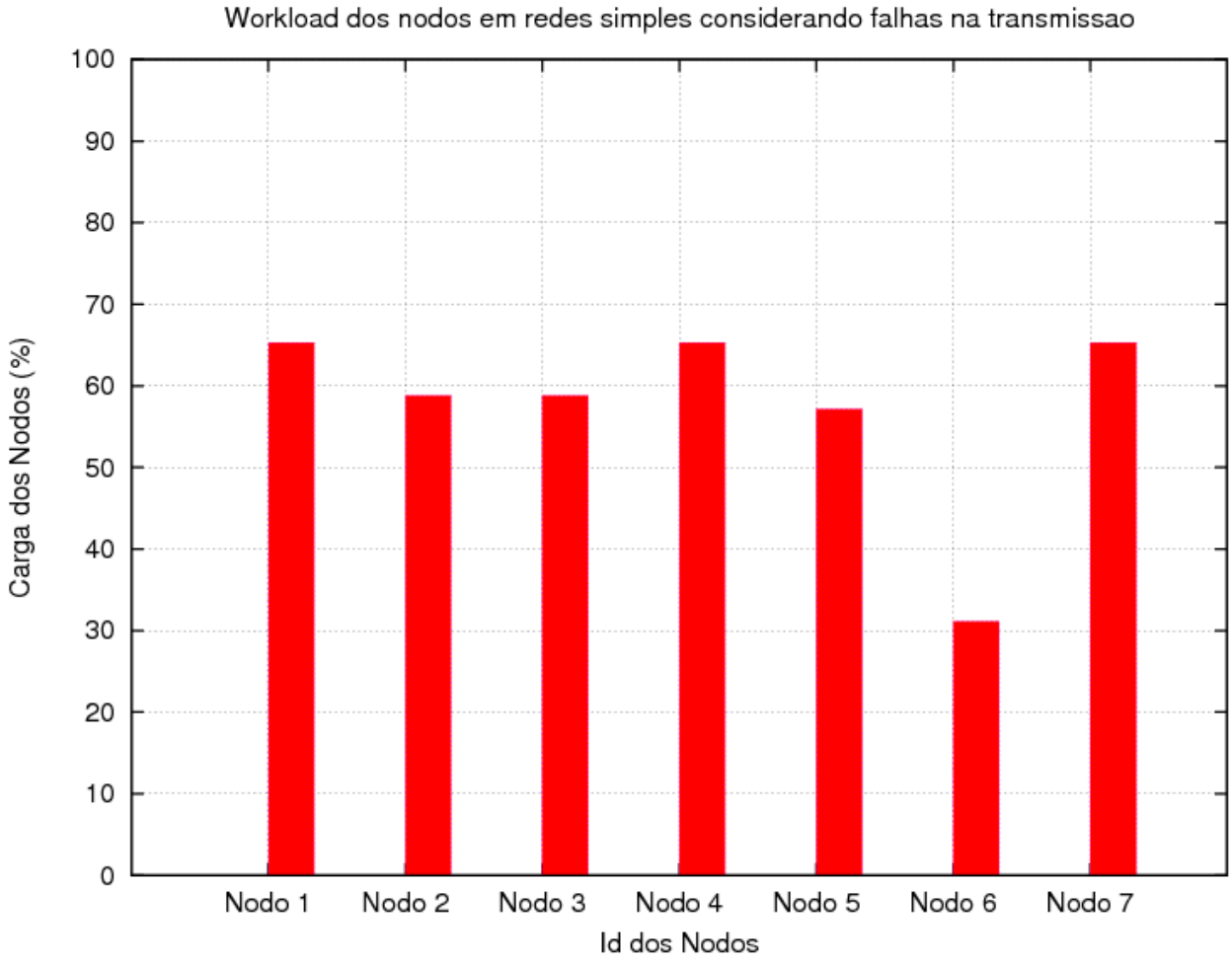


Figura 6.4: *Workload* dos nodos no processo de descoberta de rotas do modelo simples considerando falhas de transmissão

No gráfico apresentado na Figura 6.4, é possível observar que o nodo 6 apresenta um *workload* menor que dos outros nodos, que deve-se ao fato deste nodo, apesar de estar eu uma rota que possui dois nodos com participação na transmissão maiores que os demais (nodos 1 e 7), a rota é a que mostrou menor eficiência no momento do processo de descoberta de rotas, e por possuir um maior número de nodos, está mais suscetível à ausência ou falha dos nodos presentes em sua rota.

A Tabela 6.2 apresenta numericamente o *workload* dos nodos nos dois modelos simples, tanto o modelo que não leva em consideração as falhas de transmissão quanto àquele que o faz. A coluna *Id dos Nodos* é onde os nodos da rede são identificados; a coluna *Sem falhas de transmissão* apresenta as probabilidades de *workload* dos nodos no modelo que descarta falhas de transmissão,

já a coluna *Com probabilidade de falhas de transmissão* apresenta as probabilidades de *workload* dos nodos no modelo que conta com falhas de transmissão e manutenções de rotas.

Os nodos 1, 4 e 7 continuam a possuir um *workload* mais elevado que os outros, assim como no modelo que descarta falhas, contudo, existem agora variações entre os outros nodos.

Tabela 6.2: Comparação dos resultados do *workload* dos nodos para o modelo simples com e sem a possibilidade de falhas na transmissão

Id dos nodos	Sem falhas de transmissão (%)	Com probabilidade de falhas de transmissão (%)
Node_1	66.666666	65.280289
Node_2	33.333333	58.770343
Node_3	33.33333	58.770343
Node_4	66.666666	65.280289
Node_5	33.333333	57.142857
Node_6	33.333333	31.103074
Node_7	66.666666	65.280289

Para os modelos de maior complexidade, tanto o processo de descoberta de rotas quanto de utilização dos nodos apresentaram comportamentos semelhantes, tendo um *workload* maior dos nodos quando consideradas as falhas na transmissão, porém um desempenho menor das rotas no processo de descoberta das mesmas.

O modelo que analisou a técnica de escuta promíscua (*promiscuous listening*) apresentou, no processo de descoberta de rotas, um desempenho constante das rotas simples, mas um desempenho relevantemente inferior quando este uso o modo de escuta promíscua, como apresentado no gráfico da Figura 6.5.

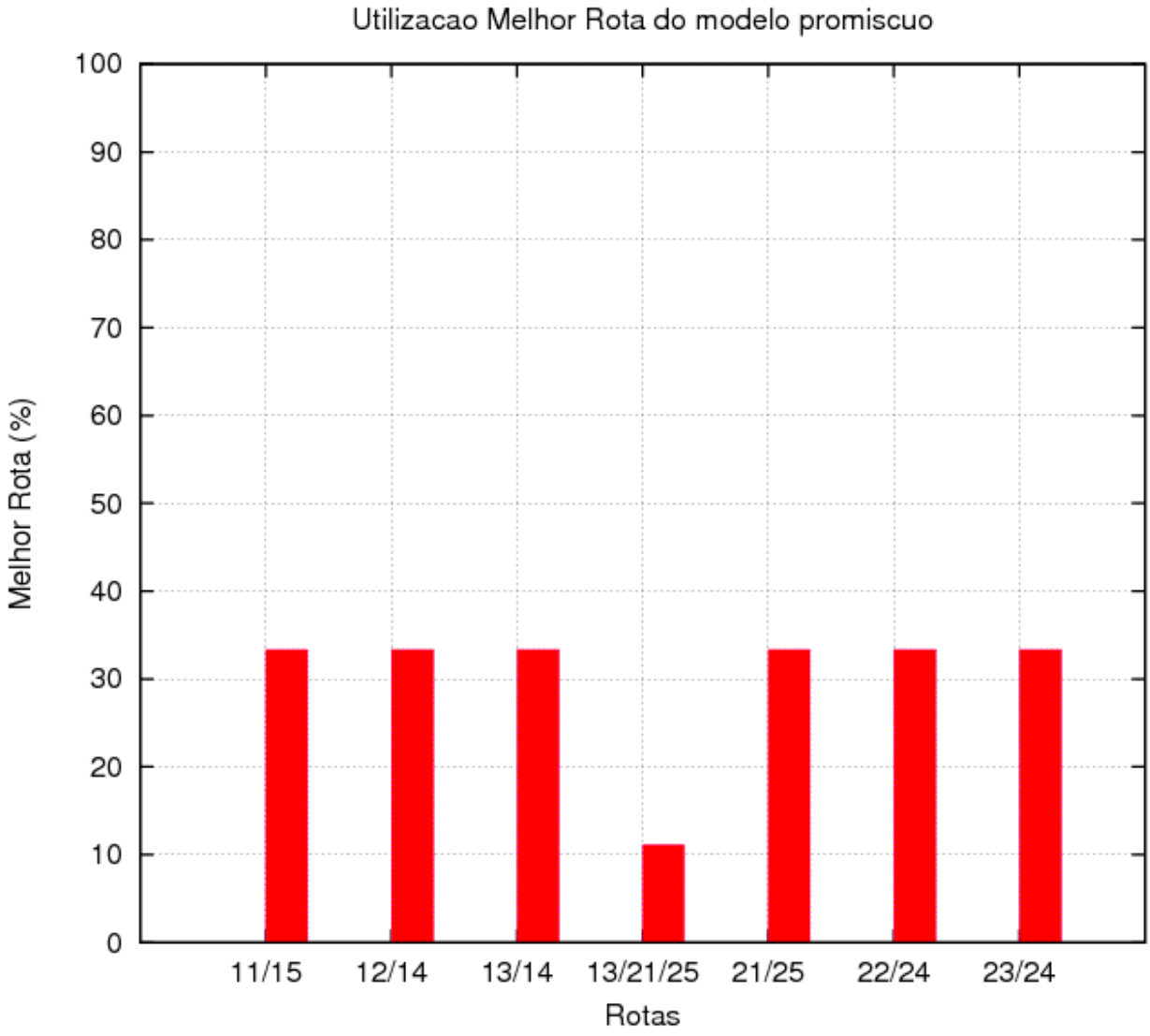


Figura 6.5: Gráfico de desempenho de rotas do modelo de escuta promíscua

Já o modelo de escuta promíscua que admite falhas na transmissão, o desempenho das rotas foi levemente inferior em relação ao modelo que desconsidera esse tipo de situação de transmissão, com exceção de uma única rota (através dos nodos 21 e 25), porém, a rota que utiliza da escuta promíscua teve um desempenho semelhante às outras rotas, o que não aconteceu no modelo anteriormente avaliado. O desempenho obtido para esse modelo pode ser visto no gráfico presente na Figura 6.6.

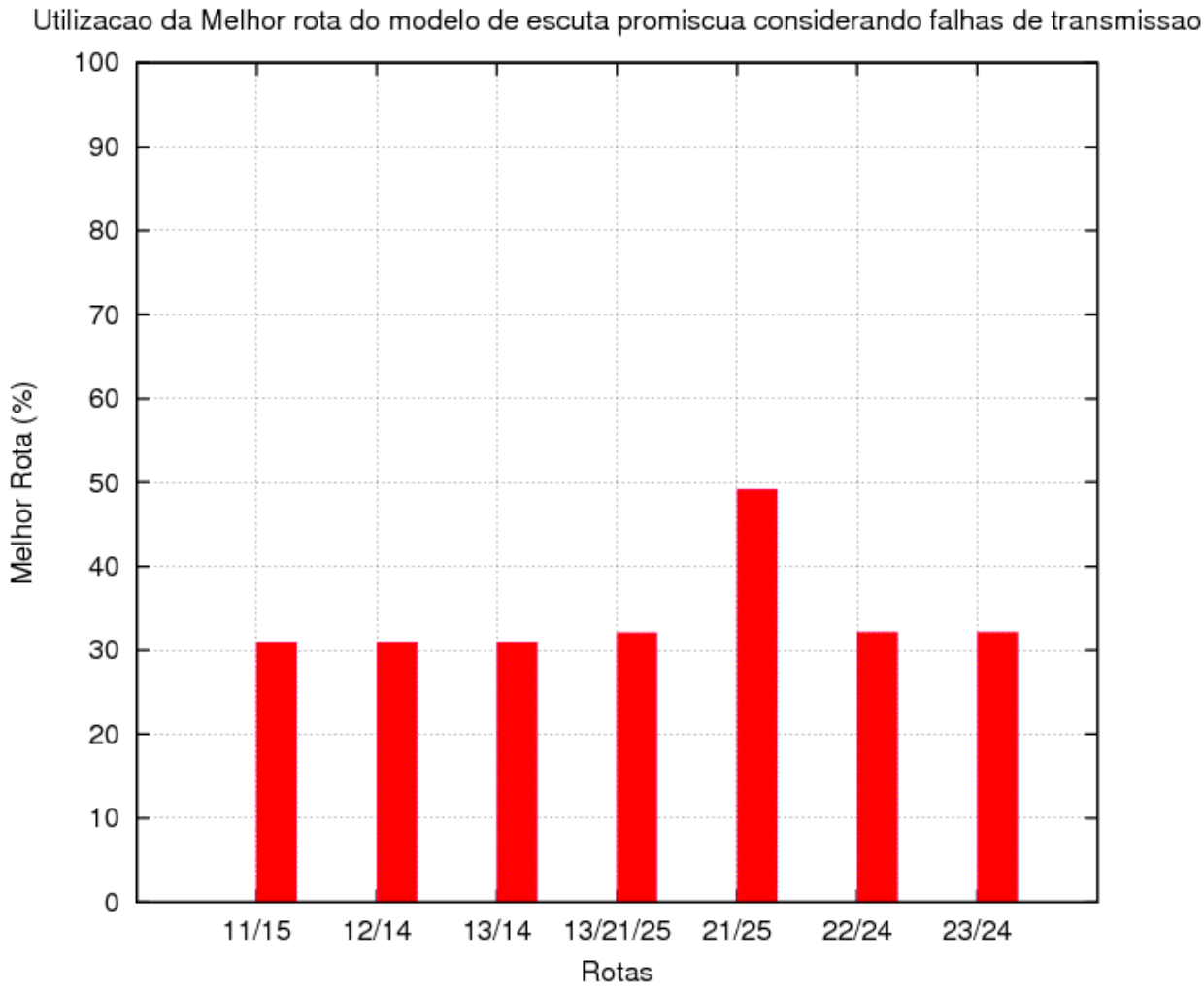


Figura 6.6: Gráfico de desempenho de rotas do modelo de escuta promíscua considerando falhas de transmissão

A Tabela 6.3 apresenta um comparativo dos resultados obtidos para as rotas na avaliação do processo de descoberta de rotas sem falhas na transmissão como os resultados do modelo que o faz.

Tabela 6.3: Comparação dos resultados do *workload* dos nodos para o modelo simples com e sem a possibilidade de falhas na transmissão

Rota	Sem falhas na transmissão (%)	Com probabilidade de falhas na transmissão (%)
11/15	33.333333	30.988861
12/14	33.333333	30.988861
13/14	33.333333	30.988861
13/21/25	11.111111	32.105856
21/25	33.333333	49.158898
22/24	33.333333	32.124488
23/24	33.333333	32.124488

Na avaliação do *workload* dos nodos presentes no modelo de escuta promíscua, os nodos 14 e 24 apresentaram uma carga relevantemente maior que os outros nodos, em decorrência destes estarem presentes em quase todas as rotas possíveis para se chegar ao nodo destino. O gráfico mostrado na Figura 6.7 apresenta o *workload* dos nodos deste modelo.

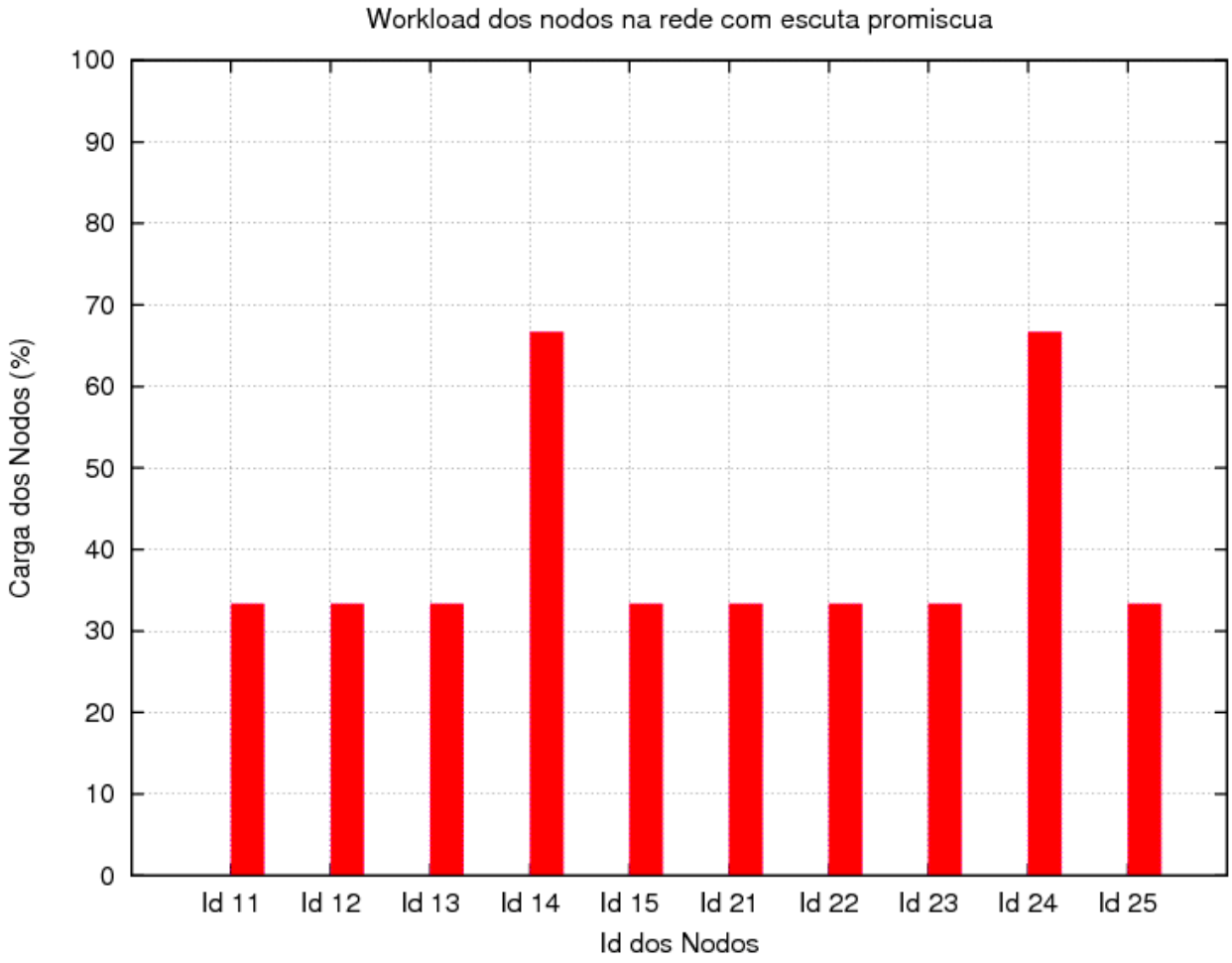


Figura 6.7: *Workload* dos nodos no processo de descoberta de rotas do modelo de escuta promíscua

Uma vez avaliado o modelo de escuta promíscua que desconsidera as falhas de transmissão, é necessário avaliar o modelo que o faz, onde o *workload* dos nodos foi consideravelmente maior que no modelo avaliado anteriormente, apresentando uma variação da carga dos nodos maior do que no outro modelo que descartava a possibilidade de falhas na transmissão dos pacotes, e também mostra um *workload* semelhante entre o nodo 14 que participa da maioria das rotas para o nodo destino e o nodo 21, que é o nodo responsável pela escuta promíscua dentro da rede. A Figura 6.8 traz o gráfico referente a essa avaliação.

Figura 6.8: *Workload* dos nodos no processo de descoberta de rotas do modelo de escuta promíscua considerando falhas de transmissão

São apresentadas as devidas comparações de desempenho quando avaliado o *workload* dos nodos na Tabela 6.4 dos modelos que utilizam a técnica de escuta promíscua. Como visto nos gráficos, quando são consideradas as falhas na transmissão dos pacotes de roteamento, a carga dos nodos é maior em relação ao modelo anterior, bem como agora o nodo 21 que é responsável pela escuta promíscua tem uma participação maior no processo de descoberta de rotas que no modelo anterior.

Tabela 6.4: Comparação dos resultados do *workload* dos nodos para o modelo de escuta promíscua com e sem a possibilidade de falhas na transmissão

Id dos Nodos	Sem falhas de transmissão (%)	Com probabilidade de falhas na transmissão (%)
Node_11	33.333332	47.758259
Node_12	33.333332	45.050745
Node_13	33.333332	54.969310

Id dos Nodos (cont.)	Sem falhas de transmissão (%) (cont.)	Com probabilidade de falhas na transmissão (%) (cont.)
Node_14	66.666664	58.588316
Node_15	33.333332	47.758259
Node_21	33.333335	59.327119
Node_22	33.333332	41.341782
Node_23	33.333332	41.341782
Node_24	66.666664	63.130825
Node_25	33.333335	59.327119

No modelo com agregação de serviços de roteamento, é possível observar que o desempenho das rotas que tem o nodo agregador sofrem uma grande perda quando comparadas às outras rotas. As Figuras 6.9 e 6.10 mostram o processo de descoberta de rotas do modelo com agregação de serviços, onde são avaliados os modelos que descartam perdas e falhas de transmissão bem como o modelo que analisa todas essas possibilidades.

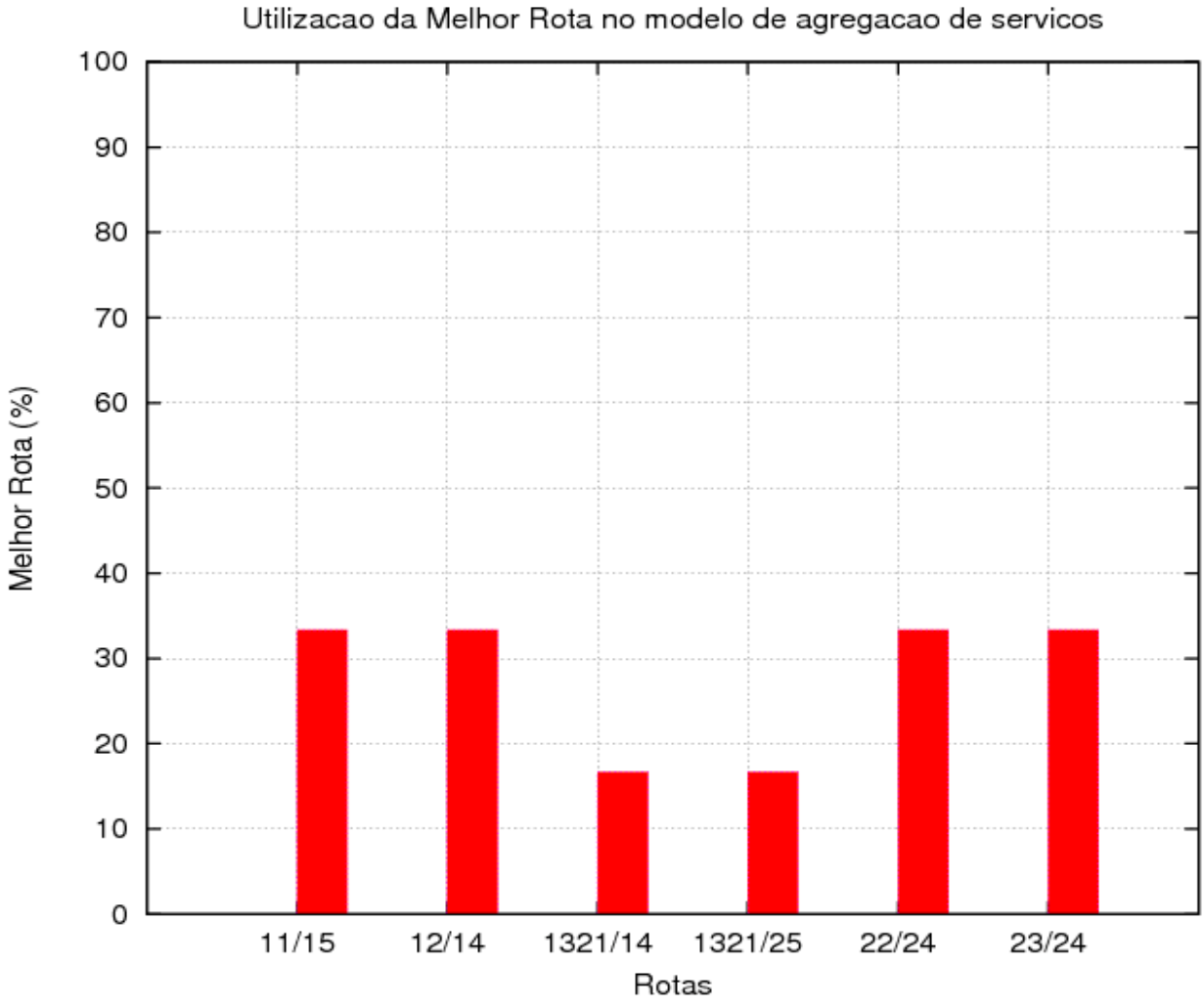


Figura 6.9: Gráfico de desempenho de rotas do modelo com agregação de serviços

Na Figura 6.9, pode ser observado que nas rotas que utilizam a agregação de serviços de roteamento, o desempenho na descoberta de rotas não é tão satisfatório quanto nas outras rotas, que tem desempenho semelhante para ambas as redes. Na Figura 6.10, onde são admitidas perdas na transmissão e processos de manutenção de rota, o desempenho no processo de descoberta de rotas é muito menor que no modelo que descartava as falhas de transmissão, sendo que para os modelos que usam a agregação de serviços é quase nulo.

Figura 6.10: Gráfico de desempenho de rotas do modelo com agregação de serviços considerando falhas de transmissão

É possível observar na Tabela 6.5 os resultados numéricos dos processos de rota para os modelos que utilizam agregação de serviços.

Tabela 6.5: Comparação dos resultados dos processos de descoberta de rota para o modelo com agregação de serviços com e sem a possibilidade de falhas na transmissão

Rota	Sem falhas na transmissão (%)	Com probabilidade de falhas na transmissão (%)
11/15	33.333334	4.492378
13/14	33.333334	3.955703
1321/14	16.666667	2.486215
1321/25	16.666667	3.442904
22/24	33.333334	4.081751
23/24	33.333334	4.081751

O *workload* do modelo com agregação de serviços apresenta um desempenho com variação significativa em relação aos processos de descoberta de rotas que não estão suscetíveis à falhas e o modelo que apresenta a possibilidade de ocorrer manutenção de rota e falhas de transmissão. As Figuras 6.11 e 6.12 apresentam esses gráficos.

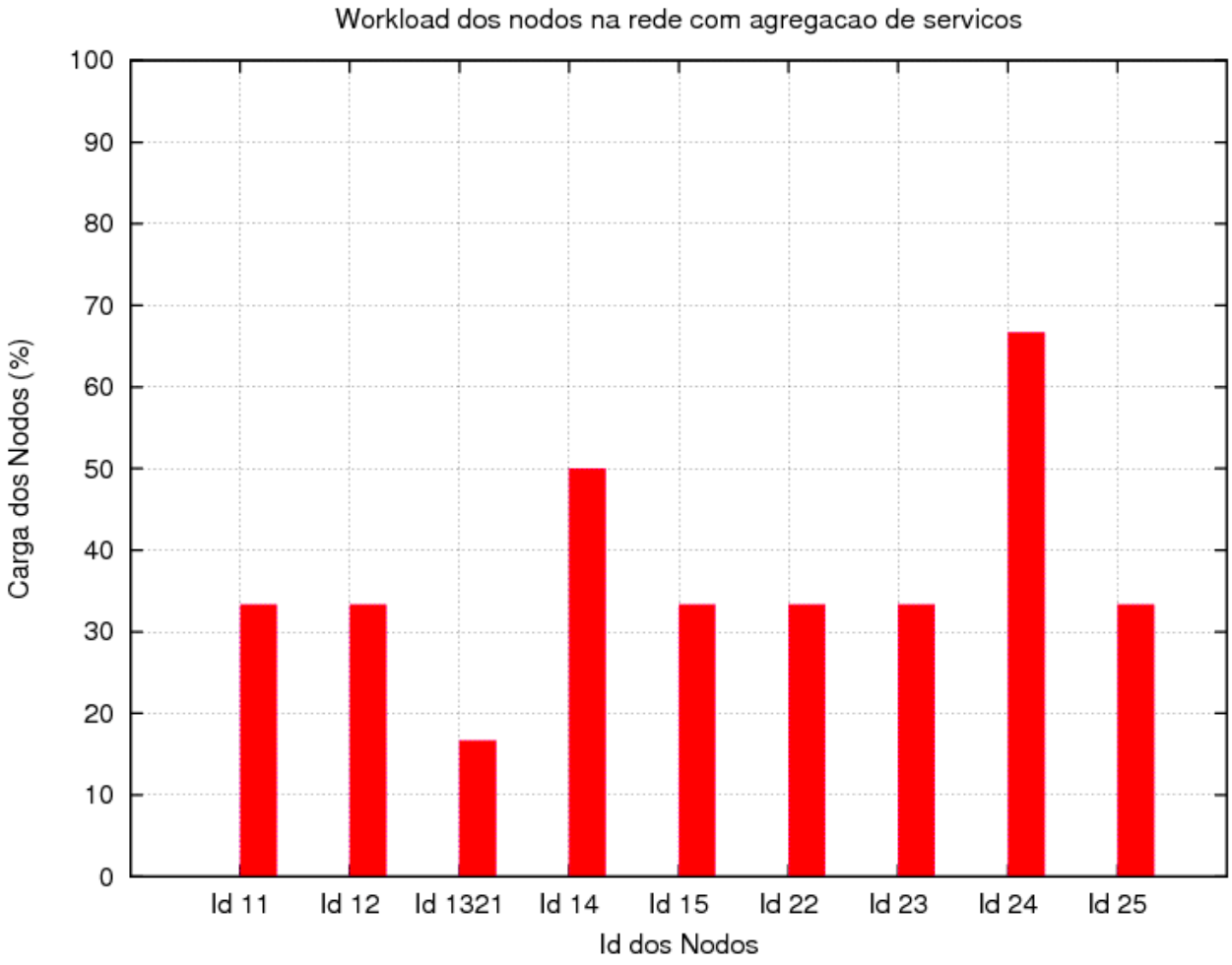


Figura 6.11: *Workload* dos nodos no processo de descoberta de rotas do modelo com agregação de serviços

Para o modelo gerado suscetível a falhas de transmissão e manutenção de rotas, o *workload* gerado para o nodo que agregou os serviços de roteamento foi consideravelmente melhor que o modelo anterior, o que o torna mais eficiente quando este assume que é possível que falhas existam no momento da descoberta de rotas, apesar do desempenho na descoberta de rotas ser consideravelmente inferior.

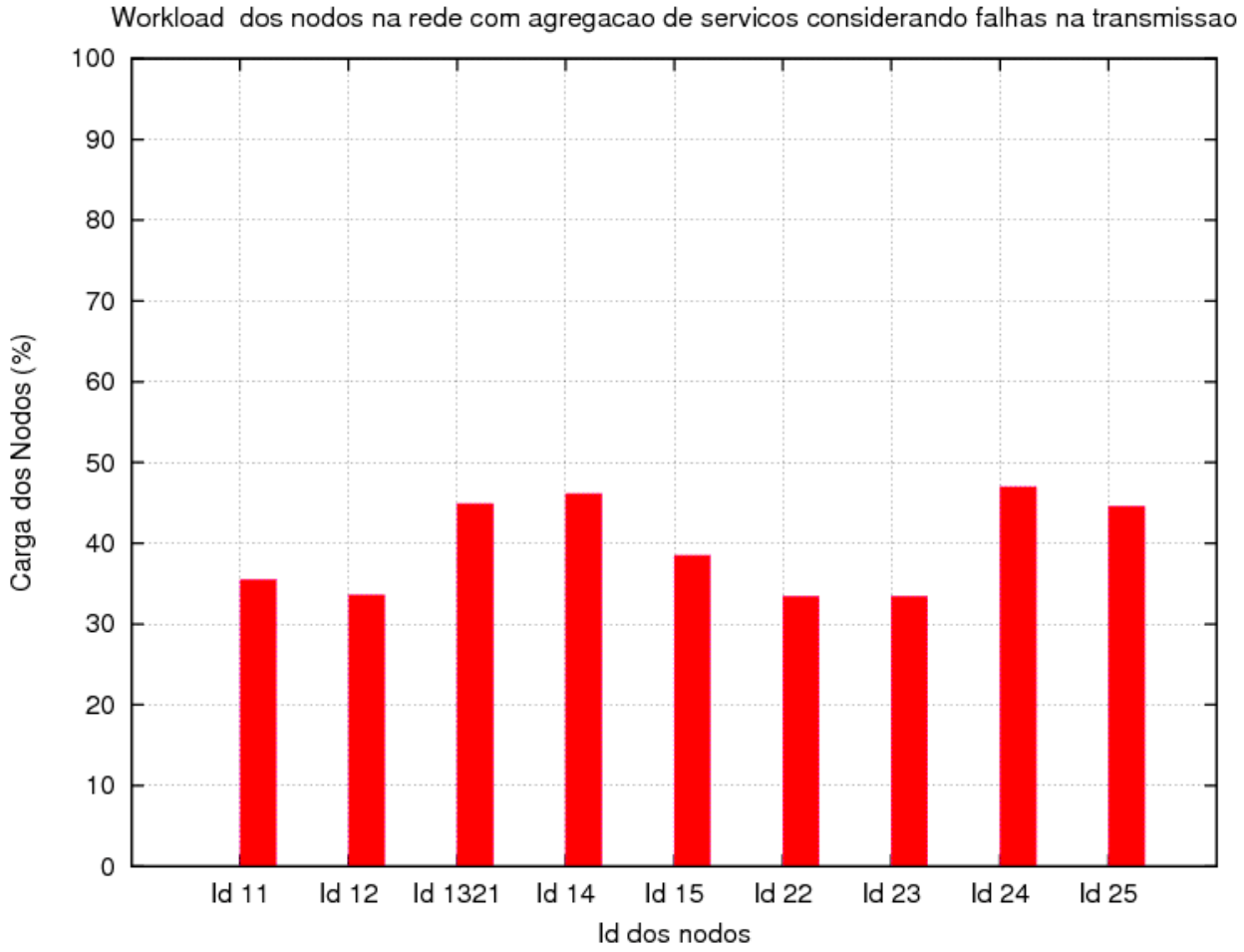


Figura 6.12: *Workload* dos nodos no processo de descoberta de rotas do modelo de agregação de serviços considerando falhas de transmissão

Na Tabela 6.6 o *workload* dos nodos nos modelos que utilizaram agregação de serviços é apresentado, tanto descartando falhas na transmissão quanto às considerando.

Tabela 6.6 Comparação dos resultados do *workload* dos nodos para o modelo com agregação de serviços com e sem a possibilidade de falhas na transmissão

Id dos Nodos	Sem falhas na transmissão (%)	Com probabilidade de falhas na transmissão (%)
Node_11	33.333334	35.515944
Node_12	33.333334	33.605069

Id dos Nodos (cont.)	Sem falhas na transmissão (%) (cont.)	Com probabilidade de falhas na transmissão (%) (cont.)
Node_1321	16.666666	44.891687
Node_14	50.000000	46.123528
Node_15	33.333334	38.500126
Node_22	33.333334	33.448859
Node_23	33.333334	33.448859
Node_24	66.666666	46.969020
Node_25	33.333334	44.587701

Estes resultados obtidos mostram que no processo de descoberta de rotas, a utilização das rotas nem sempre leva em consideração a rota mais curta, e sim, na sua maioria de tentativas, as rotas que possuem os nodos com maior carga de rede, podendo usar como base para esta afirmação o *workload* dos nodos apresentados nos gráficos, onde os nodos que possuem maior carga no roteamento de pacotes são os mesmos nodos que participam das rotas que são as mais utilizadas na transmissão dos pacotes de roteamento e de pacotes de dados.

No tópico que se segue, serão apresentados os índices de desempenho relativos a vazão dos modelos apresentados neste trabalho, que é um dos índices mais utilizados na literatura para avaliar os protocolos de roteamento, tendo a simulação como formalismo de avaliação.

6.2 Vazão (*throughput*)

6.2.1 Vazão por tamanho de pacotes

A vazão é considerada como um dos índices mais importantes na avaliação de desempenho dos protocolos de roteamento para redes *wireless Ad hoc*, estando presente na maioria dos trabalhos relacionados a este tipo de avaliação [Adi08, Ahm06, Bou04, Bro98, Das00, Das98, Lay07, Puc07].

A vazão é dada pela quantidade de bits por segundo (*bps*) que são efetivamente transmitidos. De uma forma geral, o termo vazão é definido como a razão (requisições por unidade de tempo) em que as requisições podem ser servidas por um sistema. A vazão é medida em pacotes por segundo (*pps*) ou bits por segundo (*bps*), conforma Jain em [Jai91], que também afirma que este índice pode também ser medido em bytes por segundo (Bps).

Neste trabalho, na obtenção da vazão dos pacotes de roteamento, foram considerados os seguintes elementos:

• O conjunto de pacotes de roteamento, dados pelo cabeçalho DSR fixo, apresentado na Figura 4.2 deste trabalho;

• A taxa de transmissão da rede (que neste trabalho foi admitida como sendo de 2Mbps);

• Variação no tamanho dos pacotes de dados, que foram somados aos tamanhos de cabeçalho do protocolo DSR.

A sintaxe usada para definir os elementos acima elencados nos modelos gerados na ferramenta PEPS foi assim descrita:

```
bandwidth_b = 2000000; // largura de banda em bits/segundos
bandwidth_B = bandwidth_b/8; // largura de banda em bytes/segundos
bandwidth_Mbps = bandwidth_b/1000000;     // largura de banda em
bytes/segundos
size_pack_64 = 64;          // tamanho de pacotes em bytes
size_pack_128 = 128;
size_pack_256 = 256;
size_pack_512 = 512;
NXH = 7;     // campo DSR Next Header em bytes
RESVD = 8;   //Campo DSR Reserved Header em bytes
MAC = 47;    // Campo Medium Access Control header em bytes
PALG = 16;  // Campo DSR Payload Length header em bytes
header_dsr = NXH+RESVD+MAC+PALG;         // cabeçalho DSR fixo
```

A Figura 6.13 apresenta a vazão do modelo simples quando avaliado em relação aos pacotes de dados.

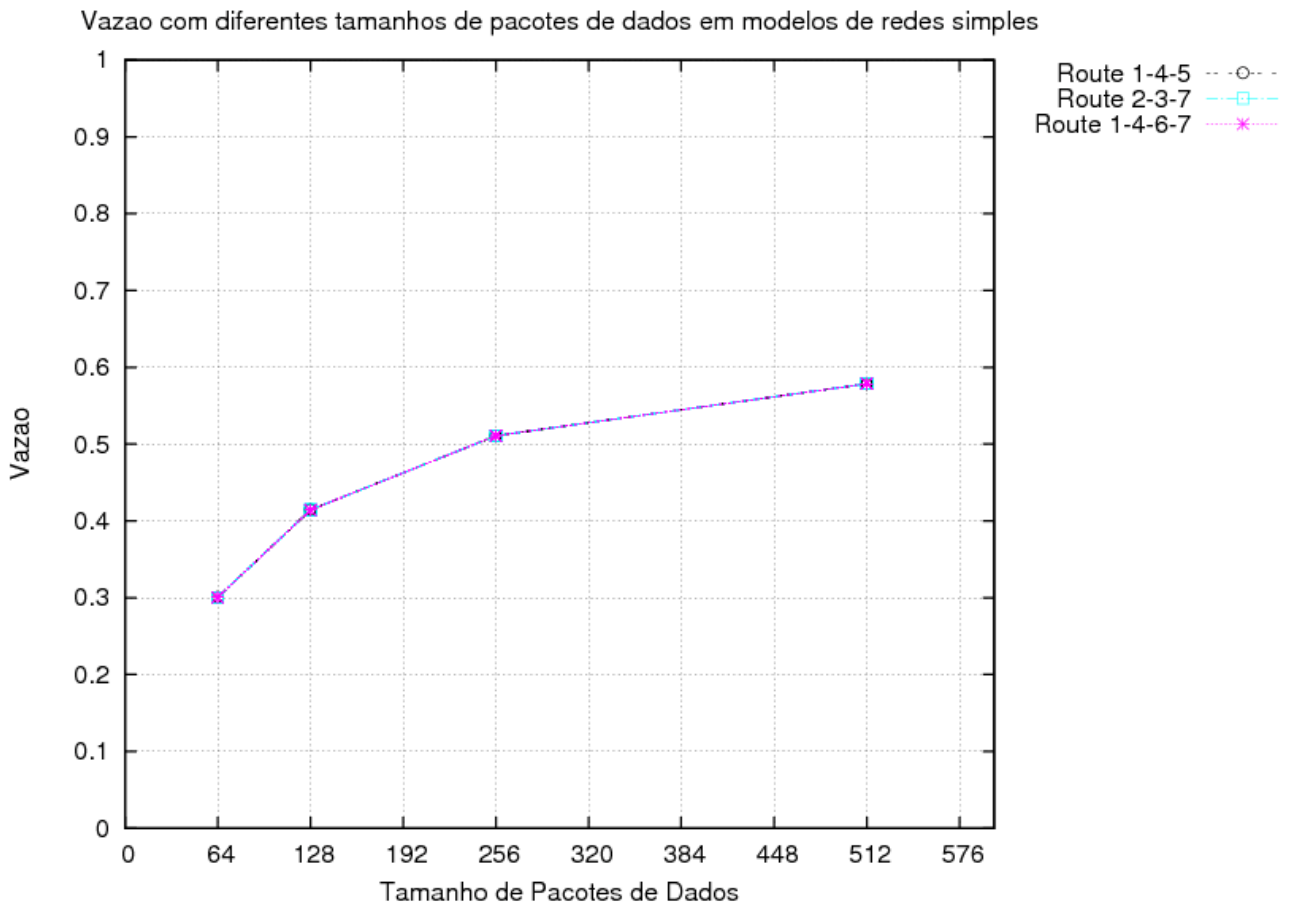


Figura 6.13: Vazão do modelo simples considerando o tamanho de pacote de dados

O gráfico apresenta um comportamento semelhante para todas as rotas, em relação aos pacotes transmitidos, uma vez que no processo de descoberta de rotas apresentou-se o desempenho de tais rotas, todas com o mesmo valor, logo, esta igualdade se repetiu na vazão dos pacotes.

Para que fosse possível obter tais resultados, foram criadas funções de integração que consideravam as variáveis inseridas no modelo, dadas pela seguinte sintaxe:

```
t_AInt_237_64  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) &&
(st      AInt_7      ==      Fw))      *      bandwidth_Mbps)      *
(size_pack_64/(size_pack_64+header_dsr)));
t_AInt_237_128  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) &&
(st      AInt_7      ==      Fw))      *      bandwidth_Mbps)      *
(size_pack_128/(size_pack_128+header_dsr)));
t_AInt_237_256  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) &&
(st      AInt_7      ==      Fw))      *      bandwidth_Mbps)      *
(size_pack_256/(size_pack_256+header_dsr)));
t_AInt_237_512  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) &&
(st      AInt_7      ==      Fw))      *      bandwidth_Mbps)      *
(size_pack_512/(size_pack_512+header_dsr)));
```

onde a expressão

```
((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 == Fw)) * bandwidth_Mbps)
```

representa o produto entre a probabilidade da mensagem ser enviada através da rota 2-3-7 e a largura de banda, e este resultado é multiplicado pela razão dada pela expressão

```
(size_pack_64/(size_pack_64+header_dsr))
```

onde têm-se o tamanho do pacote de dados (size_pack_64) dividido pela soma deste mais o valor correspondente ao tamanho do cabeçalho fixo do protocolo DSR, dado pela expressão:

```
(size_pack_64+header_dsr)
```

Para o modelo com possibilidade de falhas de transmissão, que mostrou no processo de descoberta de rotas um desempenho melhor para as rotas com menos saltos, apresentou também este resultado na vazão dos pacotes, como pode ser visto no gráfico da Figura 6.14.

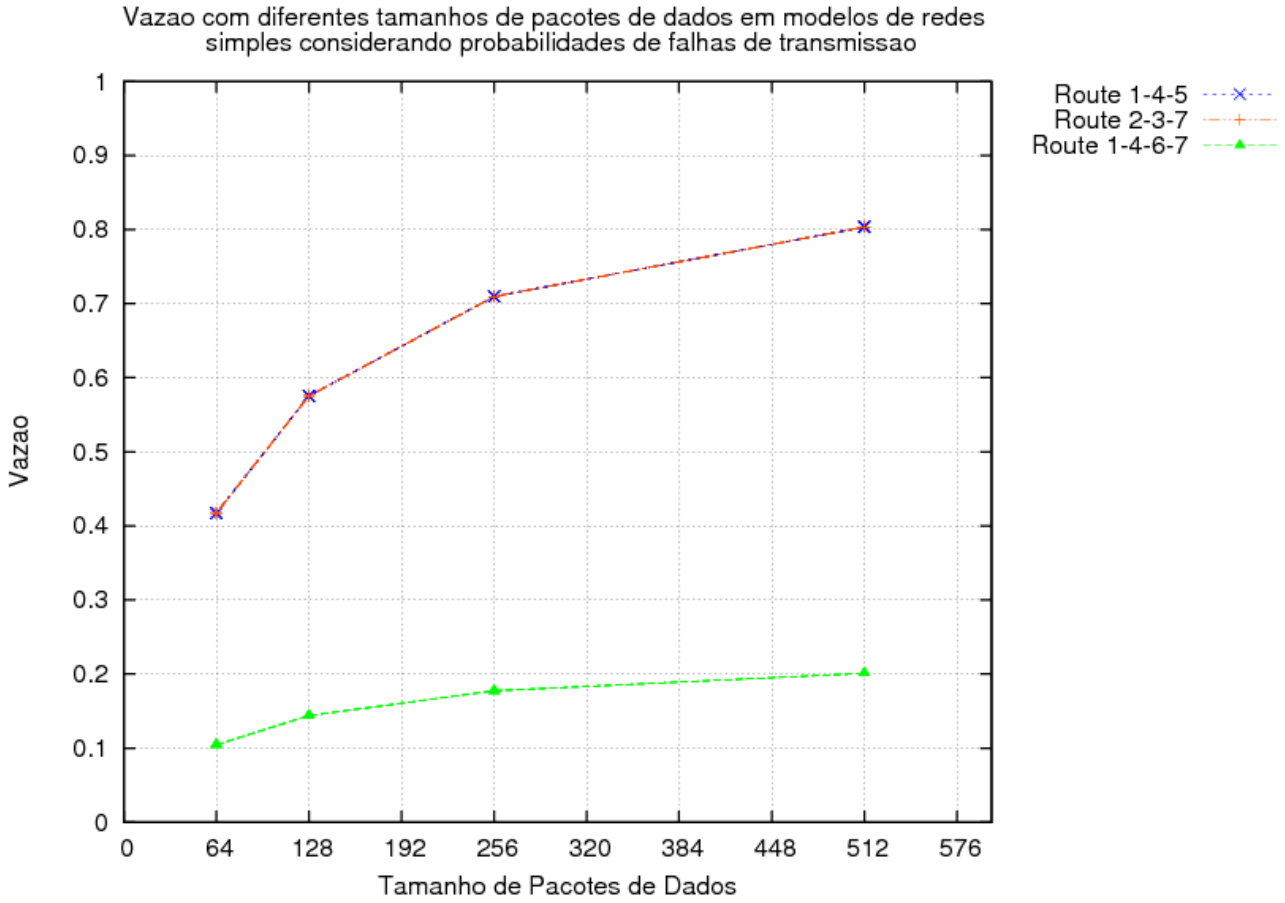


Figura 6.14: Vazão do modelo simples considerando o tamanho de pacotes de dados e possibilidade de falhas na transmissão

Os resultados numéricos para os modelos simples são apresentados na Tabela 6.7, demonstrando as vazões dos dois modelos de redes simples.

Tabela 6.7: Resultados da vazão dos modelos simples considerando tamanho de pacotes de dados

Tamanho de pacotes	**Vazão do modelo simples sem falhas na transmissão (%)**			**Vazão do modelo simples com probabilidade de falhas na transmissão (%)**		
	Route 2/3/7	**Route 1/4/5**	**Route 1/4/6/7**	**Route 2/3/7**	**Route 1/4/5**	**Route 1/4/6/7**
64kB	0.300469	0.300469	0.300469	0.417288	0.417288	0.104322
128kB	0.414239	0.414239	0.414239	0.575290	0.575290	0.143822
256kB	0.510978	0.510978	0.510978	0.709640	0.709640	0.177410
512kB	0.578531	0.578531	0.578531	0.803457	0.803457	0.200864

Para o modelo que realiza a técnica de escuta promíscua (*promiscuous listening*), a vazão obtida tem o mesmo comportamento do modelo simples, agora com um maior número de nodos (14 nodos, sendo 2 nodos fonte, 2 nodos destino e 10 nodos intermediários). Nestes modelos, a vazão não teve o mesmo desempenho em relação ao modelo simples, apresentando uma curva decrescente de desempenho conforme os tamanhos de pacotes foram aumentando, como pode ser visto na Figura 6.15.

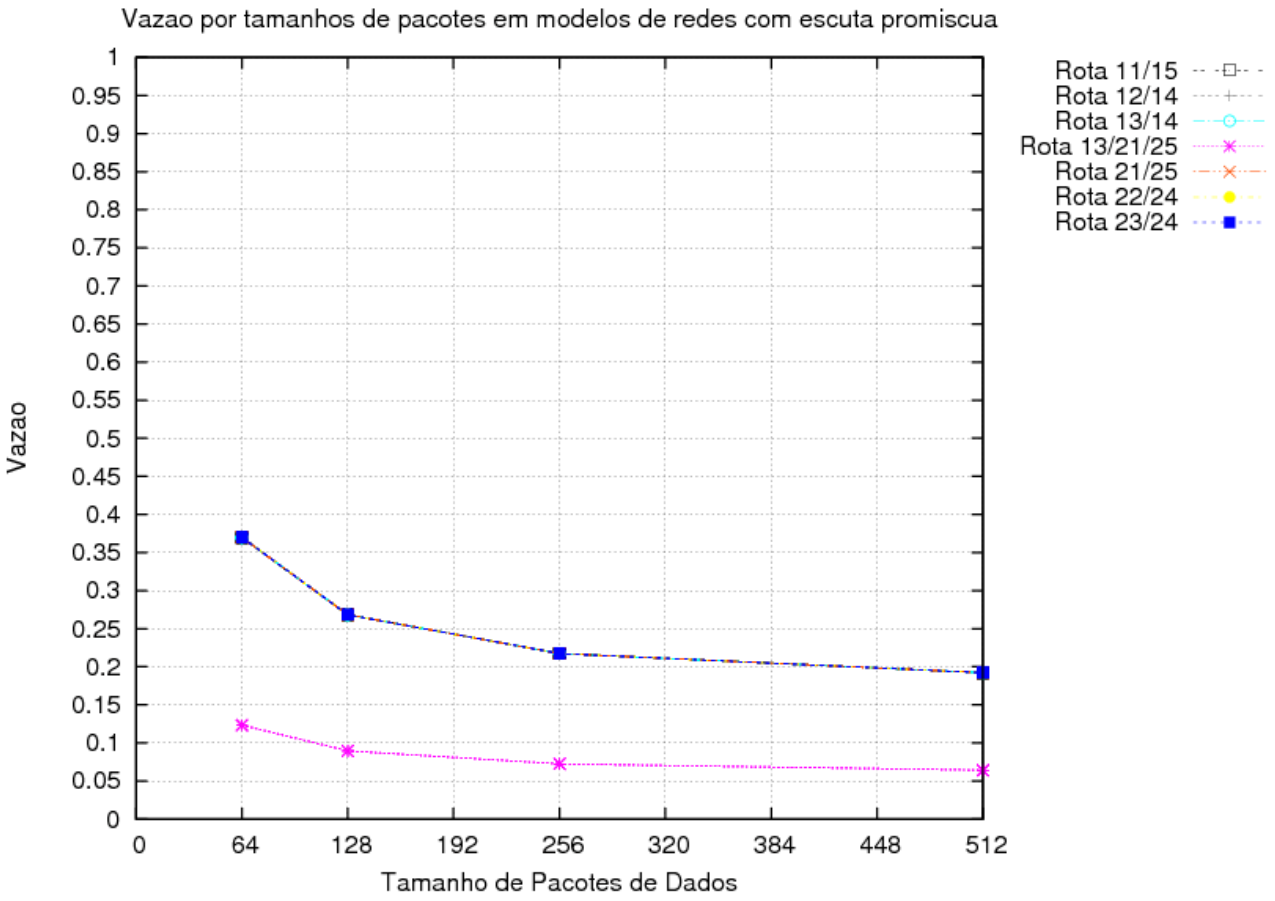


Figura 6.15: Vazão do modelo de escuta promíscua por tamanho de pacotes

A Figura 6.16 apresenta a avaliação de desempenho da vazão no modelo de escuta promíscua que considera falhas na transmissão e possíveis manutenções de rota.

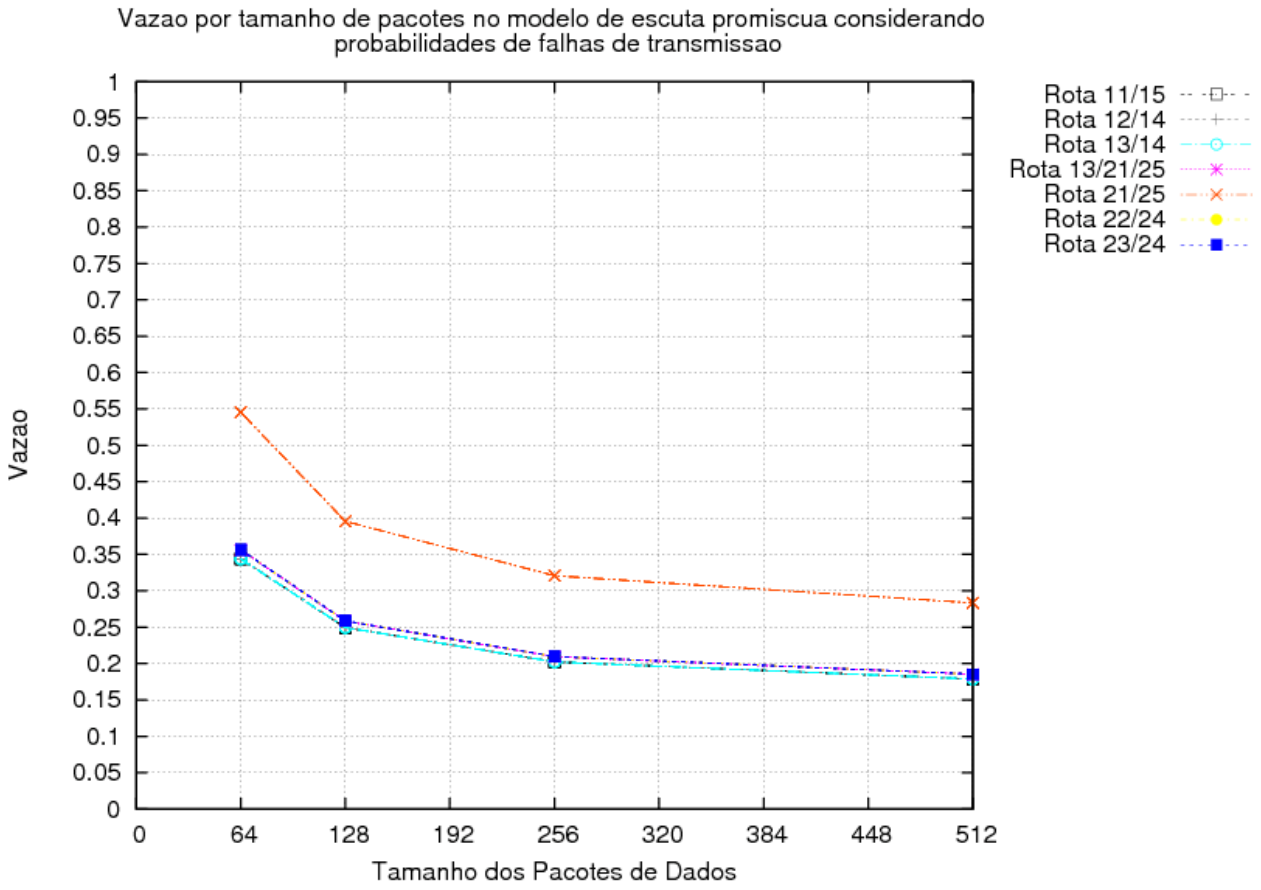


Figura 6.16: Vazão do modelo de escuta promíscua por tamanho de pacotes considerando falhas de transmissão

Neste modelo, apesar de ter comportamento semelhante ao modelo anterior, que descartava as falhas, neste pode-se observar que a vazão da rota que utiliza a escuta promíscua (rota 13/21/25) teve um desempenho muito melhor que no modelo anterior, devido ao fato que os nodos presentes na rede podem não estar disponíveis para realizar a comunicação e transmissão dos dados, fazendo assim com que outras rotas possam ser utilizadas, tornando mais igualitária à forma de requisição de rotas para os nodos. O desempenho numérico dos dois modelos pode ser visto na Tabela 6.8.

Tabela 6.8: Resultados da vazão dos modelos de escuta promíscua considerando tamanho de pacotes de dados

	Vazão do modelo com escuta promíscua sem falhas na transmissão (%)				**Vazão do modelo com escuta promíscua com probabilidade de falhas na transmissão (%)**			
Rotas/Tamanho de Pacotes	**64kB**	**128kB**	**256 kB**	**512kB**	**64kB**	**128kB**	**256 kB**	**512kB**
11/15	0.369791	0.268229	0.217447	0.192057	0.343819	0.249390	0.202175	0.178568
12/14	0.369791	0.268229	0.217447	0.192057	0.343819	0.249390	0.202175	0.178568
13/14	0.369791	0.268229	0.217447	0.192057	0.343819	0.249390	0.202175	0.178568
13/21/25	0.123263	0.089409	0.072482	0.064019	0.356093	0.258293	0.209392	0.184942

Rotas/Tamanho de Pacotes (cont.)	64kB (cont.)	128kB (cont.)	256 kB (cont.)	512kB (cont.)	64kB (cont.)	128kB (cont.)	256 kB (cont.)	512kB (cont.)
21/25	0.369791	0.268229	0.217447	0.192057	0.545295	0.395531	0.320649	0.283208
22/24	0.369791	0.268229	0.217447	0.192057	0.356411	0.258524	0.209580	0.185108
23/24	0.369791	0.268229	0.217447	0.192057	0.356411	0.258524	0.209580	0.185108

Para o modelo com agregação de serviços de transmissão, quando desconsideradas as falhas na transmissão, este apresentou uma vazão inferior ao modelo que aceita que falhas possam ocorrer bem como as rotas que se utilizam da técnica de agregação apresentaram uma vazão mais expressiva que as outras rotas presentes no modelo, contudo, diferentemente do modelo de escuta promíscua, estes modelos tiveram uma vazão crescente em relação aos tamanhos de pacotes. As Figuras 6.17 e 6.18 mostram o desempenho da vazão dos modelos que utilizam agregação de serviços, sendo que o gráfico da Figura 6.17 corresponde ao modelo que desconsidera falhas na transmissão dos pacotes.

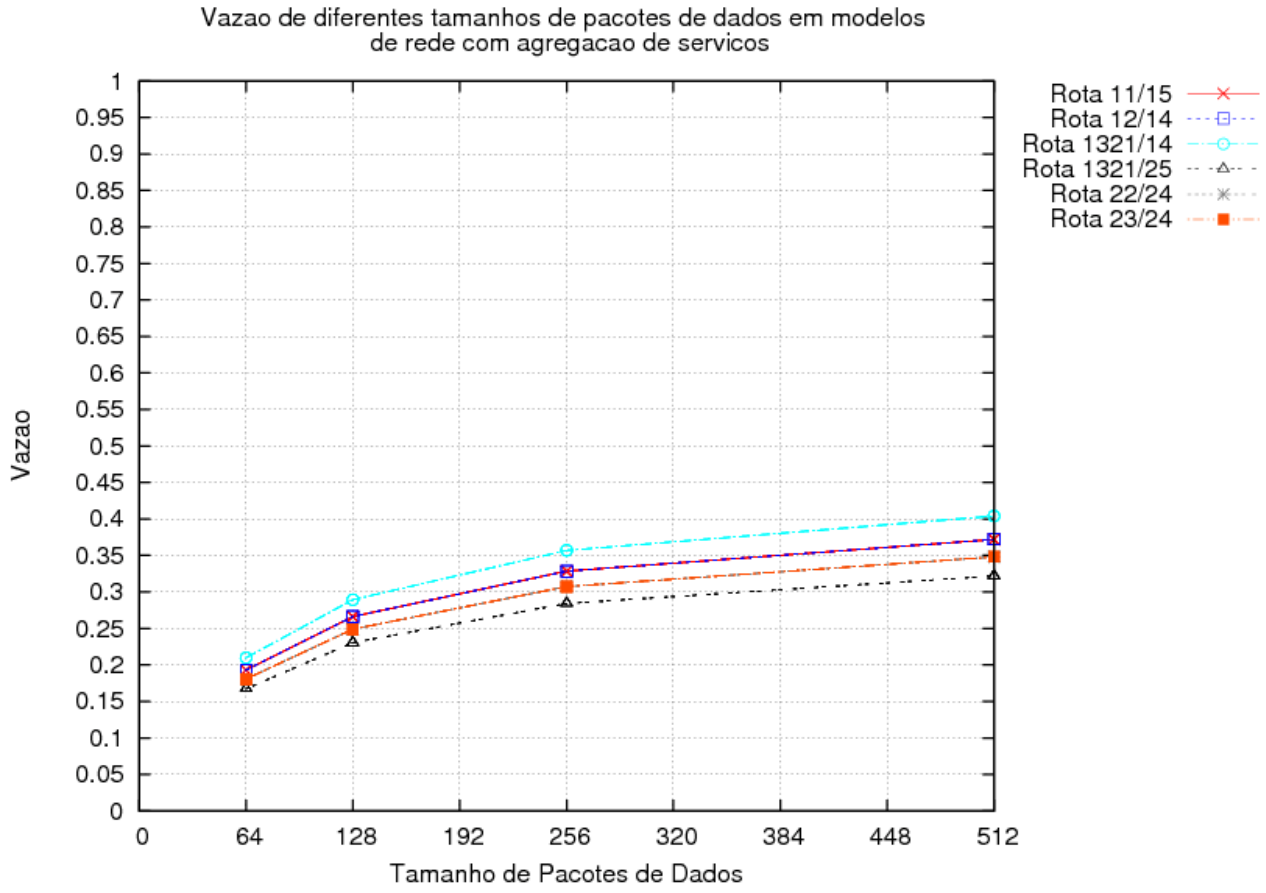


Figura 6.17: Vazão por tamanho de pacotes do modelo de agregação de serviços

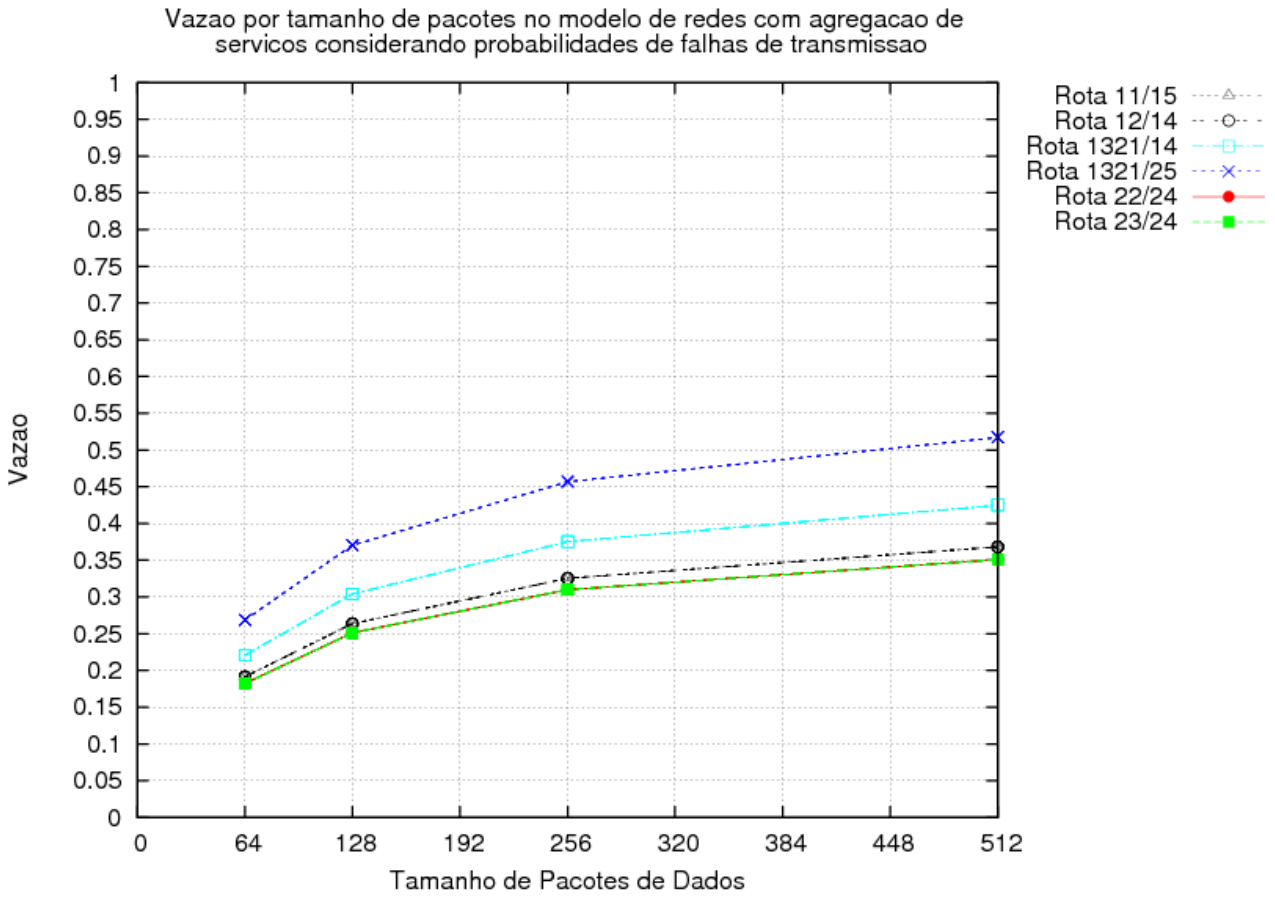


Figura 6.18: Vazão por tamanho de pacotes do modelo de agregação de serviços considerando falhas de transmissão

A Tabela 6.9 mostra os resultados numéricos dos dois modelos que utilizam escuta promíscua na avaliação da vazão por tamanho de pacotes.

Tabela 6.9: Resultados da vazão dos modelos de agregação de serviços considerando tamanho de pacotes de dados

	Vazão do modelo com agregação de serviços sem falhas na transmissão (%)				**Vazão do modelo com agregação de serviços com probabilidade de falhas na transmissão (%)**			
Rotas/Tamanho de Pacotes	**64kB**	**128kB**	**256 kB**	**512kB**	**64kB**	**128kB**	**256 kB**	**512kB**
11/15	0.193171	0.266313	0.328506	0.371936	0.191267	0.263689	0.325269	0.368271
12/14	0.193171	0.266313	0.328506	0.371936	0.191267	0.263689	0.325269	0.368271
1321/14	0.209854	0.289313	0.356877	0.404057	0.220670	0.304225	0.375271	0.424883
1321/25	0.167179	0.230480	0.284305	0.321891	0.268643	0.370362	0.456854	0.517252
22/24	0.180703	0.249125	0.307303	0.347930	0.182251	0.251258	0.309935	0.350910
23/24	0.180703	0.249125	0.307303	0.347930	0.182251	0.251258	0.309935	0.350910

6.2.2 Vazão por tempo de pausa

O uso do tempo de pausa na modelagem para a obtenção de índices de avaliação da vazão da rede tem por objetivo avaliar os nodos quando da sua movimentação dentro da área da rede, e assim, quando estas pausas acontecem em intervalos de tempos predefinidos, obtêm-se valores para a avaliação entre um intervalo e outro, onde os nodos permanecem estáticos e voltam a transmitir após este intervalo, pois, segundo [Bou04], variando-se o tempo de pausa dentro de uma rede, muda-se também a frequência de movimentação dos nodos, onde, segundo [Bro98], quando se tem um tempo de pausa equivalente a zero, existe uma movimentação constante dentro da rede, e quanto mais tal tempo de pausa é incrementado, menor será a movimentação dentro da rede.

Para que este índice pudesse ser avaliado, foram levadas em consideração algumas técnicas, como o padrão de mobilidade, sendo utilizado o padrão *random waypoint* (RWP), que foi utilizado por Johnson *et al.* em [Joh96a] na avaliação do próprio protocolo DSR, uma vez que este padrão é largamente usado na avaliação de redes *Ad hoc*, e tem como premissa o fato que realiza uma escolha aleatória de destinos no momento da transmissão de dados, configurando um processo que atua em conjunto ao processo de inundação de rede (*flooding*) usado no processo de descoberta de rotas, que considera os nodos ao alcance da transmissão, aleatoriamente.

Outro fato relevante ao uso do padrão RWP diz respeito à forma como o nodo se comporta quando este padrão é utilizado, seguindo os seguintes passos, apresentados por Delamare em [Del06]:

• O nodo escolhe (ou tem este predeterminado, assim como no protocolo DSR) o seu destino dentro da área de movimentação;

• Este então move-se até o destino com uma velocidade constante;

• Ao alcançar o destino, o nodo fica parado por um determinado tempo de pausa;

• Este então reinicia o processo acima descrito.

Conforme Broch *et al.* em [Bro98], o tempo de pausa representa a movimentação dos nodos dentro do espaço de alcance da rede. Quanto menor o tempo de pausa, maior a mobilidade dos nodos, assim como quanto maior o tempo de pausa, menor a mobilidade. Nos modelos gerados neste trabalho, os tempos de pausa foram gerados em nível de proporção ao número de nodos,

sendo analisados de acordo com os trabalhos relacionados, e proporcionalmente distribuídos a esses nodos.

Na modelagem da relação vazão por tempo de pausa, foram utilizados valores de tempo de pausa que variaram entre 0 (zero) segundos, indicando uma movimentação constante dos nodos durante a transmissão, até o intervalo de tempo de 126 segundos (sendo o último tempo de pausa admitidos para os nodos), que representa uma movimentação nula dos nodos.

Quando inserido o tempo de pausa para o modelo simples, observou-se uma vazão exponencial para todas as rotas presentes no modelo, confirmando a afirmativa de Layun *et al.* [Lay07], onde os autores dizem que em redes pequenas, o número de nodos determina uma maior ou menor vazão, o que pode ser observado nos modelos avaliados, uma vez que o número de nodos para todos os modelos é considerado pequeno (entre sete e quinze nodos), e o tempo de pausa dos nodos, que representa a movimentação dos mesmos (quanto menor o tempo de pausa, mais movimentação tem-se na rede e, quanto maior, mais estática a rede se encontra, conforme citam [Bro98, Lay07, Puc07, Per01]), mostra que com os caminhos definidos e uma menor movimentação dos nodos, a vazão tende a ser maior para todas as rotas.

Para se obter tal índice, levou-se em consideração a largura de banda da rede (assumida como sendo 2Mbps constantes, e identificada na sintaxe como *bandwidth_Mbps*), a probabilidade da rota estar transmitindo (considerando os estados dos nodos, neste caso, os nodos *AInt_1321* e *Aint_25*, utilizando a expressão "*((st AInt_1321 == Fw) && (st AInt_25 == Fw))*"), bem como o tamanho do pacote de dados (para todos os modelos foi considerado um tamanho de pacotes de 512 bytes, neste exemplo representado por size_pack_512, ou seja, um pacote de 512 bytes) e do cabeçalho do protocolo (*header_dsr*).

A sintaxe da função de integração gerada para a obtenção da vazão, agora considerando o tempo de pausa durante as transmissões é a seguinte:

```
t_AInt_132125_64 = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw))
*                          bandwidth_Mbps)                        *
(size_pack_64/(size_pack_64+header_dsr)))/tx_tpause;
```

Como pode ser observada, a sintaxe é semelhante àquela apresentada para a vazão por tamanho de pacotes, havendo agora a razão entre a expressão e o tempo de pausa, representada pelo parâmetro *tx_tpause*.

As Figuras 6.19 e 6.20 mostram o desempenho da vazão do modelo simples quando considerados os tempos de pausa dos nodos durante a transmissão dos pacotes.

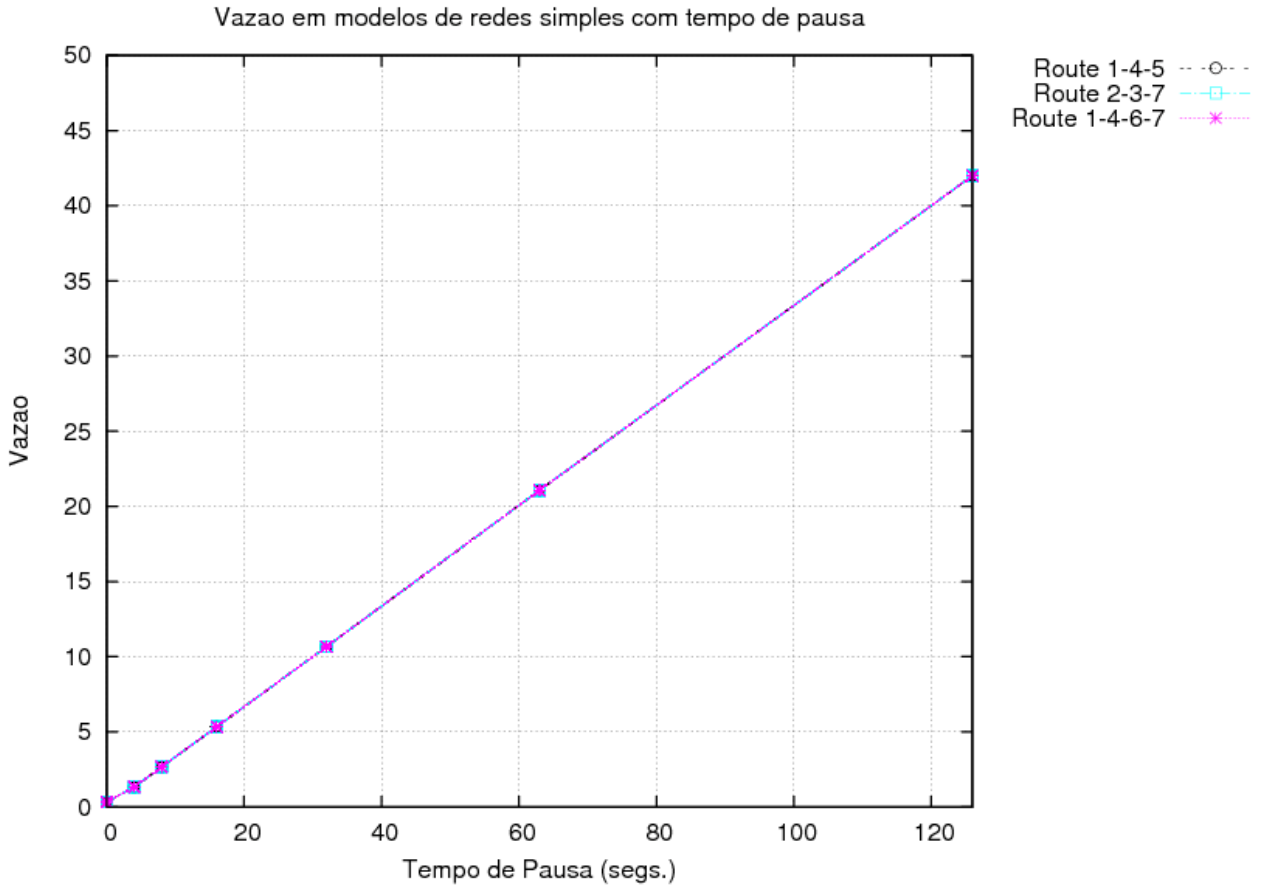


Figura 6.19: Vazão do modelo simples com tempo de pausa

O gráfico mostrado na Figura 6.20 apresenta o resultado do modelo que considera também a probabilidade de ocorrem falhas na transmissão, e com isso a manutenção de rota ser realizada, bem como novos processos de descoberta de rotas.

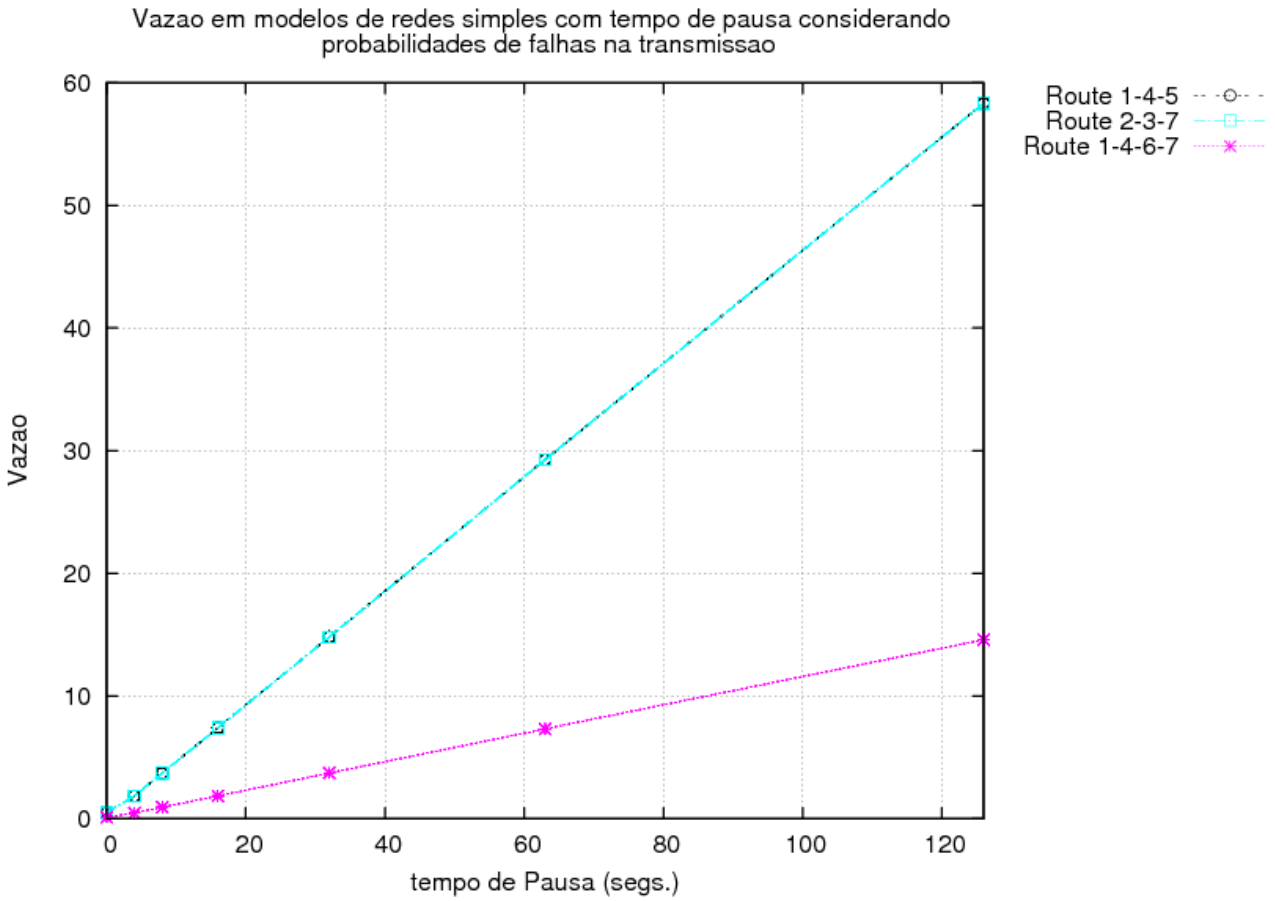


Figura 6.20: Vazão do modelo simples com tempo de pausa, considerando falhas na transmissão

Nestes modelos, os tempos de pausa utilizados foram de 4, 8, 16, 32, 63 e 126 segundos, buscando uma proporcionalidade entre a quantidade de nodos utilizados nos modelos deste trabalho, com aqueles apresentados em avaliações usando simulação como formalismo na obtenção de resultados.

No modelo que considera falhas de transmissão, pode-se observar que a rota que apresenta um maior número de saltos (rota 1/4/6/7) tem uma vazão menor em relação às outras.

Esse desempenho inferior para a rota com mais saltos entre o nodo fonte e o nodo destino está diretamente ligado ao baixo *workload* dos nodos presentes nesta rota, apresentados no início deste capítulo.

Os resultados de avaliação da vazão com tempo de pausa são apresentados na Tabela 6.10.

Tabela 6.10: Resultados da vazão dos modelos simples, considerando tempo de pausa

Tempo de pausa (segs.)	Vazão do modelo simples com tempo de pausa sem falhas na transmissão (%)			Vazão do modelo simples com tempo de pausa e probabilidade de falhas na transmissão (%)		
	Route 2/3/7	Route 1/4/5	Route 1/4/6/7	Route 2/3/7	Route 1/4/5	Route 1/4/6/7
0	0.333333	0.333333	0.333333	0.462929	0.462929	0.115732
4	1.333333	1.333333	1.333333	1.851717	1.851717	0.462929
8	2.666666	2.666666	2.666666	3.703435	3.703435	0.925858
16	5.333333	5.333333	5.333333	7.406871	7.406871	1.851717
32	10.666666	10.666666	10.666666	14.813743	14.813743	3.703435
63	21.070307	21.070307	21.070307	29.262293	29.262293	7.315573
126	42.002688	42.002688	42.002688	58.332847	58.332847	14.583211

Inserindo tempo de pausa nos modelos de escuta promíscua e agregação de serviços, estes tiveram um comportamento semelhante ao modelo simples, tendo uma vazão crescente e constante conforme o tempo de pausa era incrementado, o que indica que quanto menor a movimentação dos nodos dentro da rede, maior a vazão atingida. A Figura 6.21 mostra o gráfico de desempenho de vazão do modelo de escuta promíscua desconsiderando falhas na transmissão.

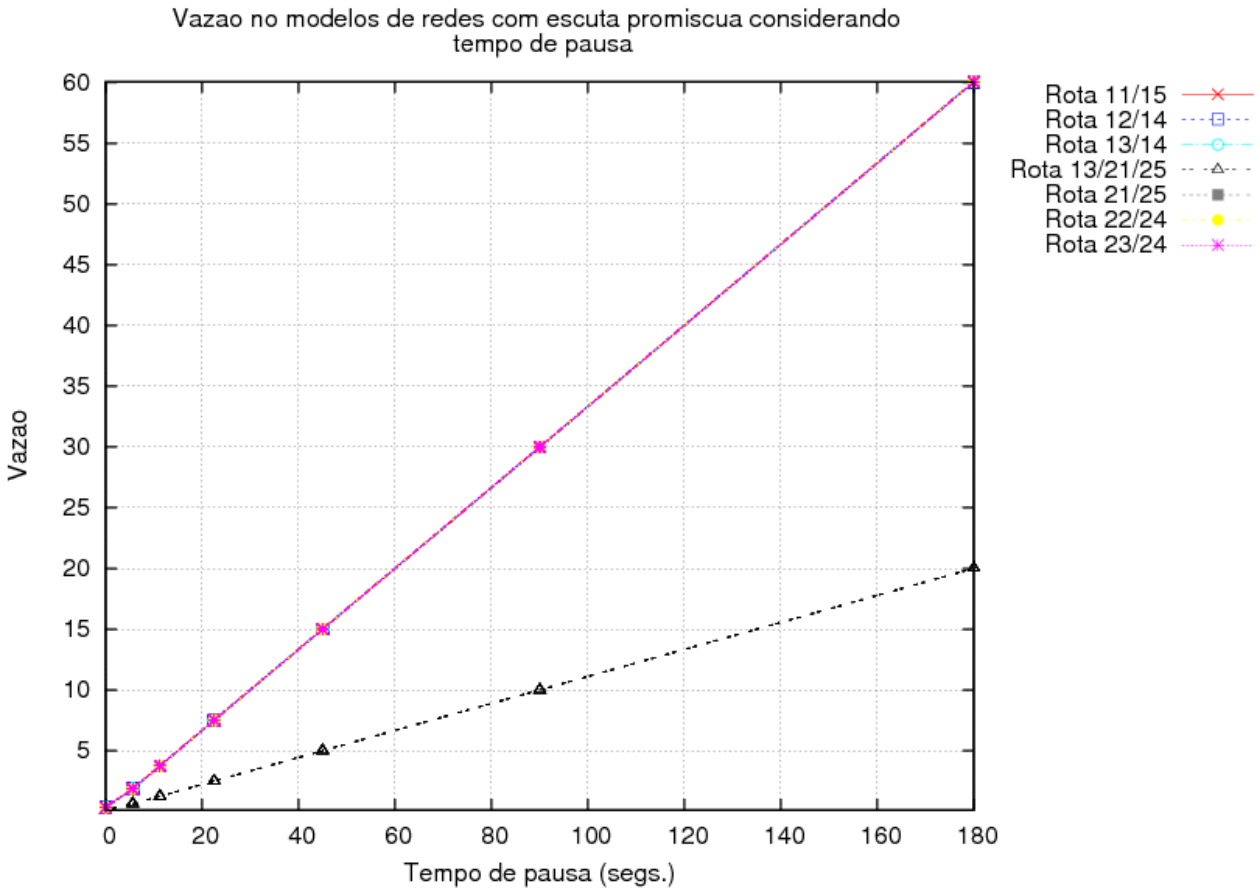


Figura 6.21: Vazão do modelo de escuta promíscua com tempo de pausa

No gráfico de desempenho, a rota que realiza a escuta promíscua apresentou um desempenho inferior às outras rotas, não mostrando a eficiência esperada neste tipo de técnica. Contudo, quando o modelo que tem a função de atingibilidade definida para possibilitar que falhas na transmissão ocorram, como mostrado no Capítulo 5, fazendo que novas descobertas de rotas sejam geradas, o caminho percorrido através da rota de escuta promíscua apresenta um desempenho semelhante às outras rotas, tornando assim o processo útil dentro da rede. A Figura 6.22 apresenta os resultados da avaliação da vazão para este modelo.

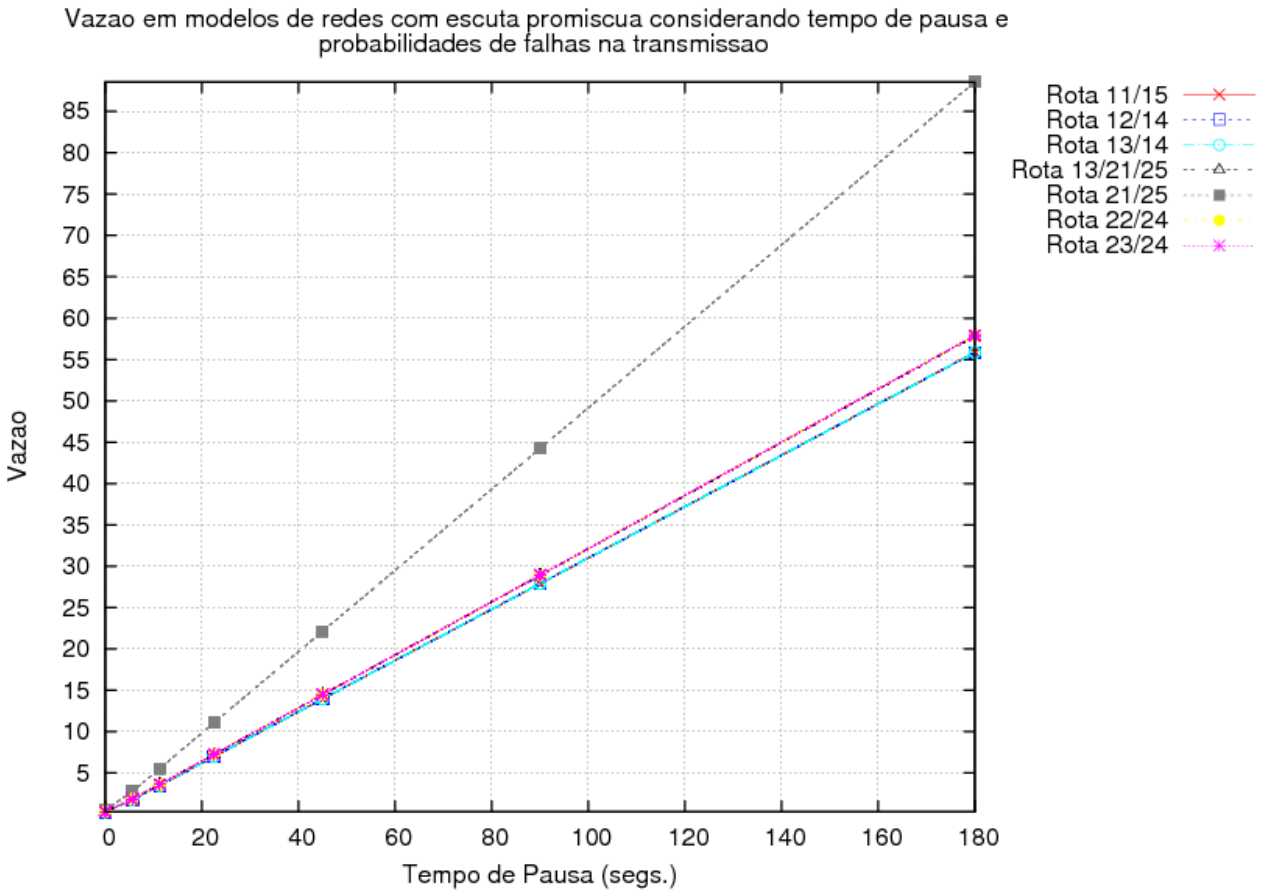


Figura 6.22: Vazão do modelo de escuta promíscua com tempo de pausa, considerando falhas na transmissão

Nas Tabelas 6.11 e 6.12 são apresentados os resultados numéricos dos modelos de escuta promíscua, sendo respectivamente mostrados os resultados do modelo que desconsidera falhas e transmissão e os resultados do modelo passível de falhas de transmissão.

Tabela 6.11: Resultados da vazão dos modelos de escuta promíscua considerando tempo de pausa

Tempo de pausa (segs.)	**Vazão do modelo simples com escuta promíscua com tempo de pausa e sem falhas na transmissão (%)**						
	Route 11/15	**Route 12/14**	**Route 13/14**	**Route 13/21/25**	**Route 21/25**	**Route 22/24**	**Route 23/24**
0	0.333333	0.333333	0.333333	0.111111	0.333333	0.333333	0.333333
5.625	1.875082	1.875082	1.875082	0.625027	1.875082	1.875082	1.875082
11.25	3.753753	3.753753	3.753753	1.251251	3.753753	3.753753	3.753753
22.5	7.507507	7.507507	7.507507	2.502502	7.507507	7.507507	7.507507

Tempo de pausa (segs.) (cont.)	Route 11/15 (cont.)	Route 12/14 (cont.)	Route 13/14 (cont.)	Route 13/21/25 (cont.)	Route 21/25 (cont.)	Route 22/24 (cont.)	Route 23/24 (cont.)
45	15.015014	15.015014	15.015014	5.005005	15.015015	15.015014	15.015014
90	30.002999	30.002999	30.002999	10.001001	30.003001	30.002999	30.002999
180	60.060059	60.060059	60.060059	20.002002	60.060061	60.060059	60.060059

Tabela 6.12: Resultados da vazão dos modelos de escuta promíscua considerando tempo de pausa e falhas na transmissão

Tempo de pausa (segs.)	Vazão do modelo simples com escuta promíscua com tempo de pausa e considerando probabilidades de falhas na transmissão (%)						
	Route 11/15	Route 12/14	Route 13/14	Route 13/21/25	Route 21/25	Route 22/24	Route 23/24
0	0.309921	0.309921	0.309921	0.320985	0.491534	0.321272	0.321272
5.625	1.744073	1.744073	1.744073	1.806334	2.766090	1.807949	1.807949
11.25	3.490111	3.490111	3.490111	3.614702	5.535296	3.617935	3.617935
22.5	6.980223	6.980223	6.980223	7.229405	11.070593	7.235871	7.235871
45	13.960446	13.960446	13.960446	14.458811	22.141187	14.471742	14.471742
90	27.895761	27.895761	27.895761	28.891594	44.242516	28.917434	28.917434
180	55.841786	55.841786	55.841786	57.835245	88.564749	57.886971	57.886971

Quando avaliados os modelos que realizam agregação de serviços de transmissão, as rotas que passam pelo nodo *AInt_1321*, que é responsável pelo serviço de agregar as tarefas de transmissão, apresentaram as maiores vazões quando falhas na transmissão foram consideradas. As Figuras 6.23 e 6.24 mostram o desempenho da vazão nestes modelos.

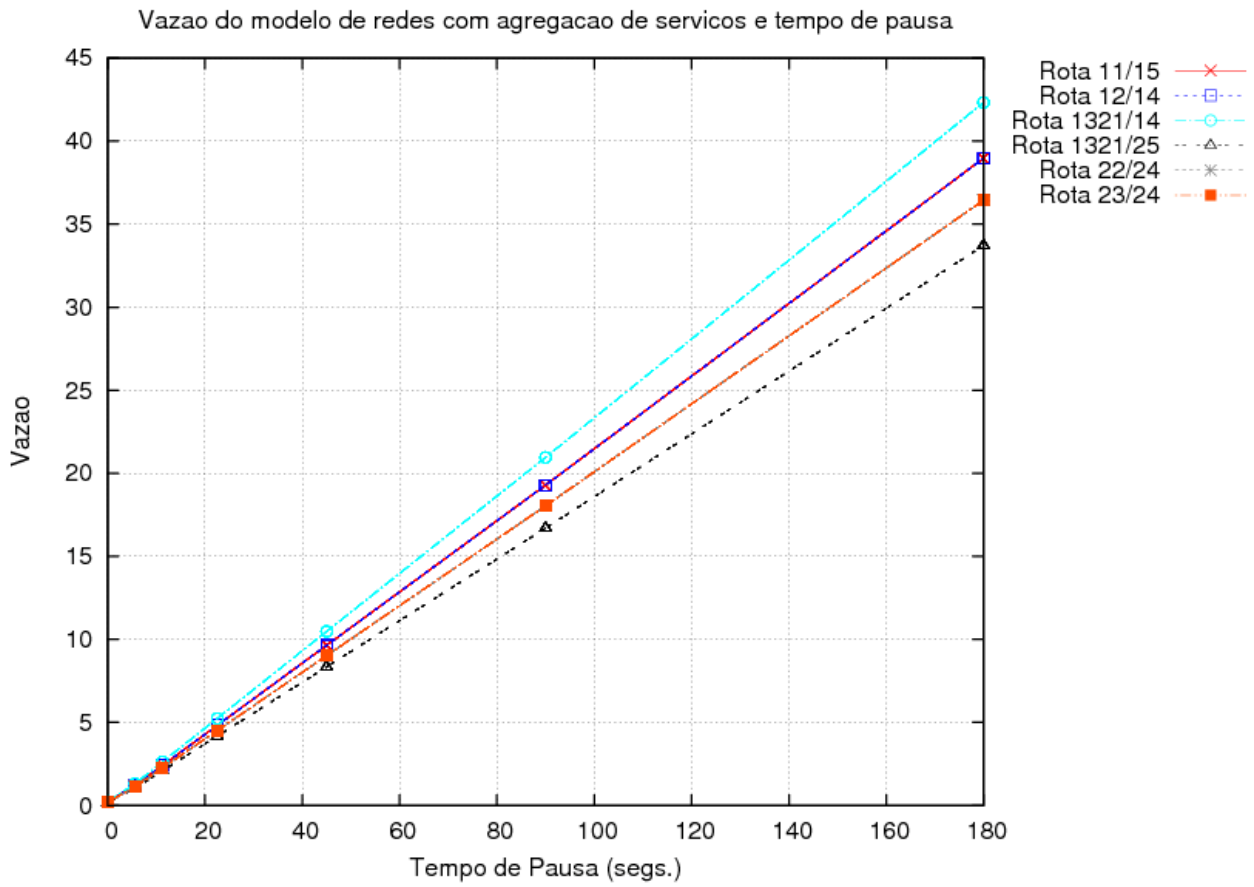


Figura 6.23: Vazão do modelo de agregação de serviços com tempo de pausa

Na Figura 6.23, apesar da rota 1321/14 obter a maior vazão, a rota 1321/25 obteve a menor delas, mostrando-se assim insatisfatória em relação às outras rotas que não utilizam técnica alguma, entretanto, quando as falhas de transmissão são consideradas, estas rotas que apresentam a agregação dos serviços de transmissão se mostram efetivamente mais eficientes que as outras rotas, como pode ser observado no gráfico da Figura 6.24.

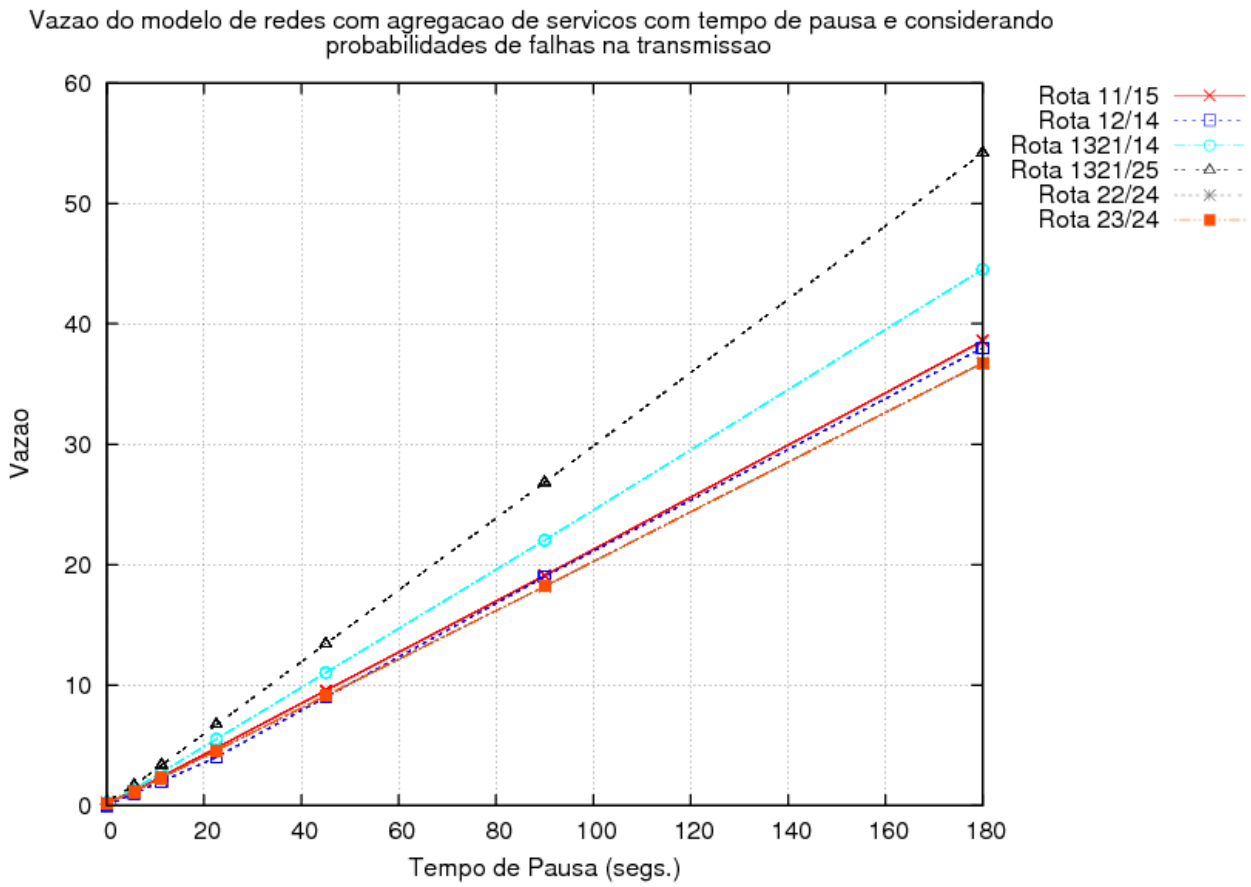


Figura 6.24: Vazão do modelo de agregação de serviços com tempo de pausa, considerando falhas na transmissão

Os resultados numéricos mostrados nas Tabelas 6.13 e 6.14 comprovam numericamente a eficiência das rotas que utilizaram de agregação de serviços, em especial quando as possíveis falhas de transmissão possam vir a ocorrer dentro da rede.

Tabela 6.13 Resultados da vazão dos modelos de agregação de serviços considerando tempo de pausa

Tempo de pausa (segs.)	**Vazão do modelo com agregação de serviços e tempo de pausa sem falhas na transmissão (%)**					
	Route 11/15	**Route 12/14**	**Route 1321/14**	**Route 1321/25**	**Route 22/24**	**Route 23/24**
0	0.214299	0.214299	0.232806	0.185464	0.200467	0.200467
5.625	1.205961	1.205961	1.310111	1.043695	1.128125	1.128125
11.25	2.413280	2.413280	2.621698	2.088567	2.257520	2.257520
22.5	4.822217	4.822217	5.238676	4.173374	4.510977	4.510977
45	9.644434	9.644434	10.477353	8.346748	9.021954	9.021954
90	19.288869	19.288869	20.954706	16.693497	18.043908	18.043908

Tempo de pausa (segs.)	Route 11/15	Route 12/14	Route 1321/14	Route 1321/25	Route 22/24	Route 23/24
180	38.963515	38.963515	42.328506	33.720865	36.448694	36.448694

Tabela 6.14 Resultados da vazão dos modelos de agregação de serviços considerando tempo de pausa

Tempo de pausa (segs.)	Vazão do modelo com agregação de serviços e tempo de pausa considerando probabilidades de falhas na transmissão (%)						
	Route 11/15	Route 12/14	Route 1321/14	Route 1321/25	Route 22/24	Route 23/24	
0	0.212187	0.212187	0.244806	0.298026	0.202184	0.202184	
5.625	1.194078	1.194078	1.377636	1.677131	1.137786	1.137786	
11.25	2.387350	2.387350	2.754343	3.353131	2.274805	2.274805	
22.5	4.779002	4.779002	5.513650	6.712305	4.553710	4.553710	
45	9.549402	9.549402	11.017374	13.412526	9.099223	9.099223	
90	19.098805	19.098805	22.034749	26.825053	18.198446	18.198446	
180	38.579587	38.579587	44.510194	54.186608	36.760861	36.760861	

Na literatura, devido ao fato que os intervalos de tempo de pausa são maiores que nos modelos gerados para a avaliação em SAN, em decorrência da quantidade de nodos utilizados nas topologias de simulação, estes trabalhos iniciavam o primeiro tempo de pausa dos nodos em 100 segundos, como mostrado no gráfico da Figura 6.25, extraído de [Bou04], onde o desempenho do protocolo DSR foi grifado para uma melhor compreensão.

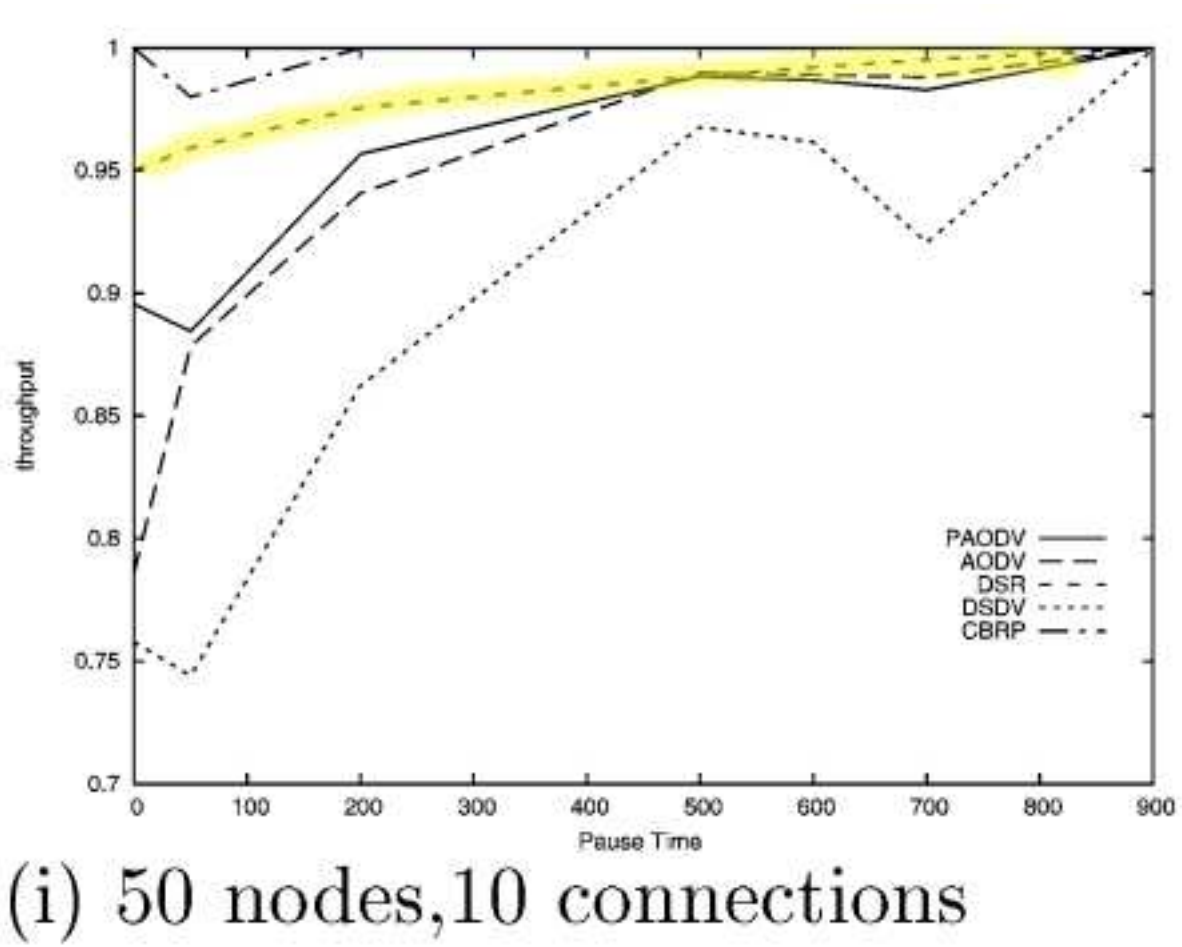


Figura 6.25: Vazão do protocolo DSR extraído de [Bou04]

No trabalho de [Per01], que utilizou o termo "*packet delivery fraction*", já que os autores consideraram as perdas na transmissão dos pacotes, e também utilizaram pontos de avaliação dentro do tempo de pausa menor que [Bou04], se aproximam mais dos resultados obtidos com os modelos gerados em SAN, apresentados anteriormente, com uma variação em relação aos resultados obtidos neste trabalho, pois a vazão inicialmente decresce para então entrar em uma curva ascendente. A Figura 6.26 mostra os resultados extraídos no trabalho dos referidos autores.

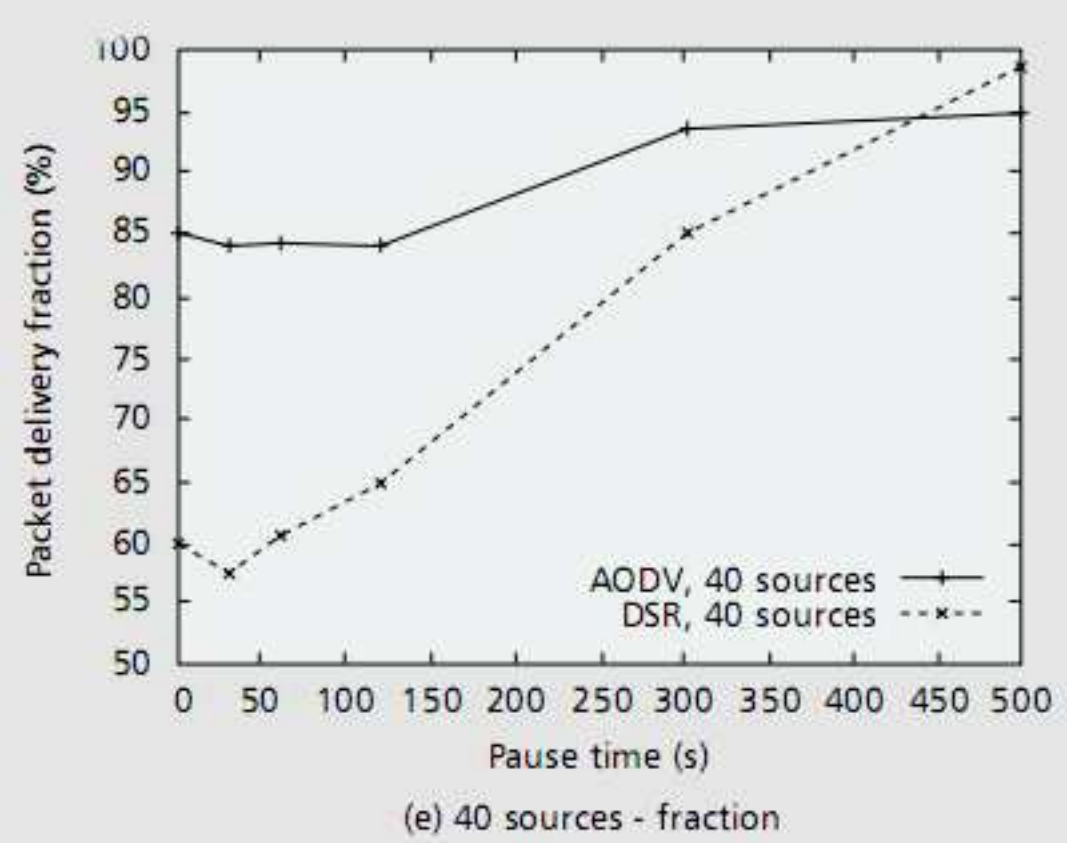


Figura 6.26: Vazão do protocolo DSR extraído de [Per01]

7 CONCLUSÕES E TRABALHOS FUTUROS

A avaliação de desempenho de redes *Ad hoc*, em especial de protocolos de roteamento, comumente realizadas por meio de simulação exige, entre outros fatores, o uso de um grande número de nodos, bem como um grande número de parâmetros e variáveis, fazendo com que esta avaliação por vezes se torne onerosa à avaliação para que se obtenham resultados confiáveis. O uso de um formalismo analítico estruturado neste trabalho, que foram as redes de autômatos estocásticos, mostrou-se eficiente na obtenção de resultados quando estes são comparados àqueles obtidos com simulação. Para avaliar o protocolo de roteamento DSR utilizando redes de autômatos estocásticos, foi necessário um entendimento do seu funcionamento para que assim, fosse possível adequar suas características principais à forma de avaliação que é imposta a SAN.

A geração dos parâmetros do protocolo DSR voltados para a aplicação em SAN, de forma a poder realizar sua modelagem, bem como sua avaliação, podem ser considerados como uma das maiores dificuldades encontradas na construção deste trabalho, pois, transportar um ambiente de rede, analisar quais seriam as variáveis inseridas, bem como as abstrações admitidas para alguns destes parâmetros apresentaram um elevado grau de complexidade, para que estes pudessem ser adequados as redes criadas.

A partir de tal modelagem, a primeira contribuição presente nesta dissertação foi a análise do processo de descoberta de rotas do protocolo DSR, uma vez que na literatura, este processo é tratado de forma trivial, sendo elencado apenas teoricamente os passos para que este processo aconteça, porém, o desempenho de como as rotas se formam, e a sua utilização para determinar como a melhor rota é escolhida, não é analisado, e neste trabalho isto foi possível através do uso do formalismo SAN.

Ainda no processo de descoberta de rotas, é possível também a presença de falhas na transmissão dos pacotes de descoberta de rota e assim, uma vez que estes ocorram, podem gerar também o processo de manutenção de rota e ainda assim prover resultados para demonstrar como esses processos efetivam-se dentro do protocolo e como as rotas são determinadas.

Ao avaliar o protocolo DSR, buscou-se elencar algumas características passíveis de modelagem, sendo a escuta promíscua a característica escolhida para ser utilizada na modelagem em SAN, pois esta apresenta especificidades relacionadas a processos de transmissão paralela, que podem ser adequadas ao formalismo SAN na obtenção de resultados que colaborassem para se avaliar o protocolo DSR em diversos índices de desempenho.

Os resultados para *workload* dos nodos apresentados foram satisfatórios o suficiente para que se tenha o entendimento de que as melhores rotas, dentro das redes, são determinadas pelo *workload* dos nodos, ou seja, quanto mais ativos dentro da rede e quanto maior a carga de roteamento que estes nodos venham a possuir, estes irão determinar a melhor rota para que os pacotes sejam transmitidos.

É importante ressaltar que com isso, pode-se ter o problema de afunilamento ou "gargalo" de dados, ou seja, se um nodo for responsável por muitas transmissões, este pode ter perdas parciais ou até mesmo globais de pacotes e gerar problemas de transmissão.

Já os resultados obtidos para índices como vazão por tamanho de pacotes e por inclusão de tempo de pausa dos nodos dentro da rede mostraram que, quando comparados com resultados já apresentados na literatura, e mesmo se valendo de uma quantidade menor de nodos, é possível extrair resultados que são passíveis de comparação com outros formalismos.

Assim como na literatura, foram inseridos vários tempos de pausa para simular a movimentação dos nodos dentro da rede, apesar deste não ser o intuito principal do trabalho, mas uma vez que eram necessários os resultados do comportamento do protocolo DSR diante da possibilidade de movimentação dos nodos, bem como a probabilidade destes estarem ou não presentes durante a transmissão dos pacotes, a movimentação foi inserida no contexto da avaliação e os modelos SAN mostraram-se eficazes com mais essa variável em sua estrutura, mesmo tendo-se observado que na vazão dos pacotes, estes apresentaram um comportamento crescente mais constante que nos modelos de simulação, mas que mesmo assim possibilitam que a avaliação de tal índice seja possível, demonstrando a eficiência do formalismo.

Na introdução deste trabalho, foi colocado como limitação para a modelagem de protocolos de roteamento de redes de computadores sem fio *Ad hoc* o fato da explosão de estados em outros formalismos analíticos como Cadeias de Markov, por exemplo, sendo que com as redes de autômatos estocásticos, foi possível realizar a avaliação de forma eficiente, contudo, para obter resultados com redes maiores, que possuam um número maior de nodos ou uma complexidade maior de processos, seria necessário buscar alternativas em novos algoritmos de processamento da ferramenta PEPS, ou então em formalismos mais robustos, como, por exemplo, simulação perfeita, a fim de possibilitar mais variações dentro da rede, como a interferência na transmissão, ou a inserção de mais variáveis associadas ao protocolo, como características específicas do processo de manutenção de rotas.

Por fim, as sugestões de trabalhos futuros remetem à análise de características mais complexas do protocolo de roteamento DSR, tais como as respostas para requisições de rede utilizando rotas armazenadas em *cache*, ou seja, já tendo as informações da rota, de que maneira a transmissão dos dados poderia ser mais eficiente, bem como buscar soluções para problemas de roteamento do protocolo como, por exemplo, quando existem terminais escondidos dentro da rede.

Outra proposta para trabalhos futuros pode ser sugerida no que tange as otimizações do protocolo DSR, uma vez que o mesmo fora implementado para atender à novas demandas de redes *Ad hoc* mais robustas, e que utilizam de mecanismos mais concisos de segurança, que não foram elencados neste trabalho, assim como a avaliação de outros protocolos, para que seja possível realizar a comparação de protocolos assim como em simulação, contudo, utilizando SAN como formalismo para avaliação. Ainda em se tratando de trabalhos futuros, com a possibilidade de poder aumentar os modelos de forma à utilizar técnicas analíticas e algoritmos diferentes dos disponíveis atualmente para verificar a eficiência dos mesmos em relação à obtenção de resultados.

A avaliação de desempenho de redes de computadores é uma área em franca expansão, e que apresenta uma gama diversificada de fatores a serem considerados ainda não explorados, para a melhoria quantitativa e qualitativa das informações, na busca da eficiência na tarefa de interligar dispositivos e compartilhar informações, e diante deste panorama, apresentar novas formas de se analisar e avaliar tais redes faz com que novas possibilidades e até mesmo novas formas de entendimento destas redes sejam possíveis.

REFERÊNCIAS BIBLIOGRÁFICAS

[Adi08] Adibi, S.; Agnew, G.B. “Multilayer flavoured dynamic source routing in mobile ad-hoc networks”. IET Communications, 2008, vol. 2, Nº 5, pp. 690-707.

[Ahm06] Ahmed, S.; Alam, M.S. “Performance evaluation of important *Ad hoc* networks protocols”. EURASIP Journal on *Wireless* Communications and Networking. Hindawi Publishing Corporation. 2006.

[Bal04] Baldo L.; Brenner L.; Fernandes L. G.; Fernandes P.; Sales A. “Performance Models for Master/Slave Parallel Programs”. In: First International Workshop on Practical Applications of Stochastic Modelling. The Royal Society, pp. 41-60. London, UK. 2004.

[Ben03] Benoit, A.; Brenner, L.; Fernandes, P.; Plateau, B.; Stewart, J. “The PEPS software tool”. In: Computer Performance Evaluation /TOOLS 2003, vol. 2794 of LNCS, pp. 98-115, Urbana, IL, USA, 2003.

[Bet03] Bettstetter, C.; Resta, G.; Santi, P. “The node distribution of the random waypoint mobility model for *wireless Ad hoc* networks”. IEEE Transactions on Mobile Computing, vol.2 (3): pp.257–269, Jul - Set 2003.

[Bou04] Boukerche, A. “Performance evaluation of routing protocols for *Ad hoc* networks”. Mobile Networks and Applications 9, pp. 333–342, 2004. Kluwer Academic Publishers. Netherlands, 2004.

[Bre05] Brenner L., Fernandes P., Sales A. “The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations”. International Journal of Simulation: Systems, Science & Technology, Vol.6, Number 3-4, p.52 - 60. Nottingham, February, 2005.

[Bre07] Brenner, L.; Fernandes, P.; Plateau B.; Sbeity, I. “PEPS2007 – Stochastic Automata Networks Software Tool”. In: Proceedings of the 4th International Conference on the Quantitative Evaluation of Systems, QEST 2007. Edinburgh, Scotland, pp. 163-164. Setembro, 2007.

[Bro98] Broch, J.; Maltz, D.A.; Jonhson, D. B.; Hu, Y. C.; Jetcheva, J. “A performance comparison of multihop *wireless Ad hoc* network routing protocols”. In: International Conference on Mobile Computing and Networking. Proceedings of the 4th annual ACM/IEEE International conference on Mobile Computing and networking. Dallas, Texas, 1998.

[Cam99] Câmara, D.; Loureiro, A.A.F. “Curso de Redes de Computação Móvel *Ad hoc*”. Jornada de Atualização em Informática, 1999.

[Cha05] Chanin, R.; Fernandes, P.; Dotti, F. L.; Sales, A. “Avaliação Quantitativa de Sistemas”. Porto Alegre, 2005, 88 p.

[Das98] Das, S.R.; Castañeda, R.; Yan, J.; Sengupta, R. “Comparative performance evaluation of routing protocols for mobile, *Ad hoc* networks”. IEEE Computer Communications and Networks, 1998. Proceedings. 7th International Conference on, 1998.

[Das00] Das, S. R.; Castañeda, R.; Yan, J. “Simulation-based performance evaluation of routing protocols for mobile *Ad hoc* networks”. Mobile Networks and Aplications, vol. 5, Baltzer Science Publishers BV, 2000.

[Del06] Delamare F.; Dotti F. L.; Fernandes P.; Nunes C. M.; Ost L. C. “Analytical Modeling of Random Waypoint Mobility Patterns”. In: Third ACM International Workshop on Performance Evaluation of *Wireless Ad hoc*, Sensor, and Ubiquitous Networks, 2006, Torremolinos. ACM PE-WASUN 2006. pp. 106 - 113.

[Dot05] Dotti F. L.; Fernandes P.; Sales A., Santos O. M. “Modular Analytical Performance Models for *Ad hoc Wireless* Networks”. In: 3rd International Symposium on Modeling and Optimization in Mobile, *Ad hoc*, and *Wireless* Networks, 2005, Trentino, Italy. WiOpt 2005. New York: IEEE Press, 2005. pp. 164-173.

[Fee99] Feeney, L. M. “A taxonomy for routing protocols in mobile *Ad hoc* networks. SICS Technical Report Bristol, UK, October, 1999.

[Fer98] Fernandes, P.; Plateau, B.; Stewart, W. J. “Efficient descriptor – Vector multiplication in stochastic automata networks”. Journal of the ACM. V. 45, n.3, Maio 1998. pp. 381-414.

[Fou06] Forouzan, B. A. “Comunicação de dados e Redes de Computadores”. Porto Alegre: Ed. Bookman, 2006, 840p.

[Iee07] IEEE Standard for Information Technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – “Specific requirements Part 11: *Wireless* LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Sponsor”, IEEE, 2007.

[Iet08] IETF. “Internet Engineering Task Force”. Capturado em: http://www.ietf.org, Dezembro 2008.

[Jai91] Jain R., “The Art of Computer Systems Performance Analysis: Techniques fo Experimental Design, Measurement, Simulation, and Modeling”. New York: Wiley - Interscience, 1991.

[Joh96a] Johnson, D.B.; Maltz, D.A. “Dynamic Source Routing in *Ad hoc Wireless* Networks, in Mobile Computing”, edited by Tomasz Imielinski and Hank Korth, Chapter 5, pp. 153-181, Kluwer Academic Publishers, 1996.

[Joh96b] Johnson, D.B.; Maltz, D.A. “Protocols for Adaptive *Wireless* and Mobile Networking”. IEEE Personal Communications, vol. 3(1): pp.34-42, Fevereiro, 1996.

[Joh07] Johnson, D.B.; Hu, Y.; Maltz, D. “The dynamic source routing protocol (DSR) for mobile *Ad hoc* networks for IPv4”. Request for Comments (RFC): 4728, Fevereiro 2007.

[Kur06] Kurose, J.; Ross, K. W. “Redes de Computadores e a Internet: uma abordagem top-down”. São Paulo: Pearson Addison Wesley, 2006, 3ª. ed, 656p.

[Lay07] Layuan, L.; Chunlin, L.; Peiyan, Y. “Performance evaluation and simulations of routing protocols in *Ad hoc* networks”. Computer Communications 30, pp. 1890-1898, Elsevier, 2007.

[Man08] ”Mobile *Ad hoc* Networks (MANET)”. Capturado em: http://www.ietf.org/html.charters/manet-charter.html, Dezembro 2008.

[Mar06] Marinoni, S.; Kari, H. H. “*Ad hoc* routing protocol's performance: a realistic simulation based study”. Telecommunications Systems (2006) 33: pp. 269-289. Springer Science + Business Media, LLC, 2006.

[Oli05] Oliveira, L. B.; Siqueira, I. G.; Loureiro, A.A.F. “On the performance of *Ad hoc* routing protocols under a peer-to-peer application”. Journal of Parallel and Distributed Computing, 2005.

[Per99] Perkins, C.E.; Royer, E.M. “Ad-hoc on-demand distance vector”. Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA ‘99. Second IEEE Workshop on, 1999.

[Per01] Perkins, C.E.; Royer, E.M., Das, S.R.; Marina, M.K. “Performance comparison of two on-demand routing protocols for *Ad hoc* networks”. IEEE Personal Communications, Fevereiro, 2001.

[Pid92] Pidd, M. “Computer simulations and management science”. John Wiley and Sons, 1992.

[Pla85] Plateau. B. “On the stochastic structure of parallelism and synchronization models for distributed algorithms”. ACM Sigmetrics Conference on Measurements and Modeling of Computer Systems, pp. 147-154, Austin, Texas, USA, 1985.

[Puc07] Pucha, H.; Das, S.M.; Hu, Y.C. “The performance impact of traffic patterns on routing protocols in mobile *Ad hoc* networks”. Computer Networks, vol. 51, Março 2007.

[Ram02] Ramanathan, R.; Redi, J. “A brief oveirview of *Ad hoc* networks: challenges and directions”. IEEE Communications Magazine. 50th Anniversary Commemorative Issue. Maio 2002.

[Rao04] Rao, R. “Integration of On-Demand Service and Route Discovery in Mobile *Ad hoc* Networks”, Tese de Doutorado, Raleigh, North Carolina State University, North Carolina, 2004, 120p.

[Roy99] Royer, E.M. “A review of current routing protocols for *Ad hoc* mobile *wireless* networks”. IEEE Personal Communications. Abril 1999.

[Ste94] Stewart, W. J. “Introduction to the numerical solution of Markov Chains”. Princeton University Press, 1994.

[Tan03] Tanenbaum, A. S. “Redes de computadores”. Rio de Janeiro: Elsevier, 2003 – 5ª Reimpressão.

[Tav94] Tavares, J.F.S. “Dicionário VERBO de Inglês Técnico e Científico”. VERBO, 1994.

[Wan03] Wang, J.; Yuan, Z.; Leung, C.; Jia, W. “A-DSR: A DSR-Based Anycast Protocol for IPv6 Flow in Mobile *Ad hoc* Networks”. In: Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th.

APÊNDICE A – SAN PARA REDES SIMPLES SEM PROBABILIDADES DE FALHAS

SAN da Figura 5.8, modelo para redes simples e sem probabilidades de falhas na transmissão.

```
identifiers
    f1      = (st AInt_1 != Off) * 0.8;
    f2      = (st AInt_1 == Off) * 0.2;
    f3      = (st AInt_2 != Off) * 0.8;
    f4      = (st AInt_2 == Off) * 0.2;
    f5      = (st AInt_3 != Off) * 0.8;
    f6      = (st AInt_3 == Off) * 0.2;
    f7      = (st AInt_4 != Off) * 0.8;
    f8      = (st AInt_4 == Off) * 0.2;
    f9      = (st AInt_5 != Off) * 0.8;
    f10     = (st AInt_5 == Off) * 0.2;
    f11     = (st AInt_6 != Off) * 0.8;
    f12     = (st AInt_6 == Off) * 0.2;
    f13     = (st AInt_7 != Off) * 0.8;
    f14     = (st AInt_7 == Off) * 0.2;
    bandwidth_b = 2000000;                    // bandwidth in bits/seconds
    bandwidth_B = bandwidth_b/8;              // bandwidth in bytes/seconds
    bandwidth_Mbps = bandwidth_b/1000000;     // bandwidth in bytes/seconds
    size_pack_64 = 64;                        // package size in bytes
    size_pack_128 = 128;
    size_pack_256 = 256;
    size_pack_512 = 512;
    NXH = 7;                                  // DSR Next Header field in bytes
    RESVD = 8;                                // DSR Reserved Header in bytes
    MAC = 47;                                 // Medium Access Control header in
bytes
    PALG = 16;                               // DSR Payload Length header in
bytes
    header_dsr = NXH+RESVD+MAC+PALG;         // headers
    ocur_rate = 500;                     // ocurrence rate of a route request
message in miliseconds
    lambda = 0.5;                 // ocurrence rate of a route request message
in seconds (1 second = 1000 miliseconds)
    tx_tpause = 0.1777;           // ocurrence rate of time pause (for 5.625
seconds, according to established time pause)


    tau     = (st ASource == T) && ((st AInt_1 == Fw) && (st AInt_1 != Off)) &&
((st AInt_4 == I) && (AInt_4 != Off) && (AInt_4 != R)) && ((AInt_3 != Fw) ||
(AInt_3 != R)) && (ADest == I) && ((AInt_2 != Fw) && (AInt_5 != Fw) && (AInt_6
!= Fw) && (AInt_7 != Fw));

    gama    = (st ASource == R) && ((st AInt_2 == Fw) && (st AInt_2 != Off) &&
(st AInt_2 != R)) && ((st AInt_3 == I) && (st AInt_3 != Off) && (st AInt_3 !=
R)) && (ADest == I) && (st ADest != R) && ((st AInt_1 != R) && (AInt_4 != R) &&
(AInt_5 != R) && (AInt_6 != R) && (AInt_7 != R));

    psi     = ((st ASource == R) && ((st AInt_3 == Fw) && (st AInt_3 != Off) &&
(st AInt_3 != R)) && ((st AInt_7 == I) && (st AInt_7 != Off) && (st AInt_7 !=
R)) && (ADest == I) && (st ADest != R) && ((AInt_1 != R) && (AInt_2 != R) &&
(AInt_4 != R) && (AInt_5 != R) && (st AInt_6 != R)));
```

```
    ro      = (st ASource == R) && ((st AInt_4 == Fw) && (st AInt_4 != Off) &&
(st AInt_4 != R)) && (ADest == I) && (st ADest != R) && (((st AInt_5 == I) &&
(st AInt_5 != Off) && (st AInt_5 != R)) || ((st AInt_6 == I) && (st AInt_6 !=
Off) && (st AInt_6 != R))) && ((AInt_2 != R) && (AInt_3 != R) && (AInt_7 != R)
&& (AInt_1 != R));

    niu     = (((st ASource == R) && ((st AInt_7 == Fw) && (st AInt_7 != Off) &&
(st AInt_7 != R)) && (st AInt_6 != Fw) && (ADest == I) && ((AInt_1 != R) &&
(AInt_2 != R) && (AInt_3 != R) && (AInt_4 != R) && (AInt_5 != R) && (AInt_6 !=
R) && (st ADest != R))));

    chi     = (((st ASource == R) && ((st AInt_5 == Fw) && (st AInt_5 != Off) &&
(st AInt_5 != R)) && ((st ADest == I) && (st ADest != R)) && ((st AInt_7 == I)
&& (st AInt_7 != Off) && (st AInt_7 != R)) && ((AInt_1 != R) && (AInt_2 != R) &&
(AInt_3 != R) && (AInt_4 != R) && (st AInt_6 != R))));

    eta     = ((st ASource == R) && ((st AInt_6 == Fw) && (st AInt_6 != Off) &&
(st AInt_6 != R)) && ((st AInt_7 == I) && (st AInt_7 != Off) && (st AInt_7 !=
R)) && ((st ADest == I) && (st ADest != R)) && ((AInt_1 != R) && (AInt_2 != R)
&& (AInt_3 != R) && (AInt_4 != R) && (AInt_5 != R)));

    epsilon = (((st ASource == R) && ((st AInt_7 == Fw) && (st AInt_7 != Off) &&
(st AInt_7 != R)) && (st AInt_3 != Fw) && (ADest == I) && ((AInt_1 != R) &&
(AInt_2 != R) && (AInt_3 != R) && (AInt_4 != R) && (AInt_5 != R) && (AInt_6 !=
R) && (st ADest != R))));

    alfa    = (st ADest == R) && ((st AInt_7 == Fw) && (st AInt_7 != Off) && (st
AInt_7 != R)) && (st ASource == R) && ((st AInt_6 != R) && (st AInt_5 != R) &&
(st AInt_4 != R) && (st AInt_2 != R) && (st AInt_3 != R) && (st AInt_1 != R));

    beta    = ((st AInt_7 == R) && (st AInt_7 != Off) && (st AInt_7 != Fw)) &&
(st ASource == R) && ((st AInt_3 == Fw) && (st AInt_3 != Off)) && ((st AInt_6 !=
R) && (st AInt_5 != R) && (st AInt_4 != R) && (st AInt_2 != R) && (st AInt_1 !=
R)) && (st ADest == I);

    delta   = (st ASource == R) && ((st AInt_3 == R) && (st AInt_3 != Off) &&
(st AInt_3 != Fw)) && (st ADest == I) && ((st AInt_2 == Fw) && (st AInt_2 !=
Off) && (st AInt_2 != R)) && ((st AInt_7 != R) && (st AInt_5 != R) && (st AInt_4
!= R) && (st AInt_1 != R) && (st AInt_6 != R));

    omega   = (st ASource == R) && ((st AInt_2 == R) && (st AInt_2 != Off) &&
(st AInt_2 != Fw)) && (st ADest == I) && ((st AInt_1 != R) && (st AInt_3 != R)
&& (st AInt_4 != R) && (st AInt_5 != R) && (st AInt_6 != R) && (st AInt_7 != R))
&& ((st AInt_3 == I) && (st AInt_7 == I));

    sigma   = (st ADest == R) && ((st AInt_5 == Fw) && (st AInt_5 != Off) && (st
AInt_5 != R)) && (st ASource == R) && ((st AInt_7 != R) && (st AInt_6 != R) &&
(st AInt_4 != R) && (st AInt_3 != R) && (st AInt_2 != R) && (st AInt_1 != R));

    sho     = ((st AInt_5 == R) && (st AInt_5 != Off) && (st AInt_5 != Fw)) &&
(st ASource == R) && (st AInt_4 == Fw) && (st AInt_4 != Off) && ((st AInt_4 !=
R)) && ((st AInt_7 != R) && (st AInt_6 != R) && (st AInt_3 != R) && (st AInt_2
!= R) && (st AInt_1 != R)) && (st ADest == I);

    kapa    = ((st AInt_4 == R) && (st AInt_4 != Off) && (st AInt_4 != Fw)) &&
(st ASource == R) && ((st AInt_1 == Fw) && (st AInt_1 != Off) && (st AInt_1 !=
R)) && ((st AInt_2 != R) && (st AInt_3 != R) && (st AInt_5 != R) && (st AInt_6
!= R) && (st AInt_7 != R)) && (st ADest == I);
```

```
    teta    = (st ASource == R) && ((st AInt_1 == R) && (st AInt_1 != Off) &&
(st AInt_1 != Fw)) && (st ADest == I) && ((st AInt_2 != R) && (st AInt_3 != R)
&& (st AInt_4!= R) && (st AInt_5 != R) && (st AInt_6 != R) && (st AInt_7 != R));

    zeta    = (st ASource == R) && ((st AInt_7 == Fw) && (st AInt_7 != Off) &&
(st AInt_7 != R)) && (st ADest == I) && ((st AInt_1 != R) && (st AInt_2 != R) &&
(st AInt_3 != R) && (st AInt_4 != R) && (st AInt_5 != R) && (st AInt_6 != R));

    upsilon = (st ASource == R) && ((st AInt_7 == R) && (st AInt_7 != Off) &&
(st AInt_7 != Fw)) && (st ADest == I) && ((st AInt_6 == Fw) && (st AInt_6 !=
Off) && (st AInt_6 != R)) && ((st AInt_5 != R) && (st AInt_4 != R) && (st AInt_3
!= R) && (st AInt_2 != R) && (st AInt_1 != R));

    iota    = ((st AInt_6 == R) && (st AInt_6 != Off) && (st AInt_6 != Fw)) &&
(st ASource == R) && (st AInt_4 == Fw) && (st AInt_4 != Off) && ((st AInt_4 !=
R)) && ((st AInt_7 != R) && (st AInt_5 != R) && (st AInt_3 != R) && (st AInt_2
!= R) && (st AInt_1 != R)) && (st ADest == I);

    stigma  = ((st AInt_4 == R) && (st AInt_4 != Off) && (st AInt_4 != Fw)) &&
(st ASource == R) && ((st AInt_1 == Fw) && (st AInt_1 != Off) && (st AInt_1 !=
R)) && ((st AInt_2 != R) && (st AInt_3 != R) && (st AInt_5 != R) && (st AInt_6
!= R) && (st AInt_7 != R)) && (st ADest == I);

    qoopa   = (st ASource == R) && ((st AInt_1 == R) && (st AInt_1 != Off) &&
(st AInt_1 != Fw)) && (st ADest == I) && ((st AInt_2 != R) && (st AInt_3 != R)
&& (st AInt_4!= R) && (st AInt_5 != R) && (st AInt_6 != R) && (st AInt_7 != R));


events

  loc on1 f1;
  loc off1 f2;
  loc on2 f3;
  loc off2 f4;
  loc on3 f5;
  loc off3 f6;
  loc on4 f7;
  loc off4 f8;
  loc on5 f9;
  loc off5 f10;
  loc on6 f11;
  loc off6 f12;
  loc on7 f13;
  loc off7 f14;

  syn rreq   lambda;
  syn s1     tau;
  syn s2     gama;
  syn s3     psi;
  syn s4     ro;
  syn s5     niu;
  syn s6     chi;
  syn s7     eta;
  syn s8     epsilon;
  syn rrep1  alfa;
  syn rrep2  beta;
  syn rrep3  delta;
  syn rrep4  omega;
  syn rrep5  sigma;
  syn rrep6  sho;
  syn rrep7  kapa;
```

```
  syn rrep8  teta;
  syn rrep9  zeta;
  syn rrep10 upsilon;
  syn rrep11 iota;
  syn rrep12 stigma;
  syn rrep13 qoopa;

partial reachability = ((st ASource == T) && (st ADest == I) && ((st AInt_1 ==
Fw) && (st AInt_4 == Fw) && (st AInt_5 == Fw)) && ((st AInt_2 != R) && (st
AInt_2 != Fw) && (st AInt_2 != Off)) && ((st AInt_3 != R) && (st AInt_3 != Fw)
&& (st AInt_3 != Off)) && ((st AInt_6 != R) && (st AInt_6 != Fw) && (st AInt_6
!= Off)) && ((st AInt_7 != R) && (st AInt_7 != Fw) && (st AInt_7 != Off))) ||
((st ASource == T) && (st ADest == I) && ((st AInt_2 == Fw) && (st AInt_3 == Fw)
&& (st AInt_7 == Fw)) && ((st AInt_1 != R) && (st AInt_1 != Fw) && (st AInt_1 !=
Off)) && ((st AInt_4 != R) && (st AInt_4 != Fw) && (st AInt_4 != Off)) && ((st
AInt_5 != R) && (st AInt_5 != Fw) && (st AInt_5 != Off)) && ((st AInt_6 != R) &&
(st AInt_6 != Fw) && (st AInt_6 != Off))) || ((st ASource == T) && (st ADest ==
I) && ((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 == Fw) && (st AInt_7
== Fw)) && ((st AInt_2 != R) && (st AInt_2 != Fw) && (st AInt_2 != Off)) && ((st
AInt_3 != R) && (st AInt_3 != Fw) && (st AInt_3 != Off)) && ((st AInt_5 != R) &&
(st AInt_5 != Fw) && (st AInt_5 != Off)));

network Adsr (continuous)
  aut ASource
    stt T to (R) rreq
    stt R to (T) rrep4 rrep8 rrep13

  aut AInt_1
    stt I to (Off) off1
          to (Fw) rreq
    stt Off to (I) on1
    stt Fw to (Fw) s1
           to (R) rrep7 rrep12
    stt R to (I) rrep8 rrep13


  aut AInt_2
    stt I to (Off) off2
          to (Fw) rreq
    stt Off to (I) on2
    stt Fw to (Fw) s2
           to (R) rrep3
    stt R to (I) rrep4

  aut AInt_3
    stt I to (Off) off3
          to (Fw) s2
    stt Off to (I) on3
    stt Fw to (Fw) s3
           to (R) rrep2
    stt R to (I) rrep3

  aut AInt_4
    stt I to (Off) off4
          to (Fw) s1
    stt Off to (I) on4
    stt Fw to (Fw) s4
           to (R) rrep6 rrep11
    stt R to (I) rrep7 rrep12

  aut AInt_5
```

```
    stt I to (Off) off5
          to (Fw) s4
    stt Off to (I) on5
    stt Fw to (Fw) s6
           to (R) rrep5
    stt R to (I) rrep6

  aut AInt_6
    stt I to (Off) off6
          to (Fw) s4
    stt Off to (I) on6
    stt Fw to (Fw) s7
           to (R) rrep10
    stt R to (I) rrep11

  aut AInt_7
    stt I to (Off) off7
          to (Fw) s3 s7
    stt Off to (I) on7
    stt Fw to (Fw) s5 s8
           to (R) rrep1 rrep9
    stt R to (I) rrep2 rrep10

  aut ADest
    stt I to (R) s5 s6 s8
    stt R to (I) rrep1 rrep5 rrep9

results
  t_AInt_237_64  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 ==
Fw)) * bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr)))/ tx_pause;
  t_AInt_237_128  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 ==
Fw)) * bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))/ tx_pause;
  t_AInt_237_256  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 ==
Fw)) * bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))/ tx_pause;
  t_AInt_237_512  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 ==
Fw)) * bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))/ tx_pause;

  t_AInt_145_64  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 ==
Fw)) * bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr)))/ tx_pause;
  t_AInt_145_128  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 ==
Fw)) * bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))/ tx_pause;
  t_AInt_145_256  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 ==
Fw)) * bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))/ tx_pause;
  t_AInt_145_512  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 ==
Fw)) * bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))/ tx_pause;

  t_AInt_1467_64  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 ==
Fw) && (st AInt_7 == Fw)) * bandwidth_Mbps) *
(size_pack_64/(size_pack_64+header_dsr)))/ tx_pause;
  t_AInt_1467_128  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 ==
Fw) && (st AInt_7 == Fw)) * bandwidth_Mbps) *
(size_pack_128/(size_pack_128+header_dsr)))/ tx_pause;
  t_AInt_1467_256  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 ==
Fw) && (st AInt_7 == Fw)) * bandwidth_Mbps) *
(size_pack_256/(size_pack_256+header_dsr)))/ tx_pause;
  t_AInt_1467_512  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 ==
Fw) && (st AInt_7 == Fw)) * bandwidth_Mbps) *
(size_pack_512/(size_pack_512+header_dsr)))/ tx_pause;

  t_route_237  = ((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 == Fw));
  t_route_145  = ((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 == Fw));
```

```
  t_route_1467 = ((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 == Fw) &&
(st AInt_7 == Fw));

  t_Asource = (st ASource == T);

  t_AInt_1_Off = (st AInt_1 == Off);
  t_AInt_2_Off = (st AInt_2 == Off);
  t_AInt_3_Off = (st AInt_3 == Off);
  t_AInt_4_Off = (st AInt_4 == Off);
  t_AInt_5_Off = (st AInt_5 == Off);
  t_AInt_6_Off = (st AInt_6 == Off);
  t_AInt_7_Off = (st AInt_7 == Off);

  t_AInt_1_Rep = (st AInt_1 == R);
  t_AInt_2_Rep = (st AInt_2 == R);
  t_AInt_3_Rep = (st AInt_3 == R);
  t_AInt_4_Rep = (st AInt_4 == R);
  t_AInt_5_Rep = (st AInt_5 == R);
  t_AInt_6_Rep = (st AInt_6 == R);
  t_AInt_7_Rep = (st AInt_7 == R);

  t_AInt_1_Fw = (st AInt_1 == Fw);
  t_AInt_2_Fw = (st AInt_2 == Fw);
  t_AInt_3_Fw = (st AInt_3 == Fw);
  t_AInt_4_Fw = (st AInt_4 == Fw);
  t_AInt_5_Fw = (st AInt_5 == Fw);
  t_AInt_6_Fw = (st AInt_6 == Fw);
  t_AInt_7_Fw = (st AInt_7 == Fw);
```

APÊNDICE B – SAN PARA REDES SIMPLES COM PROBABILIDADES DE FALHAS

SAN da Figura 5.8, modelo para redes simples com probabilidades de falhas na transmissão.

```
identifiers
    f1      = (st AInt_1 != Off) * 0.8;
    f2      = (st AInt_1 == Off) * 0.2;
    f3      = (st AInt_2 != Off) * 0.8;
    f4      = (st AInt_2 == Off) * 0.2;
    f5      = (st AInt_3 != Off) * 0.8;
    f6      = (st AInt_3 == Off) * 0.2;
    f7      = (st AInt_4 != Off) * 0.8;
    f8      = (st AInt_4 == Off) * 0.2;
    f9      = (st AInt_5 != Off) * 0.8;
    f10     = (st AInt_5 == Off) * 0.2;
    f11     = (st AInt_6 != Off) * 0.8;
    f12     = (st AInt_6 == Off) * 0.2;
    f13     = (st AInt_7 != Off) * 0.8;
    f14     = (st AInt_7 == Off) * 0.2;
    bandwidth_b = 2000000;                    // bandwidth in bits/seconds
    bandwidth_B = bandwidth_b/8;              // bandwidth in bytes/seconds
    bandwidth_Mbps = bandwidth_b/1000000;     // bandwidth in bytes/seconds
    size_pack_64 = 64;                          // package size in bytes
    size_pack_128 = 128;
    size_pack_256 = 256;
    size_pack_512 = 512;
    NXH = 7;                                  // DSR Next Header field in bytes
    RESVD = 8;                                // DSR Reserved Header in bytes
    MAC = 47;                                 // Medium Access Control header in
bytes
    PALG = 16;                              // DSR Payload Length header in bytes
    header_dsr = NXH+RESVD+MAC+PALG;          // headers
    ocur_rate = 500;                        // ocurrence rate of a route request
message in miliseconds
    lambda = 0.5;                 // ocurrence rate of a route request message
in seconds (1 second = 1000 miliseconds)
    tx_tpause = 0.1777;           // ocurrence rate of time pause (for 5.625
seconds, according to established time pause)

    tau     = (st ASource == R) && ((st AInt_1 == Fw) && (st AInt_1 != Off)) &&
((st AInt_4 == I) && (AInt_4 != Off) && (AInt_4 != R)) && ((AInt_3 != Fw) ||
(AInt_3 != R)) && (ADest == I) && (st ADest != R) && ((AInt_2 != R) && (AInt_3
!= R) && (AInt_5 != R) && (AInt_6 != R) && (AInt_7 != R));

    gama    = (st ASource == R) && ((st AInt_2 == Fw) && (st AInt_2 != Off) &&
(st AInt_2 != R)) && ((st AInt_3 == I) && (st AInt_3 != Off) && (st AInt_3 !=
R)) && (ADest == I) && (st ADest != R) && ((st AInt_1 != R) && (AInt_4 != R) &&
(AInt_5 != R) && (AInt_6 != R) && (AInt_7 != R));

    psi     = ((st ASource == R) && ((st AInt_3 == Fw) && (st AInt_3 != Off) &&
(st AInt_3 != R)) && ((st AInt_7 == I) && (st AInt_7 != Off) && (st AInt_7 !=
R)) && (ADest == I) && (st ADest != R) && ((AInt_1 != R) && (AInt_2 != R) &&
(AInt_4 != R) && (AInt_5 != R) && (st AInt_6 != R)));

    ro      = (st ASource == R) && ((st AInt_4 == Fw) && (st AInt_4 != Off) &&
(st AInt_4 != R)) && (ADest == I) && (st ADest != R) && (((st AInt_5 == I) &&
(st AInt_5 != Off) && (st AInt_5 != R)) || ((st AInt_6 == I) && (st AInt_6 !=
```

```
Off) && (st AInt_6 != R))) && ((AInt_2 != R) && (AInt_3 != R) && (AInt_7 != R)
&& (AInt_1 != R));

    niu     = (((st ASource == R) && ((st AInt_7 == Fw) && (st AInt_7 != Off) &&
(st AInt_7 != R)) && (st AInt_6 != Fw) && (ADest == I) && ((AInt_1 != R) &&
(AInt_2 != R) && (AInt_3 != R) && (AInt_4 != R) && (AInt_5 != R) && (AInt_6 !=
R) && (st ADest != R))));

    chi     = (((st ASource == R) && ((st AInt_5 == Fw) && (st AInt_5 != Off) &&
(st AInt_5 != R)) && ((st ADest == I) && (st ADest != R)) && ((st AInt_7 == I)
&& (st AInt_7 != Off) && (st AInt_7 != R)) && ((AInt_1 != R) && (AInt_2 != R) &&
(AInt_3 != R) && (AInt_4 != R) && (st AInt_6 != R))));

    eta     = ((st ASource == R) && ((st AInt_6 == Fw) && (st AInt_6 != Off) &&
(st AInt_6 != R)) && ((st AInt_7 == I) && (st AInt_7 != Off) && (st AInt_7 !=
R)) && ((st ADest == I) && (st ADest != R)) && ((AInt_1 != R) && (AInt_2 != R)
&& (AInt_3 != R) && (AInt_4 != R) && (AInt_5 != R)));

    epsilon = (((st ASource == R) && ((st AInt_7 == Fw) && (st AInt_7 != Off) &&
(st AInt_7 != R)) && (st AInt_3 != Fw) && (ADest == I) && ((AInt_1 != R) &&
(AInt_2 != R) && (AInt_3 != R) && (AInt_4 != R) && (AInt_5 != R) && (AInt_6 !=
R) && (st ADest != R))));

    alfa    = (st ADest == R) && ((st AInt_7 == Fw) && (st AInt_7 != Off) && (st
AInt_7 != R)) && (st ASource == R) && ((st AInt_6 != R) && (st AInt_5 != R) &&
(st AInt_4 != R) && (st AInt_2 != R) && (st AInt_3 != R) && (st AInt_1 != R));

    beta    = ((st AInt_7 == R) && (st AInt_7 != Off) && (st AInt_7 != Fw)) &&
(st ASource == R) && ((st AInt_3 == Fw) && (st AInt_3 != Off)) && ((st AInt_6 !=
R) && (st AInt_5 != R) && (st AInt_4 != R) && (st AInt_2 != R) && (st AInt_1 !=
R)) && (st ADest == I);

    delta   = (st ASource == R) && ((st AInt_3 == R) && (st AInt_3 != Off) &&
(st AInt_3 != Fw)) && (st ADest == I) && ((st AInt_2 == Fw) && (st AInt_2 !=
Off) && (st AInt_2 != R)) && ((st AInt_7 != R) && (st AInt_5 != R) && (st AInt_4
!= R) && (st AInt_1 != R) && (st AInt_6 != R));

    omega   = (st ASource == R) && ((st AInt_2 == R) && (st AInt_2 != Off) &&
(st AInt_2 != Fw)) && (st ADest == I) && ((st AInt_1 != R) && (st AInt_3 != R)
&& (st AInt_4 != R) && (st AInt_5 != R) && (st AInt_6 != R) && (st AInt_7 != R))
&& ((st AInt_3 == I) && (st AInt_7 == I));

    sigma   = (st ADest == R) && ((st AInt_5 == Fw) && (st AInt_5 != Off) && (st
AInt_5 != R)) && (st ASource == R) && ((st AInt_7 != R) && (st AInt_6 != R) &&
(st AInt_4 != R) && (st AInt_3 != R) && (st AInt_2 != R) && (st AInt_1 != R));

    sho     = ((st AInt_5 == R) && (st AInt_5 != Off) && (st AInt_5 != Fw)) &&
(st ASource == R) && (st AInt_4 == Fw) && (st AInt_4 != Off) && ((st AInt_4 !=
R)) && ((st AInt_7 != R) && (st AInt_6 != R) && (st AInt_3 != R) && (st AInt_2
!= R) && (st AInt_1 != R)) && (st ADest == I);

    kapa    = ((st AInt_4 == R) && (st AInt_4 != Off) && (st AInt_4 != Fw)) &&
(st ASource == R) && ((st AInt_1 == Fw) && (st AInt_1 != Off) && (st AInt_1 !=
R)) && ((st AInt_2 != R) && (st AInt_3 != R) && (st AInt_5 != R) && (st AInt_6
!= R) && (st AInt_7 != R)) && (st ADest == I);

    teta    = (st ASource == R) && ((st AInt_1 == R) && (st AInt_1 != Off) &&
(st AInt_1 != Fw)) && (st ADest == I) && ((st AInt_2 != R) && (st AInt_3 != R)
&& (st AInt_4!= R) && (st AInt_5 != R) && (st AInt_6 != R) && (st AInt_7 != R));
```

```
    zeta    = (st ASource == R) && ((st AInt_7 == Fw) && (st AInt_7 != Off) &&
(st AInt_7 != R)) && (st ADest == I) && ((st AInt_1 != R) && (st AInt_2 != R) &&
(st AInt_3 != R) && (st AInt_4 != R) && (st AInt_5 != R) && (st AInt_6 != R));

    upsilon = (st ASource == R) && ((st AInt_7 == R) && (st AInt_7 != Off) &&
(st AInt_7 != Fw)) && (st ADest == I) && ((st AInt_6 == Fw) && (st AInt_6 !=
Off) && (st AInt_6 != R)) && ((st AInt_5 != R) && (st AInt_4 != R) && (st AInt_3
!= R) && (st AInt_2 != R) && (st AInt_1 != R));

    iota    = ((st AInt_6 == R) && (st AInt_6 != Off) && (st AInt_6 != Fw)) &&
(st ASource == R) && (st AInt_4 == Fw) && (st AInt_4 != Off) && ((st AInt_4 !=
R)) && ((st AInt_7 != R) && (st AInt_5 != R) && (st AInt_3 != R) && (st AInt_2
!= R) && (st AInt_1 != R)) && (st ADest == I);

    stigma  = ((st AInt_4 == R) && (st AInt_4 != Off) && (st AInt_4 != Fw)) &&
(st ASource == R) && ((st AInt_1 == Fw) && (st AInt_1 != Off) && (st AInt_1 !=
R)) && ((st AInt_2 != R) && (st AInt_3 != R) && (st AInt_5 != R) && (st AInt_6
!= R) && (st AInt_7 != R)) && (st ADest == I);

    qoopa   = (st ASource == R) && ((st AInt_1 == R) && (st AInt_1 != Off) &&
(st AInt_1 != Fw)) && (st ADest == I) && ((st AInt_2 != R) && (st AInt_3 != R)
&& (st AInt_4!= R) && (st AInt_5 != R) && (st AInt_6 != R) && (st AInt_7 != R));


events

  loc on1 f1;
  loc off1 f2;
  loc on2 f3;
  loc off2 f4;
  loc on3 f5;
  loc off3 f6;
  loc on4 f7;
  loc off4 f8;
  loc on5 f9;
  loc off5 f10;
  loc on6 f11;
  loc off6 f12;
  loc on7 f13;
  loc off7 f14;

  syn rreq   lambda;
  syn s1     tau;
  syn s2     gama;
  syn s3     psi;
  syn s4     ro;
  syn s5     niu;
  syn s6     chi;
  syn s7     eta;
  syn s8     epsilon;
  syn rrep1  alfa;
  syn rrep2  beta;
  syn rrep3  delta;
  syn rrep4  omega;
  syn rrep5  sigma;
  syn rrep6  sho;
  syn rrep7  kapa;
  syn rrep8  teta;
  syn rrep9  zeta;
  syn rrep10 upsilon;
  syn rrep11 iota;
```

```
  syn rrep12 stigma;
  syn rrep13 qoopa;

partial reachability = ((st ASource == T) && (st ADest == I) && ((st AInt_1 ==
Fw) && (st AInt_4 == Fw) && (st AInt_5 == Fw))) || ((st ASource == T) && (st
ADest == I) && ((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 == Fw))) ||
((st ASource == T) && (st ADest == I) && ((st AInt_1 == Fw) && (st AInt_4 == Fw)
&& (st AInt_6 == Fw) && (st AInt_7 == Fw)));

network Adsr (continuous)
  aut ASource
    stt T to (R) rreq
    stt R to (T) rrep4 rrep8 rrep13

  aut AInt_1
    stt I to (Off) off1
          to (Fw) rreq
    stt Off to (I) on1
    stt Fw to (Fw) s1
           to (R) rrep7 rrep12
    stt R to (I) rrep8 rrep13


  aut AInt_2
    stt I to (Off) off2
          to (Fw) rreq
    stt Off to (I) on2
    stt Fw to (Fw) s2
           to (R) rrep3
    stt R to (I) rrep4

  aut AInt_3
    stt I to (Off) off3
          to (Fw) s2
    stt Off to (I) on3
    stt Fw to (Fw) s3
           to (R) rrep2
    stt R to (I) rrep3

  aut AInt_4
    stt I to (Off) off4
          to (Fw) s1
    stt Off to (I) on4
    stt Fw to (Fw) s4
           to (R) rrep6 rrep11
    stt R to (I) rrep7 rrep12

  aut AInt_5
    stt I to (Off) off5
          to (Fw) s4
    stt Off to (I) on5
    stt Fw to (Fw) s6
           to (R) rrep5
    stt R to (I) rrep6

  aut AInt_6
    stt I to (Off) off6
          to (Fw) s4
    stt Off to (I) on6
    stt Fw to (Fw) s7
           to (R) rrep10
```

```
    stt R to (I) rrep11

  aut AInt_7
    stt I to (Off) off7
          to (Fw) s3 s7
    stt Off to (I) on7
    stt Fw to (Fw) s5 s8
           to (R) rrep1 rrep9
    stt R to (I) rrep2 rrep10

  aut ADest
    stt I to (R) s5 s6 s8
    stt R to (I) rrep1 rrep5 rrep9

results
  t_AInt_237_64  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 ==
Fw)) * bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr)))/ tx_pause;
  t_AInt_237_128  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 ==
Fw)) * bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))/ tx_pause;
  t_AInt_237_256  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 ==
Fw)) * bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))/tx_pause;
  t_AInt_237_512  = ((((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 ==
Fw)) * bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))/ tx_pause;

  t_AInt_145_64  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 ==
Fw)) * bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr))) / tx_pause;
  t_AInt_145_128  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 ==
Fw)) * bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr))) / tx_pause;
  t_AInt_145_256  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 ==
Fw)) * bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr))) / tx_pause;
  t_AInt_145_512  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 ==
Fw)) * bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr))) / tx_pause;

  t_AInt_1467_64  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 ==
Fw) && (st AInt_7 == Fw)) * bandwidth_Mbps) *
(size_pack_64/(size_pack_64+header_dsr))) / tx_pause;
  t_AInt_1467_128  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 ==
Fw) && (st AInt_7 == Fw)) * bandwidth_Mbps) *
(size_pack_128/(size_pack_128+header_dsr))) / tx_pause;
  t_AInt_1467_256  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 ==
Fw) && (st AInt_7 == Fw)) * bandwidth_Mbps) *
(size_pack_256/(size_pack_256+header_dsr))) / tx_pause;
  t_AInt_1467_512  = ((((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 ==
Fw) && (st AInt_7 == Fw)) * bandwidth_Mbps) *
(size_pack_512/(size_pack_512+header_dsr))) / tx_pause;

  t_route_237  = ((st AInt_2 == Fw) && (st AInt_3 == Fw) && (st AInt_7 == Fw));
  t_route_145  = ((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_5 == Fw));
  t_route_1467 = ((st AInt_1 == Fw) && (st AInt_4 == Fw) && (st AInt_6 == Fw) &&
(st AInt_7 == Fw));

  t_Asource = (st ASource == T);

  t_AInt_1_Off = (st AInt_1 == Off);
  t_AInt_2_Off = (st AInt_2 == Off);
  t_AInt_3_Off = (st AInt_3 == Off);
  t_AInt_4_Off = (st AInt_4 == Off);
  t_AInt_5_Off = (st AInt_5 == Off);
  t_AInt_6_Off = (st AInt_6 == Off);
  t_AInt_7_Off = (st AInt_7 == Off);
```

```
  t_AInt_1_Rep = (st AInt_1 == R);
  t_AInt_2_Rep = (st AInt_2 == R);
  t_AInt_3_Rep = (st AInt_3 == R);
  t_AInt_4_Rep = (st AInt_4 == R);
  t_AInt_5_Rep = (st AInt_5 == R);
  t_AInt_6_Rep = (st AInt_6 == R);
  t_AInt_7_Rep = (st AInt_7 == R);

  t_AInt_1_Fw = (st AInt_1 == Fw);
  t_AInt_2_Fw = (st AInt_2 == Fw);
  t_AInt_3_Fw = (st AInt_3 == Fw);
  t_AInt_4_Fw = (st AInt_4 == Fw);
  t_AInt_5_Fw = (st AInt_5 == Fw);
  t_AInt_6_Fw = (st AInt_6 == Fw);
  t_AInt_7_Fw = (st AInt_7 == Fw);
```

APÊNDICE C – SAN PARA REDES COM ESCUTA PROMÍSCUA SEM PROBABILIDADES DE FALHAS

SAN da Figura 5.11, modelo para redes com escuta promíscua sem probabilidade de falhas na transmissão.

```
identifiers
    f11      = (st AInt_11 != Off) * 0.8;
    f12      = (st AInt_11 == Off) * 0.2;
    f13      = (st AInt_12 != Off) * 0.8;
    f14      = (st AInt_12 == Off) * 0.2;
    f15      = (st AInt_13 != Off) * 0.8;
    f16      = (st AInt_13 == Off) * 0.2;
    f17      = (st AInt_14 != Off) * 0.8;
    f18      = (st AInt_14 == Off) * 0.2;
    f19      = (st AInt_15 != Off) * 0.8;
    f110     = (st AInt_15 == Off) * 0.2;
    f21      = (st AInt_21 != Off) * 0.8;
    f22      = (st AInt_21 == Off) * 0.2;
    f23      = (st AInt_22 != Off) * 0.8;
    f24      = (st AInt_22 == Off) * 0.2;
    f25      = (st AInt_23 != Off) * 0.8;
    f26      = (st AInt_23 == Off) * 0.2;
    f27      = (st AInt_24 != Off) * 0.8;
    f28      = (st AInt_24 == Off) * 0.2;
    f29      = (st AInt_25 != Off) * 0.8;
    f210     = (st AInt_25 == Off) * 0.2;
    bandwidth_b = 2000000;                    // bandwidth in bits/seconds
    bandwidth_B = bandwidth_b/8;              // bandwidth in bytes/seconds
    bandwidth_Mbps = bandwidth_b/1000000;     // bandwidth in bytes/seconds
    size_pack_64 = 64;                        // package size in bytes
    size_pack_128 = 128;
    size_pack_256 = 256;
    size_pack_512 = 512;
    NXH = 7;                                  // DSR Next Header field in bytes
    RESVD = 8;                                // DSR Reserved Header in bytes
    MAC = 47;                                 // Medium Access Control header in
bytes
    PALG = 16;                              // DSR Payload Length header in bytes
    dsr_header = NXH+RESVD+MAC+PALG;           // headers
    ocur_rate = 500;                        // ocurrence rate of a route request
message in miliseconds
    lambda = 0.5;                  // ocurrence rate of a route request message
in seconds (1 second = 1000 miliseconds)
    tx_tpause = 0.1777;            // ocurrence rate of time pause (for 5.625
seconds, according to established time pause)

    lambda1 = lambda;

    tau1     = (st ASource1 == T) && ((st AInt_11 == Fw) && (st AInt_11 != Off)
&& (st AInt_11 != R)) && ((st AInt_15 == I) && (st AInt_15 != Off) && (st
AInt_15 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_12 != R) &&
(st AInt_13 != R) && (st AInt_14 != R));

    gama1    = (((st ASource1 == T) && ((st AInt_15 == Fw) && (st AInt_15 !=
Off) && (st AInt_15 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_13 != R) && (st AInt_14 != R))));
```

```
    psi1     = (st ASource1 == T) && ((st AInt_12 == Fw) && (st AInt_12 != Off)
&& (st AInt_12 != R)) && ((st AInt_14 == I) && (st AInt_14 != Off) && (st
AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) &&
(st AInt_13 != R) && (st AInt_15 != R));

    ro1      = (((st ASource1 == T) && ((st AInt_14 == Fw) && (st AInt_14 !=
Off) && (st AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_13 != R) && (st AInt_15 != R))));

    niu1     = (st ASource1 == T) && ((st AInt_13 == Fw) && (st AInt_13 != Off)
&& (st AInt_13 != R))  && ((st AInt_14 == I) && (st AInt_14 != Off) && (st
AInt_14 != R))  && ((st AInt_21 == I) && (st AInt_21 != Off) && (st AInt_21 !=
R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) && (st AInt_12
!= R) && (st AInt_15 != R));

    chi1     = (((st ASource1 == T) && ((st AInt_14 == Fw) && (st AInt_14 !=
Off) && (st AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_13 != R) && (st AInt_15 != R))));

    eta1     = (st ASource1 == T) && ((st AInt_21 == Fw) && (st AInt_21 != Off)
&& (st AInt_21 != R))   && ((st AInt_25 == I) && (st AInt_25 != Off) && (st
AInt_25 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) &&
(st AInt_12 != R) && (st AInt_13 != R) && (st AInt_14 != R) && (st AInt_15 !=
R));

    epsilon1  = (((st ASource1 == T) && ((st AInt_25 == Fw) && (st AInt_25 !=
Off) && (st AInt_25 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_13 != R) && (st AInt_14 != R) &&
(st AInt_15 != R))));

    alfa1    = (st ADest1 == R) && ((st AInt_15 == Fw) && (st AInt_15 != Off) &&
(st AInt_15 != R)) && (st ASource1 == T) && ((st AInt_14 != R) && (st AInt_13 !=
R) && (st AInt_12 != R) && (st AInt_11 != R));

    beta1    = (st ADest1 == R) && ((st AInt_15 == R) && (st AInt_15 != Off) &&
(st AInt_15 != Fw)) && (st ASource1 == T) && ((st AInt_11 == Fw) && (st AInt_11
!= Off) && (st AInt_11 != R)) && ((st AInt_14 != R) && (st AInt_13 != R) && (st
AInt_11 != R));

    delta1   = (st ADest1 == R) && ((st AInt_11 == R) && (st AInt_11 != Off) &&
(st AInt_11 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_13 != R)) && (st AInt_12 != R);

    omega1   = (st ADest1 == R) && ((st AInt_14 == Fw) && (st AInt_14 != Off) &&
(st AInt_14 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_13 !=
R) && (st AInt_12 != R) && (st AInt_11 != R));

    sigma1   = (st ADest1 == R) && ((st AInt_14 == R) && (st AInt_14 != Off) &&
(st AInt_14 != Fw)) && (st ASource1 == T) && ((st AInt_12 == Fw) && (st AInt_12
!= Off) && (st AInt_12 != R)) && ((st AInt_15 != R) && (st AInt_13 != R) && (st
AInt_11 != R));

    sho1     = (st ADest1 == R) && ((st AInt_12 == R) && (st AInt_12 != Off) &&
(st AInt_12 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_13 != R) && (st AInt_11 != R));

    kapa1    = (st ADest1 == R) && ((st AInt_14 == Fw) && (st AInt_14 != Off) &&
(st AInt_14 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_13 !=
R) && (st AInt_12 != R) && (st AInt_11 != R));
```

```
    teta1    = (st ADest1 == R) && ((st AInt_14 == R) && (st AInt_14 != Off) &&
(st AInt_14 != Fw)) && ((st AInt_13 == Fw) && (st AInt_13 != Off) && (st AInt_13
!= R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_12 != R) && (st
AInt_11 != R));

    zeta1    = (st ADest1 == R) && ((st AInt_13 == R) && (st AInt_13 != Off) &&
(st AInt_13 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    upsilon1 = (st ADest1 == R) && ((st AInt_25 == Fw) && (st AInt_25 != Off) &&
(st AInt_25 != R)) && (st ASource1 == T) &&  ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_13 != R) && (st AInt_12 != R) && (st AInt_11 != R));

    iota1    = (st ADest1 == R) && ((st AInt_25 == R) && (st AInt_25 != Off) &&
(st AInt_25 != Fw)) &&  ((st AInt_21 == Fw) && (st AInt_21 != Off) && (st
AInt_21 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_14 != R)
&& (st AInt_13 != R) && (st AInt_12 != R) && (st AInt_11 != R));

    stigma1  = (st ADest1 == R) && ((st AInt_21 == R) && (st AInt_21 != Off) &&
(st AInt_21 != Fw)) && (st ASource1 == T) && ((st AInt_13 == Fw) && (st AInt_13
!= Off) && (st AInt_13 != R)) && ((st AInt_15 != R) && (st AInt_14 != R) && (st
AInt_12 != R) && (st AInt_11 != R));

    qoopa1   = (st ADest1 == R) && ((st AInt_13 == R) && (st AInt_13 != Off) &&
(st AInt_13 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    lambda2  = lambda;

    tau2     = (st ASource2 == T) && ((st AInt_21 == Fw) && (st AInt_21 != Off)
&& (st AInt_21 != R)) && ((st AInt_25 == I) && (AInt_25 != Off) && (AInt_25 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_22 != R) && (AInt_23 != R) &&
(AInt_24 != R));

    gama2    = (st ASource2 == T) && ((st AInt_25 == Fw) && (st AInt_25 != Off)
&& (st AInt_25 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) &&
(AInt_22 != R) && (AInt_23 != R) && (AInt_24 != R));

    psi2     = (st ASource2 == T) && ((st AInt_22 == Fw) && (st AInt_22 != Off)
&& (st AInt_22 != R)) && ((st AInt_24 == I) && (AInt_24 != Off) && (AInt_24 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) && (AInt_23 != R) &&
(AInt_25 != R));

    ro2      = (st ASource2 == T) && ((st AInt_24 == Fw) && (st AInt_24 != Off)
&& (st AInt_24 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) &&
(AInt_22 != R) && (AInt_23 != R) && (AInt_25 != R));

    niu2     = (st ASource2 == T) && ((st AInt_23 == Fw) && (st AInt_23 != Off)
&& (st AInt_23 != R)) && ((st AInt_24 == I) && (AInt_24 != Off) && (AInt_24 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) && (AInt_22 != R) &&
(AInt_25 != R));

    chi2     = (st ASource2 == T) && ((st AInt_24 == Fw) && (st AInt_24 != Off)
&& (st AInt_24 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) &&
(AInt_22 != R) && (AInt_23 != R) && (AInt_25 != R));

    alfa2    = (st ADest2 == R) && ((st AInt_25 == Fw) && (st AInt_25 != Off) &&
(st AInt_25 != R)) && (st ASource2 == T) && ((st AInt_24 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_21 != R));
```

```
    beta2    = (st ADest2 == R) && ((st AInt_25 == R) && (st AInt_25 != Off) &&
(st AInt_25 != Fw)) && (st ASource2 == T) && ((st AInt_21 == Fw) && (st AInt_21
!= Off) && (st AInt_21 != R)) && ((st AInt_24 != R) && (st AInt_23 != R) && (st
AInt_22 != R));

    delta2   = (st ADest2 == R) && ((st AInt_21 == R) && (st AInt_21 != Off) &&
(st AInt_21 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_23 != R) && (st AInt_22 != R));

    omega2   = (st ADest2 == R) && ((st AInt_24 == Fw) && (st AInt_24 != Off) &&
(st AInt_24 != R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_21 != R));

    sigma2   = (st ADest2 == R) && ((st AInt_24 == R) && (st AInt_24 != Off) &&
(st AInt_24 != Fw)) && (st ASource2 == T) && ((st AInt_22 == Fw) && (st AInt_22
!= Off) && (st AInt_22 != R)) && ((st AInt_25 != R) && (st AInt_23 != R) && (st
AInt_21 != R));

    sho2     = (st ADest2 == R) && ((st AInt_22 == R) && (st AInt_22 != Off) &&
(st AInt_22 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_23 != R) && (st AInt_21 != R));

    kapa2    = (st ADest2 == R) && ((st AInt_24 == Fw) && (st AInt_24 != Off) &&
(st AInt_24 != R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_21 != R));

    teta2    = (st ADest2 == R) && ((st AInt_24 == R) && (st AInt_24 != Off) &&
(st AInt_24 != Fw)) && ((st AInt_23 == Fw) && (AInt_23 != Off) && (AInt_23 !=
R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_22 != R) && (st
AInt_21 != R));

    zeta2    = (st ADest2 == R) && ((st AInt_23 == R) && (st AInt_23 != Off) &&
(st AInt_23 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_22 != R) && (st AInt_21 != R));

events

  loc on11  f11;
  loc off11 f12;
  loc on12  f13;
  loc off12 f14;
  loc on13  f15;
  loc off13 f16;
  loc on14  f17;
  loc off14 f18;
  loc on15  f19;
  loc off15 f110;
  loc on21  f21;
  loc off21 f22;
  loc on22  f23;
  loc off22 f24;
  loc on23  f25;
  loc off23 f26;
  loc on24  f27;
  loc off24 f28;
  loc on25  f29;
  loc off25 f210;

  syn rreq1   lambda1;
  syn s11     tau1;
  syn s12     gama1;
```

```
  syn s13     psi1;
  syn s14     ro1;
  syn s15     niu1;
  syn s16     chi1;
  syn s151    eta1;
  syn s152    epsilon1;
  syn rrep11  alfa1;
  syn rrep12  beta1;
  syn rrep13  delta1;
  syn rrep14  omega1;
  syn rrep15  sigma1;
  syn rrep16  sho1;
  syn rrep17  kapa1;
  syn rrep18  teta1;
  syn rrep19  zeta1;
  syn rrep153 upsilon1;
  syn rrep154 iota1;
  syn rrep155 stigma1;
  syn rrep156 qoopa1;
  syn rreq2   lambda2;
  syn s21     tau2;
  syn s22     gama2;
  syn s23     psi2;
  syn s24     ro2;
  syn s25     niu2;
  syn s26     chi2;
  syn rrep21  alfa2;
  syn rrep22  beta2;
  syn rrep23  delta2;
  syn rrep24  omega2;
  syn rrep25  sigma2;
  syn rrep26  sho2;
  syn rrep27  kapa2;
  syn rrep28  teta2;
  syn rrep29  zeta2;

partial reachability = (((st ASource1 == T) && (st ADest1 == I) && ((st AInt_11
== Fw) && (st AInt_15 == Fw)) && ((st AInt_12 != R) && (st AInt_12 != Fw) && (st
AInt_12 != Off)) && ((st AInt_13 != R) && (st AInt_13 != Fw) && (st AInt_13 !=
Off)) && ((st AInt_14 != R) && (st AInt_14 != Fw) && (st AInt_14 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) && (st AInt_14 ==
Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) && (st AInt_11 != Off)) && ((st
AInt_13 != R) && (st AInt_13 != Fw) && (st AInt_13 != Off)) && ((st AInt_15 !=
R) && (st AInt_15 != Fw) && (st AInt_15 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) && (st AInt_14 ==
Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) && (st AInt_11 != Off)) && ((st
AInt_12 != R) && (st AInt_12 != Fw) && (st AInt_12 != Off)) && ((st AInt_15 !=
R) && (st AInt_15 != Fw) && (st AInt_15 != Off))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) && (st AInt_21 ==
Fw) && (st AInt_25 == Fw)) && ((st AInt_11 != R) && (st AInt_11 != Fw) && (st
AInt_11 != Off)) && ((st AInt_12 != R) && (st AInt_12 != Fw) && (st AInt_12 !=
Off)) && ((st AInt_14 != R) && (st AInt_14 != Fw) && (st AInt_14 != Off)) &&
((st AInt_15 != R) && (st AInt_15 != Fw) && (st AInt_15 != Off)))) &&
(((st ASource2 == T) && (st ADest2 == I) && ((st AInt_21 == Fw) && (st AInt_25
== Fw)) && ((st AInt_22 != R) && (st AInt_22 != Fw) && (st AInt_22 != Off)) &&
((st AInt_23 != R) && (st AInt_23 != Fw) && (st AInt_23 != Off)) && ((st AInt_24
!= R) && (st AInt_24 != Fw) && (st AInt_24 != Off))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) && (st AInt_24 ==
Fw)) && ((st AInt_21 != R) && (st AInt_21 != Fw) && (st AInt_21 != Off)) && ((st
AInt_23 != R) && (st AInt_23 != Fw) && (st AInt_23 != Off)) && ((st AInt_25 !=
R) && (st AInt_25 != Fw) && (st AInt_25 != Off))) ||
```

```
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) && (st AInt_24 ==
Fw)) && ((st AInt_22 != R) && (st AInt_22 != Fw) && (st AInt_22 != Off)) && ((st
AInt_21 != R) && (st AInt_21 != Fw) && (st AInt_21 != Off)) && ((st AInt_25 !=
R) && (st AInt_25 != Fw) && (st AInt_25 != Off))));

network Adsr (continuous)

  aut ASource1
    stt T to (R) rreq1
    stt R to (T) rrep13 rrep16 rrep19 rrep156

  aut AInt_11
    stt I to (Off) off11
          to (Fw) rreq1
    stt Off to (I) on11
    stt Fw to (Fw) s11
           to (R) rrep12
    stt R to (I) rrep13

  aut AInt_12
    stt I to (Off) off12
          to (Fw) rreq1
    stt Off to (I) on12
    stt Fw to (Fw) s13
           to (R) rrep15
    stt R  to (I) rrep16

  aut AInt_13
    stt I to (Off) off13
          to (Fw) rreq1
    stt Off to (I) on13
    stt Fw to (Fw) s15
           to (R) rrep18 rrep155
    stt R  to (I) rrep19 rrep156

  aut AInt_14
    stt I to (Off) off14
          to (Fw) s13 s15
    stt Off to (I) on14
    stt Fw to (Fw) s14 s16
           to (R) rrep14 rrep17
    stt R  to (I) rrep15 rrep18

  aut AInt_15
    stt I to (Off) off15
          to (Fw) s11
    stt Off to (I) on15
    stt Fw to (Fw) s12
           to (R) rrep11
    stt R to (I) rrep12

  aut ADest1
    stt I to (R) s12 s14 s16 s152
    stt R to (I) rrep11 rrep14 rrep17 rrep153

  aut ASource2
    stt T to (R) rreq2
    stt R to (T) rrep23 rrep26 rrep29

  aut AInt_21
    stt I to (Off) off21
```

```
          to (Fw) rreq2 s15
    stt Off to (I) on21
    stt Fw to (Fw) s21 s151
           to (R) rrep22 rrep154
    stt R to (I) rrep23 rrep155

  aut AInt_22
    stt I to (Off) off22
          to (Fw) rreq2
    stt Off to (I) on22
    stt Fw to (Fw) s23
           to (R) rrep25
    stt R to (I) rrep26

  aut AInt_23
    stt I to (Off) off23
          to (Fw) rreq2
    stt Off to (I) on23
    stt Fw to (Fw) s25
           to (R) rrep28
    stt R to (I) rrep29

  aut AInt_24
    stt I to (Off) off24
          to (Fw) s23 s25
    stt Off to (I) on24
    stt Fw to (Fw) s24 s26
           to (R) rrep24 rrep27
    stt R to (I) rrep25 rrep28

  aut AInt_25
    stt I to (Off) off25
          to (Fw) s21 s16 s151
    stt Off to (I) on25
    stt Fw to (Fw) s22 s152
           to (R) rrep21 rrep153
    stt R to (I) rrep22 rrep154

  aut ADest2
    stt I to (R) s22 s24 s26
    stt R to (I) rrep21 rrep24 rrep27

results

  t_AInt_1115_64  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_1115_128  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))/ tx_pause;
  t_AInt_1115_256  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))/ tx_pause;
  t_AInt_1115_512  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))/ tx_pause;

  t_AInt_1214_64  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_1214_128  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))/ tx_pause;
  t_AInt_1214_256  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))/ tx_pause;
  t_AInt_1214_512  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))/ tx_pause;
```

```
  t_AInt_1314_64  = ((((st AInt_13 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_1314_128  = ((((st AInt_13 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))/ tx_pause;
  t_AInt_1314_256  = ((((st AInt_13 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))/ tx_pause;
  t_AInt_1314_512  = ((((st AInt_13 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))/ tx_pause;

  t_AInt_132125_64  = ((((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st AInt_25
== Fw)) * bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/
tx_pause;
  t_AInt_132125_128  = ((((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st
AInt_25 == Fw)) * bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))
/ tx_pause;
  t_AInt_132125_256  = ((((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st
AInt_25 == Fw)) * bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))
/ tx_pause;
  t_AInt_132125_512  = ((((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st
AInt_25 == Fw)) * bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))
/ tx_pause;

  t_Asource1 = (st ASource1 == Fw);

  t_AInt_11_Off = (st AInt_11 == Off);
  t_AInt_12_Off = (st AInt_12 == Off);
  t_AInt_13_Off = (st AInt_13 == Off);
  t_AInt_14_Off = (st AInt_14 == Off);
  t_AInt_15_Off = (st AInt_15 == Off);

  t_AInt_11_Rep = (st AInt_11 == R);
  t_AInt_12_Rep = (st AInt_12 == R);
  t_AInt_13_Rep = (st AInt_13 == R);
  t_AInt_14_Rep = (st AInt_14 == R);
  t_AInt_15_Rep = (st AInt_15 == R);

  t_AInt_11_Fw = (st AInt_11 == Fw);
  t_AInt_12_Fw = (st AInt_12 == Fw);
  t_AInt_13_Fw = (st AInt_13 == Fw);
  t_AInt_14_Fw = (st AInt_14 == Fw);
  t_AInt_15_Fw = (st AInt_15 == Fw);

  t_AInt_2125_64  = ((((st AInt_21 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_2125_128  = ((((st AInt_21 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))/ tx_pause;
  t_AInt_2125_256  = ((((st AInt_21 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))/ tx_pause;
  t_AInt_2125_512  = ((((st AInt_21 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))/ tx_pause;

  t_AInt_2224_64  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_2224_128  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))/ tx_pause;
  t_AInt_2224_256  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))/ tx_pause;
  t_AInt_2224_512  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))/ tx_pause;
```

```
  t_AInt_2324_64  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_2324_128  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))/ tx_pause;
  t_AInt_2324_256  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))/ tx_pause;
  t_AInt_2324_512  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))/ tx_pause;


  t_Asource2 = (st ASource2 == Fw);

  t_AInt_21_Off = (st AInt_21 == Off);
  t_AInt_22_Off = (st AInt_22 == Off);
  t_AInt_23_Off = (st AInt_23 == Off);
  t_AInt_24_Off = (st AInt_24 == Off);
  t_AInt_25_Off = (st AInt_25 == Off);

  t_AInt_21_Rep = (st AInt_21 == R);
  t_AInt_22_Rep = (st AInt_22 == R);
  t_AInt_23_Rep = (st AInt_23 == R);
  t_AInt_24_Rep = (st AInt_24 == R);
  t_AInt_25_Rep = (st AInt_25 == R);

  t_AInt_21_Fw = (st AInt_21 == Fw);
  t_AInt_22_Fw = (st AInt_22 == Fw);
  t_AInt_23_Fw = (st AInt_23 == Fw);
  t_AInt_24_Fw = (st AInt_24 == Fw);
  t_AInt_25_Fw = (st AInt_25 == Fw);

  t_Route_1115_Fw = ((st AInt_11 == Fw) && (st AInt_15 == Fw));
  t_Route_1214_Fw = ((st AInt_12 == Fw) && (st AInt_14 == Fw));
  t_Route_1314_Fw = ((st AInt_13 == Fw) && (st AInt_14 == Fw));
  t_Route_132125_Fw = ((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st AInt_25
== Fw));
  t_Route_2125_Fw = ((st AInt_21 == Fw) && (st AInt_25 == Fw));
  t_Route_2224_Fw = ((st AInt_22 == Fw) && (st AInt_24 == Fw));
  t_Route_2324_Fw = ((st AInt_23 == Fw) && (st AInt_24 == Fw));
```

APÊNDICE D – SAN PARA REDES COM ESCUTA PROMÍSCUA COM PROBABILIDADES DE FALHAS

SAN da Figura 5.11, modelo para redes com escuta promíscua com probabilidades de falhas na transmissão.

```
identifiers
    f11      = (st AInt_11 != Off) * 0.8;
    f12      = (st AInt_11 == Off) * 0.2;
    f13      = (st AInt_12 != Off) * 0.8;
    f14      = (st AInt_12 == Off) * 0.2;
    f15      = (st AInt_13 != Off) * 0.8;
    f16      = (st AInt_13 == Off) * 0.2;
    f17      = (st AInt_14 != Off) * 0.8;
    f18      = (st AInt_14 == Off) * 0.2;
    f19      = (st AInt_15 != Off) * 0.8;
    f110     = (st AInt_15 == Off) * 0.2;
    f21      = (st AInt_21 != Off) * 0.8;
    f22      = (st AInt_21 == Off) * 0.2;
    f23      = (st AInt_22 != Off) * 0.8;
    f24      = (st AInt_22 == Off) * 0.2;
    f25      = (st AInt_23 != Off) * 0.8;
    f26      = (st AInt_23 == Off) * 0.2;
    f27      = (st AInt_24 != Off) * 0.8;
    f28      = (st AInt_24 == Off) * 0.2;
    f29      = (st AInt_25 != Off) * 0.8;
    f210     = (st AInt_25 == Off) * 0.2;
    bandwidth_b = 2000000;                    // bandwidth in bits/seconds
    bandwidth_B = bandwidth_b/8;              // bandwidth in bytes/seconds
    bandwidth_Mbps = bandwidth_b/1000000;     // bandwidth in bytes/seconds
    size_pack_64 = 64;                        // package size in bytes
    size_pack_128 = 128;
    size_pack_256 = 256;
    size_pack_512 = 512;
    NXH = 7;                                  // DSR Next Header field in bytes
    RESVD = 8;                                // DSR Reserved Header in bytes
    MAC = 47;                                 // Medium Access Control header in
bytes
    PALG = 16;                              // DSR Payload Length header in bytes
    dsr_header = NXH+RESVD+MAC+PALG;          // headers
    ocur_rate = 500;                        // ocurrence rate of a route request
message in miliseconds
    lambda = 0.5;                 // ocurrence rate of a route request message
in seconds (1 second = 1000 miliseconds)
    tx_tpause = 0.1777;           // ocurrence rate of time pause (for 5.625
seconds, according to established time pause)

    lambda1 = lambda;

    tau1     = (st ASource1 == T) && ((st AInt_11 == Fw) && (st AInt_11 != Off)
&& (st AInt_11 != R)) && ((st AInt_15 == I) && (st AInt_15 != Off) && (st
AInt_15 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_12 != R) &&
(st AInt_13 != R) && (st AInt_14 != R));

    gama1    = (((st ASource1 == T) && ((st AInt_15 == Fw) && (st AInt_15 !=
Off) && (st AInt_15 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_13 != R) && (st AInt_14 != R))));
```

```
    psi1     = (st ASource1 == T) && ((st AInt_12 == Fw) && (st AInt_12 != Off)
&& (st AInt_12 != R)) && ((st AInt_14 == I) && (st AInt_14 != Off) && (st
AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) &&
(st AInt_13 != R) && (st AInt_15 != R));

    ro1      = (((st ASource1 == T) && ((st AInt_14 == Fw) && (st AInt_14 !=
Off) && (st AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_13 != R) && (st AInt_15 != R))));

    niu1     = (st ASource1 == T) && ((st AInt_13 == Fw) && (st AInt_13 != Off)
&& (st AInt_13 != R))  && ((st AInt_14 == I) && (st AInt_14 != Off) && (st
AInt_14 != R))  && ((st AInt_21 == I) && (st AInt_21 != Off) && (st AInt_21 !=
R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) && (st AInt_12
!= R) && (st AInt_15 != R));

    chi1     = (((st ASource1 == T) && ((st AInt_14 == Fw) && (st AInt_14 !=
Off) && (st AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_13 != R) && (st AInt_15 != R))));

    eta1     = (st ASource1 == T) && ((st AInt_21 == Fw) && (st AInt_21 != Off)
&& (st AInt_21 != R))   && ((st AInt_25 == I) && (st AInt_25 != Off) && (st
AInt_25 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) &&
(st AInt_12 != R) && (st AInt_13 != R) && (st AInt_14 != R) && (st AInt_15 !=
R));

    epsilon1  = (((st ASource1 == T) && ((st AInt_25 == Fw) && (st AInt_25 !=
Off) && (st AInt_25 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_13 != R) && (st AInt_14 != R) &&
(st AInt_15 != R))));

    alfa1    = (st ADest1 == R) && ((st AInt_15 == Fw) && (st AInt_15 != Off) &&
(st AInt_15 != R)) && (st ASource1 == T) && ((st AInt_14 != R) && (st AInt_13 !=
R) && (st AInt_12 != R) && (st AInt_11 != R));

    beta1    = (st ADest1 == R) && ((st AInt_15 == R) && (st AInt_15 != Off) &&
(st AInt_15 != Fw)) && (st ASource1 == T) && ((st AInt_11 == Fw) && (st AInt_11
!= Off) && (st AInt_11 != R)) && ((st AInt_14 != R) && (st AInt_13 != R) && (st
AInt_11 != R));

    delta1   = (st ADest1 == R) && ((st AInt_11 == R) && (st AInt_11 != Off) &&
(st AInt_11 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_13 != R)) && (st AInt_12 != R);

    omega1   = (st ADest1 == R) && ((st AInt_14 == Fw) && (st AInt_14 != Off) &&
(st AInt_14 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_13 !=
R) && (st AInt_12 != R) && (st AInt_11 != R));

    sigma1   = (st ADest1 == R) && ((st AInt_14 == R) && (st AInt_14 != Off) &&
(st AInt_14 != Fw)) && (st ASource1 == T) && ((st AInt_12 == Fw) && (st AInt_12
!= Off) && (st AInt_12 != R)) && ((st AInt_15 != R) && (st AInt_13 != R) && (st
AInt_11 != R));

    sho1     = (st ADest1 == R) && ((st AInt_12 == R) && (st AInt_12 != Off) &&
(st AInt_12 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_13 != R) && (st AInt_11 != R));

    kapa1    = (st ADest1 == R) && ((st AInt_14 == Fw) && (st AInt_14 != Off) &&
(st AInt_14 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_13 !=
R) && (st AInt_12 != R) && (st AInt_11 != R));
```

```
    teta1    = (st ADest1 == R) && ((st AInt_14 == R) && (st AInt_14 != Off) &&
(st AInt_14 != Fw)) && ((st AInt_13 == Fw) && (st AInt_13 != Off) && (st AInt_13
!= R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_12 != R) && (st
AInt_11 != R));

    zeta1    = (st ADest1 == R) && ((st AInt_13 == R) && (st AInt_13 != Off) &&
(st AInt_13 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    upsilon1 = (st ADest1 == R) && ((st AInt_25 == Fw) && (st AInt_25 != Off) &&
(st AInt_25 != R)) && (st ASource1 == T) &&  ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_13 != R) && (st AInt_12 != R) && (st AInt_11 != R));

    iota1    = (st ADest1 == R) && ((st AInt_25 == R) && (st AInt_25 != Off) &&
(st AInt_25 != Fw)) &&  ((st AInt_21 == Fw) && (st AInt_21 != Off) && (st
AInt_21 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_14 != R)
&& (st AInt_13 != R) && (st AInt_12 != R) && (st AInt_11 != R));

    stigma1  = (st ADest1 == R) && ((st AInt_21 == R) && (st AInt_21 != Off) &&
(st AInt_21 != Fw)) && (st ASource1 == T) && ((st AInt_13 == Fw) && (st AInt_13
!= Off) && (st AInt_13 != R)) && ((st AInt_15 != R) && (st AInt_14 != R) && (st
AInt_12 != R) && (st AInt_11 != R));

    qoopa1   = (st ADest1 == R) && ((st AInt_13 == R) && (st AInt_13 != Off) &&
(st AInt_13 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    lambda2  = lambda;

    tau2     = (st ASource2 == T) && ((st AInt_21 == Fw) && (st AInt_21 != Off)
&& (st AInt_21 != R)) && ((st AInt_25 == I) && (AInt_25 != Off) && (AInt_25 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_22 != R) && (AInt_23 != R) &&
(AInt_24 != R));

    gama2    = (st ASource2 == T) && ((st AInt_25 == Fw) && (st AInt_25 != Off)
&& (st AInt_25 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) &&
(AInt_22 != R) && (AInt_23 != R) && (AInt_24 != R));

    psi2     = (st ASource2 == T) && ((st AInt_22 == Fw) && (st AInt_22 != Off)
&& (st AInt_22 != R)) && ((st AInt_24 == I) && (AInt_24 != Off) && (AInt_24 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) && (AInt_23 != R) &&
(AInt_25 != R));

    ro2      = (st ASource2 == T) && ((st AInt_24 == Fw) && (st AInt_24 != Off)
&& (st AInt_24 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) &&
(AInt_22 != R) && (AInt_23 != R) && (AInt_25 != R));

    niu2     = (st ASource2 == T) && ((st AInt_23 == Fw) && (st AInt_23 != Off)
&& (st AInt_23 != R)) && ((st AInt_24 == I) && (AInt_24 != Off) && (AInt_24 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) && (AInt_22 != R) &&
(AInt_25 != R));

    chi2     = (st ASource2 == T) && ((st AInt_24 == Fw) && (st AInt_24 != Off)
&& (st AInt_24 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_21 != R) &&
(AInt_22 != R) && (AInt_23 != R) && (AInt_25 != R));

    alfa2    = (st ADest2 == R) && ((st AInt_25 == Fw) && (st AInt_25 != Off) &&
(st AInt_25 != R)) && (st ASource2 == T) && ((st AInt_24 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_21 != R));
```

```
    beta2    = (st ADest2 == R) && ((st AInt_25 == R) && (st AInt_25 != Off) &&
(st AInt_25 != Fw)) && (st ASource2 == T) && ((st AInt_21 == Fw) && (st AInt_21
!= Off) && (st AInt_21 != R)) && ((st AInt_24 != R) && (st AInt_23 != R) && (st
AInt_22 != R));

    delta2   = (st ADest2 == R) && ((st AInt_21 == R) && (st AInt_21 != Off) &&
(st AInt_21 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_23 != R) && (st AInt_22 != R));

    omega2   = (st ADest2 == R) && ((st AInt_24 == Fw) && (st AInt_24 != Off) &&
(st AInt_24 != R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_21 != R));

    sigma2   = (st ADest2 == R) && ((st AInt_24 == R) && (st AInt_24 != Off) &&
(st AInt_24 != Fw)) && (st ASource2 == T) && ((st AInt_22 == Fw) && (st AInt_22
!= Off) && (st AInt_22 != R)) && ((st AInt_25 != R) && (st AInt_23 != R) && (st
AInt_21 != R));

    sho2     = (st ADest2 == R) && ((st AInt_22 == R) && (st AInt_22 != Off) &&
(st AInt_22 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_23 != R) && (st AInt_21 != R));

    kapa2    = (st ADest2 == R) && ((st AInt_24 == Fw) && (st AInt_24 != Off) &&
(st AInt_24 != R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_21 != R));

    teta2    = (st ADest2 == R) && ((st AInt_24 == R) && (st AInt_24 != Off) &&
(st AInt_24 != Fw)) && ((st AInt_23 == Fw) && (AInt_23 != Off) && (AInt_23 !=
R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_22 != R) && (st
AInt_21 != R));

    zeta2    = (st ADest2 == R) && ((st AInt_23 == R) && (st AInt_23 != Off) &&
(st AInt_23 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_22 != R) && (st AInt_21 != R));

events

  loc on11  f11;
  loc off11 f12;
  loc on12  f13;
  loc off12 f14;
  loc on13  f15;
  loc off13 f16;
  loc on14  f17;
  loc off14 f18;
  loc on15  f19;
  loc off15 f110;
  loc on21  f21;
  loc off21 f22;
  loc on22  f23;
  loc off22 f24;
  loc on23  f25;
  loc off23 f26;
  loc on24  f27;
  loc off24 f28;
  loc on25  f29;
  loc off25 f210;

  syn rreq1   lambda1;
  syn s11     tau1;
  syn s12     gama1;
```

```
  syn s13     psi1;
  syn s14     ro1;
  syn s15     niu1;
  syn s16     chi1;
  syn s151    eta1;
  syn s152    epsilon1;
  syn rrep11  alfa1;
  syn rrep12  beta1;
  syn rrep13  delta1;
  syn rrep14  omega1;
  syn rrep15  sigma1;
  syn rrep16  sho1;
  syn rrep17  kapa1;
  syn rrep18  teta1;
  syn rrep19  zeta1;
  syn rrep153 upsilon1;
  syn rrep154 iota1;
  syn rrep155 stigma1;
  syn rrep156 qoopa1;
  syn rreq2   lambda2;
  syn s21     tau2;
  syn s22     gama2;
  syn s23     psi2;
  syn s24     ro2;
  syn s25     niu2;
  syn s26     chi2;
  syn rrep21  alfa2;
  syn rrep22  beta2;
  syn rrep23  delta2;
  syn rrep24  omega2;
  syn rrep25  sigma2;
  syn rrep26  sho2;
  syn rrep27  kapa2;
  syn rrep28  teta2;
  syn rrep29  zeta2;

partial reachability = (((st ASource1 == T) && (st ADest1 == I) && ((st AInt_11
== Fw) && (st AInt_15 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) && (st AInt_14 ==
Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) && (st AInt_14 ==
Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_13 == Fw) && (st AInt_21 ==
Fw) && (st AInt_25 == Fw)))) &&
(((st ASource2 == T) && (st ADest2 == I) && ((st AInt_21 == Fw) && (st AInt_25
== Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) && (st AInt_24 ==
Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) && (st AInt_24 ==
Fw))));

network Adsr (continuous)

  aut ASource1
    stt T to (R) rreq1
    stt R to (T) rrep13 rrep16 rrep19 rrep156

  aut AInt_11
    stt I to (Off) off11
          to (Fw) rreq1
    stt Off to (I) on11
```

```
  stt Fw to (Fw) s11
         to (R) rrep12
  stt R to (I) rrep13

aut AInt_12
  stt I to (Off) off12
        to (Fw) rreq1
  stt Off to (I) on12
  stt Fw to (Fw) s13
         to (R) rrep15
  stt R  to (I) rrep16

aut AInt_13
  stt I to (Off) off13
        to (Fw) rreq1
  stt Off to (I) on13
  stt Fw to (Fw) s15
         to (R) rrep18 rrep155
  stt R  to (I) rrep19 rrep156

aut AInt_14
  stt I to (Off) off14
        to (Fw) s13 s15
  stt Off to (I) on14
  stt Fw to (Fw) s14 s16
         to (R) rrep14 rrep17
  stt R  to (I) rrep15 rrep18

aut AInt_15
  stt I to (Off) off15
        to (Fw) s11
  stt Off to (I) on15
  stt Fw to (Fw) s12
         to (R) rrep11
  stt R to (I) rrep12

aut ADest1
  stt I to (R) s12 s14 s16 s152
  stt R to (I) rrep11 rrep14 rrep17 rrep153

aut ASource2
  stt T to (R) rreq2
  stt R to (T) rrep23 rrep26 rrep29

aut AInt_21
  stt I to (Off) off21
        to (Fw) rreq2 s15
  stt Off to (I) on21
  stt Fw to (Fw) s21 s151
         to (R) rrep22 rrep154
  stt R to (I) rrep23 rrep155

aut AInt_22
  stt I to (Off) off22
        to (Fw) rreq2
  stt Off to (I) on22
  stt Fw to (Fw) s23
         to (R) rrep25
  stt R to (I) rrep26

aut AInt_23
```

```
    stt I to (Off) off23
          to (Fw) rreq2
    stt Off to (I) on23
    stt Fw to (Fw) s25
           to (R) rrep28
    stt R to (I) rrep29

  aut AInt_24
    stt I to (Off) off24
          to (Fw) s23 s25
    stt Off to (I) on24
    stt Fw to (Fw) s24 s26
           to (R) rrep24 rrep27
    stt R to (I) rrep25 rrep28

  aut AInt_25
    stt I to (Off) off25
          to (Fw) s21 s16 s151
    stt Off to (I) on25
    stt Fw to (Fw) s22 s152
           to (R) rrep21 rrep153
    stt R to (I) rrep22 rrep154

  aut ADest2
    stt I to (R) s22 s24 s26
    stt R to (I) rrep21 rrep24 rrep27

results

  t_AInt_1115_64  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_1115_128  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))/ tx_pause;
  t_AInt_1115_256  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))/ tx_pause;
  t_AInt_1115_512  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))/ tx_pause;

  t_AInt_1214_64  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_1214_128  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))/ tx_pause;
  t_AInt_1214_256  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))/ tx_pause;
  t_AInt_1214_512  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))/ tx_pause;

  t_AInt_1314_64  = ((((st AInt_13 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_1314_128  = ((((st AInt_13 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))/ tx_pause;
  t_AInt_1314_256  = ((((st AInt_13 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))/ tx_pause;
  t_AInt_1314_512  = ((((st AInt_13 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))/ tx_pause;

  t_AInt_132125_64  = ((((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st AInt_25
== Fw)) * bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) /
tx_pause;
```

```
  t_AInt_132125_128  = ((((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st
AInt_25 == Fw)) * bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header))))
/ tx_pause;
  t_AInt_132125_256  = ((((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st
AInt_25 == Fw)) * bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header))))
/ tx_pause;
  t_AInt_132125_512  = ((((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st
AInt_25 == Fw)) * bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header))))
/ tx_pause;

  t_Asource1 = (st ASource1 == Fw);

  t_AInt_11_Off = (st AInt_11 == Off);
  t_AInt_12_Off = (st AInt_12 == Off);
  t_AInt_13_Off = (st AInt_13 == Off);
  t_AInt_14_Off = (st AInt_14 == Off);
  t_AInt_15_Off = (st AInt_15 == Off);

  t_AInt_11_Rep = (st AInt_11 == R);
  t_AInt_12_Rep = (st AInt_12 == R);
  t_AInt_13_Rep = (st AInt_13 == R);
  t_AInt_14_Rep = (st AInt_14 == R);
  t_AInt_15_Rep = (st AInt_15 == R);

  t_AInt_11_Fw = (st AInt_11 == Fw);
  t_AInt_12_Fw = (st AInt_12 == Fw);
  t_AInt_13_Fw = (st AInt_13 == Fw);
  t_AInt_14_Fw = (st AInt_14 == Fw);
  t_AInt_15_Fw = (st AInt_15 == Fw);

  t_AInt_2125_64  = ((((st AInt_21 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) / tx_pause;
  t_AInt_2125_128  = ((((st AInt_21 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_2125_256  = ((((st AInt_21 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_2125_512  = ((((st AInt_21 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;

  t_AInt_2224_64  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) / tx_pause;
  t_AInt_2224_128  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_2224_256  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_2224_512  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;

  t_AInt_2324_64  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) / tx_pause;
  t_AInt_2324_128  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_2324_256  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_2324_512  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;


  t_Asource2 = (st ASource2 == Fw);

  t_AInt_21_Off = (st AInt_21 == Off);
```

```
  t_AInt_22_Off = (st AInt_22 == Off);
  t_AInt_23_Off = (st AInt_23 == Off);
  t_AInt_24_Off = (st AInt_24 == Off);
  t_AInt_25_Off = (st AInt_25 == Off);

  t_AInt_21_Rep = (st AInt_21 == R);
  t_AInt_22_Rep = (st AInt_22 == R);
  t_AInt_23_Rep = (st AInt_23 == R);
  t_AInt_24_Rep = (st AInt_24 == R);
  t_AInt_25_Rep = (st AInt_25 == R);

  t_AInt_21_Fw = (st AInt_21 == Fw);
  t_AInt_22_Fw = (st AInt_22 == Fw);
  t_AInt_23_Fw = (st AInt_23 == Fw);
  t_AInt_24_Fw = (st AInt_24 == Fw);
  t_AInt_25_Fw = (st AInt_25 == Fw);

  t_Route_1115_Fw = ((st AInt_11 == Fw) && (st AInt_15 == Fw));
  t_Route_1214_Fw = ((st AInt_12 == Fw) && (st AInt_14 == Fw));
  t_Route_1314_Fw = ((st AInt_13 == Fw) && (st AInt_14 == Fw));
  t_Route_132125_Fw = ((st AInt_13 == Fw) && (st AInt_21 == Fw) && (st AInt_25
== Fw));
  t_Route_2125_Fw = ((st AInt_21 == Fw) && (st AInt_25 == Fw));
  t_Route_2224_Fw = ((st AInt_22 == Fw) && (st AInt_24 == Fw));
  t_Route_2324_Fw = ((st AInt_23 == Fw) && (st AInt_24 == Fw));
```

APÊNDICE E – SAN PARA REDES COM AGREGAÇÃO DE SERVIÇOS DE ROTEAMENTO SEM PROBABILIDADES DE FALHAS

SAN da Figura 5.13, modelo para redes com agregação de serviços de roteamento sem probabilidades de falhas na transmissão.

```
identifiers
    f11      = (st AInt_11 != Off) * 0.8;
    f12      = (st AInt_11 == Off) * 0.2;
    f13      = (st AInt_12 != Off) * 0.8;
    f14      = (st AInt_12 == Off) * 0.2;
    f15      = (((st AInt_1321 != Off) * 0.8));
    f16      = (((st AInt_1321 == Off) * 0.2));
    f17      = (st AInt_14 != Off) * 0.8;
    f18      = (st AInt_14 == Off) * 0.2;
    f19      = (st AInt_15 != Off) * 0.8;
    f110     = (st AInt_15 == Off) * 0.2;
    f23      = (st AInt_22 != Off) * 0.8;
    f24      = (st AInt_22 == Off) * 0.2;
    f25      = (st AInt_23 != Off) * 0.8;
    f26      = (st AInt_23 == Off) * 0.2;
    f27      = (st AInt_24 != Off) * 0.8;
    f28      = (st AInt_24 == Off) * 0.2;
    f29      = (st AInt_25 != Off) * 0.8;
    f210     = (st AInt_25 == Off) * 0.2;
    bandwidth_b = 2000000;                    // bandwidth in bits/seconds
    bandwidth_B = bandwidth_b/8;              // bandwidth in bytes/seconds
    bandwidth_Mbps = bandwidth_b/1000000;     // bandwidth in bytes/seconds
    size_pack_64 = 64;                        // package size in bytes
    size_pack_128 = 128;
    size_pack_256 = 256;
    size_pack_512 = 512;
    NXH = 7;                                  // DSR Next Header field in bytes
    RESVD = 8;                                // DSR Reserved Header in bytes
    MAC = 47;                                 // Medium Access Control header in
bytes
    PALG = 16;                              // DSR Payload Length header in bytes
    dsr_header = NXH+RESVD+MAC+PALG;           // headers
    ocur_rate = 500;                        // ocurrence rate of a route request
message in miliseconds
    lambda = 0.5;                  // ocurrence rate of a route request message
in seconds (1 second = 1000 miliseconds)
    tx_tpause = 0.1777;            // ocurrence rate of time pause (for 5.625
seconds, according to established time pause)

    lambda1 = lambda;

    tau1     = (st ASource1 == T) && ((st AInt_11 == Fw) && (st AInt_11 != Off)
&& (st AInt_11 != R)) && ((st AInt_15 == I) && (st AInt_15 != Off) && (st
AInt_15 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_12 != R) &&
(st AInt_1321 != R) && (st AInt_14 != R));

    gama1    = (((st ASource1 == T) && ((st AInt_15 == Fw) && (st AInt_15 !=
Off) && (st AInt_15 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_1321 != R) && (st AInt_14 != R))));
```

```
    psi1     = (st ASource1 == T) && ((st AInt_12 == Fw) && (st AInt_12 != Off)
&& (st AInt_12 != R)) && ((st AInt_14 == I) && (st AInt_14 != Off) && (st
AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) &&
(st AInt_12 != R) && (st AInt_15 != R));

    ro1      = (((st ASource1 == T) && ((st AInt_14 == Fw) && (st AInt_14 !=
Off) && (st AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_1321 != R) && (st AInt_15 !=
R))));

    niu1     = (st ASource1 == T) && ((st AInt_1321 == Fw) && (st AInt_1321 !=
Off) && (st AInt_1321 != R))  && ((st AInt_14 == I) && (st AInt_14 != Off) &&
(st AInt_14 != R))  && ((st AInt_25 == I) && (st AInt_25 != Off) && (st AInt_25
!= R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) && (st
AInt_12 != R) && (st AInt_15 != R));

    chi1     = (((st ASource1 == T) && ((st AInt_14 == Fw) && (st AInt_14 !=
Off) && (st AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_1321 != R) && (st AInt_15 !=
R))));

    eta1     = (st ASource1 == T) && ((st AInt_25 == Fw) && (st AInt_25 != Off)
&& (st AInt_25!= R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 !=
R) && (st AInt_12 != R) && (st AInt_1321 != R) && (st AInt_14 != R) && (st
AInt_15 != R));

    alfa1    = (st ADest1 == R) && ((st AInt_15 == Fw) && (st AInt_15 != Off) &&
(st AInt_15 != R)) && (st ASource1 == T) && ((st AInt_14 != R) && (st AInt_1321
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    beta1    = (st ADest1 == R) && ((st AInt_15 == R) && (st AInt_15 != Off) &&
(st AInt_15 != Fw)) && (st ASource1 == T) && ((st AInt_11 == Fw) && (st AInt_11
!= Off) && (st AInt_11 != R)) && ((st AInt_14 != R) && (st AInt_1321 != R) &&
(st AInt_11 != R));

    delta1   = (st ADest1 == R) && ((st AInt_11 == R) && (st AInt_11 != Off) &&
(st AInt_11 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_1321 != R)) && (st AInt_12 != R);

    omega1   = (st ADest1 == R) && ((st AInt_14 == Fw) && (st AInt_14 != Off) &&
(st AInt_14 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_1321
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    sigma1   = (st ADest1 == R) && ((st AInt_14 == R) && (st AInt_14 != Off) &&
(st AInt_14 != Fw)) && (st ASource1 == T) && ((st AInt_12 == Fw) && (st AInt_12
!= Off) && (st AInt_12 != R)) && ((st AInt_15 != R) && (st AInt_1321 != R) &&
(st AInt_11 != R));

    sho1     = (st ADest1 == R) && ((st AInt_12 == R) && (st AInt_12 != Off) &&
(st AInt_12 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_1321 != R) && (st AInt_11 != R));

    kapa1    = (st ADest1 == R) && ((st AInt_14 == Fw) && (st AInt_14 != Off) &&
(st AInt_14 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_1321
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    teta1    = (st ADest1 == R) && ((st AInt_14 == R) && (st AInt_14 != Off) &&
(st AInt_14 != Fw)) && ((st AInt_1321 == Fw) && (st AInt_1321 != Off) && (st
AInt_1321 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_12 !=
R) && (st AInt_11 != R));
```

```
    zeta1    = (st ADest1 == R) && ((st AInt_1321 == R) && (st AInt_1321 != Off)
&& (st AInt_1321 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st
AInt_14 != R) && (st AInt_12 != R) && (st AInt_11 != R));

    upsilon1 = (st ADest1 == R) && ((st AInt_25 == Fw) && (st AInt_25 != Off) &&
(st AInt_25 != R)) && (st ASource1 == T) &&  ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_1321 != R) && (st AInt_12 != R) && (st AInt_11 != R));

    iota1    = (st ADest1 == R) && ((st AInt_25 == R) && (st AInt_25 != Off) &&
(st AInt_25 != Fw)) &&  ((st AInt_1321 == Fw) && (st AInt_1321 != Off) && (st
AInt_1321 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_14 !=
R) && (st AInt_12 != R) && (st AInt_11 != R));

    stigma1  = (st ADest1 == R) && ((st AInt_1321 == R) && (st AInt_1321 != Off)
&& (st AInt_1321 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st
AInt_14 != R) && (st AInt_12 != R) && (st AInt_11 != R));

   lambda2  = lambda;

    tau2     = (st ASource2 == T) && ((st AInt_1321 == Fw) && (st AInt_1321 !=
Off) && (st AInt_1321 != R)) && ((st AInt_25 == I) && (AInt_25 != Off) &&
(AInt_25 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_22 != R) &&
(AInt_23 != R) && (AInt_24 != R));

    gama2    = (st ASource2 == T) && ((st AInt_25 == Fw) && (st AInt_25 != Off)
&& (st AInt_25 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R)
&& (AInt_22 != R) && (AInt_23 != R) && (AInt_24 != R));

    psi2     = (st ASource2 == T) && ((st AInt_22 == Fw) && (st AInt_22 != Off)
&& (st AInt_22 != R)) && ((st AInt_24 == I) && (AInt_24 != Off) && (AInt_24 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R) && (AInt_23 != R)
&& (AInt_25 != R));

    ro2      = (st ASource2 == T) && ((st AInt_24 == Fw) && (st AInt_24 != Off)
&& (st AInt_24 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R)
&& (AInt_22 != R) && (AInt_23 != R) && (AInt_25 != R));

    niu2     = (st ASource2 == T) && ((st AInt_23 == Fw) && (st AInt_23 != Off)
&& (st AInt_23 != R)) && ((st AInt_24 == I) && (AInt_24 != Off) && (AInt_24 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R) && (AInt_22 != R)
&& (AInt_25 != R));

    chi2     = (st ASource2 == T) && ((st AInt_24 == Fw) && (st AInt_24 != Off)
&& (st AInt_24 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R)
&& (AInt_22 != R) && (AInt_23 != R) && (AInt_25 != R));

    alfa2    = (st ADest2 == R) && ((st AInt_25 == Fw) && (st AInt_25 != Off) &&
(st AInt_25 != R)) && (st ASource2 == T) && ((st AInt_24 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_1321 != R));

    beta2    = (st ADest2 == R) && ((st AInt_25 == R) && (st AInt_25 != Off) &&
(st AInt_25 != Fw)) && (st ASource2 == T) && ((st AInt_1321 == Fw) && (st
AInt_1321 != Off) && (st AInt_1321 != R)) && ((st AInt_24 != R) && (st AInt_23
!= R) && (st AInt_22 != R));

    delta2   = (st ADest2 == R) && ((st AInt_1321 == R) && (st AInt_1321 != Off)
&& (st AInt_1321 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st
AInt_24 != R) && (st AInt_23 != R) && (st AInt_22 != R));
```

```
    omega2   = (st ADest2 == R) && ((st AInt_24 == Fw) && (st AInt_24 != Off) &&
(st AInt_24 != R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_1321 != R));

    sigma2   = (st ADest2 == R) && ((st AInt_24 == R) && (st AInt_24 != Off) &&
(st AInt_24 != Fw)) && (st ASource2 == T) && ((st AInt_22 == Fw) && (st AInt_22
!= Off) && (st AInt_22 != R)) && ((st AInt_25 != R) && (st AInt_23 != R) && (st
AInt_1321 != R));

    sho2     = (st ADest2 == R) && ((st AInt_22 == R) && (st AInt_22 != Off) &&
(st AInt_22 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_23 != R) && (st AInt_1321 != R));

    kapa2    = (st ADest2 == R) && ((st AInt_24 == Fw) && (st AInt_24 != Off) &&
(st AInt_24 != R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_1321 != R));

    teta2    = (st ADest2 == R) && ((st AInt_24 == R) && (st AInt_24 != Off) &&
(st AInt_24 != Fw)) && ((st AInt_23 == Fw) && (AInt_23 != Off) && (AInt_23 !=
R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_22 != R) && (st
AInt_1321 != R));

    zeta2    = (st ADest2 == R) && ((st AInt_23 == R) && (st AInt_23 != Off) &&
(st AInt_23 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_22 != R) && (st AInt_1321 != R));

events

  loc on11  f11;
  loc off11 f12;
  loc on12  f13;
  loc off12 f14;
  loc on1321  f15;
  loc off1321 f16;
  loc on14  f17;
  loc off14 f18;
  loc on15  f19;
  loc off15 f110;
  loc on22  f23;
  loc off22 f24;
  loc on23  f25;
  loc off23 f26;
  loc on24  f27;
  loc off24 f28;
  loc on25  f29;
  loc off25 f210;

  syn rreq1   lambda1;
  syn s11     tau1;
  syn s12     gama1;
  syn s13     psi1;
  syn s14     ro1;
  syn s15     niu1;
  syn s16     chi1;
  syn s151    eta1;
  syn rrep11  alfa1;
  syn rrep12  beta1;
  syn rrep13  delta1;
  syn rrep14  omega1;
  syn rrep15  sigma1;
  syn rrep16  sho1;
```

```
  syn rrep17  kapa1;
  syn rrep18  teta1;
  syn rrep19  zeta1;
  syn rrep152 upsilon1;
  syn rrep153 iota1;
  syn rrep154 stigma1;
  syn rreq2   lambda2;
  syn s21     tau2;
  syn s22     gama2;
  syn s23     psi2;
  syn s24     ro2;
  syn s25     niu2;
  syn s26     chi2;
  syn rrep21  alfa2;
  syn rrep22  beta2;
  syn rrep23  delta2;
  syn rrep24  omega2;
  syn rrep25  sigma2;
  syn rrep26  sho2;
  syn rrep27  kapa2;
  syn rrep28  teta2;
  syn rrep29  zeta2;

partial reachability = (((st ASource1 == T) && (st ADest1 == I) && ((st AInt_11
== Fw) && (st AInt_15 == Fw)) && (((st AInt_1321 != R) && (st AInt_1321 != Fw)
&& (st AInt_1321 != Off)) && ((st AInt_12 != R) && (st AInt_12 != Fw) && (st
AInt_12 != Off)) && ((st AInt_14 != R) && (st AInt_14 != Fw) && (st AInt_14 !=
Off)))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) && (st AInt_14 ==
Fw)) && (((st AInt_1321 != R) && (st AInt_1321 != Fw) && (st AInt_1321 != Off))
&& ((st AInt_11 != R) && (st AInt_11 != Fw) && (st AInt_11 != Off)) && ((st
AInt_15 != R) && (st AInt_15 != Fw) && (st AInt_15 != Off)))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_1321 == Fw) && (st AInt_14
== Fw)) && (((st AInt_11 != R) && (st AInt_11 != Fw) && (st AInt_11 != Off)) &&
((st AInt_12 != R) && (st AInt_12 != Fw) && (st AInt_12 != Off)) && ((st AInt_15
!= R) && (st AInt_15 != Fw) && (st AInt_15 != Off)))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_1321 == Fw) && (st AInt_25
== Fw)) && (((st AInt_22 != R) && (st AInt_22 != Fw) && (st AInt_22 != Off)) &&
((st AInt_23 != R) && (st AInt_23 != Fw) && (st AInt_23 != Off)) && ((st AInt_24
!= R) && (st AInt_24 != Fw) && (st AInt_24 != Off)))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) && (st AInt_24 ==
Fw)) && (((st AInt_1321 != R) && (st AInt_1321 != Fw) && (st AInt_1321 != Off))
&& ((st AInt_23 != R) && (st AInt_23 != Fw) && (st AInt_23 != Off)) && ((st
AInt_25 != R) && (st AInt_25 != Fw) && (st AInt_25 != Off)))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) && (st AInt_24 ==
Fw)) && (((st AInt_1321 != R) && (st AInt_1321 != Fw) && (st AInt_1321 != Off))
&& ((st AInt_22 != R) && (st AInt_22 != Fw) && (st AInt_22 != Off)) && ((st
AInt_25 != R) && (st AInt_25 != Fw) && (st AInt_25 != Off))));



network Adsr (continuous)

  aut ASource1
    stt T to (R) rreq1
    stt R to (T) rrep13 rrep16 rrep19 rrep154

  aut AInt_11
    stt I to (Off) off11
          to (Fw) rreq1
    stt Off to (I) on11
```

```
    stt Fw to (Fw) s11
           to (R) rrep12
    stt R to (I) rrep13

  aut AInt_12
    stt I to (Off) off12
          to (Fw) rreq1
    stt Off to (I) on12
    stt Fw to (Fw) s13
           to (R) rrep15
    stt R  to (I) rrep16

  aut AInt_1321
    stt I to (Off) off1321
          to (Fw) rreq1 rreq2
    stt Off to (I) on1321
    stt Fw to (Fw) s15 s21
           to (R) rrep18 rrep153 rrep22
    stt R  to (I) rrep19 rrep154 rrep23

  aut AInt_14
    stt I to (Off) off14
          to (Fw) s13 s15
    stt Off to (I) on14
    stt Fw to (Fw) s14 s16
           to (R) rrep14 rrep17
    stt R  to (I) rrep15 rrep18

  aut AInt_15
    stt I to (Off) off15
          to (Fw) s11
    stt Off to (I) on15
    stt Fw to (Fw) s12
           to (R) rrep11
    stt R to (I) rrep12

  aut ADest1
    stt I to (R) s12 s14 s16 s151
    stt R to (I) rrep11 rrep14 rrep17 rrep152

  aut ASource2
    stt T to (R) rreq2
    stt R to (T) rrep23 rrep26 rrep29

  aut AInt_22
    stt I to (Off) off22
          to (Fw) rreq2
    stt Off to (I) on22
    stt Fw to (Fw) s23
           to (R) rrep25
    stt R to (I) rrep26

  aut AInt_23
    stt I to (Off) off23
          to (Fw) rreq2
    stt Off to (I) on23
    stt Fw to (Fw) s25
           to (R) rrep28
    stt R to (I) rrep29

  aut AInt_24
```

```
    stt I to (Off) off24
          to (Fw) s23 s25
    stt Off to (I) on24
    stt Fw to (Fw) s24 s26
           to (R) rrep24 rrep27
    stt R to (I) rrep25 rrep28

  aut AInt_25
    stt I to (Off) off25
          to (Fw) s15 s21
    stt Off to (I) on25
    stt Fw to (Fw) s22 s151
           to (R) rrep21 rrep152
    stt R to (I) rrep22 rrep153

  aut ADest2
    stt I to (R) s22 s24 s26
    stt R to (I) rrep21 rrep24 rrep27

results

  t_AInt_1115_64  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header))))/ tx_pause;
  t_AInt_1115_128  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_1115_256  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_1115_512  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;

  t_AInt_1214_64  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) / tx_pause;
  t_AInt_1214_128  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_1214_256  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_1214_512  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;

  t_AInt_132114_64  = ((((st AInt_1321 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) / tx_pause;
  t_AInt_132114_128  = ((((st AInt_1321 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_132114_256  = ((((st AInt_1321 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_132114_512  = ((((st AInt_1321 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;

  t_AInt_132125_64  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) / tx_pause;
  t_AInt_132125_128  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_132125_256  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_132125_512  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;

  t_Asource1 = (st ASource1 == Fw);

  t_AInt_11_Off = (st AInt_11 == Off);
  t_AInt_12_Off = (st AInt_12 == Off);
```

```
  t_AInt_1321_Off = (st AInt_1321 == Off);
  t_AInt_14_Off = (st AInt_14 == Off);
  t_AInt_15_Off = (st AInt_15 == Off);

  t_AInt_11_Rep = (st AInt_11 == R);
  t_AInt_12_Rep = (st AInt_12 == R);
  t_AInt_1321_Rep = (st AInt_1321 == R);
  t_AInt_14_Rep = (st AInt_14 == R);
  t_AInt_15_Rep = (st AInt_15 == R);

  t_AInt_11_Fw = (st AInt_11 == Fw);
  t_AInt_12_Fw = (st AInt_12 == Fw);
  t_AInt_1321_Fw = (st AInt_1321 == Fw);
  t_AInt_14_Fw = (st AInt_14 == Fw);
  t_AInt_15_Fw = (st AInt_15 == Fw);

  t_AInt_132125_64  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) / tx_pause;
  t_AInt_132125_128  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_132125_256  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_132125_512  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;

  t_AInt_2224_64  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) / tx_pause;
  t_AInt_2224_128  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_2224_256  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_2224_512  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;

  t_AInt_2324_64  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+dsr_header)))) / tx_pause;
  t_AInt_2324_128  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+dsr_header)))) / tx_pause;
  t_AInt_2324_256  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+dsr_header)))) / tx_pause;
  t_AInt_2324_512  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+dsr_header)))) / tx_pause;


  t_Asource2 = (st ASource2 == Fw);

  t_AInt_1321_Off = (st AInt_1321 == Off);
  t_AInt_22_Off = (st AInt_22 == Off);
  t_AInt_23_Off = (st AInt_23 == Off);
  t_AInt_24_Off = (st AInt_24 == Off);
  t_AInt_25_Off = (st AInt_25 == Off);

  t_AInt_1321_Rep = (st AInt_1321 == R);
  t_AInt_22_Rep = (st AInt_22 == R);
  t_AInt_23_Rep = (st AInt_23 == R);
  t_AInt_24_Rep = (st AInt_24 == R);
  t_AInt_25_Rep = (st AInt_25 == R);

  t_AInt_1321_Fw = (st AInt_1321 == Fw);
  t_AInt_22_Fw = (st AInt_22 == Fw);
  t_AInt_23_Fw = (st AInt_23 == Fw);
```

```
t_AInt_24_Fw = (st AInt_24 == Fw);
t_AInt_25_Fw = (st AInt_25 == Fw);

t_Route1115 = ((st AInt_11 == Fw) && (st AInt_15 == Fw));
t_Route1214 = ((st AInt_12 == Fw) && (st AInt_14 == Fw));
t_Route132114 = ((st AInt_1321 == Fw) && (st AInt_14 == Fw));
t_Route132125 = ((st AInt_1321 == Fw) && (st AInt_25 == Fw));
t_Route2224 = ((st AInt_22 == Fw) && (st AInt_24 == Fw));
t_Route2324 = ((st AInt_23 == Fw) && (st AInt_24 == Fw));
```

APÊNDICE F – SAN PARA REDES COM AGREGAÇÃO DE SERVIÇOS DE ROTEAMENTO COM PROBABILIDADES DE FALHAS

SAN da Figura 5.13, modelo para redes com agregação de serviços de roteamento com probabilidades de falhas na transmissão.

```
identifiers
    f11      = (st AInt_11 != Off) * 0.8;
    f12      = (st AInt_11 == Off) * 0.2;
    f13      = (st AInt_12 != Off) * 0.8;
    f14      = (st AInt_12 == Off) * 0.2;
    f15      = (((st AInt_1321 != Off) * 0.8));
    f16      = (((st AInt_1321 == Off) * 0.2));
    f17      = (st AInt_14 != Off) * 0.8;
    f18      = (st AInt_14 == Off) * 0.2;
    f19      = (st AInt_15 != Off) * 0.8;
    f110     = (st AInt_15 == Off) * 0.2;
    f23      = (st AInt_22 != Off) * 0.8;
    f24      = (st AInt_22 == Off) * 0.2;
    f25      = (st AInt_23 != Off) * 0.8;
    f26      = (st AInt_23 == Off) * 0.2;
    f27      = (st AInt_24 != Off) * 0.8;
    f28      = (st AInt_24 == Off) * 0.2;
    f29      = (st AInt_25 != Off) * 0.8;
    f210     = (st AInt_25 == Off) * 0.2;
    bandwidth_b = 2000000;                    // bandwidth in bits/seconds
    bandwidth_B = bandwidth_b/8;              // bandwidth in bytes/seconds
    bandwidth_Mbps = bandwidth_b/1000000;     // bandwidth in bytes/seconds
    size_pack_64 = 64;                        // package size in bytes
    size_pack_128 = 128;
    size_pack_256 = 256;
    size_pack_512 = 512;
    NXH = 7;                                  // DSR Next Header field in bytes
    RESVD = 8;                                // DSR Reserved Header in bytes
    MAC = 47;                                 // Medium Access Control header in
bytes
    PALG = 16;                              // DSR Payload Length header in bytes
    header_dsr = NXH+RESVD+MAC+PALG;           // headers
    ocur_rate = 500;                        // ocurrence rate of a route request
message in miliseconds
    lambda = 0.5;                  // ocurrence rate of a route request message
in seconds (1 second = 1000 miliseconds)
    tx_tpause = 0.1777;            // ocurrence rate of time pause (for 5.625
seconds, according to established time pause)

    lambda1  = lambda;

    tau1     = (st ASource1 == T) && ((st AInt_11 == Fw) && (st AInt_11 != Off)
&& (st AInt_11 != R)) && ((st AInt_15 == I) && (st AInt_15 != Off) && (st
AInt_15 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_12 != R) &&
(st AInt_1321 != R) && (st AInt_14 != R));

    gama1    = (((st ASource1 == T) && ((st AInt_15 == Fw) && (st AInt_15 !=
Off) && (st AInt_15 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_1321 != R) && (st AInt_14 != R))));
```

```
    psi1     = (st ASource1 == T) && ((st AInt_12 == Fw) && (st AInt_12 != Off)
&& (st AInt_12 != R)) && ((st AInt_14 == I) && (st AInt_14 != Off) && (st
AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) &&
(st AInt_12 != R) && (st AInt_15 != R));

    ro1      = (((st ASource1 == T) && ((st AInt_14 == Fw) && (st AInt_14 !=
Off) && (st AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_1321 != R) && (st AInt_15 !=
R))));

    niu1     = (st ASource1 == T) && ((st AInt_1321 == Fw) && (st AInt_1321 !=
Off) && (st AInt_1321 != R))  && ((st AInt_14 == I) && (st AInt_14 != Off) &&
(st AInt_14 != R))  && ((st AInt_25 == I) && (st AInt_25 != Off) && (st AInt_25
!= R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 != R) && (st
AInt_12 != R) && (st AInt_15 != R));

    chi1     = (((st ASource1 == T) && ((st AInt_14 == Fw) && (st AInt_14 !=
Off) && (st AInt_14 != R)) && (st ADest1 == I) && (st ADest1 != R) && ((st
AInt_11 != R) && (st AInt_12 != R) && (st AInt_1321 != R) && (st AInt_15 !=
R))));

    eta1     = (st ASource1 == T) && ((st AInt_25 == Fw) && (st AInt_25 != Off)
&& (st AInt_25!= R)) && (st ADest1 == I) && (st ADest1 != R) && ((st AInt_11 !=
R) && (st AInt_12 != R) && (st AInt_1321 != R) && (st AInt_14 != R) && (st
AInt_15 != R));

    alfa1    = (st ADest1 == R) && ((st AInt_15 == Fw) && (st AInt_15 != Off) &&
(st AInt_15 != R)) && (st ASource1 == T) && ((st AInt_14 != R) && (st AInt_1321
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    beta1    = (st ADest1 == R) && ((st AInt_15 == R) && (st AInt_15 != Off) &&
(st AInt_15 != Fw)) && (st ASource1 == T) && ((st AInt_11 == Fw) && (st AInt_11
!= Off) && (st AInt_11 != R)) && ((st AInt_14 != R) && (st AInt_1321 != R) &&
(st AInt_11 != R));

    delta1   = (st ADest1 == R) && ((st AInt_11 == R) && (st AInt_11 != Off) &&
(st AInt_11 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_1321 != R)) && (st AInt_12 != R);

    omega1   = (st ADest1 == R) && ((st AInt_14 == Fw) && (st AInt_14 != Off) &&
(st AInt_14 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_1321
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    sigma1   = (st ADest1 == R) && ((st AInt_14 == R) && (st AInt_14 != Off) &&
(st AInt_14 != Fw)) && (st ASource1 == T) && ((st AInt_12 == Fw) && (st AInt_12
!= Off) && (st AInt_12 != R)) && ((st AInt_15 != R) && (st AInt_1321 != R) &&
(st AInt_11 != R));

    sho1     = (st ADest1 == R) && ((st AInt_12 == R) && (st AInt_12 != Off) &&
(st AInt_12 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_1321 != R) && (st AInt_11 != R));

    kapa1    = (st ADest1 == R) && ((st AInt_14 == Fw) && (st AInt_14 != Off) &&
(st AInt_14 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_1321
!= R) && (st AInt_12 != R) && (st AInt_11 != R));

    teta1    = (st ADest1 == R) && ((st AInt_14 == R) && (st AInt_14 != Off) &&
(st AInt_14 != Fw)) && ((st AInt_1321 == Fw) && (st AInt_1321 != Off) && (st
AInt_1321 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_12 !=
R) && (st AInt_11 != R));
```

```
    zeta1    = (st ADest1 == R) && ((st AInt_1321 == R) && (st AInt_1321 != Off)
&& (st AInt_1321 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st
AInt_14 != R) && (st AInt_12 != R) && (st AInt_11 != R));

    upsilon1 = (st ADest1 == R) && ((st AInt_25 == Fw) && (st AInt_25 != Off) &&
(st AInt_25 != R)) && (st ASource1 == T) &&  ((st AInt_15 != R) && (st AInt_14
!= R) && (st AInt_1321 != R) && (st AInt_12 != R) && (st AInt_11 != R));

    iota1    = (st ADest1 == R) && ((st AInt_25 == R) && (st AInt_25 != Off) &&
(st AInt_25 != Fw)) &&  ((st AInt_1321 == Fw) && (st AInt_1321 != Off) && (st
AInt_1321 != R)) && (st ASource1 == T) && ((st AInt_15 != R) && (st AInt_14 !=
R) && (st AInt_12 != R) && (st AInt_11 != R));

    stigma1  = (st ADest1 == R) && ((st AInt_1321 == R) && (st AInt_1321 != Off)
&& (st AInt_1321 != Fw)) && (st ASource1 == R) && ((st AInt_15 != R) && (st
AInt_14 != R) && (st AInt_12 != R) && (st AInt_11 != R));

   lambda2  = lambda;

    tau2     = (st ASource2 == T) && ((st AInt_1321 == Fw) && (st AInt_1321 !=
Off) && (st AInt_1321 != R)) && ((st AInt_25 == I) && (AInt_25 != Off) &&
(AInt_25 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_22 != R) &&
(AInt_23 != R) && (AInt_24 != R));

    gama2    = (st ASource2 == T) && ((st AInt_25 == Fw) && (st AInt_25 != Off)
&& (st AInt_25 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R)
&& (AInt_22 != R) && (AInt_23 != R) && (AInt_24 != R));

    psi2     = (st ASource2 == T) && ((st AInt_22 == Fw) && (st AInt_22 != Off)
&& (st AInt_22 != R)) && ((st AInt_24 == I) && (AInt_24 != Off) && (AInt_24 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R) && (AInt_23 != R)
&& (AInt_25 != R));

    ro2      = (st ASource2 == T) && ((st AInt_24 == Fw) && (st AInt_24 != Off)
&& (st AInt_24 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R)
&& (AInt_22 != R) && (AInt_23 != R) && (AInt_25 != R));

    niu2     = (st ASource2 == T) && ((st AInt_23 == Fw) && (st AInt_23 != Off)
&& (st AInt_23 != R)) && ((st AInt_24 == I) && (AInt_24 != Off) && (AInt_24 !=
R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R) && (AInt_22 != R)
&& (AInt_25 != R));

    chi2     = (st ASource2 == T) && ((st AInt_24 == Fw) && (st AInt_24 != Off)
&& (st AInt_24 != R)) && (ADest2 == I) && (st ADest2 != R) && ((AInt_1321 != R)
&& (AInt_22 != R) && (AInt_23 != R) && (AInt_25 != R));

    alfa2    = (st ADest2 == R) && ((st AInt_25 == Fw) && (st AInt_25 != Off) &&
(st AInt_25 != R)) && (st ASource2 == T) && ((st AInt_24 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_1321 != R));

    beta2    = (st ADest2 == R) && ((st AInt_25 == R) && (st AInt_25 != Off) &&
(st AInt_25 != Fw)) && (st ASource2 == T) && ((st AInt_1321 == Fw) && (st
AInt_1321 != Off) && (st AInt_1321 != R)) && ((st AInt_24 != R) && (st AInt_23
!= R) && (st AInt_22 != R));

    delta2   = (st ADest2 == R) && ((st AInt_1321 == R) && (st AInt_1321 != Off)
&& (st AInt_1321 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st
AInt_24 != R) && (st AInt_23 != R) && (st AInt_22 != R));
```

```
    omega2   = (st ADest2 == R) && ((st AInt_24 == Fw) && (st AInt_24 != Off) &&
(st AInt_24 != R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_1321 != R));

    sigma2   = (st ADest2 == R) && ((st AInt_24 == R) && (st AInt_24 != Off) &&
(st AInt_24 != Fw)) && (st ASource2 == T) && ((st AInt_22 == Fw) && (st AInt_22
!= Off) && (st AInt_22 != R)) && ((st AInt_25 != R) && (st AInt_23 != R) && (st
AInt_1321 != R));

    sho2     = (st ADest2 == R) && ((st AInt_22 == R) && (st AInt_22 != Off) &&
(st AInt_22 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_23 != R) && (st AInt_1321 != R));

    kapa2    = (st ADest2 == R) && ((st AInt_24 == Fw) && (st AInt_24 != Off) &&
(st AInt_24 != R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_23 !=
R) && (st AInt_22 != R) && (st AInt_1321 != R));

    teta2    = (st ADest2 == R) && ((st AInt_24 == R) && (st AInt_24 != Off) &&
(st AInt_24 != Fw)) && ((st AInt_23 == Fw) && (AInt_23 != Off) && (AInt_23 !=
R)) && (st ASource2 == T) && ((st AInt_25 != R) && (st AInt_22 != R) && (st
AInt_1321 != R));

    zeta2    = (st ADest2 == R) && ((st AInt_23 == R) && (st AInt_23 != Off) &&
(st AInt_23 != Fw)) && (st ASource2 == R) && ((st AInt_25 != R) && (st AInt_24
!= R) && (st AInt_22 != R) && (st AInt_1321 != R));

events

  loc on11  f11;
  loc off11 f12;
  loc on12  f13;
  loc off12 f14;
  loc on1321  f15;
  loc off1321 f16;
  loc on14  f17;
  loc off14 f18;
  loc on15  f19;
  loc off15 f110;
  loc on22  f23;
  loc off22 f24;
  loc on23  f25;
  loc off23 f26;
  loc on24  f27;
  loc off24 f28;
  loc on25  f29;
  loc off25 f210;

  syn rreq1   lambda1;
  syn s11     tau1;
  syn s12     gama1;
  syn s13     psi1;
  syn s14     ro1;
  syn s15     niu1;
  syn s16     chi1;
  syn s151    eta1;
  syn rrep11  alfa1;
  syn rrep12  beta1;
  syn rrep13  delta1;
  syn rrep14  omega1;
  syn rrep15  sigma1;
  syn rrep16  sho1;
```

```
  syn rrep17  kapa1;
  syn rrep18  teta1;
  syn rrep19  zeta1;
  syn rrep152 upsilon1;
  syn rrep153 iota1;
  syn rrep154 stigma1;
  syn rreq2   lambda2;
  syn s21     tau2;
  syn s22     gama2;
  syn s23     psi2;
  syn s24     ro2;
  syn s25     niu2;
  syn s26     chi2;
  syn rrep21  alfa2;
  syn rrep22  beta2;
  syn rrep23  delta2;
  syn rrep24  omega2;
  syn rrep25  sigma2;
  syn rrep26  sho2;
  syn rrep27  kapa2;
  syn rrep28  teta2;
  syn rrep29  zeta2;

partial reachability = ((st ASource1 == T) && (st ADest1 == I) && ((st AInt_11
== Fw) && (st AInt_15 == Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_12 == Fw) && (st AInt_14 ==
Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_1321 == Fw) && (st AInt_14
== Fw))) ||
((st ASource1 == T) && (st ADest1 == I) && ((st AInt_1321 == Fw) && (st AInt_25
== Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_22 == Fw) && (st AInt_24 ==
Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_23 == Fw) && (st AInt_24 ==
Fw))) ||
((st ASource2 == T) && (st ADest2 == I) && ((st AInt_1321 == Fw) && (st AInt_25
== Fw)));

network Adsr (continuous)

  aut ASource1
    stt T to (R) rreq1
    stt R to (T) rrep13 rrep16 rrep19 rrep154

  aut AInt_11
    stt I to (Off) off11
          to (Fw) rreq1
    stt Off to (I) on11
    stt Fw to (Fw) s11
           to (R) rrep12
    stt R to (I) rrep13

  aut AInt_12
    stt I to (Off) off12
          to (Fw) rreq1
    stt Off to (I) on12
    stt Fw to (Fw) s13
           to (R) rrep15
    stt R  to (I) rrep16

  aut AInt_1321
```

```
  stt I to (Off) off1321
        to (Fw) rreq1 rreq2
  stt Off to (I) on1321
  stt Fw to (Fw) s15 s21
         to (R) rrep18 rrep153 rrep22
  stt R  to (I) rrep19 rrep154 rrep23

aut AInt_14
  stt I to (Off) off14
        to (Fw) s13 s15
  stt Off to (I) on14
  stt Fw to (Fw) s14 s16
         to (R) rrep14 rrep17
  stt R  to (I) rrep15 rrep18

aut AInt_15
  stt I to (Off) off15
        to (Fw) s11
  stt Off to (I) on15
  stt Fw to (Fw) s12
         to (R) rrep11
  stt R to (I) rrep12

aut ADest1
  stt I to (R) s12 s14 s16 s151
  stt R to (I) rrep11 rrep14 rrep17 rrep152

aut ASource2
  stt T to (R) rreq2
  stt R to (T) rrep23 rrep26 rrep29

aut AInt_22
  stt I to (Off) off22
        to (Fw) rreq2
  stt Off to (I) on22
  stt Fw to (Fw) s23
         to (R) rrep25
  stt R to (I) rrep26

aut AInt_23
  stt I to (Off) off23
        to (Fw) rreq2
  stt Off to (I) on23
  stt Fw to (Fw) s25
         to (R) rrep28
  stt R to (I) rrep29

aut AInt_24
  stt I to (Off) off24
        to (Fw) s23 s25
  stt Off to (I) on24
  stt Fw to (Fw) s24 s26
         to (R) rrep24 rrep27
  stt R to (I) rrep25 rrep28

aut AInt_25
  stt I to (Off) off25
        to (Fw) s15 s21
  stt Off to (I) on25
  stt Fw to (Fw) s22 s151
         to (R) rrep21 rrep152
```

```
    stt R to (I) rrep22 rrep153

  aut ADest2
    stt I to (R) s22 s24 s26
    stt R to (I) rrep21 rrep24 rrep27

results

  t_AInt_1115_64  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr))))/ tx_pause;
  t_AInt_1115_128  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))) / tx_pause;
  t_AInt_1115_256  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))) / tx_pause;
  t_AInt_1115_512  = ((((st AInt_11 == Fw) && (st AInt_15 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))) / tx_pause;

  t_AInt_1214_64  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr)))) / tx_pause;
  t_AInt_1214_128  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))) / tx_pause;
  t_AInt_1214_256  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))) / tx_pause;
  t_AInt_1214_512  = ((((st AInt_12 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))) / tx_pause;

  t_AInt_132114_64  = ((((st AInt_1321 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr)))) / tx_pause;
  t_AInt_132114_128  = ((((st AInt_1321 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))) / tx_pause;
  t_AInt_132114_256  = ((((st AInt_1321 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))) / tx_pause;
  t_AInt_132114_512  = ((((st AInt_1321 == Fw) && (st AInt_14 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))) / tx_pause;

  t_AInt_132125_64  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr)))) / tx_pause;
  t_AInt_132125_128  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))) / tx_pause;
  t_AInt_132125_256  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))) / tx_pause;
  t_AInt_132125_512  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))) / tx_pause;

  t_Asource1 = (st ASource1 == Fw);

  t_AInt_11_Off = (st AInt_11 == Off);
  t_AInt_12_Off = (st AInt_12 == Off);
  t_AInt_1321_Off = (st AInt_1321 == Off);
  t_AInt_14_Off = (st AInt_14 == Off);
  t_AInt_15_Off = (st AInt_15 == Off);

  t_AInt_11_Rep = (st AInt_11 == R);
  t_AInt_12_Rep = (st AInt_12 == R);
  t_AInt_1321_Rep = (st AInt_1321 == R);
  t_AInt_14_Rep = (st AInt_14 == R);
  t_AInt_15_Rep = (st AInt_15 == R);

  t_AInt_11_Fw = (st AInt_11 == Fw);
  t_AInt_12_Fw = (st AInt_12 == Fw);
  t_AInt_1321_Fw = (st AInt_1321 == Fw);
```

```
  t_AInt_14_Fw = (st AInt_14 == Fw);
  t_AInt_15_Fw = (st AInt_15 == Fw);

  t_AInt_132125_64  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr)))) / tx_pause;
  t_AInt_132125_128  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))) / tx_pause;
  t_AInt_132125_256  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))) / tx_pause;
  t_AInt_132125_512  = ((((st AInt_1321 == Fw) && (st AInt_25 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))) / tx_pause;

  t_AInt_2224_64  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr)))) / tx_pause;
  t_AInt_2224_128  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))) / tx_pause;
  t_AInt_2224_256  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))) / tx_pause;
  t_AInt_2224_512  = ((((st AInt_22 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))) / tx_pause;

  t_AInt_2324_64  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_64/(size_pack_64+header_dsr)))) / tx_pause;
  t_AInt_2324_128  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_128/(size_pack_128+header_dsr)))) / tx_pause;
  t_AInt_2324_256  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_256/(size_pack_256+header_dsr)))) / tx_pause;
  t_AInt_2324_512  = ((((st AInt_23 == Fw) && (st AInt_24 == Fw)) *
bandwidth_Mbps) * (size_pack_512/(size_pack_512+header_dsr)))) / tx_pause;


  t_Asource2 = (st ASource2 == Fw);

  t_AInt_1321_Off = (st AInt_1321 == Off);
  t_AInt_22_Off = (st AInt_22 == Off);
  t_AInt_23_Off = (st AInt_23 == Off);
  t_AInt_24_Off = (st AInt_24 == Off);
  t_AInt_25_Off = (st AInt_25 == Off);

  t_AInt_1321_Rep = (st AInt_1321 == R);
  t_AInt_22_Rep = (st AInt_22 == R);
  t_AInt_23_Rep = (st AInt_23 == R);
  t_AInt_24_Rep = (st AInt_24 == R);
  t_AInt_25_Rep = (st AInt_25 == R);

  t_AInt_1321_Fw = (st AInt_1321 == Fw);
  t_AInt_22_Fw = (st AInt_22 == Fw);
  t_AInt_23_Fw = (st AInt_23 == Fw);
  t_AInt_24_Fw = (st AInt_24 == Fw);
  t_AInt_25_Fw = (st AInt_25 == Fw);

  t_Route1115 = ((st AInt_11 == Fw) && (st AInt_15 == Fw));
  t_Route1214 = ((st AInt_12 == Fw) && (st AInt_14 == Fw));
  t_Route132114 = ((st AInt_1321 == Fw) && (st AInt_14 == Fw));
  t_Route132125 = ((st AInt_1321 == Fw) && (st AInt_25 == Fw));
  t_Route2224 = ((st AInt_22 == Fw) && (st AInt_24 == Fw));
  t_Route2324 = ((st AInt_23 == Fw) && (st AInt_24 == Fw));
```