ANDSR - PROTOCOLO DE ROTEAMENTO ANÔNIMO PARA REDES AD HOC

Andréa Martins Araujo

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Prof. Julius Cesar Barreto Leite, PhD.

Prof. Luís Henrique Maciel Kosmalski Costa, Dr.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2005

ARAUJO, ANDRÉA MARTINS

ANDSR - Protocolo de Roteamento Anônimo para Redes Ad Hoc [Rio de Janeiro] 2005

XV, 133 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Elétrica, 2005)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Roteamento

2. Anonimato e Segurança

3. LOTOS

I. COPPE/UFRJ II. Título (série)

Aos familiares e amigos.

Agradecimentos

A Deus.

À minha família, pelo apoio em todas as minhas decisões.

Ao Professor Aloysio de Castro Pinto Pedroza, pelo incentivo e confiança.

Aos Professores do GTA pela excelente formação adquirida durante o curso.

Ao amigo David Fernandez Cruz Moura, pela colaboração na realização deste trabalho.

Ao amigo Bruno Eugênio Ronzani que cedeu sua máquina para instalação da ferramenta CADP, enquanto a máquina que usava estava em manutenção.

Ao amigo Pedro Braconnot Velloso que apresentou meu artigo no WSEG.

Ao Professor Hubert Garavel do INRIA Rhone-Alpes na França, pela colaboração e atenção no esclarecimento de dúvidas sobre a ferramenta CADP.

Aos colegas de curso e toda equipe do GTA, em especial às amigas Ingrid e Ivana, pela amizade e ajuda em diveros momentos,

Aos professores que participaram da Comissão Examinadora.

À FAPERJ pelo financiamento de minha bolsa, durante a vigência da qual se desenvolveu esta tese.

Aos funcionários do Departamento que de uma forma ou de outra contribuíram para o êxito deste trabalho.

A todos aqueles que tenham contribuído de alguma maneira e porventura não tenham sido mencionados.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ANDSR - PROTOCOLO DE ROTEAMENTO ANÔNIMO PARA REDES AD HOC

Andréa Martins Araujo

Março/2005

Orientador: Aloysio de Castro Pinto Pedroza

Programa: Engenharia Elétrica

As primeiras pesquisas em redes ad hoc estudaram os problemas do roteamento em um cenário livre de adversários assumindo um ambiente confiável. Por causa da característica dinâmica, essas redes estão vulneráveis a vários ataques, incluindo análise de tráfego. Usar caminhos anônimos para comunicação proporciona segurança e privacidade contra análise de tráfego.

Esse trabalho apresenta a criação e o desenvolvimento do ANDSR (Anonymous Dynamic Source Routing), um protocolo de roteamento anônimo para redes ad hoc. O ANDSR tem suas origens no protocolo DSR (Dynamic Source Routing). É uma extensão do protocolo DSR, pois traz algumas modificações que permitem ao algoritmo melhor resistência à análise de tráfego. A técnica de descrição formal, a linguagem LOTOS (Language of Temporal Ordering Specification), é usada para especificar e validar (empregando simulações, testes e verificações) o protocolo proposto. LOTOS é apropriada para lidar com especificações de protocolos de segurança com alto nível de abstração. Além disso, possui uma rigorosa validação. O conjunto de ferramentas CADP (Caesar/Aldebaran Development Package) é usado de maneira a validar o protocolo.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ANDSR - ANONYMOUS ROUTING PROTOCOL FOR AD HOC NETWORKS

Andréa Martins Araujo

March/2005

Advisor: Aloysio de Castro Pinto Pedroza

Department: Electrical Engineering

Prior research in ad hoc networking has generally studied the routing problem in a non-adversarial setting assuming a trusted environment. Because of the dynamic characteristic, these networks are vulnerable to several attaques, including traffic analysis. Using anonymous paths for communication provides security and privacy against traffic analysis.

This works presents the creation and the development of ANDSR (Anonymous Dynamic Source Routing), an anonymous routing protocol for wireless ad hoc network. ANDSR has its origins in DSR (Dynamic Source Routing) protocol. It is an extension of DSR, which brings some modifications that provide the algorithm best resistance to the traffic analysis. The formal description techique, LOTOS (Language of Temporal Ordering Specification) language, is used to specificate and validate (employing simulations, testing, and verifications) the proposed protocol. LOTOS is appropriate to handle the specification of the security protocols with a high level of abstration. Besides, it has a rigorous validation. The CADP tool (Caesar/Aldebaran Development Package) is used in order to validate the protocol.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de acrônimos

ANDSR : *Anonymous Dynamic Source Routing*;

AODV : *Ad Hoc On Demand Distance Vector*;

CA : *Certificate Autority*;

CADP : *CAESAR/ALDEBARAN Development Package*;

DSDV : *Destination Sequenced Distance Vector*;

DSR : *Dynamic Source Routing*;

FDT : *Formal Description Techniques*;

IETF : *Internet Engineering Task Force*;

ISO : *International Organization for Standardization*;

LOTOS : *Language of Temporal Ordering Specification*;

LTS : *Labelled Transition System*;

RREP : *Route Reply*;

RREQ : *Route Request*;

Capítulo 1

Introdução

A PROLIFERAÇÃO da computação móvel e dos dispositivos de comunicação sem fio (telefones celulares, palms, etc) está revolucionando a sociedade da informação. Atualmente, a maioria das conexões entre os dispositivos sem fio são estabelecida através de uma infraestrutura fixa, ou redes privadas. Enquanto essas redes proporcionam um bom caminho para o acesso aos serviços da rede, a configuração necessária para essa infraestrutura demanda tempo e um alto custo. Adicionalmente, existem situações onde a conectividade e certos serviços de rede não estão disponíveis.

Recentemente, novas alternativas para entrega de serviços estão sendo estudadas. Mais especificamente, os dispositivos móveis se conectam uns aos outros através de configurações automáticas e sem qualquer tipo de infraestrutura fixa. As redes que possuem essas características são chamadas *redes ad hoc*.

Segurança também é um aspecto bastante importante para a utilização dessas redes. O presente trabalho propõe um protocolo de roteamento capaz de prover anonimato no estabelecimento de rotas entre os dispositivos das redes ad hoc. Essa proposta pretende dificultar a ação de eventuais intrusos que queiram adquirir informações sobre a rede.

1.1 Motivação

Com o desenvolvimento da computação móvel, a privacidade (sigilo na troca de informações) e o anonimato (sigilo da identidade das partes envolvidas) na comunicação estão se tornando necessários em grande parte das aplicações. A criptografia é normalmente usada para esconder o conteúdo trocado numa comunicação e manter a integridade do sistema. Porém, esconder a identidade das partes da comunicação contra espionagem não é muito comum. Um dos primeiros estudos sobre anonimato na Internet foi desenvolvido por D. Chaum [1] em 1981. A partir daí surgiram outros trabalhos sobre anonimato na Internet [2, 3, 4] e, mais recentemente, pesquisadores estão tentando transportar esse conceito para redes ad hoc. Entretanto, prover anonimato nas redes ad hoc é bastante difícil devido à sua natureza dinâmica.

1.2 Trabalhos Relacionados

Na literatura não existem muitos trabalhos que tratam do anonimato em redes ad hoc, pois a pesquisa nesse campo é recente. Chaum [1] foi um dos pioneiros no estudo sobre anonimato. Sua proposta consiste em prover anonimato em correspondência eletrônica através de um método conhecido como *Mix Method*, desenvolvido por ele. Neste método a origem da mensagem não a envia diretamente para o destino, mas para um misturador que é responsável por manter o anonimato na comunicação escondendo os endereços das partes envolvidas. O trabalho de Chaum serviu de base para outros trabalhos sobre anonimato na Internet [2, 3, 4]. O objetivo dos pesquisadores, agora, é transportar o conceito de anonimato para redes ad hoc.

Jiang et al. [5] propõem algoritmos que permitem a comunicação anônima entre os nós em redes ad hoc. Essa proposta também se baseou no trabalho desenvolvido por D. Chaum em 1981. Trata-se de dois algoritmos de busca de misturadores na rede, quando um nó, chamado fonte, necessita entregar um pacote ele deve executar um algoritmo de busca de misturadores para localizar e conhecer os misturadores presentes na rede. Esses algoritmos são independentes do protocolo de roteamento usado, sendo executados antes do protocolo de roteamento.

El-Khatib et al. [6] propõem um protocolo de roteamento anônimo para redes ad hoc usando o conceito de CA (*certificate autority*) e assinatura digital.

Kong et. al. [7] desenvolveram um protocolo chamado ANODR (*ANonymous On Demand Routing with Untraceable Routes for Mobile Ad hoc Networks*). Esse protocolo estabelece rotas anônimas usando o conceito de *broadcast with trapdoor information* [7] (mecanismo que introduz informações falsas, conhecidas somente pelos nós receptores, onde os dados podem ser entregues anonimamente a eles) e *pseudonymity* [7, 8] (cada nó possui um pseudonimo associado a ele).

Com relação aos métodos formais empregados, existem diversos trabalhos que utilizam técnicas formais para verificação de protocolos, porém, são poucos os trabalhos que tratam da verificação formal em redes ad hoc. Destaca-se o trabalho de Leduc et. al. [9] que desenvolveram uma metodologia para a verificação formal de protocolos de segurança usando a linguagem LOTOS [10].

Dos Santos et al. [11] descrevem uma metodologia para verificação de protocolos de roteamento para redes ad hoc. Essa metodologia propõe uma modelagem independente do número de nós participantes da rede. Considerando uma rede com dez nós, onde existe um nó fonte, um destino e os demais são intermediários, existirão três grupos de nós. O primeiro grupo representa os nós fontes, o segundo representa os nós destinos e o terceiro os intermediários. Na modelagem existirão somente três entidades: fonte, destino e intermediários. É como se só existissem três nós. Na verdade, cada entidade modela uma nuvem de nós, a entidade intermediários representa aqueles oito nós da rede imaginária. Essa metodologia pode ser usada para modelar redes com qualquer quantidade de nós.

Bagatelli et. al. [12] apresentam a descrição formal de uma arquitetura de suporte à descoberta de serviço em redes ad hoc, usando a linguagem LOTOS e o conjunto de ferramentas CADP [13].

1.3 Objetivo

Este trabalho tem como objetivo prover anonimato em redes ad hoc, escondendo, para tal, a identidade da origem e do destino da comunicação, bem como as informações trocadas entre eles. É proposto, então, um protocolo de roteamento anônimo para redes ad hoc chamado ANDSR (*Anonymous Dynamic Source Routing*). O ANDSR é responsável por manter o anonimato na troca de dados entre dois dispositivos ou nós, dificultanto o trabalho de intrusos que queiram obter informações a respeito dos nós e seus relacionamentos (com quem o nó se comunica) na rede. Para o estabelecimento de rotas anônimas são usados nós especiais, chamados misturadores, que irão manter o anonimato no roteamento dos pacotes. Diferente da proposta de Jiang et al. [5], no protocolo proposto a busca por misturadores está incluída no próprio algoritmo de roteamento. Também não há necessidade do conhecimento prévio de todos os misturadores da rede. Além disso, o ANDSR não utiliza autoridades certificadoras e assinaturas digitais como o trabalho de El-Khatib et al. [6]. Seu algoritmo baseia-se na proposta de Chaum [1], adaptando os conceitos apresentados neste trabalho para redes ad hoc. Para obter um modelo correto e validado do protocolo é usada a FDT LOTOS e um pacote de ferramentas para análise, o pacote CADP, adequado à engenharia de protocolos.

Este trabalho está organizado da seguinte forma: no capítulo 2 são apresentadas as redes ad hoc e o protocolo ANDSR (*Anonymous Dynamic Source Routing*), o capítulo 3 descreve o projeto abordando a linguagem LOTOS (*Language of Temporal Ordering Specification*), o software de análise CADP, a metodologia adotada e a modelagem do protocolo, o capítulo 4 apresenta o processo de verficação e os resultados e, por fim, o capítulo 5 conclui o trabalho.

Capítulo 2

ANDSR - Anonymous Dynamic Source Routing

ORIGINALMENTE , as redes ad hoc foram usadas em operações militares para melhorar a comunicação no campo de batalha. A natureza dinâmica das operações militares não pode depender de uma infraestrutura fixa. A maioria das redes ad hoc, excluindo a arena militar, são desenvolvidas em ambientes acadêmicos. Recentemente, algumas soluções começam a aparecer comercialmente [14].

A privacidade na comunicação está se tornando um requisito de segurança cada vez mais importante. Enquanto a criptografia é usada normalmente para proteger o conteúdo trocado entre os usuários, a observação do tráfego pode revelar informações acerca dos usuários e seus relacionamentos na rede. O propósito da análise de tráfego é descobrir quem se comunica com quem. Assim, a comunicação anônima deve ser resistente à análise de tráfego, isto é, deve criar dificuldades para observadores que queiram adquirir informações sobre a rede. A idéia é esconder a identidade da fonte e do destino da comunicação, mantendo o anonimato e a privacidade dos usuários.

Prover anonimato em redes ad hoc é um desafio devido à natureza dinâmica dessas redes. As pesquisas nesse sentido são recentes e tentam transportar e adaptar conceitos conhecidos e utilizados na Internet. Este capítulo apresenta as redes ad hoc e o protocolo de roteamento anônimo ANDSR (*Anonymous Dynamic Source Routing*), proposto e modelado formalmente neste projeto. Na seção 2.1 são apresentadas as redes ad hoc, o protocolo de roteamento DSR (*Dynamic Source Routing*) e os aspectos de segurança dessas redes. A seção 2.2 faz algumas considerações a respeito do DSR, apresenta os príncipios básicos do algoritmo proposto, o funcionamento do protocolo ANDSR, trata dos pacotes usados no protocolo ANDSR e trata, também, dos métodos criptográficos usados. Por fim, a seção 2.3 faz alguns comentários acerca dos assuntos abordados neste capítulo.

2.1 Redes Ad Hoc

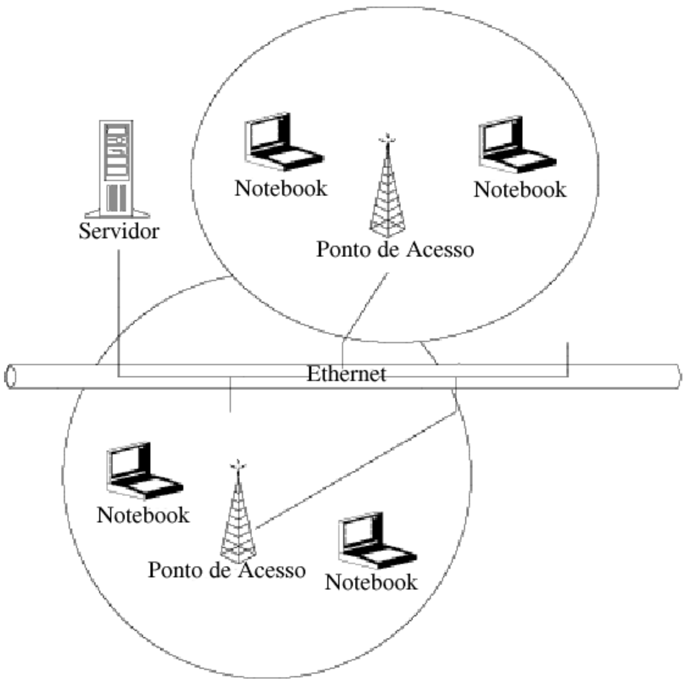
Uma rede ad hoc, vide Figura 2.1(a), é um conjunto de dispositivos (ou nós) móveis que formam um sistema autônomo onde os nós estão conectados através de enlaces sem fio, sem o uso de qualquer tipo de infraestrutura fixa ou administração centralizada, tais como estações bases ou pontos de acesso, Figura 2.1(b). Em resumo, os nós atuam como roteadores.

Neste tipo de rede, os nós podem se mover livre e aleatoriamente, desta forma, a topologia da rede pode mudar rapidamente e imprevisivelmente.

Em geral, as rotas entre os nós podem incluir múltiplos saltos. Um nó é capaz de se comunicar diretamente com um ou mais nós, desde que os nós estejam dentro de seu raio de alcance. Para a comunicação com os nós fora do raio de alcance, é necessário o uso de nós intermediários que cooperarão no encaminhamento de pacotes salto a salto.

Essas redes são usadas quando não há possibilidade do uso de infraestruturas fixas ou quando há necessidade de estabelecer uma rede rapidamente. As aplicações mais conhecidas são as militares, as operações de salvamento em desastres, incursões em terrenos hostis em geral e conferências.

(a) **Rede Ad Hoc**



(b) **Rede Infraestruturada**

Figura 2.1: **Redes móveis sem fio**

2.1.1 Roteamento

A camada de rede é responsável pelo roteamento dos pacotes de um nó origem para um ou mais nós destinos. O algoritmo de roteamento é a parte do software da camada de rede que decide sobre a linha de saída a ser usada na transmissão dos pacotes de entrada [15].

Em redes ad hoc, onde a topologia muda freqüentemente, os algoritmos de roteamento podem ser separados em três classes:

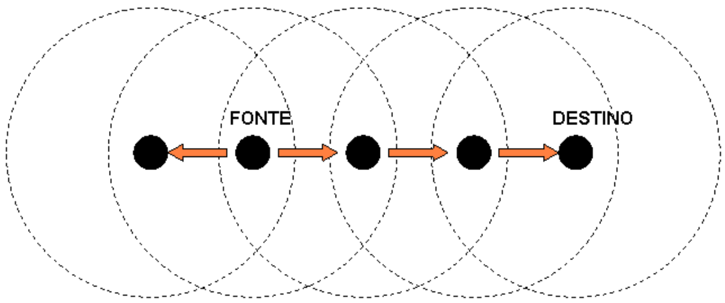
- Reativos - são aqueles que realizam o estabelecimento de uma rota apenas quando ela é solicitada pelo nó fonte ou origem. Um processo de descoberta de rota é iniciado quando um determinado destino deve ser alcançado e não existe rota estabelecida para o mesmo. Esse processo é finalizado quando o destino é alcançado ou quando, após tentativas todas as combinações de rotas possíveis, nenhuma é encontrada.
- Pró-ativos - mantêm informações sobre as rotas para todos os nós da rede, mesmo que o nó que executa esse algoritmo nunca tenha utilizado muitas dessas rotas, tanto para enviar seus próprios pacotes como para enviar pacotes de outros nós, fazendo papel de roteador. São usadas mensagens periódicas que são trocadas entre os nós da rede com o objetivo de manter a tabela de rotas de cada nó atualizada.
- Híbridos - apenas uma parte dos nós faz atualizações periódicas.

Levando em consideração as mudanças constantes de topologia, bem como as mudanças de relacionamento (com quais nós um certo nó se comunica), e além do mecanismo de descoberta de rotas, os protocolos de roteamento necessitam incorporar o mecanismo de manutenção de rotas. Esse mecanismo tentará restabelecer o caminho caso haja uma quebra de enlace devido a um nó que tenha saído do alcance ou da rede.

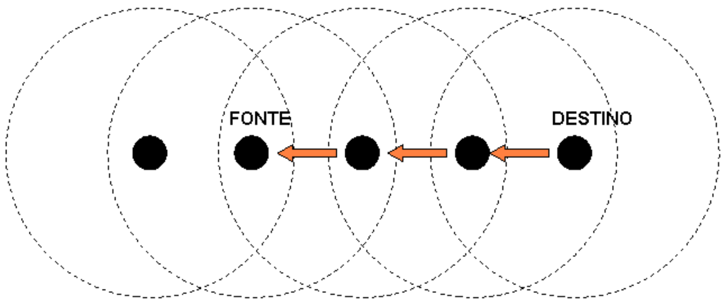
Vários protocolos de roteamento foram propostos, dentre eles destacam-se o AODV (*Ad Hoc On Demand Distance Vector*) [16, 17, 18, 19, 20], o DSDV (*Destination Sequenced Distance Vector*) [16, 17, 21] e o DSR (*Dynamic Souce Routing*) [16, 17, 18, 22] como os mais populares. O DSR serviu de base para o desenvolvimento do ANDSR e está descrito brevemente a seguir.

2.1.2 DSR - Dynamic Source Routing

O DSR [16, 17, 18, 22] é o protocolo mais antigo do IETF (*Internet Engineering Task Force*) para roteamento em redes ad hoc e o mais difundo e testado entre os vários protocolos de roteamento existentes. Ele opera sob demanda, ou seja, sua tabela de rotas é montada somente se houver necessidade de envio de mensagens e é composto por dois mecanismos principais: a descoberta de rota (*route discovery*) e a manutenção de rota (*route maintenance*).



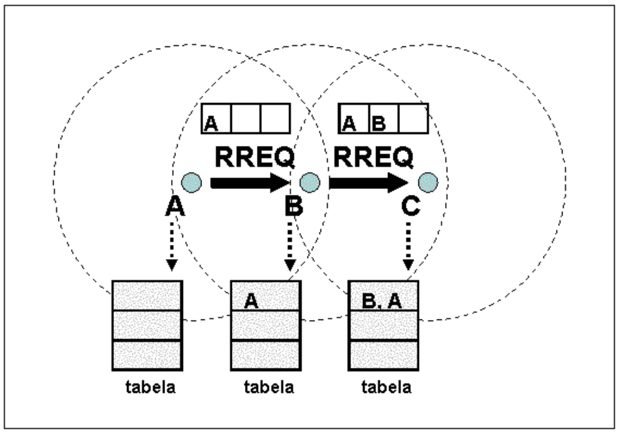
(a) **Propagação do RREQ**



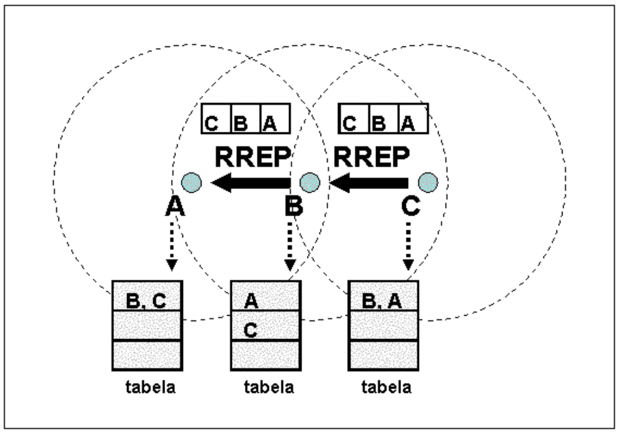
(b) **Propagação do RREP**

Figura 2.2: **Propagação de Mensagens**

A descoberta de rota é iniciada quando um nó recebe um pacote a ser enviado e não possui rota armazenada para o destino. Então, o nó, chamado fonte, envia uma requisição de rota (*RREQ - Route REQuest*) por difusão. Cada nó que recebe esse pacote verifica se ele já foi recebido anteriormente. Caso afirmativo, o pacote é descartado. Caso contrário, o nó, que não é o nó destino, coloca seu endereço como próximo nó intermediário e retransmite o pacote, por inundação (*flooding*) e envia uma confirmação de recebimento (*ACK - Acknowledge*) ao nó anterior. O próximo nó que receber o RREQ fará o mesmo até que o pacote alcance o destino, vide propagação do RREQ na Figura 2.2(a). Este nó, ao receber o RREQ, envia um (*RREP - Route REPly*) para a fonte através do caminho reverso, vide propagação do RREP na Figura 2.2(b).



(a) **Aprendizado de rotas na propagação do RREQ**



(b) **Aprendizado de rotas na propagação do RREP**

Figura 2.3: **Aprendizado de Rotas**

A manutenção de rotas é feita quando cada nó que transmite um pacote é responsável por armazenar o *ACK* recebido do nó seguinte, sem que haja sua retransmissão até o nó fonte. Caso ocorra quebra de um enlace, o nó que encaminha um pacote e ao não receber o *ACK*, procura uma nova rota para o destino em sua tabela. Caso não tenha esta rota, o nó envia uma mensagem de erro (*RERR - Route ERRor*) ao seu nó antecessor que age da mesma forma até que o RERR alcance o nó fonte. Quando o nó fonte receber esse pacote, verifica se existe outra rota para o destino. Caso negativo, deve-se iniciar nova descoberta de rota.

O DSR é capaz de aprender rotas com os pacotes recebidos. Quando um nó A acha uma rota para C através de B, B aprenderá uma rota para A ao receber o RREQ de A, e C aprenderá uma rota para A passando por B ao receber o RREQ de B, como na Figura 2.3(a). Além disso, no caminho reverso, B aprenderá uma rota para C com o recebimento do RREP de C e A aprenderá a rota para C ao receber o RREP, vide Figura 2.3(b).

Um nó executando o algoritmo DSR também possui a capacidade de aprender rotas mesmo quando o pacote não é endereçado a ele. Isto acontece quando esse nó está em modo promíscuo. Nesse modo, o nó fica escutando o tráfego dos nós dentro de seu raio de alcance. É vantajoso, pois o nó aprende rotas, porém o nó consome bastante energia de sua bateria e acaba favorecendo a análise de tráfego.

2.1.3 Aspectos Importantes

A pesquisa dessas redes tem focado seus estudos nas áreas a seguir.

Consumo de Energia

As limitações das baterias, típicas das redes ad hoc, reduzem o número de participantes nas transmissões tornando essencial algum mecanismo de conservação de energia [23]. Para minimizar esses problemas vários protocolos para gerenciamento eficiente de energia estão sendo propostos para redes ad hoc e para redes de sensores que é uma situação particular das redes ad hoc [24].

Qualidade de Serviço

A qualidade de serviço (QoS) é usualmente definida como um conjunto de requerimentos de serviços que uma rede deve encontrar enquanto transporta pacotes entre fonte e destino [23]. Em redes ad hoc, o roteamento com qualidade de serviço é difícil devido à constante mudança de topologia. Com isso, as informações disponíveis sobre o estado da rede são completamente imprecisas. Quanto mais rápido a topologia mudar, a provisão de QoS se torna muito difícil. Algumas técnicas estão sendo pesquisadas e propostas para tornar QoS viável em redes ad hoc [25, 26].

Segurança

Pesquisas em redes ad hoc costumam se basear em preocupações como gerar um mínimo de tráfego possível e consumir o mínimo de recursos dos dispositivos, como baterias, memória e processador. Mesmo assim, por depender dos próprios usuários da rede para realizar o roteamento, o consumo dos recursos e a quantidade de tráfego gerado são relativamente altos. Implementar qualquer tipo de segurança em redes com essas características é bastante complicado.

A natureza dinâmica e não infraestruturada das redes ad hoc as tornam vulneráveis a uma variedade de ataques. Pode-se distinguir essencialmente dois grandes conjuntos de vulnerabilidades, as relacionadas aos mecanismos básicos e as relacionadas aos mecanismos de segurança. A primeira se refere a mecanismos de operação da rede, onde o mais crítico é o roteamento. A segunda diz respeito às falhas nos próprios mecanismos que deveriam proteger a rede contra ataques e ameaças [27].

Focando nas vulnerabilidades dos mecanismos de roteamento, pode-se classificar os ataques aos protocolos de roteamento em duas categorias principais [28, 29]:

- Ataques Passivos - são aqueles nos quais o intruso não participa ativamente na degradação da operação normal da rede. Um intruso somente espiona o tráfego da rede para obter informações sobre os nós e seus relacionamentos (com quais nós um certo nó se comunica) na rede. Ele também pode repassar as informações obtidas a um cúmplice que poderá usá-las para iniciar ataques que destruam a operação da rede. Como exemplo de ataque passivo tem-se a análise de tráfego. Esse ataque consiste em capturar os pacotes no intuito de observar suas características e conteúdo. É bastante difícil de ser detectado.

- Ataques Ativos - são aqueles nos quais o intruso participa ativamente na destruição do funcionamento normal da rede e seus serviços. Como exemplo de ataques desse tipo cita-se:

  – Identidade Falsa (*Impersonation*) - Nós maliciosos podem se juntar à rede sem serem detectados podendo enviar falsas informações de roteamento como se fosse um outro nó.

  – Negação de Serviço (*Denial-of-Service*) - Um exemplo de ataque desse tipo seria quando um nó malicioso tenta criar rotas para nós não existentes oprimindo o funcionamento dos protocolos de roteamento.

2.2 ANDSR - Anonymous Dynamic Source Routing

Esta seção apresenta o protocolo de roteamento anônimo ANDSR (*Anonymous Dynamic Source Routing*) descrevendo seu funcionamento e características.

2.2.1 Considerações do Projeto

O protocolo de roteamento proposto, ANDSR, é uma extensão do protocolo DSR, pois traz algumas modificações que permitem ao algoritmo maior resistência à análise de tráfego. O DSR serviu de base para esta proposta devido a alguns fatores principais:

- É um dos mais antigos protocolos de roteamento do IETF (*Internet Engineering Task Force*) e desta forma, o DSR foi amplamente estudado tornando-se bastante conhecido pela comunidade científica;

- Sob o ponto de vista de desempenho, as pesquisas mostram que possui um bom desempenho em redes de baixa à média mobilidade [16, 18, 17, 30];

- Com relação à análise de tráfego, é bastante vulnerável devido à escuta promíscua do meio de comunicação e ao fato deste protocolo armazenar o caminho (rota) no pacote de roteamento.

O ANDSR utiliza o conceito de nós misturadores *(mix nodes)* [1] que são nós confiáveis e responsáveis pelo encaminhamento dos pacotes de maneira anônima. Esses nós misturadores formam um grupo que divide o mesmo par de chaves assimétricas, uma chave pública e outra privada. Considerando o pior caso, um misturador comprometido, serão descobertas as suas chaves e com isso a identidade do nó destino de um pacote RREQ (*Route REQuest*) que chegue a ele. Os ataques passivos são invisíveis aos usuários da rede e têm como objetivo espionar ou bisbilhotar informações a respeito dos nós e suas identidades. Mesmo que um nó misturador seja comprometido e as chaves compartilhadas sejam descobertas, a única informação importante que certamente seria descoberta é a identidade do nó destino. Isto porque os misturadores não conhecem as identidades dos nós fontes das comunicações, elas estão criptografadas com a chaves dos nós destinos. O fato dos misturadores compartilharem as mesmas chaves torna o algoritmo mais simples, as mensagens para os misturadores podem ser enviadas em difusão e não individualmente a cada misturador, além disso, os nós da rede necessitariam conhecer todos misturadores.

O processo de distribuição de chaves e o protocolo utilizado para tal não fazem parte do escopo deste projeto, desta maneira, considera-se que as chaves já foram distribuídas com sucesso. As chaves públicas como o próprio nome sugere são de domínio público e, portanto, são do conhecimento de qualquer nó da rede.

A distribuição de nós misturadores na rede é totalmente aleatória, ou seja, eles se movimentam livremente e não há a necessidade de um conhecimento prévio de sua posição. Deve existir misturadores suficientes para que se tenha pelo menos um deles dentro do raio de alcance de um certo nó. Caso não haja nó misturador no alcance do nó fonte da comunicação, este deve iniciar o algoritmo novamente até que pelo menos um misturador esteja sob seu alcance.

Considerou-se que todos os nós da rede ad hoc são dispositivos ou máquinas exatamente iguais sob o ponto de vista computacional. Assim, nós intrusos possuem as mesmas capacidades computacionais que qualquer nó legítimo da rede. Neste projeto, os ataques são somente passivos do tipo análise de tráfego, onde os nós maliciosos observam o tráfego com intuito de obter informações. Nós legítimos podem ser violados sem que isso cause dano ao funcionamento da rede ou seja percebido pelos demais nós.

As falhas de comunicação tratadas neste trabalho se referem às quebras de enlace entre os nós devido a movimentação dos mesmos.

Dentre as aplicações onde pode ser utilizado o ANDSR, pode-se citar as aplicações militares, ou em qualquer aplicação que se deseje preservar o anonimato das partes envolvidas na comunicação.

2.2.2 Notação

Esta seção apresenta as notações usadas na modelagem do protocolo ANDSR. Elas aparecem na seção posterior e em todos os códigos da especificação do ANDSR.

São elas:

- S - Representa o nó fonte (*source*).
- T - Representa o nó destino (*target*) da comunicação.
- I - Representa o nó intruso.
- M - Representa o nó misturador.
- Kx - Representa a chave pública do nó X. Por exemplo, o nó destino T possui chave pública Kt.
- R - Representa um número aleatório gerado por um nó.
- Ax - Representa o endereço do nó X. Por exemplo, o nó fonte S possui endereço As.

2.2.3 Protocolo Proposto

O funcionamento básico do protocolo ANDSR está descrito a seguir:

- Passo 1 - O nó fonte *source* S deseja estabelecer comunicação anônima com o nó destino *target* T, porém não tem caminho para T. Então, S deve iniciar a descoberta de rota com encaminhamento de pacotes através de nós misturadores. S não sabe quais nós são misturadores, então, envia um pacote de pedido chamado *mix request* em difusão.

- Passo 2 - Os nós que receberem o *mix request*, caso sejam misturadores, responderão com um pacote chamado *mix reply* como na Figura 2.4, caso contrário descartam o pacote. As mensagens *mix request* e *mix reply* não são criptografadas.

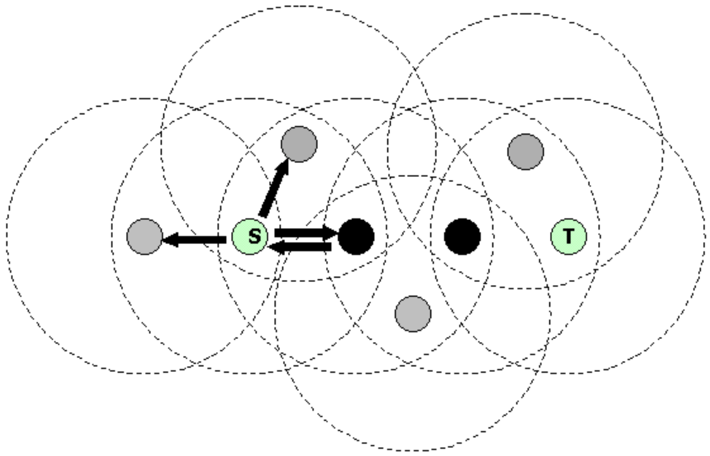


Figura 2.4: **Envio do Mix Request e recebimento do Mix Reply**

- Passo 3 - Ao receber um *mix reply*, S monta um *route request* que contém: o endereço de origem (endereço de S) criptografado por Kt, o endereço de destino final (endereço de T) criptografado por Km e as informações destinadas somente a T criptografadas por Kt (chave pública de T). S envia o *route request* para o misturador do qual recebeu um *mix reply*.

- Passo 4 - Quando um misturador recebe um *route request*, ele decifra o endereço de destino final através de sua chave privada e verifica se é vizinho de T. Caso negativo, ele encripta novamente o endereço de T com Km e envia o pacote em difusão, vide Figura 2.5.

- Passo 5 - O próximo misturador age da mesma forma, até que o pacote seja recebido por um misturador vizinho a T. Neste caso, o misturador criptografa o endereço de T com Kt e envia por difusão. Somente T pode decifrar a mensagem, os demais nós descartam o pacote.

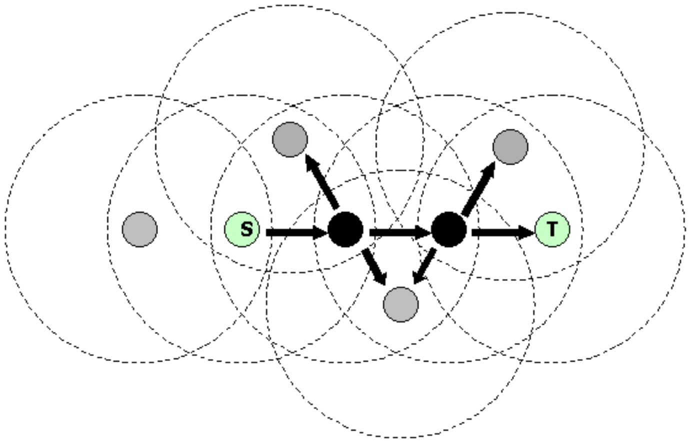


Figura 2.5: **Envio do Route Request**

- Passo 6 - T monta seu *route reply* que contém o endereço de origem (endereço de T) e o endereço de destino final (endereço de S) criptografados por Ks. T envia o *route reply* para S pelo caminho reverso, vide Figura 2.6.

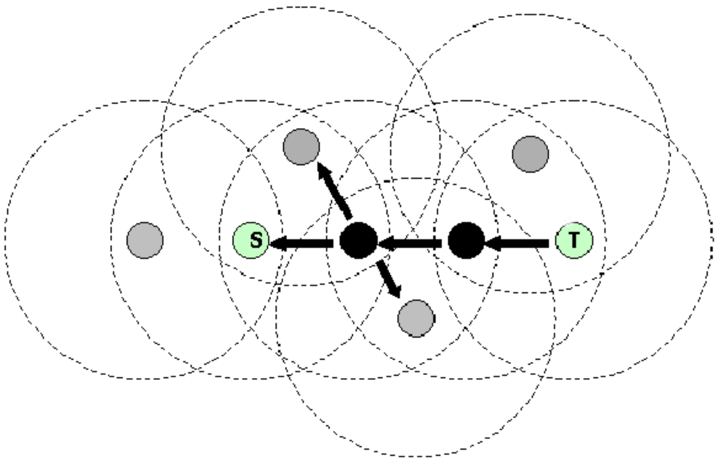


Figura 2.6: **Envio do Route Reply**

- Passo 7 - Como o endereço de destino final do *route reply* está criptografado por Ks, o misturador vizinho a S não sabe para quem entregar o pacote. Então, o pacote é enviado por difusão. Somente o nó S é capaz de decifrá-lo, os demais nós descartam o pacote. Fica estabelecido, então, o caminho anônimo entre S e T.

- Passo 8 - Caso haja uma quebra de enlace, o nó misturador deve enviar um *route error* para seu nó antecessor (do qual recebeu um *route request*) e assim por diante até alcançar o misturador vizinho ao nó S. Como esse nó misturador desconhece a

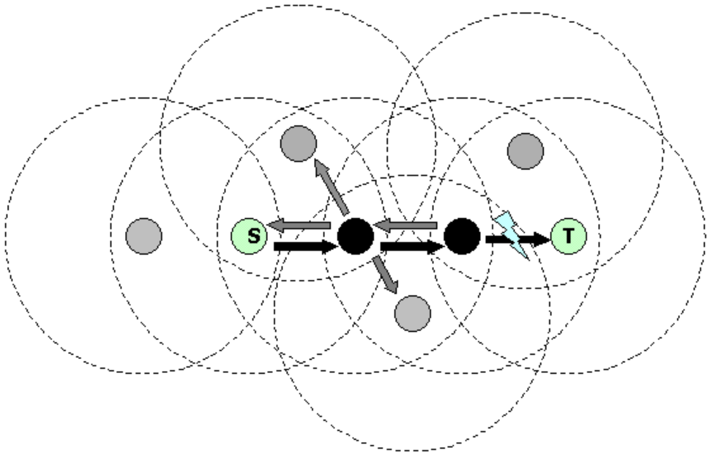


Figura 2.7: **Envio do Route Error**

identidade da fonte, ele envia o *route error* por difusão, vide Figura 2.7. As setas cinzas representam o pacote *route error*. Todos os nós na vizinhança recebem esse pacote, que só terá significado para S. Caso não exista outra rota armazenda para T, S deve reiniciar o processo de descoberta de rota.

2.2.4 Pacotes

Nesta seção, apresenta-se os pacotes de roteamento usados pelo protocolo proposto ANDSR. Os pacotes de RREQ (*Route Request*) e RREP (*Route Reply*) seguem o mesmo formato utilizado pelos pacotes usados no protocolo de roteamento DSR, porém existem algumas modificações. Além disso, o protocolo ANDSR possui pacotes novos (*Mix Request* e *Mix Reply*), não existentes no DSR. Todas essas modificações têm como objetivo manter o anonimato no roteamento.

As Figuras 2.8(a) e 2.8(b) mostram, repectivamente, as opções de cabeçalho do pacote RREQ (*Route Request Options Header*) e as opções de cabeçalho do pacote RREP (*Route Reply Options Header*), ambos do protocolo DSR . Observando essas figuras fica fácil perceber a facilidade que um intruso teria em adquirir informações sobre a rota, visto que ela fica armazenada no próprio pacote.

O significado de cada um desses campos está descrito a seguir:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     tipo      |  op. tamanho  |         Identificação         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Endereço de Destino                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Endereço [1]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Endereço [2]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Endereço [n]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(a) **Opções do Cabeçalho do RREQ do DSR**

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                |     tipo      |  op. tamanho  |L|  reservado  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Endereço [1]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Endereço [2]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Endereço [n]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(b) **Opções do Cabeçalho do RREP do DSR**

Figura 2.8: **Cabeçalho dos Pacotes**

- **Tipo** - Este campo designa o tipo de pacote de roteamento.
- **Op. Tamanho** - Tamanho das opções em octetos, excluindo o campo tipo e o próprio campo Op. Tamanho.
- **Identificação** - Valor gerado pela fonte da comunicação ao iniciar um RREQ. A cada vez que se iniciar um RREQ é gerado um novo valor de identificação. Assim, um pacote que receber uma cópia de um RREQ pode verificar se já havia recebido antes e então, descartá-lo.
- **Endereço do Destino** - Endereço do destino final da comunicação no pacote RREQ.
- **Endereço [1..n]** - Endereços dos nós que encaminham os pacotes até o destino final no pacote RREQ e o caminho reverso no pacote RREP.
- **Reservado** - Deve ser zero e ignorado na recepção.

- **L (*Last Hop External*)**- Indique o último salto dado por um pacote RREP fora da rede DSR.

Além desses campos, todo pacote de roteamento possui os seguintes campo IP:

- **Endereço da Fonte**
- **Endereço do Destino Intermediário** - Endereço do próximo salto ou endereço *broadcast*.
- **TTL (*Time to Live*)** - Limite de Saltos

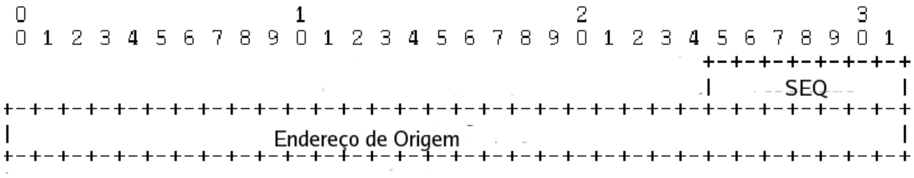
O protocolo ANDSR propõe algumas alterações nos campos destes pacotes. Ao invés do endereço dos nós tem-se os endereços criptografados dos nós de acordo com a descrição do funcionamento na seção anterior. Além disso, o campo identificação no pacote RREQ também deve ser criptografado com a chave dos misturadores para evitar que um intruso saiba de qual rota se trata. Não há problema se um nó comum qualquer receber cópias do mesmo pacote, pois todos os pacotes de roteamento recebidos por eles devem ser descartados, visto que somente os misturadores encaminham os pacotes de roteamento.

O ANDSR introduz novos tipos de pacotes, o *Mix Request* e o *Mix Reply*, cujos cabeçalhos são sugeridos, como mostra a Figura 2.9.

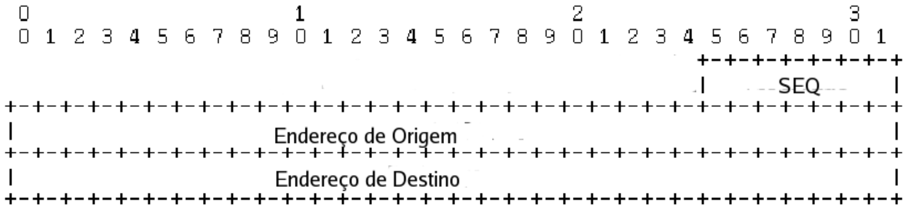
Na figura 2.9(a), o campo *Endereço de Origem* representa o endereço da fonte que inicia o mecanismo de descoberta de rotas através da busca por misturadores. O campo *Seq* é um valor gerado pelo iniciador do requerimento. Da mesma forma, na figura 2.9(b), o campo *Endereço de Origem* representa o endereço do misturador que recebeu um *Mix Request*, o campo *Endereço de Destino* representa o endereço de destino, no caso, o endereço do iniciador do requerimento *Mix Request* e, por fim, o campo *Seq* é o mesmo do pacote *Mix Request*.

2.2.5 Criptografia

As técnicas de criptografia podem ser usadas como meio efetivo de proteção de informações suscetíveis a ataques, estejam elas armazenadas em um dispositivo (máquina) ou



(a) **Mix Request**



(b) **Mix Reply**

Figura 2.9: **Pacotes do ANDSR**

sendo transmitidas pela rede. Seu objetivo principal é prover uma comunicação segura, garantindo também a privacidade e integridade dos dados.

A segurança de um sistema de criptografia não deve ser baseada somente nos algoritmos que cifram as mensagens, mas também no tamanho das chaves usadas. Além disso, um algoritmo de criptografia é considerado forte, caso seja impossível quebrá-lo em um certo espaço de tempo em que as informações ainda sejam relevantes e possam ser usadas por usuários não autorizados. Para as redes ad hoc, o tamanho das chaves deve ser grande o suficiente para ser seguro e ao mesmo tempo não pesar no processamento.

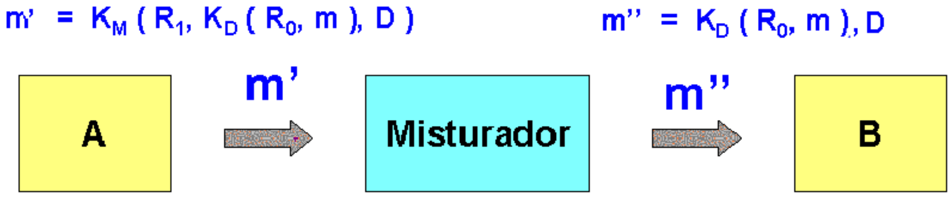


Figura 2.10: **Esquema proposto por D. Chaum**

Como mencionado anteriormente, um dos pioneiros nos estudos sobre anonimato foi D. Chaum que em seu trabalho [1] apresentou uma solução para o problema da análise de tráfego baseada em criptografia de chave pública [31]. Nesse trabalho, um usuário A que deseje enviar uma mensagem para um usuário B via e-mail, não enviará a mensagem diretamente. Sua mensagem será enviada para uma máquina chamada misturador (*mix*) e esse misturador entregará a mensagem ao usuário B. A função do misturador é esconder a identidade dos usuários como mostra o esquema da Figura 2.10. Nesse esquema K_M é chave pública do Misturador, K_D é a chave pública do destino, R_0 e R_1 são números randômicos gerados e D é o endereço do destino. Baseado nesses conceitos e idéias que foi proposto o ANDSR, na tentativa de dificultar a análise de tráfego em redes ad hoc.

2.3 Comentários

Neste capítulo foram apresentadas as redes ad hoc e o protocolo de roteamento anônimo ANDSR (*Anonymous Dynamic Source Routing*) que estabelece rotas anônimas entre dois dispositivos, dificultando uma eventual análise de tráfego executada por um ou mais nós maliciosos. O capítulo posterior trata do projeto do protocolo proposto, descrevendo, também, a linguagem formal LOTOS (*Language of Temporal Ordering Specification*) usada para a modelagem, o software de análise CADP (*CAESAR/ALDEBARAN Development Package*), a metodologia adotada e o desenvolvimento da modelagem do protocolo ANDSR.

Capítulo 3

Projeto

MUITOS protocolos são utilizados com sucesso, porém quase todos sofrem de alguma falha não prevista, como bloqueios *deadlocks* e estados não alcançados *livelocks*. É comum que protocolos propostos como documentos padrões (*draft standards*) possam ser vistos como uma seqüência de versões, onde imperfeições e erros são corrigidos. Isso ocorre freqüentemente, pois as especificações desses protocolos foram descritas em linguagem natural e não podem ser verificadas para a correção de erros e ambigüidades. Qualquer erro contido numa padronização pode ter um custo elevado quando usado para propósitos industriais. Essas são algumas das razões pelas quais a ISO (*International Organization for Standardization*) começou a desenvolver as FDTs (*Formal Description Techniques*) em meados dos anos 80. O objetivo era tornar as especificações livres de ambigüidades, habilitar a capacidade de análise de especificações, atuar como um guia para implementações e atuar como base para testes de conformidade.

Algumas das FDTs mais conhecidas e utilizadas são: FSM (*Finite State Machine*)[32], SDL (*Specification and Description Language*)[32], Estelle (*Extended Finite State Machine Language*)[32], Esterel [32], TL (*Temporal Logic*) [32], Promela [32] e LOTOS (*Language of Temporal Ordering Specification*)[10, 32].

A modelagem de um protocolo envolve a especificação do mesmo através do uso de uma técnica de descrição formal, a verificação das propriedades e características e a sua validação. Tudo com o objetivo de uma possível e futura implementação. Este capítulo aborda o desenvolvimento do projeto, a metodologia usada e as técnicas adotadas para modelagem do protocolo proposto. A seção 3.1 descreve a Linguagem LOTOS (*Language of Temporal Ordering Specification*) usada neste projeto. A seção 3.2 apresenta o software usado, o CADP. A seção 3.3 apresenta a metodologia e a seção 3.4 trata da modelagem do protocolo ANDSR. Por fim, a seção 3.5 faz os comentários finais.

3.1 Linguagem LOTOS

LOTOS (*Language of Temporal Ordering Specification*) [10] é uma técnica de descrição formal desenvolvida pela ISO (International Organization for Standardization) para a especificação formal de sistemas distribuídos e concorrentes em geral, além de ser adequada para especificação de protocolos.

A idéia básica desenvolvida pela linguagem LOTOS é que os sistemas podem ser especificados pela definição de relações temporais através de interações que constituem o comportamento externamente observável de um sistema. Na verdade, essa técnica não esta relacionada com a lógica temporal, mas é baseada em métodos de processos algébricos. Tais métodos foram inicialmente introduzidos pelo trabalho de Milner em CCS (*Calculus of Communicating System*) [33] e seguidos por outras teorias que se referem à álgebra de processos.

LOTOS é composta por dois componentes:

- LOTOS básico - Sua representação é inspirada em álgebra de processos CCS (*Calculus of Communicating System*) e CSP (*Communicating Sequential Process*) e descreve o modelo comportamental do sistema.
- LOTOS completo - Inclui a linguagem ACT ONE (*Abstract Data Type Formalism*), onde os tipos de dados são descritos por seus operadores e equações, as quais são especificadas mediante o emprego de operações algébricas. Isso permite a definiçao de variáveis e troca de dados entre processos.

A Tabela 3.1 apresenta uma lista contendo alguns operadores em LOTOS.

OPERADOR	DESCRIÇÃO
A[f,g]	Instanciação do processo A, onde f e g são as portas por onde dados são trocados
A[f,g](S,T)	Chamada do processo A com portas f e g e parâmetros de valores S e T
hide g, h in	Portas serão escondidas na apresentação dos resultados
p!V?X:T	Interação pela porta p, envio do valor V e recepção em X de um valor do tipo T
A ||| B	Funcionamento dos processos A e B são independentes e em paralelo
A || B	Sincronismo entre os processos A e B
A | [p] | B	Sincronismo entre os processos A e B através da porta p
A [ ] B	Escolha entre os processos A e B, com igual probabilidade
A >> B	O processo A habilita o processo B
exit	Termina com sucesso
stop	Parada de processo

Tabela 3.1: **Alguns operadores de LOTOS**

Em LOTOS, um sistema distribuído é modelado como um processo que pode ser decomposto em diversos subprocessos. Um processo é uma entidade capaz de realizar não só ações internas não observáveis, mas também interações com outros processos externos, os quais compõem seu ambiente observável. Essas interações são descritas unidades de sincronismo, os denominados eventos ou ações que são executados através de pontos de interações (portas de comunicação).

Considerando o exemplo a Figura 3.1, o processo *Max3* é definido como a composição em paralelo de duas instâncias do processo *Max2*. Cada um dos processos pode interagir com seu próprio ambiente, que consiste da outra instância de *Max2*, através de três portas, mas a única porta de sincronização entre os processos é *mid*. Essa porta também faz parte do processo *Max3*. Como o processo *Max3* é uma caixa preta, a porta *mid* não é visível de fora do processo *Max3*.

As duas instâncias do processo *Max2* interagem independentemente através de todas as portas exceto a porta *mid*. Nessa porta eles se sincronizam um com o outro sem a visualização dessas interações, pois *mid* está escondida. Essas interações se tornam ações internas ao sistema.

O trecho a seguir representa o sistema da Figura 3.1 descrito em LOTOS.

```
Process Max3 [in1, in2, in3, out]:=

hide mid in
```

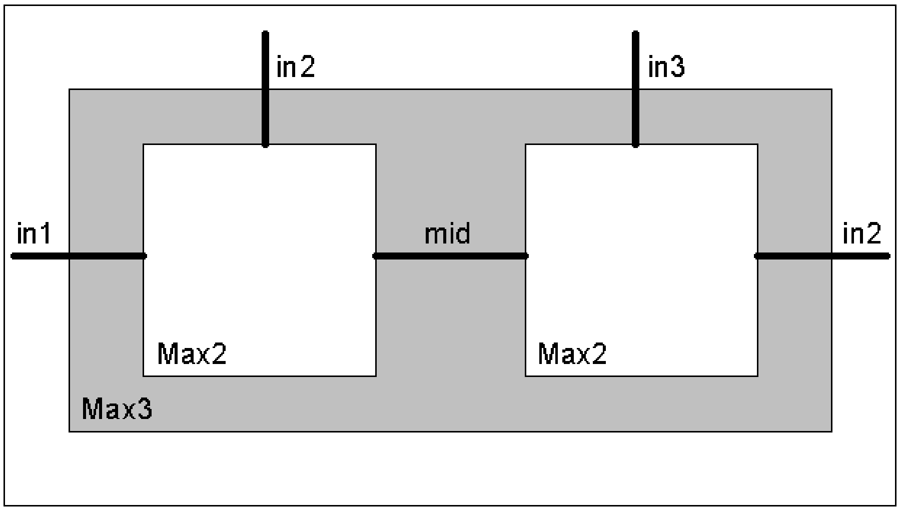


Figura 3.1: **Representação Espacial do Processo Max3**

```
( Max2 [in1, in2, mid]
  |[mid]|
  Max2 [mid, in3, out]
)

where
.
.
endproc (* Fim do Processo Max3 *)
```

3.2 O Pacote de Ferramentas CADP

O CADP (*CAESAR/ALDEBARAN Development Package*) [13] é um conjunto de ferramentas adequado à engenharia de protocolos. Foi desenvolvido pelo grupo de pesquisa VASY (*Validation of Systems*) do Instituto INRIA Rhône-Alpes juntamente com os laboratórios *Verimag* e *Bull*. Dedica-se à eficiente compilação, simulação, verificação formal e testes de descrições escritas em LOTOS. Dentre as ferramentas disponíveis nesse pacote destacam-se:

- CAESAR - É um compilador que traduz a parte comportamental da especificação em LOTOS num programa em C que será executado/simulado gerando um LTS (*Labelled Transition System*) que será verificado usando ferramentas de bissimulação (relações de equivalência).

- CAESAR.ADT - É um compilador que traduz a parte de dados das especificações escritas em LOTOS em bibliotecas de funções e tipos em C.
- ALDEBARAN - É a ferramenta para a verificação de sistemas comunicantes representados por LTSs. Permite a comparação, avaliação e redução dos LTSs através de diversas relações de equivalência entre o modelo do sistema e do respectivo serviço.
- BCG (*Binary Coded Graphs*) - É simultaneamente o formato para a representação dos LTSs e uma coleção de bibliotecas e programas que lidam com esse formato. As ferramentas a seguir estão disponíveis para o formato:
  - BCG_DRAW - Provê uma representação gráfica 2D do grafo BCG com um *layout* automático de estados e transições.
  - BCG_EDIT - É um editor interativo que permite a modificação manual da representação gerada pela ferrramenta BCG_DRAW.
  - BCG_IO - Executa conversões entre o formato BCG e vários outros formatos.
  - BCG_INFO - Provê informações sobre o grafo BCG, tais como o número de estados e transições, etc.
  - BCG_MIN - Minimiza o grafo de acordo com forte bissimulação.

Além dessas ferramentas o CADP possui uma interface TCL-TK chamada Eucalyptos, Figura 3.2, que proporciona acesso fácil a todas as ferramentas e aquivos do projeto.

A Figura 3.3 apresenta um esquema ilustrando o procedimento para a especificação formal usando a ferramenta CADP.

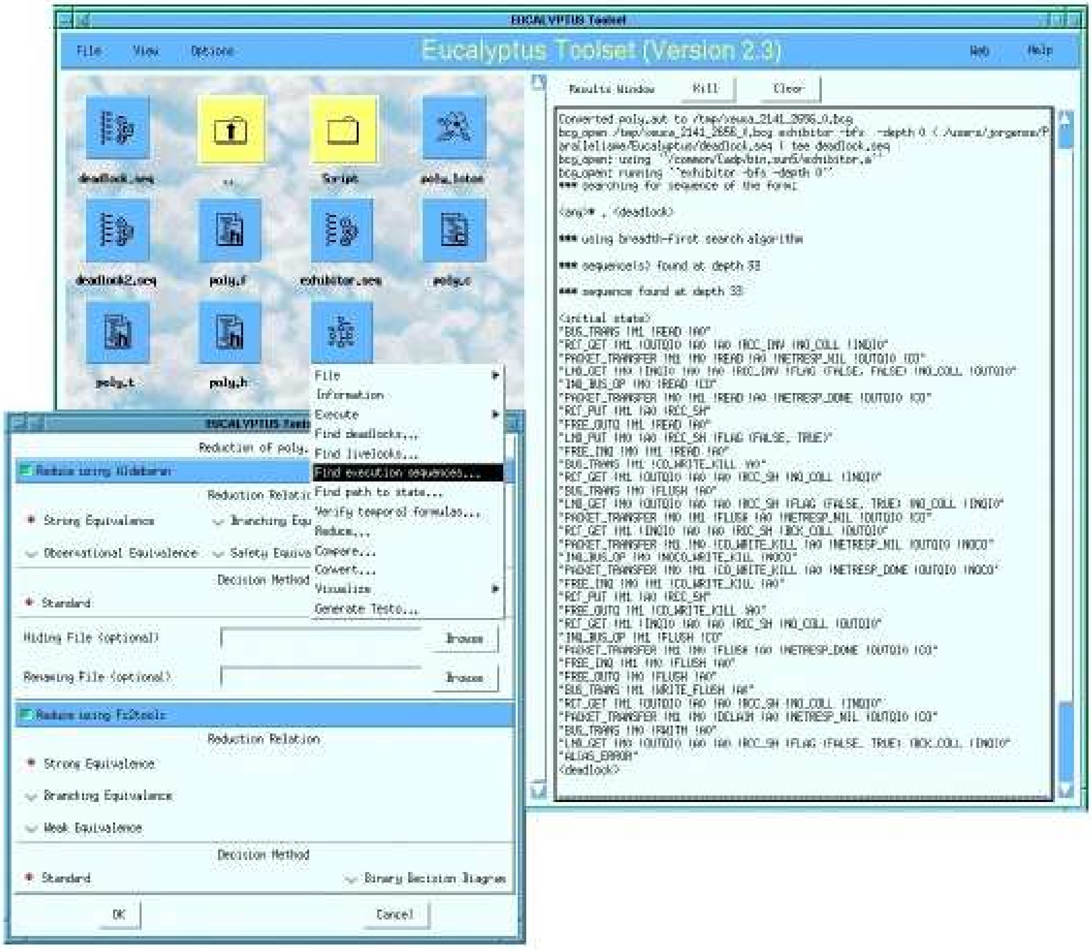


Figura 3.2: **Interface Eucalyptos**

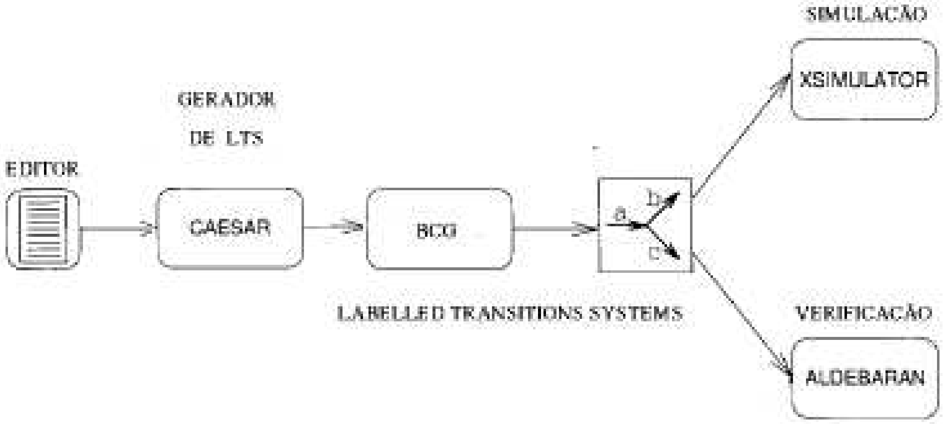


Figura 3.3: **Ferramentas CADP**

3.3 Metodologia

Uma metodologia apresenta uma regra de aplicação de um método independentemente de como a implementação é fundamentada, dando bastante liberdade àqueles que irão usá-la [11]. A metodologia apresentada neste projeto mostra uma maneira de modelar protocolos e propriedades que possam ser verificadas com o modelo desenvolvido. Essa metodologia utiliza, inicialmente, vários estudos de casos para modelar os diversos comportamentos e situações que o protocolo pode enfrentar. Os estudos de casos estão baseados em topologias específicas definidas pelo projetista. A grande vantagem do uso de estudos de casos é ganhar experiência no domínio da linguagem formal adotada, além de detalhar o funcionamento do protocolo. Neste projeto, os estudos de casos são evolutivos, ou seja, a topologia de rede selecionada é alterada com a inserção de novos nós, oferecendo maior consistência ao modelo. Cada nó inserido representa um novo comportamento modelado. Porém, os estudos de casos podem se tornar bastante complexos e o número de estados tende a crescer consideravelmente a cada inserção de nó.

Como as redes ad hoc são dinâmicas, verificar todas as combinações possíves de topologia mesmo para um número pequeno de nós é inviável. Surge a necessidade de um modelo mais abrangente, genérico. Após a modelagem dos estudos de casos criou-se experiência suficiente para desenvolver o modelo genérico do protocolo ANDSR. Esse modelo é independente da topologia da rede, do número de nós participantes e de suas posições na rede. Além disso, baseou-se na metodologia desenvolvida no trabalho [11].

3.4 Modelagem do Protocolo ANDSR

A modelagem do protocolo ANDSR foi feita usando a linguagem formal LOTOS. Sua escolha se deve, sobretudo, ao seu rigor matemático na descrição de eventos. A ferramenta de análise usada para validação foi o conjunto de ferramentas CADP, devido a sua interface amigável, ao suporte técnico dos desenvolvedores da ferramenta na França e por ser gratuita.

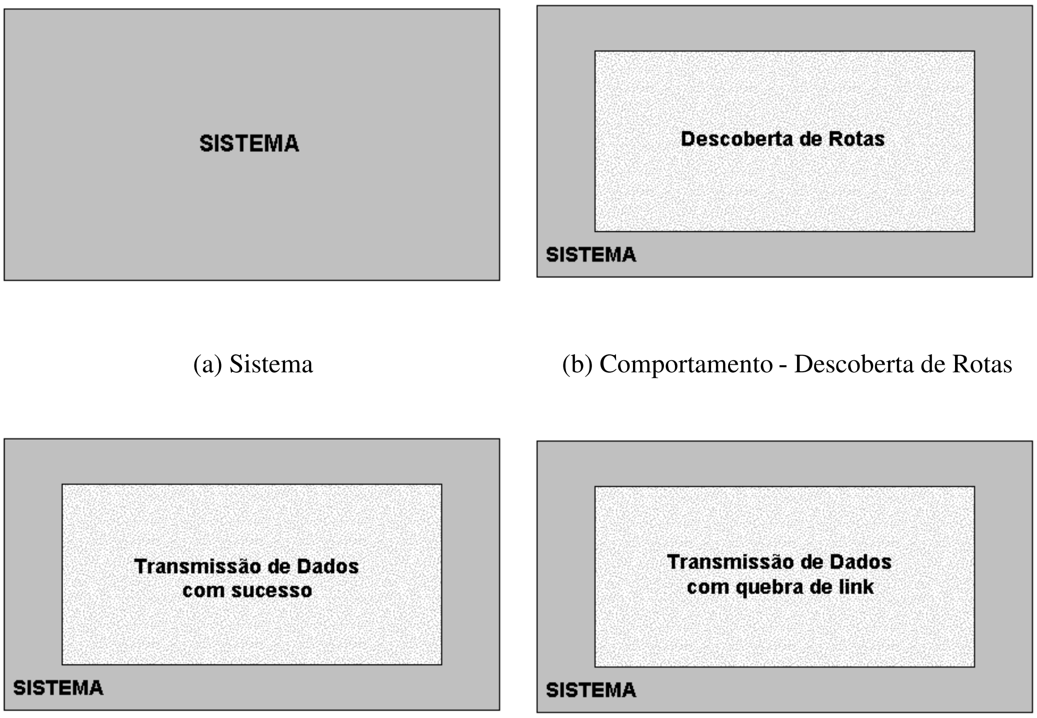
Após a definição da linguagem e ferramentas, parte-se para a modelagem do protocolo, definindo os nós, o canal de comunicação entre eles, etc. Para a descrição do protocolo proposto em LOTOS, considera-se que os protocolos da camada imediatamente inferior a sua lhe fornecem os serviços necessários e corretos para um perfeito funcionamento. Outro dado importante é que, em LOTOS, o tempo não pode ser expresso explicitamente. A maneira usada no projeto para criar o efeito de tempo foi criar uma mensagem chamada *Sincronismo*. Para compreender melhor, considere que um nó A envie uma mensagem RREQ em broadcast. Dois nós, B e C, que estão no alcance de A, receberão o RREQ. Considere também que B receberá o RREQ primeiro. Então, quando A enviar o RREQ envia também a mensagem Sincronismo. O nó B receberá as duas mensagens de A e enviará a mensagem Sincronismo a C. Ao receber esse Sincronismo, C estará apto a receber o RREQ de A. Em todas as especificações do protocolo ANDSR usou-se desse artifício para criar a idéia de tempo.

Este projeto pode ser dividido em duas fases, a primeira referente à experiência dos estudos de casos e a segunda referente ao desenvolvimento de um modelo genérico do protocolo. Os estudos de casos se basearam numa topologia específica. A partir de uma dada rede ad hoc, modelava-se a especificação do protocolo, onde cada nó representava um tipo de comportamento. Entretanto, este modelo, a medida que o número de nós aumenta, torna-se bastante complexo. Com isso, houve a necessidade de um modelo genérico, independente da topologia e do número de nós considerados. A modelagem dessas duas fases está descrita a seguir.

3.4.1 Estudos de Casos

O ponto de partida da modelagem foi apenas dois nós. Isto pode parecer desnecessário, mas é fundamental para a correção de erros, melhor domínio da linguagem e ferramenta e para a verificação de comportamentos passo a passo. Nesta fase, a adição de nós na rede representa a adição de novos comportamentos, ou seja, cada nó adicionado à rede representa mais um comportamento especificado [34, 35].

O roteamento de pacotes é função da camada de rede (Modelo OSI), logo o protocolo ANDSR reside na camada rede. Essa camada é o sistema modelado formalmente. O sistema é considerado uma caixa preta como pode ser visto na Figura 3.4(a). Dentro do sistema é executado o algoritmo ANDSR que é composto pelo mecanismo de descoberta de rotas e manutenção de rotas. A manutenção de rotas, no projeto, é representada por uma transmissão de dados com sucesso e por uma transmissão de dados com quebra de



(a) Sistema

(b) Comportamento - Descoberta de Rotas

(c) Comportamento - Transmissão de Dados com Sucesso

(d) Comportamento - Quebra de Enlace

Figura 3.4: **Modelagem**

enlace. A partir disso, considera-se que o sistema possui três comportamentos: descoberta de rotas, transmissão de dados com sucesso e transmissão de dados com quebra de enlace.

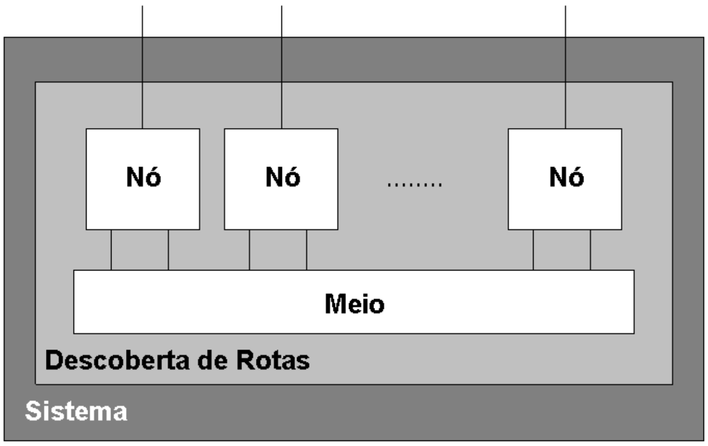


Figura 3.5: **Subprocessos e comportamentos**

Cada um desses comportamentos é representado como uma caixa preta dentro do sistema, vide Figura 3.4.

Dentro da caixa preta descoberta de rotas existe várias caixas pretas representando os nós da rede e o canal de comunicação entre eles durante o mecanismo de descobertas de rotas, vide Figura 3.5. Nessa Figura as linhas negras representam as portas de comunicação. O mesmo é considerado para a transmissão de dados com sucesso e com quebra de Enlace. O esboço da especificação do protocolo tem o seguinte aspecto:

```
Specification ANDSR ...
library ... (* declaração da biblioteca *)
behaviour
  Sistema[...]
where
        process Sistema ...
        .
        .
        where
            process Descoberta_de_Rotas ...
            .
            .
            where
                 process no_1 ...
                 .  (* descrição do nó 1 *)
                 .
                 endproc
                 .
                 .
                 process no_n ...
                 .  (* descrição do nó n *)
                 .
                 endproc
                 process Meio ...
                 .  (* descrição do canal de comunicação *)
                 .
                 endproc
            endproc
```

```
                .    (* O mesmo p/ outros processos *)
                .
          endproc
endproc

endspec
```

Exemplo - Estudo de Caso: Rede com cinco nós

Para ilustrar a metodologia empregada toma-se como exemplo a especificação do protocolo para uma rede com cinco nós (um nó fonte, um destino, dois misturadores e um nó qualquer) [35]. A Figura 3.6 apresenta a topologia utilizada.

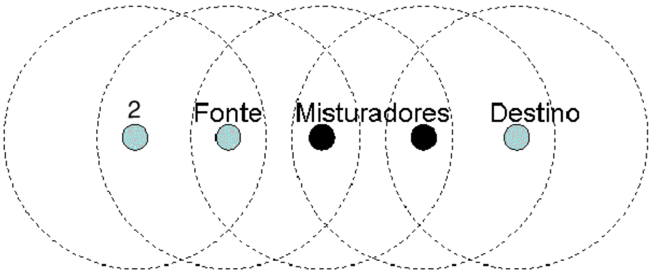


Figura 3.6: **Rede Ad Hoc simples**

Com o objetivo de facilitar a análise das propriedades e comportamentos do protocolo, foram consideradas duas situações: um ambiente sem intrusos (rede sem intrusos) e um ambiente comprometido (rede com nós maliciosos ou intrusos). Foram feitas especificações separadas para cada uma das situações, cuja diferença reside na adição de processos que representem nós maliciosos na modelagem de um ambiente comprometido.

O processo que representa a descoberta de rotas é composto por vários subprocessos que representam cada nó da rede executando o protocolo. Além desses subprocessos, existe um subprocesso chamado **Meio** que representa o canal de comunicação, por onde são enviados os pacotes. Da mesma forma, o processo transmissão de dados com sucesso e o processo transmissão de dados com falha também são compostos por subprocessos que representam cada nó da rede durante a transmissão de dados. O trecho a seguir apresenta a declaração do processo de descoberta de rotas (*Route_Discovery*).

```
process Route_Discovery[U0, U1, U2, U3, Ut] : exit :=

hide M0_in, M0_out, M1_in, M1_out, M2_in, M2_out, M3_in, M3_out, Mt_in,
     Mt_out in
(
(
Source[U0, M0_in, M0_out] (Kt, Km, Ks, R, Ss, As, At, Ab, A1)
 |||
No1_mix[U1, M1_in, M1_out] (Kt, Km, Ks, R, Sm, As, At, Ab, A1)
 |||
No2[U2, M2_in, M2_out](As, Ab)
 |||
No3_mix[U3, M3_in, M3_out] (Kt, Km, Ks, R, Sm, As, At, Ab, A1, A3)
 |||
Target[Ut, Mt_in, Mt_out](Kt, Km, Ks, R, St, As, At, Ab, A1, A3)
)

|[M0_in, M0_out, M1_in, M1_out, M2_in, M2_out, M3_in, M3_out, Mt_in,
  Mt_out]|

(
Meio[M0_in, M0_out, M1_in, M1_out, M2_in, M2_out, M3_in, M3_out, Mt_in,
     Mt_out](Kt, Km, Ks, R, As, At, Ab, A1, A3)
)
)
```

As portas de comunicação **U0, ..., Ut** ligam a camada de rede (camada responsável pelo roteamento) com a sua camada superior. As interações entre as camandas são interações externas. As interações internas são aquelas que ocorrem entre os nós durante a execução do protocolo. Essas interações não devem ser visíveis externamente e, por isso, usa-se o operador **hide** para esconder as portas nas quais elas ocorrem. O operador ||| demonstra que os processos possuem comportamentos independentes e paralelos, enquanto o operador | **[M0_in, ..., Mt_out]** | representa o sincronismo entre os processos através das portas de comunicação **M0_in, ..., Mt_out**.

O serviço proposto para este modelo tem por objetivo estabelecer rota para o destino e restabelecer a transmissão de dados, caso ocorra uma quebra num enlace, através do mecanismo de manutenção de rotas. O próximo código representa o modelo do serviço oferecido pelo protocolo.

```
Specification ANDSR_SERVICE [U0, U1, U2, U3, Ut] : noexit

library ANDSR_LIB endlib    (* declaração da biblioteca *)

behaviour

SERVICE [U0, U1, U2, U3, Ut]

where

process SERVICE [U0, U1, U2, U3, Ut] : noexit :=

(
(Discovery[U0, U1, U2, U3, Ut]
>> Maintenance [U0, U1, U2, U3, Ut]
>> SERVICE [U0, U1, U2, U3, Ut]
)

 [ ]

(Discovery[U0, U1, U2, U3, Ut]
>> Maintenance [U0, U1, U2, U3, Ut]
>> Falha [U0, U1, U2, U3, Ut]
>> SERVICE [U0, U1, U2, U3, Ut]
)

 [ ]

(Discovery[U0, U1, U2, U3, Ut]
>> Falha [U0, U1, U2, U3, Ut]
>> SERVICE [U0, U1, U2, U3, Ut]
)
)
where
```

```
process Discovery [U0, U1, U2, U3, Ut] : exit :=
U0 ! Rota_para_Destino; U0 ! RotaEncontrada; exit
endproc

process Maintenance [U0, U1, U2, U3, Ut] : exit :=
U0 ! Iniciar_Transmissao; U0 ! msgRecebida; exit
endproc

process Falha [U0, U1, U2, U3, Ut] : exit :=
 U0 ! Iniciar_Transmissao; U0 ! Link_Error; exit
endproc
endproc
endspec
```

Este código é a especificação do serviço para a topologia de rede proposta, vide Figura 3.6. Nele estão representados as ações externas, aquelas entre as camadas, por isso só aparecem as portas **U0, ..., Ut**. Além disso, algumas opções de sequência de ações ou processos foram modeladas. A primeira opção descrita é a execução do mecanismo de descoberta de rotas cujo término habilita o início de uma transmissão de dados normal. Ao final desta transmissão, o protocolo volta ao estado inicial tendo novamente as três opções para escolher. O operador $>>$ demonstra que ao final de um processo o seguinte está habilitado a ser executado e o operador **[ ]** representa a escolha entre uma sequência de ações ou processos.

3.4.2 Modelo Genérico

O modelo genérico, ao invés de modelar cada nó da rede, modela um grupo de nós que tiver o mesmo tipo de comportamento diante dos eventos. São cinco grupos modelados nesta fase: grupo de nós fonte, misturadores, intrusos, comuns e destino. Agora, não se modela um nó que represente um determinado comportamento, mas modela-se um comportamento que pode ser desempenhado por um grupo de nós.

Da mesma maneira que na fase anterior, faz-se a abstração das caixas pretas para iniciar a modelagem, vide Figura 3.7. Porém, neste caso, considera-se mais processos: Busca_Misturador, Busca_Rota, TD(transmissão de dados) e Quebra_link, representando a busca por misturadores, a descoberta de rotas, a transmissão de dados com sucesso, e a quebra de enlace durante a transmissão, respectivamente. Para cada um desses processos temos subprocessos que representam os grupos de nós existentes na rede (fonte, destino, misturador, intruso e nó comum). A idéia é usar subprocessos que representem nós de um determinado tipo, por exemplo, qualquer nó fonte na rede executa as mesmas ações, então o subprocesso Fonte representa o comportamento de qualquer nó fonte. Cada processo representa não um nó, mas todos os nós de um mesmo tipo.

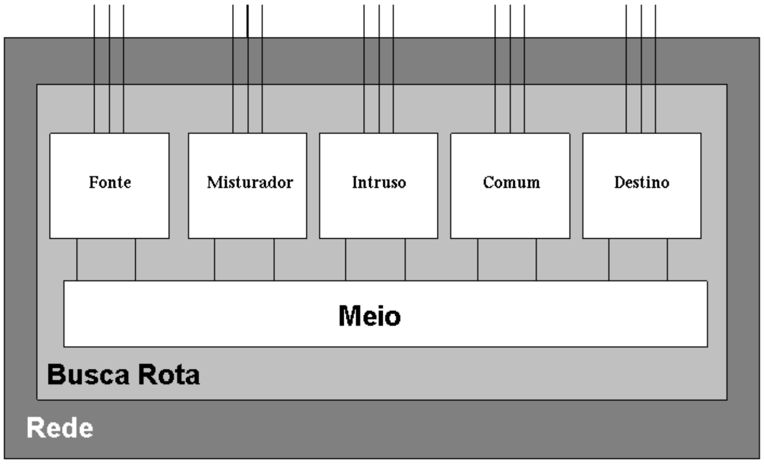


Figura 3.7: **Subprocessos e comportamentos no modelo genérico**

Esta modelagem chama o processo Sistema (modelagem dos estudos de casos) de Rede para a representação camada rede (Modelo OSI). O esboço desse modelo, em LOTOS, toma a seguinte forma:

```
Specification ANDSR ...
library ... (* declaração da biblioteca *)
behaviour
  Rede[...]
where
        process Rede ...
        .
        .
```

```
        where
            process Busca_Misturador ...
            .
            .
            where
                 process Fonte ...
                    (* descrição do comportamento
                        dos nós fontes *)
                 endproc
                 process Misturador ...
                    (* descrição do comportamento
                        dos nós misturadores *)
                 endproc
                 process ... (* o mesmo para Intruso, Comum
                                Destino e Meio *)
 .
 .
            endproc
            .  (* O mesmo p/ os outros processos,
                  Busca_Rota, ... *)
            .
        endproc
endproc
endspec
```

Usando essa metodologia elimina-se um dos maiores problemas na verificação de protocolos para redes móveis ad hoc, as mudanças constantes e aleatórias na topologia da rede. Pois é bastante difícil modelar todas as topologias possíveis de uma rede ad hoc.

Exemplo - Modelo Genérico

Para exemplificar o uso dessa modelagem toma-se o modelo final do protocolo ANDSR. O trecho a seguir mostra a declaração de subprocessos do processo Busca_Rota.

```
.
.
process Busca_Rota_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
        Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out, Mi_in,
     Mi_out, Mt_in, Mt_out in

(
Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks, R, As,
                                   At, Ab, Ai, Ac, Am)
|||
Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km, Ks, R, As,
                                       At, Ab, Ai, Ac, Am)
|||
Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km, Ks, R, As,
                                 At, Ab, Ai, Ac, Am)
|||
Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km, Ks, R, As,
                                   At, Ab, Ai, Ac, Am)
|||
Target[Ut, Pt, Qt, Mt_in, Mt_out](Kt, Km, Ks, R, As,
                                  At, Ab, Ai, Ac, Am)
)

|[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out, Mi_in,
Mi_out, Mt_in, Mt_out]|

(
Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out, Mi_in,
     Mi_out, Mt_in, Mt_out](Kt, Km, Ks, R, As, At, Ab,
                            Ai, Ac, Am)
)
where
.
.
.
```

Como mencionado anteriormente, os processos Source, Misturador, Comum, Intruso e Meio representam um grupo de nós. Diferente da modelagem anterior, os processos possuem mais portas de comunicação, compare a Figura 3.5 com a Figura 3.7. Para este exemplo, verifica-se que foram acrescentadas as portas **Px**, **Qx**, onde x representa um dos grupos de nós, podendo ser do tipo s (fonte), m (misturadores), c (comum), i (intruso) ou t (destino). A porta **Px** representa interações com a tabela de roteamento e a porta **Qx** representa interações com a memória. A criação destas portas facilita a modelagem, pois elimina a obrigação de modelar a tabela de roteamento e a memória no interior do nó. Com estas portas externas abstrai-se de sua modelagem. As variáveis entre () são usadas pelos nós e definidas na biblioteca de tipos e funções desenvolvida, também, pelo projetista.

O serviço proposto para este modelo, da mesma forma que na fase inicial, tem por objetivo estabelecer rota para o destino e restabelecer a comunicação, caso ocorra uma quebra num enlace, através do mecanismo de manutenção de rotas. O código completo do protocolo é apresentado no apêndice B, o código do serviço está no apêndice C e o da biblioteca no apêndice A. Observando esses códigos, verifica-se que o serviço é definido por várias opções de comportamentos e o operador >> apresentado habilita o próximo processo ao término com sucesso do primeiro. Além disso, aparecem processos diferentes com nomes iniciados como Busca_misturador, isso se deve aos casos particulares como por exemplo: existência de apenas um misturador entre fonte e destino, a presença de dois ou mais misturadores entre fonte e destino, a ausência de misturadores no alcance do nó fonte, etc.

3.5 Comentários

Este capítulo descreveu o desenvolvimento do projeto do protocolo ANDSR (*Anonymous Dynamic Source Routing*), apresentando a importância do uso de técnicas de descrição formal no desenvolvimento de sistemas, a linguagem formal adotada, as ferramentas usadas e a metodologia empregada no desenvolvimento do modelo do protocolo. O processo de modelagem é descrito através de casos de exemplo para uma melhor compreensão. O capítulo posterior apresenta o processo de verificação e seus resultados nos diversos estudos de casos realizados e no modelo genérico final.

Capítulo 4

Verificação e Resultados

NESTE capítulo são apresentados os resultados no processo de verificação da especificação em LOTOS do protocolo ANDSR (*Anonymous Dynamic Source Routing*) e sua validação. Os experimentos realizados foram conduzidos de forma incremental, pois cada estudo de caso agrega funcionalidades em relação aos anteriores. Essas funcionalidades podem ser inserções de nós, pacotes, funções, mudanças nas funções existentes, etc. A partir desses experimentos foi possível especificar o protocolo de maneira abrangente, o modelo genérico. A seção 4.1 descreve o processo de verificação, a seção 4.2 apresenta resultados dos estudos de casos da fase inicial e os resultados do modelo genérico e, por fim, a seção 4.3 faz os comentários finais.

4.1 Processo de Verificação

O processo de verificação consiste em checar se a especificação satisfaz as características e propriedades definidas no início do projeto. A semântica operacional das especificações deve ser definida em termos de sistemas de transições. As técnicas de verificação podem ser usadas no contexto de geração de testes através da ferramenta CADP [36].

Após a compilação da especificação em LOTOS usando as ferramentas CAESAR.ADT e CAESAR, obteve-se um grafo chamado LTS (*Labelled Transition System*). Esse grafo LTS contém os estados e transições rotuladas como ações entre os estados, ou seja, contém exatamente as possíveis sequências de execução do protocolo. Com o auxílio das ferramentas BCG (integrante do pacote CADP) deve-se fazer a minimização do grafo LTS. Essa minimização é usada para reduzir o grafo LTS retirando, então, as redundâncias e permitindo uma análise mais simples. Um método simples para testar o grafo LTS é o *transition tour* [36], onde as transições são testadas através de um *tour* por todas as transições da especificação sem checar o estado alvo. Esse teste é realmente usado na indústria de protocolos e pode ser feito manualmente ou com o auxílio de um simulador. O teste manual se torna inviável se a especificação possuir um sistema LTS com grande número de transições.

No pacote de ferrramentas CADP pode-se usar dois simuladores: o XSimulator e o Open/Caesar Simulator. Com eles pode-se acompanhar passo a passo a sequência de execução do protocolo. Esse procedimento conferirá se a sequência de execução gerada é a que se esperava. E a partir dessa conferência pode-se fazer as alterações necessárias e inserção de outros ou mais eventos.

A ferrramenta ALDEBARAN é o último estágio do processo de verificação. Ela é responsável pela comparação entre dois grafos LTS. A verificação requer a equivalência observacional [33] dos grafos LTS, isto é, a equivalência entre o grafo LTS do modelo do protocolo e o do modelo do serviço oferecido pelo protocolo à camada superior. A equivalência observacional é usada para verificar se o protocolo especificado representa os serviços discriminados inicialmente como requisitos de projeto. Caso exista essa equivalência entre protocolo e serviço a ferramenta ALDEBARAN retornará como resposta a palavra TRUE.

4.2 Desenvolvimento

A evolução da especificação do protocolo se deu através de sucessivos estudos de casos que tinham por objetivos modelar o protocolo em determinados cenários físicos e de comportamento. Essa experiência culminou num modelo mais abrangente do protocolo ANDSR, um modelo genérico, que cabe em qualquer topologia com qualquer quantidade de nós participantes, pois se baseia puramente no comportamento dos nós. A seguir é apresentado o processo de verificação realizado em cada etapa do desenvolvimento do protocolo.

4.2.1 Estudos de Casos Iniciais

No primeiro estudo de caso a rede ad hoc só possui dois nós onde só foi modelada uma troca de mensagem bastante simples. Essa modelagem é muito importante para o aprendizado da ferramenta e da linguagem, porém os resultados da verificação e validação não são relevantes ao projeto.

Um segundo estudo de caso apresentava apenas três nós, um nó fonte, um nó misturador e um nó destino, onde o misturador se encontrava entre a fonte e o destino. Da mesma forma, a importância da modelagem recai no aprendizado, mas o modelo já começa a ter características do protocolo atual. O modelo foi crescendo a medida que eram acrescentados nós e novas funções.

Um outro estudo de caso do protocolo [34] leva em consideração somente a descoberta de rota (*route discovery*), numa rede sem presença de intrusos. Nesse modelo foram definidos cinco processos, que representam os cinco nós interligados por um meio de comunicação simples para troca de dados. A topologia dessa rede está na Figura 4.1

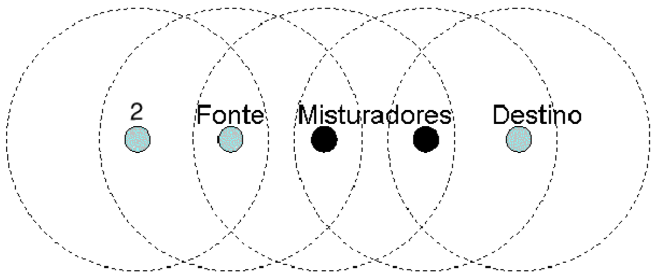


Figura 4.1: **Topologia inicial do protocolo ANDSR**

A Tabela 4.1 apresenta o número de estados e transições desse modelo, antes e depois da minimização de redundâncias.

MODELO	ESTADOS	TRANSIÇÕES	MINIMIZADO
Route Discovery	44	44	não
Route Discovery	43	43	sim
Serviço	2	2	não
Serviço	2	2	sim

Tabela 4.1: **Geração dos LTS - Labelled Transition System**

O grafo LTS da Figura 4.2 mostra a sequência de execução de eventos no protocolo. *U0 ! ROTA_PARA_DESTINO* e *U0 ! ROTAENCONTRADA* são as únicas interações do protocolo com sua camada superior ou camada usuário de seus serviços. Por isso, usa-se a letra *U* (de usuário) para essa representação. Essas interações mostram que o serviço foi prestado à camada superior, ou seja, foi encontrada rota para o destino após um pedido de rota feito por um nó. As demais transições deste grafo são apresentadas com a letra *i* demonstrando que são iterações internas do protocolo. Elas são as trocas de mensagens entre os nós através do meio de comunicação ou as ações internas dos nós, como consulta às tabelas, construção de pacotes e armazenamento de informações. A Figura 4.3 mostra o grafo LTS do modelo do serviço proposto. Este grafo apresenta somente as interações externas, ou seja, as interações com a camada superior. Os círculos representam os estados, as setas representam as transições e os rótulos representam as ações executadas. Para visualizar e verificar todas as transições e estados usou-se o Open/Caesar Simulator. Nesse simulador foi feito o *transition tour* e desta forma foi verificada a sequência de execução do protocolo. Verifica-se a existência de algum bloqueio (*deadlock*), algum estado ou transição não alcançado e se o protocolo é reinicializável.

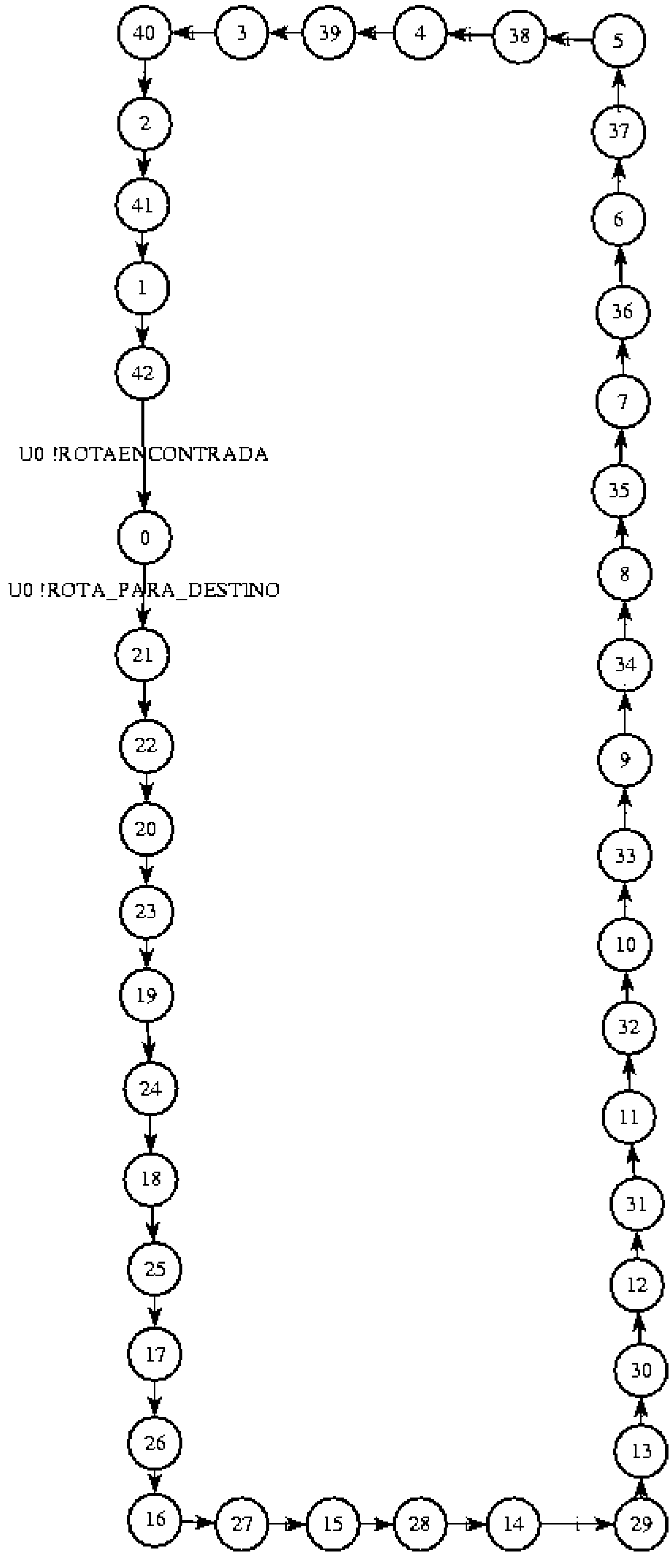


Figura 4.2: **Grafo LTS minimizado do protocolo**

Este estudo de caso do protocolo não possui nenhum *deadlock* como mostra a Figura 4.4. Além disso, a equivalência observacional entre protocolo e serviço obteve resultado

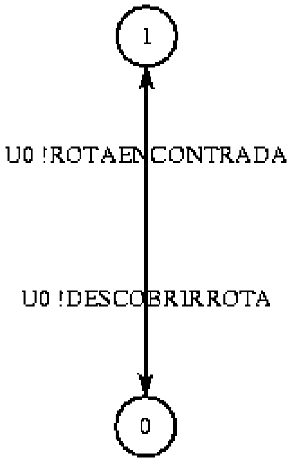


Figura 4.3: **Grafo LTS do serviço**

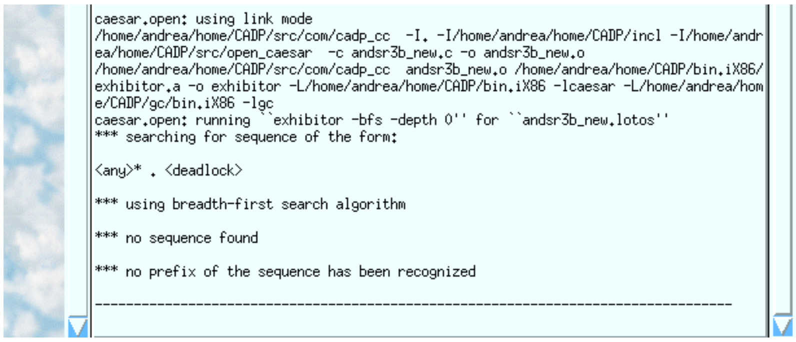


Figura 4.4: **Deadlocks**

TRUE, mostrando que o protocolo atende aos requisitos do serviço imaginado.

4.2.2 Estudo de Caso - Rede com cinco nós

Esse estudo de caso continua usando uma rede ad hoc com cinco nós, porém foram adicionadas mais funcionalidades ao protocolo [35]. Considerando que não existem intrusos na rede, a topologia é a mesma da Figura 4.1, vista na seção anterior. A Tabela 4.2 apresenta o número de estados e transições alcançados por este estudo de caso.

Observando as duas tabelas (Tabela 4.1 e Tabela 4.2) constata-se que apesar dos modelos serem baseados na mesma topologia a inserção de mais funções aumenta consideravelmente o número de estados e transições. O grafo LTS (*Labelled Transition System*) do modelo do serviço deste estudo de caso é apresentado na Figura 4.5.

MODELO	ESTADOS	TRANSIÇÕES	MINIMIZADO
ANDSR	243	236	não
ANDSR	211	213	sim
Serviço	19	21	não
Serviço	14	16	sim

Tabela 4.2: **Geração dos LTS - rede c/ 5 nós e s/ intrusos**

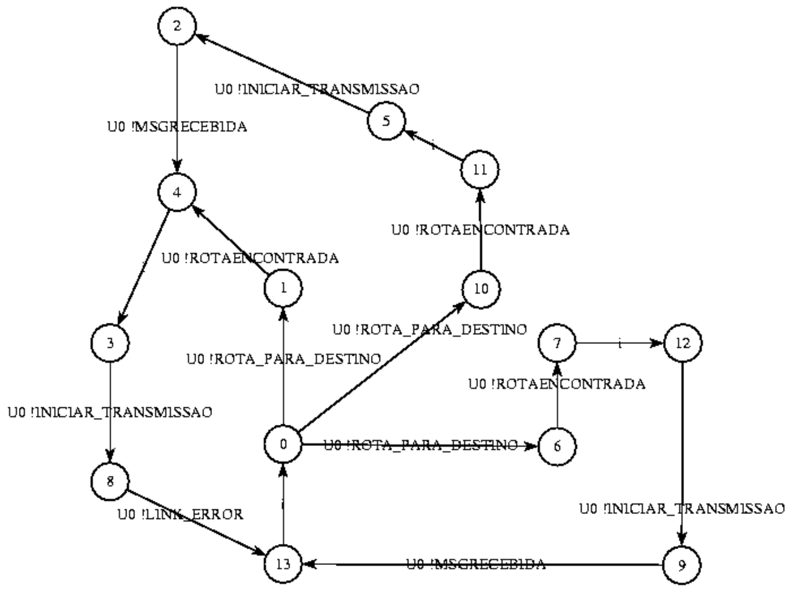


Figura 4.5: **LTS minimizado do serviço rede c/ 5 nós e s/ intrusos**

O grafo LTS do protocolo possui 211 estados (após a minimização), esse número torna difícil a visualização adequada do mesmo. Então, a melhor maneira de conferir os eventos e sua sequência de execução é usar o simulador XSimulator ou o OPEN/CAESAR Simulator. A Figura 4.6 mostra a sequência de execução de eventos do protocolo usando o OPEN/CAESAR Simulator.

Dados os mesmos cinco nós, considerou-se também o caso em que um dos nós fosse malicioso e passasse a tentar ouvir o tráfego de roteamento, vide Figura 4.7. Para esta situação analisou-se as seguintes topologias: nó malicioso entre o nó fonte e o primeiro

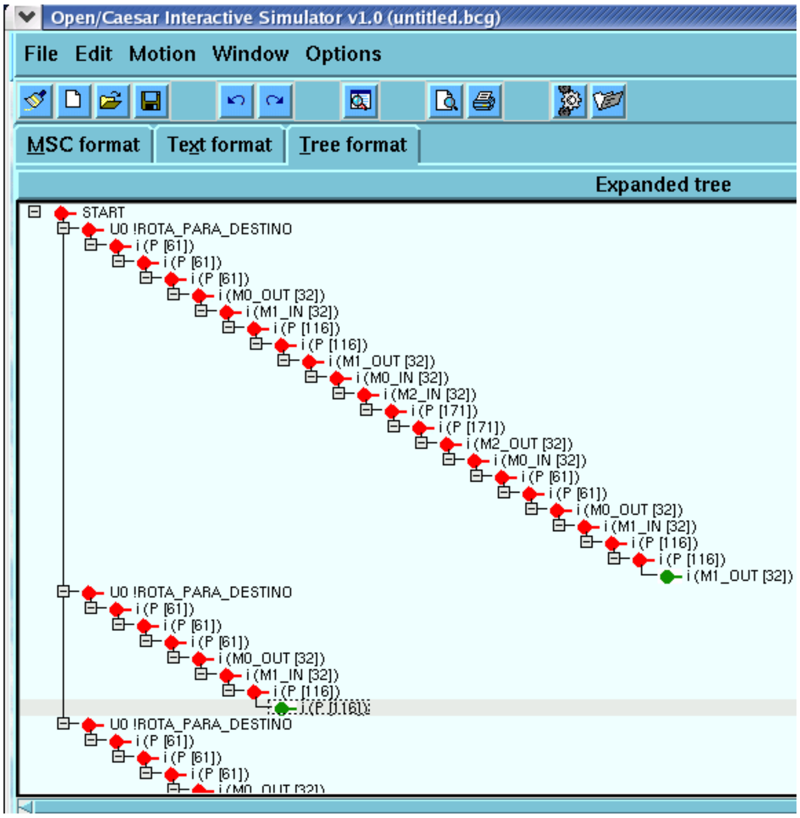


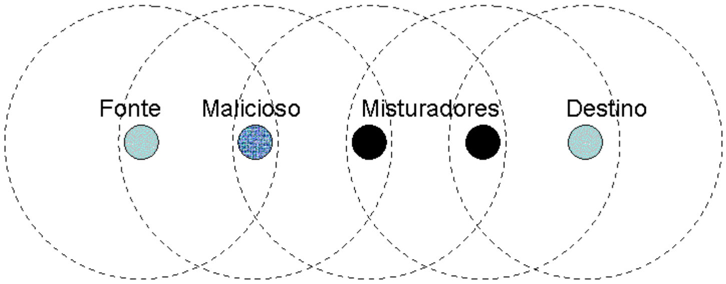
Figura 4.6: **Seqüência de execução - rede c/ 5 nós e s/ intrusos**

misturador, nó malicioso entre os dois misturadores e nó malicioso entre o segundo misturador e o nó destino. O número de estados e transições para este modelo é apresentado na tabela 4.3, porém só foi modelado o mecanismo de descoberta de rotas (*route discovery*).

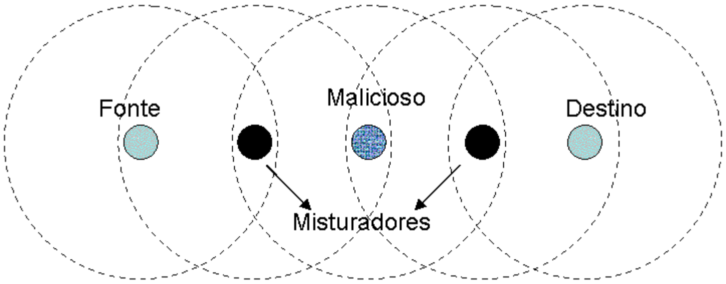
MODELO	INTRUSO	ESTADOS	TRANSIÇÕES	MINIMIZADO
Route Discovery	Entre a fonte e o primeiro misturador	55	55	não
Route Discovery	Entre a fonte e o primeiro misturador	54	54	sim
Serviço	Entre a fonte e o primeiro misturador	5	5	não
Serviço	Entre a fonte e o primeiro misturador	5	5	sim
Route Discovery	Entre o primeiro misturador e o segundo	46	46	não
Route Discovery	Entre o primeiro misturador e o segundo	45	45	sim
Serviço	Entre o primeiro misturador e o segundo	3	3	não
Serviço	Entre o primeiro misturador e o segundo	3	3	sim
Route Discovery	Entre o segundo misturador e o destino	46	46	não
Route Discovery	Entre o segundo misturador e o destino	45	45	sim
Serviço	Entre o segundo misturador e o destino	3	3	não
Serviço	Entre o segundo misturador e o destino	3	3	sim

Tabela 4.3: **Geração dos LTS - rede c/ 5 nós e c/ intruso**

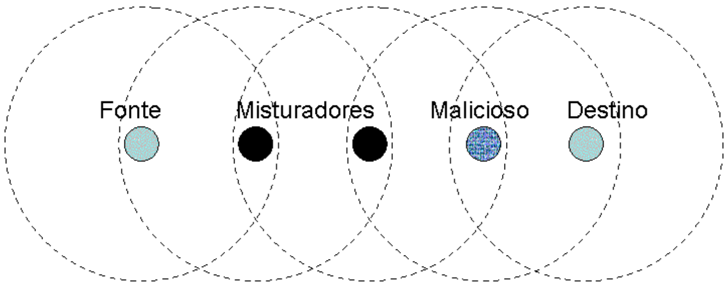
O grafo LTS do serviço deste modelo é visto na Figura 4.8. Este grafo se refere à situação onde existe um nó com comportamento malicioso entre o nó fonte e o primeiro misturador, como pode ser visto na tabela 4.3 através do número de estados e transições.



(a) **Nó malicioso entre fonte e misturador**



(b) **Nó malicioso entre os misturadores**



(c) **Nó malicioso entre misturador e destino**

Figura 4.7: **Rede com nó malicioso**

O presente estudo de caso do protocolo também possui modelo do protocolo e serviços equivalentes observacionalmente e não possui *deadlocks* e *livelocks*, vide Figura 4.9.

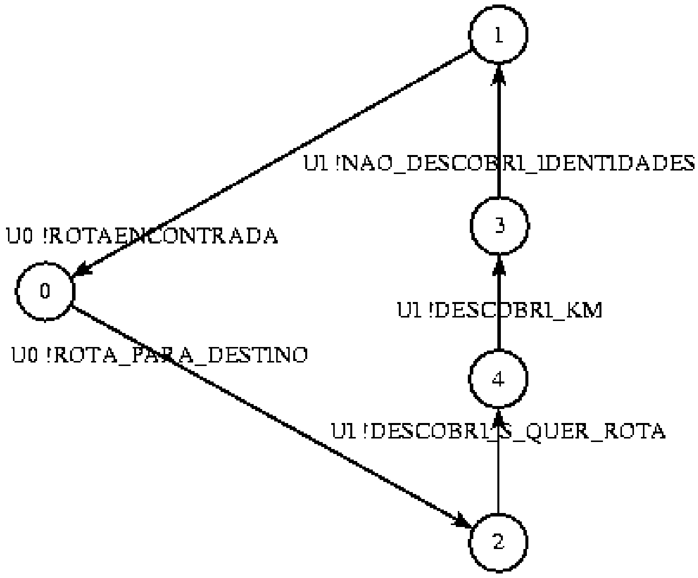


Figura 4.8: **LTS minimizado do Serviço - rede c/ 5 nós e c/ intruso**

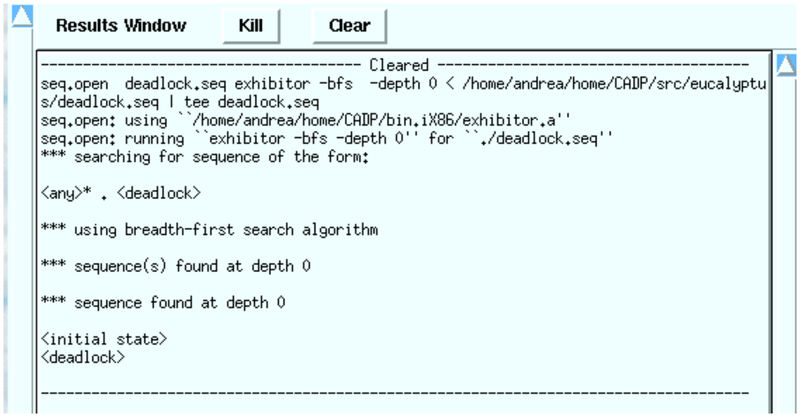


Figura 4.9: **Propriedades - rede c/ 5 nós**

4.2.3 Estudo de Caso - Rede com seis nós

O estudo de caso com 6 nós do ANDSR se baseia na topologia da Figura 4.10. Foi acrescentado um nó no alcance de ambos os misturadores da rede com objetivo de representar o comportamento de descarte de pacotes vindo desses misturadores. Considerando o caso em que a rede não possua intrusos, o número de estados e transições é apresentado na Tabela 4.4.

A inserção de um único nó aumenta bastante o número de estados e transições alcançados pelo protocolo. Analisando a Tabela 4.4 e a Tabela 4.2 verifica-se esse fato.

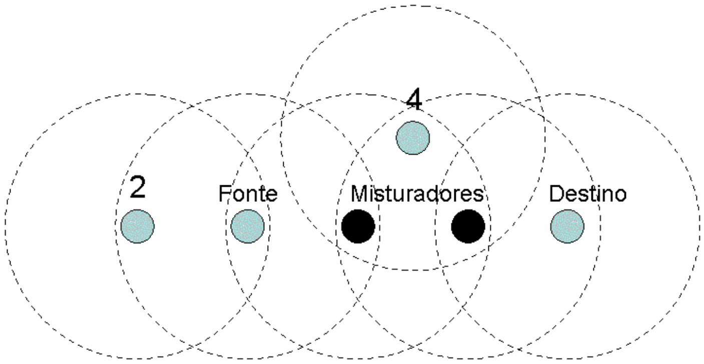


Figura 4.10: **Topologia - rede c/ 6 nós**

MODELO	ESTADOS	TRANSIÇÕES	MINIMIZADO
ANDSR	372	374	não
ANDSR	326	328	sim
Serviço	37	39	não
Serviço	31	33	sim

Tabela 4.4: **Geração dos LTS - rede c/ 6 nós**

O grafo LTS minimizado do serviço proposto é apresentado na Figura 4.11. As ações externas executadas pelo nó 4 são aquelas indicadas por *U4*. Observando o grafo vê-se o descarte de pacotes executado pelo nó 4 ao receber um pacote duplicado ou não endereçado a ele. Além disso, verifica-se a existência de três opções de início do algoritmo: descoberta de rota seguida por uma transmissão de dados normal, descoberta de rota seguida por quebra de enlace no caminho de ida das mensagens e descoberta de rota seguida de transmissão de dados com quebra de enlace no caminho de volta das mensagens.

Os modelos deste estudo de caso não possuem nenhum *deadlock* ou *livelock*, como mostra a Figura 4.12, e são equivalentes observacionalmente, isto é, o modelo de protocolo fornece à camada superior o serviço proposto no projeto.

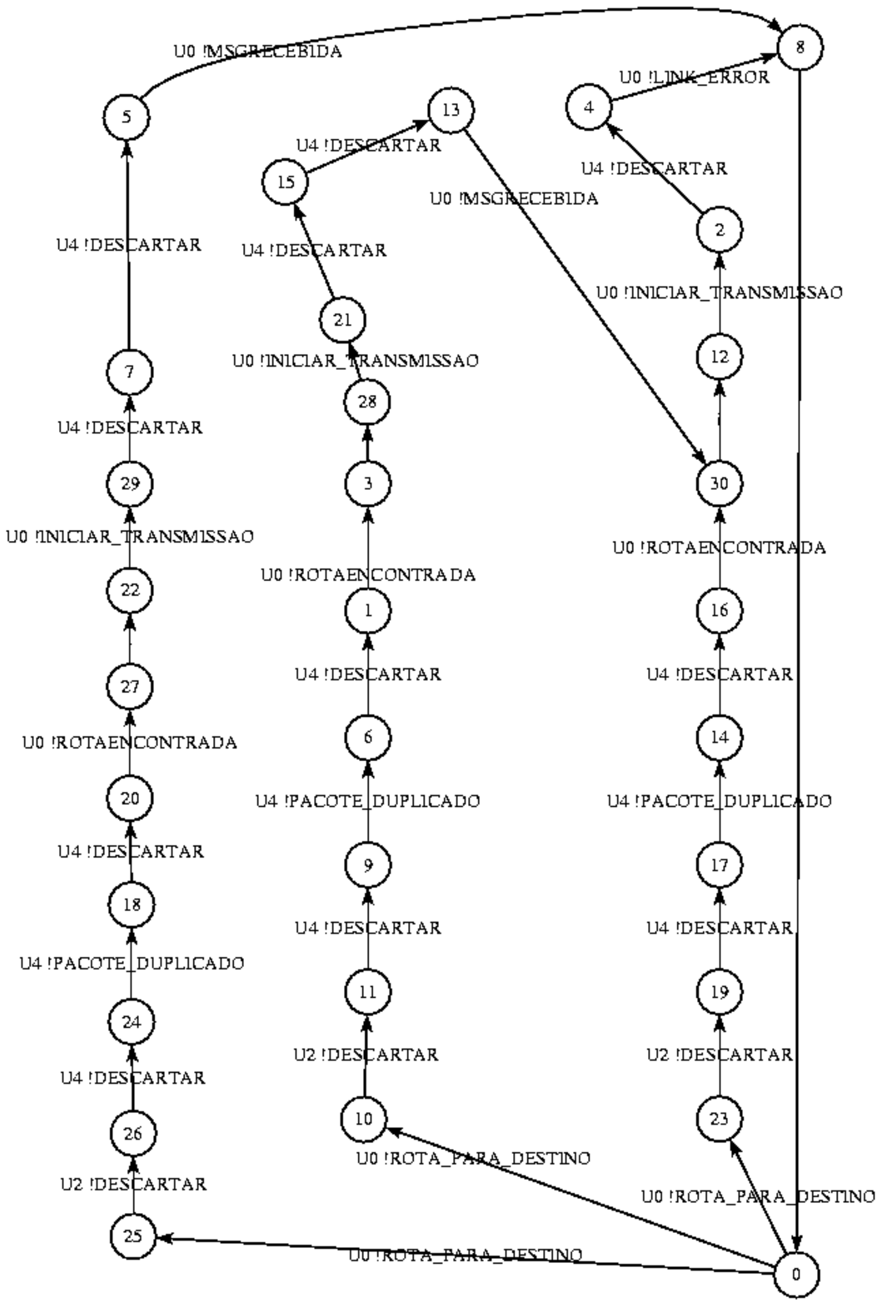


Figura 4.11: **Grafo LTS minimizado do serviço - rede c/ 6 nós**

```
Results Window   Kill   Clear
------------------------------------ Cleared ------------------------------------
bcg_open andsr7b.bcg exhibitor -bfs  -depth 0 < /home/andrea/home/CADP/src/eucalyptu
s/deadlock.seq | tee deadlock.seq
bcg_open: using ``/home/andrea/home/CADP/bin.iX86/exhibitor.a''
bcg_open: running ``exhibitor -bfs -depth 0'' for ``./andsr7b.bcg''
*** searching for sequence of the form:

<any>* . <deadlock>

*** using breadth-first search algorithm

*** no sequence found

*** no prefix of the sequence has been recognized

--------------------------------------------------------------------------------
bcg_open andsr7b.bcg evaluator   -verbose -diag livelock.bcg "/home/andrea/home/CADP
/src/xtl/livelock.mcl"
bcg_open: using ``/home/andrea/home/CADP/bin.iX86/evaluator.a''
bcg_open: running ``evaluator -verbose -diag livelock.bcg /home/andrea/home/CADP/src
/xtl/livelock.mcl'' for ``./andsr7b.bcg''
-- evaluator 3.0 -- R. Mateescu and M. Sighireanu (INRIA Rhone-Alpes) --

evaluator: preprocessing of ``livelock''
evaluator: syntax analysis of ``livelock''
evaluator: semantic analysis of ``livelock''
evaluator:    - variable binding
evaluator:    - translation in positive normal form
evaluator:    - test of alternation 1
evaluator:    - translation into regular modal equation systems
evaluator:    - regular expression elimination
evaluator:    - translation into modal equation systems
evaluator: resolution of ``livelock''

FALSE
(consult diagnostic in file ``livelock.bcg'')
--------------------------------------------------------------------------------
```

Figura 4.12: **Propriedades - rede c/ 6 nós**

4.2.4 Estudo de Caso - Rede com sete nós

Neste estudo de caso, a topologia especificada foi a da Figura 4.13 e foram avaliadas três situações: rede sem intrusos, rede com um dos nós com comportamento malicioso e um conluio de misturadores [37]. O número de estados e transições para a primeira situação é apresentado na Tabela 4.5.

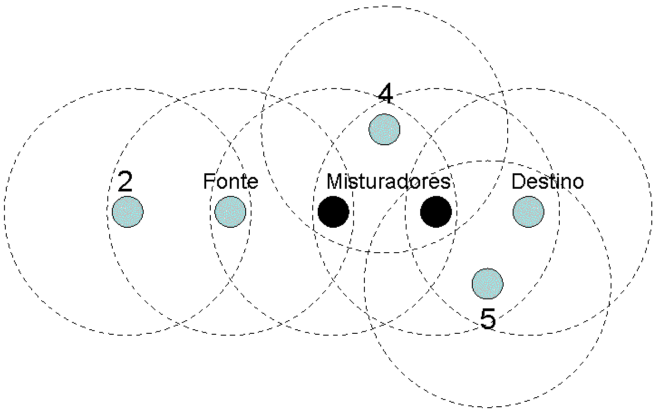


Figura 4.13: **Topologia - rede c/ 7 nós**

MODELO	ESTADOS	TRANSIÇÕES	MINIMIZADO
ANDSR	503	502	não
ANDSR	457	459	sim
Serviço	51	53	não
Serviço	45	47	sim

Tabela 4.5: **Geração dos LTS - rede c/ 7 nós e s/ intrusos**

A Figura 4.14 apresenta o grafo LTS minimizado do serviço oferecido pelo protocolo à camada superior. Esse grafo representa o serviço numa rede sem intrusos.

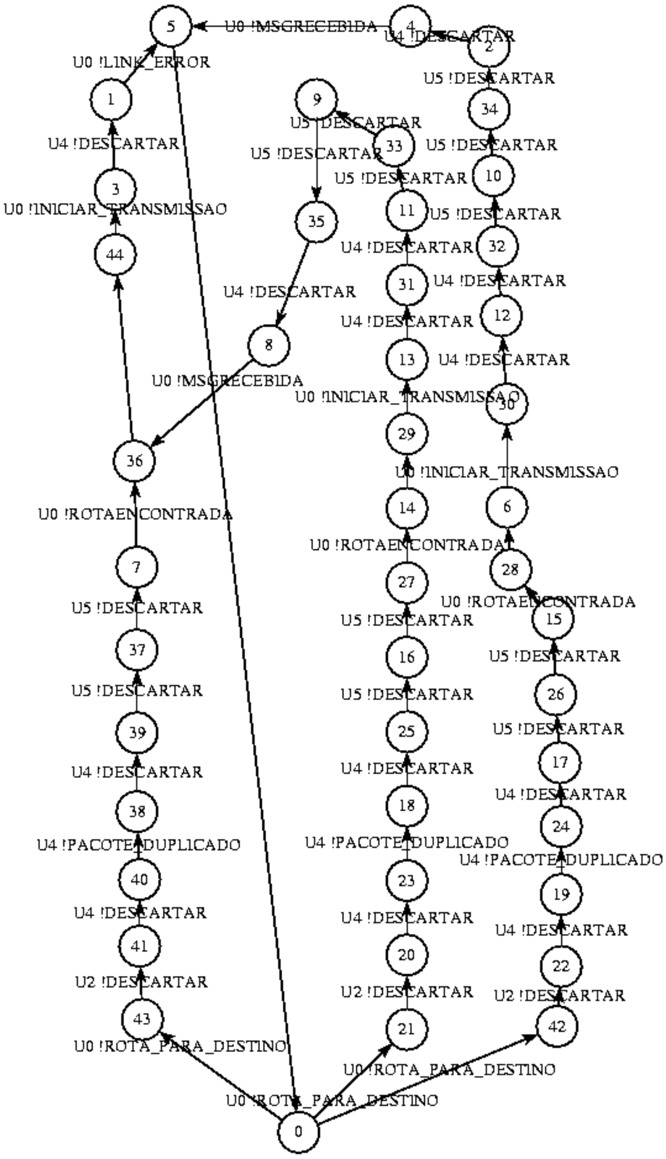


Figura 4.14: **Grafo LTS minimizado do serviço - rede c/ 7 nós e s/ intrusos**

A Tabela 4.6 apresenta os resultados obtidos para a rede com o nó malicioso (nó 4) no alcance dos misturadores.

MODELO	ESTADOS	TRANSIÇÕES	MINIMIZADO
ANDSR	538	540	não
ANDSR	485	487	sim
Serviço	56	58	não
Serviço	49	51	sim

Tabela 4.6: **Geração dos LTS - rede c/ 7 nós e c/ intruso**

A Figura 4.15 mostra parte do grafo LTS minimizado do serviço fornecido pelo protocolo especificado à camada superior. Este grafo LTS mostra a sequência de execução do mecanismo de descoberta de rotas. Para visualizar melhor o comportamento do protocolo e pontos importantes da especificação foram inseridos nos códigos das especificações eventos especiais. Esses eventos tem como objetivo mostrar o comportamento de ataque dos nós maliciosos. Dado o grafo da Figura 4.15 vê-se que o evento representado no rótulo *U4 ! NAO_DESCOBRI_IDENTIDADES* é um desses eventos especiais. *U4 ! NAO_DESCOBRI_IDENTIDADES* informa que o nó 4 não foi capaz de descobrir dados sobre a identidade dos nós através dos pacotes recebidos até aquele momento. Naturalmente, essa afirmação se refere ao modelo de intruso desenvolvido neste projeto.

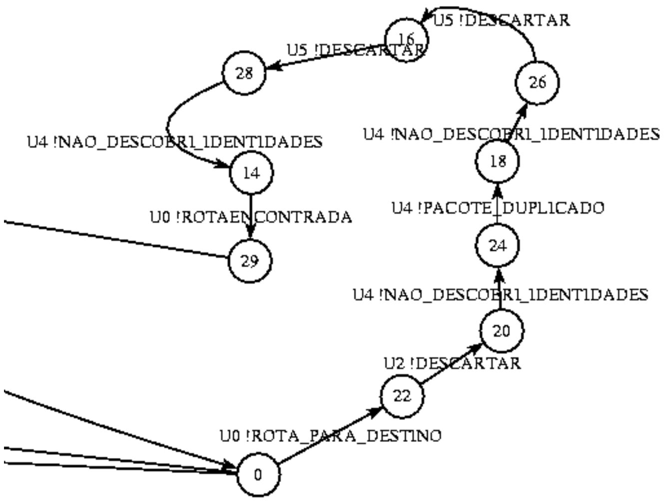


Figura 4.15: **Grafo LTS minimizado do serviço - nó 4 intruso**

MODELO	ESTADOS	TRANSIÇÕES	MINIMIZADO
ANDSR	530	532	não
ANDSR	483	485	sim
Serviço	69	71	não
Serviço	62	64	sim

Tabela 4.7: **Geração dos LTS - conluio de misturadores**

Considera-se, agora, que todos os nós misturadores formam um conluio. Eles tentam descobrir alguma informação consistente para repassar aos demais misturadores até conseguirem identificar dados relevantes ao seu interesse. A Tabela 4.7 apresenta os resultados para este caso.

A Figura 4.16 mostra parte do grafo LTS minimizado do serviço fornecido à camada superior no caso de conluio de misturadores. Observando esse grafo constata-se a possível identificação do endereço do nó destino da comunicação durante o processo de descoberta de rotas. A sequência de execução mostra esse fato.

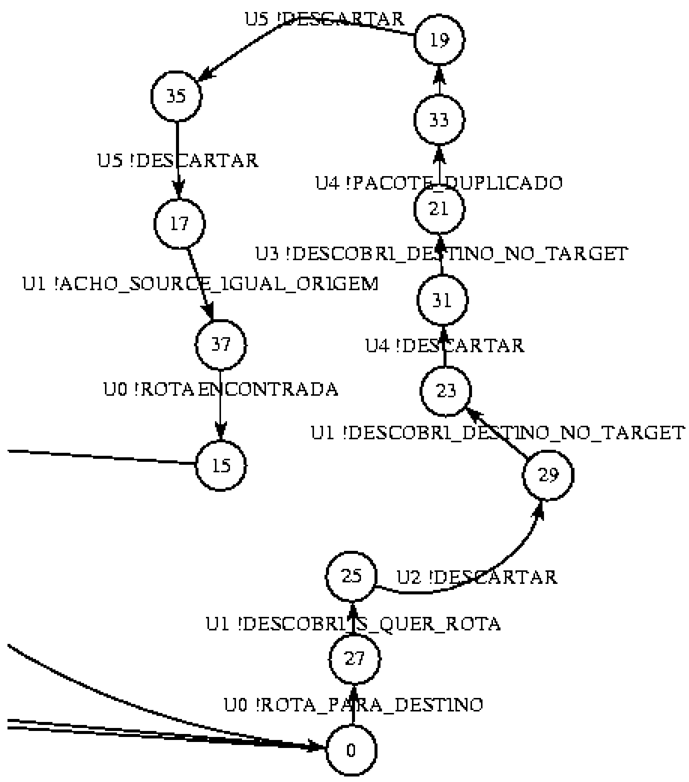


Figura 4.16: **Grafo LTS minimizado do serviço - conluio de misturadores**

O rótulo *U1 ! DESCOBRI_DESTINO_NO_TARGET* indica que o primeiro misturador descobriu a identidade do nó destino. Da mesma forma, o rótulo *U3 ! DESCOBRI_DESTINO_NO_TARGET* demonstra que o outro misturador da rede descobriu a identidade do destino. Além disso, os misturadores mantém uma desconfiança sobre a identidade do nó fonte, mas não podem comprovar o fato, pois não possuem o endereço do nó fonte. O rótulo *U1 ! ACHO_SOURCE_IGUAL_ORIGEM* demonstra essa situação.

A partir desse estudo, pensou-se em usar pares de chaves assimétricas distintas para os misturadores com o objetivo de tornar o protocolo mais resistente a ataques. Aparentemente essa idéia tornaria o algoritmo mais eficaz, porém o modo como o intruso foi modelado neste projeto o faz capaz de somente descobrir o endereço do destino se o intruso estiver recebendo os pacotes dos misturadores. O fato de possuir ou não chaves distintas não faria muita diferença sob este ponto de vista. A única informação a respeito da rota que os misturadores sabem é o endereço de destino. Além disso, o uso de chaves distintas não permitiria que as mensagens fossem repassadas em difusão, fazendo com que fossem construídas mensagens individuais para cada misturador dentro do raio de alcance.

Figura 4.17: **Deadlocks e Livelocks - rede c/ 7 nós e s/ intrusos**

A etapa final desse estudo de caso termina com a equivalência observacional entre os grafos LTS do protocolo e serviço. Através do resultado TRUE, conclui-se que o modelo do protocolo desse estudo de caso atende aos requisitos do serviço proposto. O protocolo descrito também não possui *deadlocks* e *livelocks* como é verificado na Figura 4.17.

4.2.5 Estudo de Caso - Rede com oitos nós

Este estudo de caso possui oito nós na rede, onde três são misturadores de acordo com a Figura 4.18. Nessa versão, o modelo do protocolo foi desenvolvido para estabelecer duas rotas, escolher a menor delas para uso e armazenar a outra. Além disso, foi considerada uma rede sem intrusos. A Tabela 4.8 mostra o número de estados e transições alcançados por este modelo.

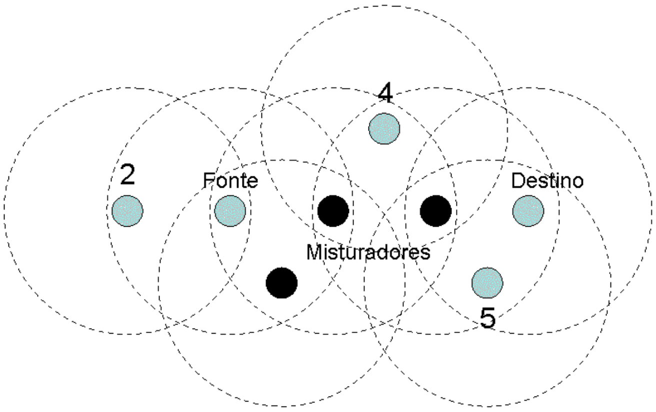


Figura 4.18: **Topologia - rede c/ 8 nós**

MODELO	ESTADOS	TRANSIÇÕES	MINIMIZADO
ANDSR	1225	1228	não
ANDSR	1076	1079	sim
Serviço	107	110	não
Serviço	92	95	sim

Tabela 4.8: **Geração dos LTS - rede c/ 8 nós**

O grafo LTS minimizado do serviço oferecido pelo protocolo à camada superior é apresentado na Figura 4.19.

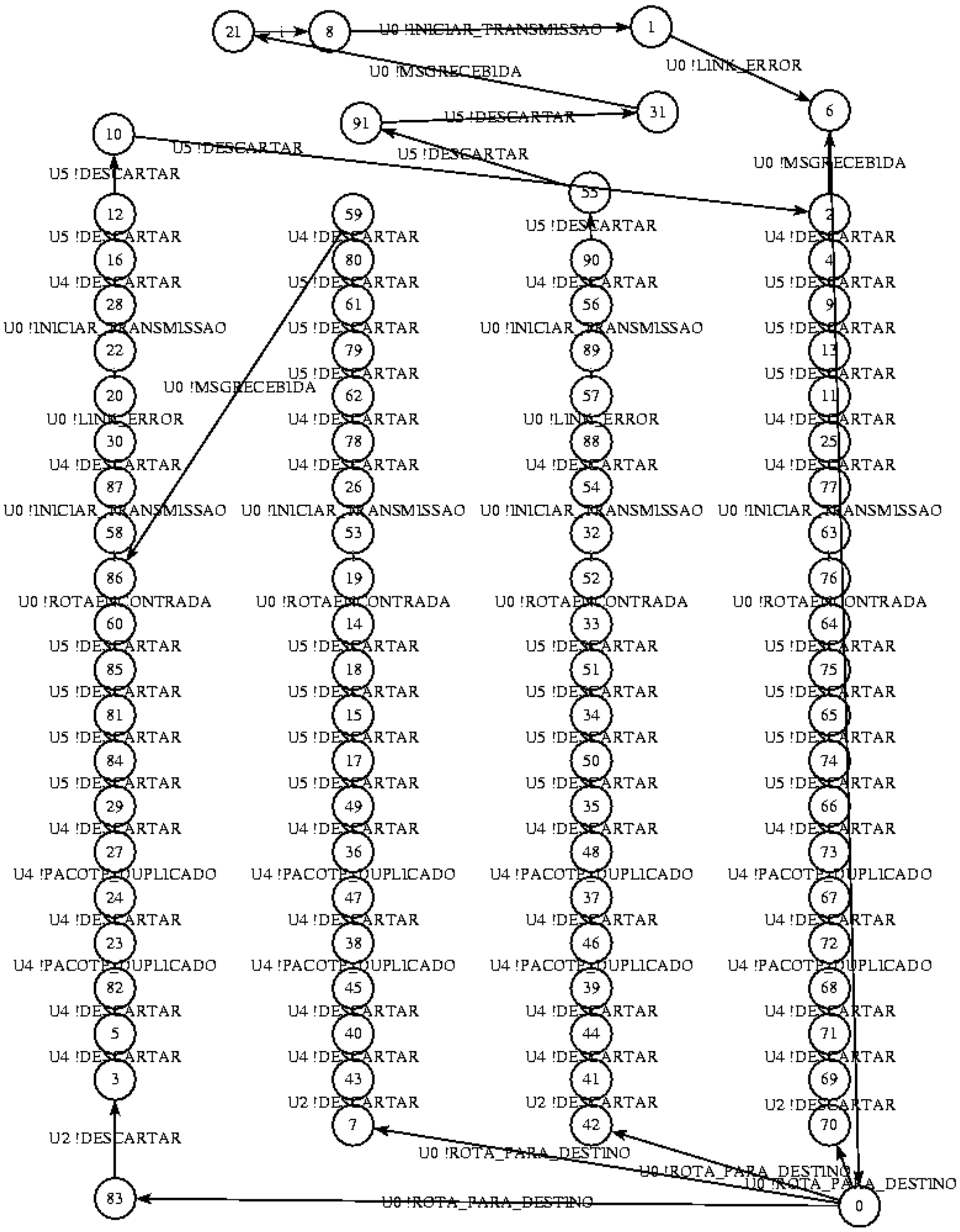


Figura 4.19: **Grafo LTS minimizado do serviço - rede c/ 8 nós**

Esta versão do ANDSR também não apresenta *deadlocks* e *livelocks*. O modelo do protocolo especificado é equivalente observacionalmente ao modelo do serviço, provando que o protocolo oferece o serviço à camada superior da maneira como foi idealizada.

Percebe-se a cada inserção de nó que o número de estados aumenta consideravelmente. Esta ainda é uma rede bastante pequena, considere o número de estados alcançados por uma topologia com 20 ou 30 nós. Isso acabaria causando uma explosão de estados, sem mencionar que talvez a maioria dos comportamentos não estivessem especificados. O objetivo da modelagem formal de um protocolo, de maneira geral, é analisar seu comportamento caracterizando dados qualitativos e não quantitativos. A solução encontrada para essas dificuldades foi criar um modelo genérico capaz de modelar o protocolo independentemente da topologia e número de nós na rede. Na verdade, o que será modelado será uma entidade que traduz o comportamento de um grupo de nós de um mesmo tipo. Nesse sentido, o ANDSR possui cinco tipos de nós: fontes, misturadores, destinos, intrusos e nós comuns. Todo comportamento referente a um nó do tipo misturador foi modelado por uma entidade chamada misturadores e assim foi feito para os demais grupos. Usando essa metodologia, o modelo do protocolo se torna mais enxuto. A próxima seção apresenta a versão final já usando essa metodologia.

4.2.6 Modelo Genérico

O modelo genérico representa o funcionamento e o comportamento do protocolo em qualquer rede ad hoc, independente de topologia e quantidade de nós. Esse modelo cobre várias possibilidades de situações e comportamentos suficientes para a validação do ANDSR. O modelo inclui situações de intrusão através de eventos especiais modelados através de interações externas (interações com a camada superior). Este modelo não perde nenhum tipo de informação em relação aos estudos de casos apresentados. Na verdade, possui mais comportamentos e menos estados e transições que o estudo de caso com oito nós.

A Tabela 4.9 apresenta o numero de estados e transições alcançados pelo protocolo.

MODELO	ESTADOS	TRANSIÇÕES	MINIMIZADO
ANDSR	558	570	não
ANDSR	370	392	sim
Serviço	150	162	não
Serviço	79	88	sim

Tabela 4.9: **Geração dos LTS - modelo genérico**

Ao se observar os valores da Tabela 4.9 verifica-se que eles são bem menores, se comparados a alguns apresentados nos estudos de caso anteriores. Como este modelo não está vinculado a uma topologia específica ou a qualquer quantidade de nós, o aumento do número de estados/transições se deve somente à inserção de funcionalidades e comportamentos. A Figura 4.20 apresenta o grafo LTS do protocolo. Como pode ser visto a quantidade de estados e transição é grande e por isso difícil de visualizar adequadamente a seqüência de execução. A Figura 4.21 apresenta um trecho desse grafo focando no início da sequência de execução do protocolo. A visualização do grafo não é a maneira mais adequada para verificar a sequência de execução do protocolo nesta situação.

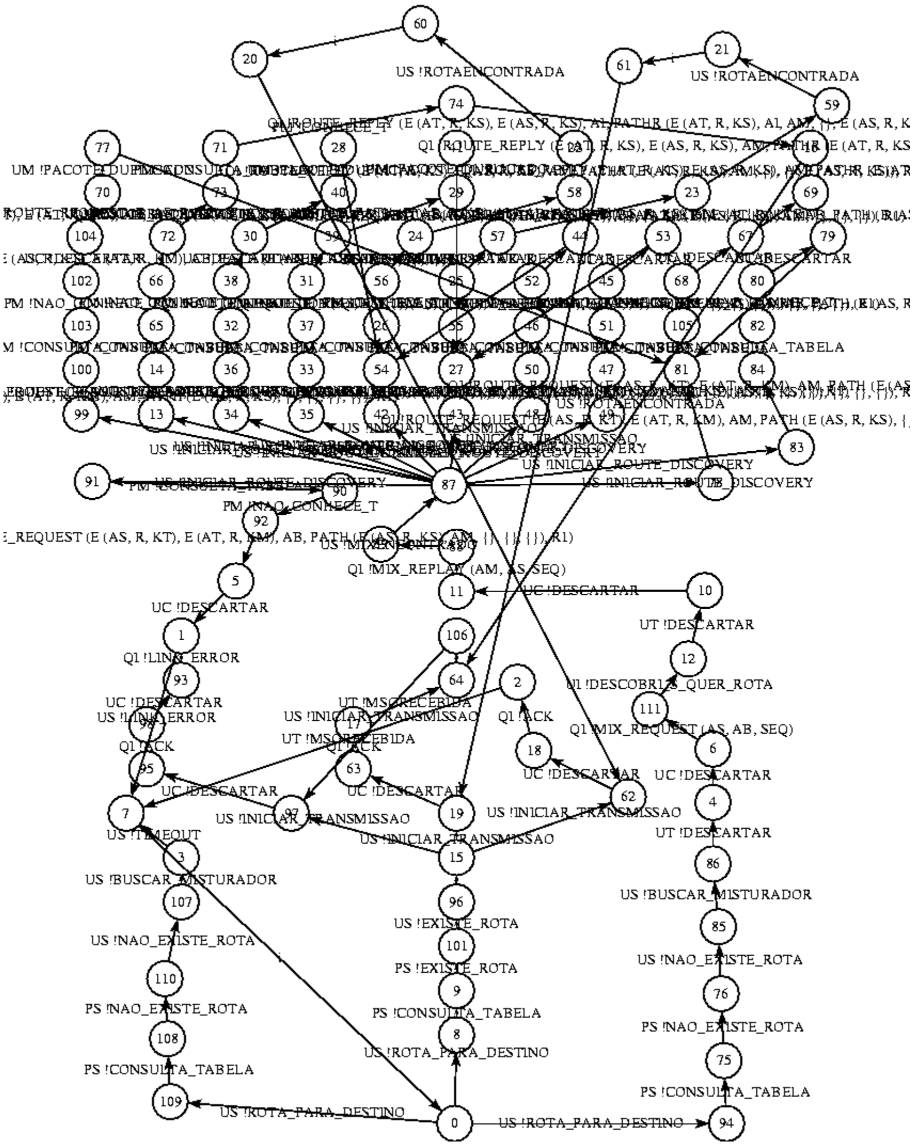


Figura 4.20: **Geração do grafo LTS do serviço do protocolo**

O chamado *Tour Transition* pode ser realizado manualmente através do grafo, porém como o número de transições é grande usou-se o Open/Caesar Simulator para verificar a

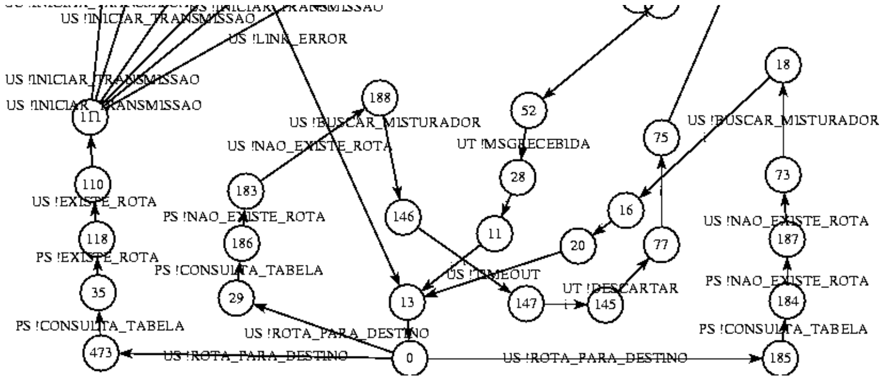


Figura 4.21: **Seqüência de execução do protocolo - Grafo LTS parcial**

sequência de execução do protocolo ANDSR. A Figura 4.22 apresenta um trecho dessa sequência sendo verificada. Esta é a forma mais adequada de verificação.

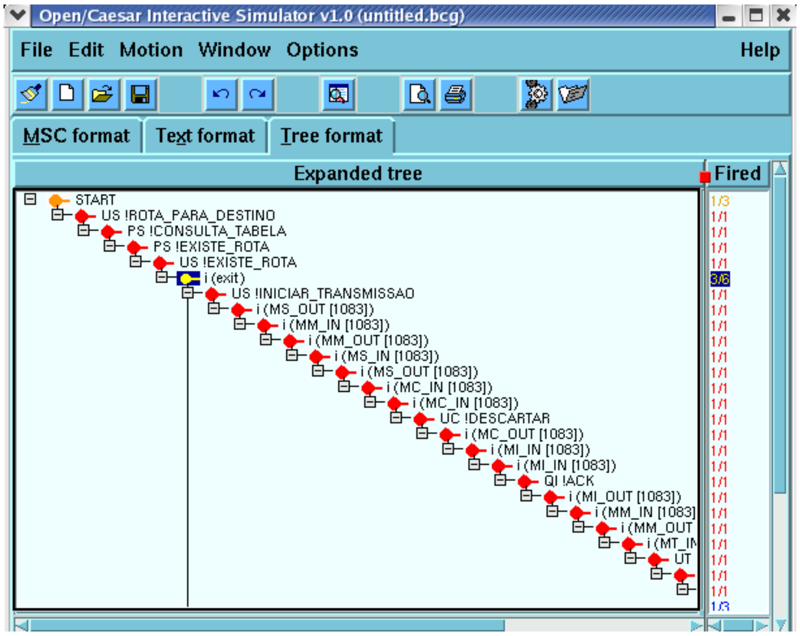


Figura 4.22: **Seqüência de execução do protocolo**

Existe a possibilidade de verificar somente as interações com a camada superior, ou seja verificar a sequência de execução do serviço, como mostra a Figura 4.23. Na verificação transição por transição é possível saber se existe algum bloqueio (*deadlock*) e alguma transição não alcançada (*livelock*). Para comprovar a inexistência dessas características

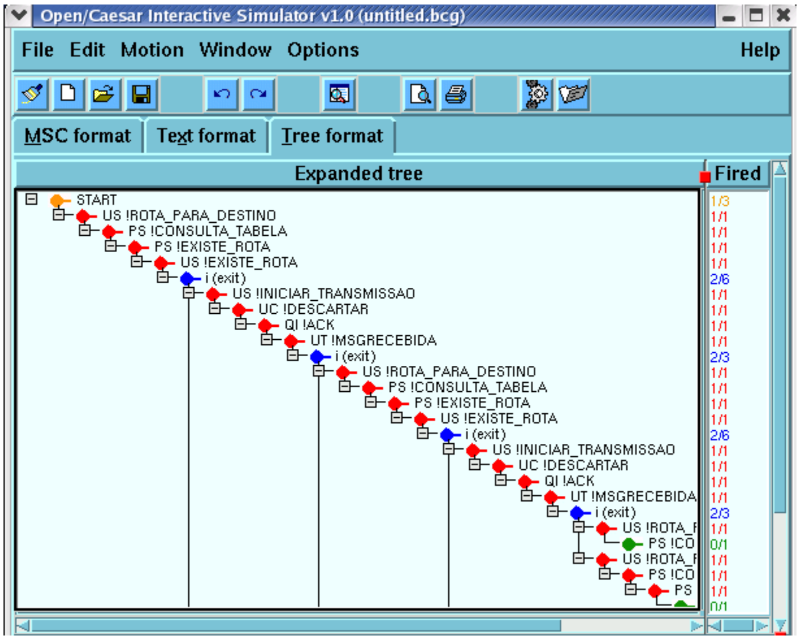


Figura 4.23: **Seqüência de execução do serviço**

no protoloto listamos as sequências de *deadlocks* e *livelocks*, vide Figuras 4.24 e 4.25.

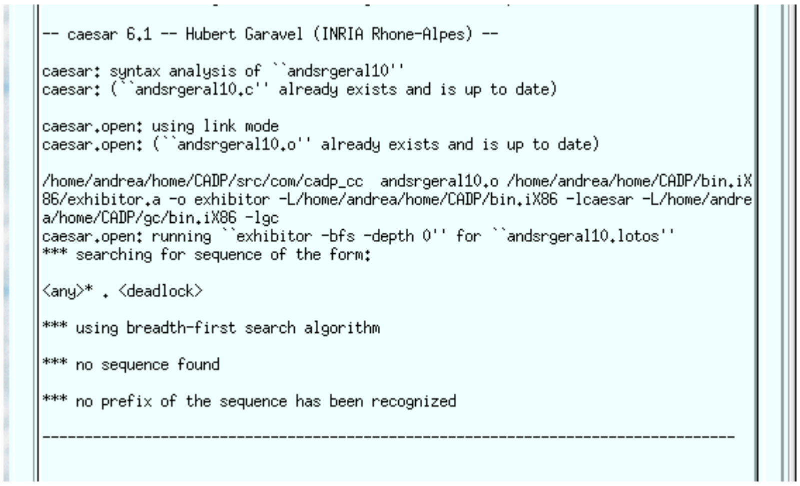
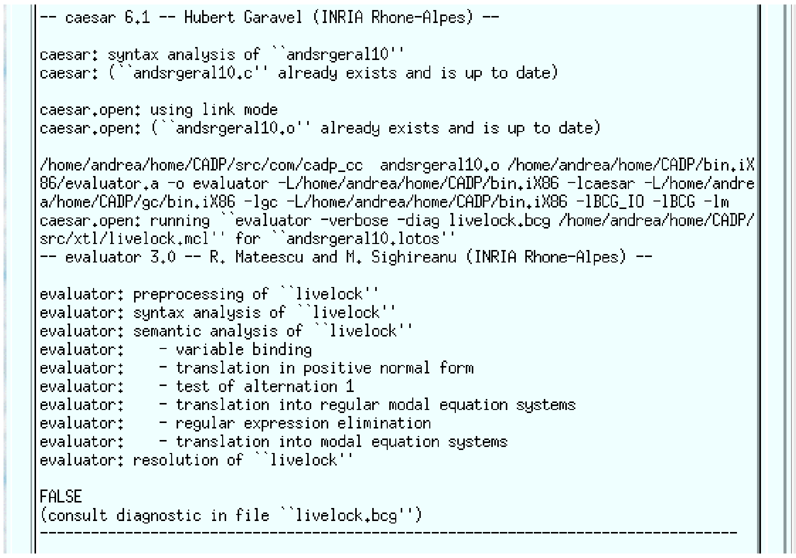

```
-- caesar 6.1 -- Hubert Garavel (INRIA Rhone-Alpes) --

caesar: syntax analysis of ``andsrgeral10''
caesar: (``andsrgeral10.c'' already exists and is up to date)

caesar.open: using link mode
caesar.open: (``andsrgeral10.o'' already exists and is up to date)

/home/andrea/home/CADP/src/com/cadp_cc  andsrgeral10.o /home/andrea/home/CADP/bin.iX
86/exhibitor.a -o exhibitor -L/home/andrea/home/CADP/bin.iX86 -lcaesar -L/home/andre
a/home/CADP/gc/bin.iX86 -lgc
caesar.open: running ``exhibitor -bfs -depth 0'' for ``andsrgeral10.lotos''
*** searching for sequence of the form:

<any>* . <deadlock>

*** using breadth-first search algorithm

*** no sequence found

*** no prefix of the sequence has been recognized

------------------------------------------------------------------------------------
```

Figura 4.24: **Deadlock**

Utilizando a ferramenta ALDEBARAN verificou-se a equivalência do modelo do protocolo e do modelo do serviço. Como resultado obteve-se TRUE significando que o protocolo e serviço são equivalentes observacionalmente [33], ou seja, o protocolo atende aos requisitos do serviço proposto no início do projeto. Diante dessa verificação conclui-se que o modelo do protocolo é correto, ou seja, está validado.



```
-- caesar 6.1 -- Hubert Garavel (INRIA Rhone-Alpes) --

caesar: syntax analysis of ``andsrgeral10''
caesar: (``andsrgeral10.c'' already exists and is up to date)

caesar.open: using link mode
caesar.open: (``andsrgeral10.o'' already exists and is up to date)

/home/andrea/home/CADP/src/com/cadp_cc  andsrgeral10.o /home/andrea/home/CADP/bin.iX
86/evaluator.a -o evaluator -L/home/andrea/home/CADP/bin.iX86 -lcaesar -L/home/andre
a/home/CADP/gc/bin.iX86 -lgc -L/home/andrea/home/CADP/bin.iX86 -lBCG_IO -lBCG -lm
caesar.open: running ``evaluator -verbose -diag livelock.bcg /home/andrea/home/CADP/
src/xtl/livelock.mcl'' for ``andsrgeral10.lotos''
-- evaluator 3.0 -- R. Mateescu and M. Sighireanu (INRIA Rhone-Alpes) --

evaluator: preprocessing of ``livelock''
evaluator: syntax analysis of ``livelock''
evaluator: semantic analysis of ``livelock''
evaluator:    - variable binding
evaluator:    - translation in positive normal form
evaluator:    - test of alternation 1
evaluator:    - translation into regular modal equation systems
evaluator:    - regular expression elimination
evaluator:    - translation into modal equation systems
evaluator: resolution of ``livelock''

FALSE
(consult diagnostic in file ``livelock.bcg'')
------------------------------------------------------------------------------------
```

Figura 4.25: **Livelock**

4.3 Comentários

O presente capítulo apresentou os resultados obtidos durante o desenvolvimento e validação do protocolo proposto. Cobriu-se um grande número de possíveis comportamentos que caracterizam o funcionamento do protocolo. O algoritmo modelado é seguro de acordo com o modelo de intruso desenvolvido, ele não previne contra ataques ativos, pois ainda não foi modelado para prevenir essas situações. Além disso, o modelo do protocolo foi projetado para inserção de futuras modificações. O próximo capítulo concluirá o trabalho fazendo algumas considerações a respeito do presente projeto e propostas para futuros trabalhos.

Capítulo 5

Conclusão

A NATUREZA das redes sem fio ad hoc gera muito estudos sobre segurança no projeto dessas redes. Vulnerabilidades dos canais de comunicação e nós, a ausência de infraestrutura e mudanças dinâmicas na topologia fazem da segurança em redes ad hoc uma questão difícil. Por ser uma rede sem qualquer infraestrutura fixa ou centralizada, soluções clássicas baseadas em autoridades certificadoras e servidores *on-line* são complicadas de serem aplicadas. Além disso, a segurança dos protocolos de roteamento em redes ad hoc é bastante complexa.

Compreender as possíveis formas de ataques é o primeiro passo para o desenvolvimento de boas soluções de segurança. Em redes ad hoc, os vários tipos de ataques podem ser separados em dois grupos: ataques passivos e ativos. Os ataques passivos não interrompem ou destrõem a operação da rede, porém podem revelar informações importantes sobre ela (como as identidades dos nós). Já os ataques ativos envolvem ações (por exemplo, modificação, destruição e inserção de mensagens) que prejudicam ou terminam com o funcionamento da rede.

Grande parte da pesquisa em segurança em redes ad hoc se dedica a propor soluções para os ataques ativos, devido aos danos evidentes que eles provocam (como destruição da operação da rede). A análise de tráfego é um ataque passivo que tenta descobrir informações sobre os nós (endereço, localização, etc.) através da escuta do tráfego de roteamento. Num ambiente sem fio como o das redes ad hoc é quase impossível percebê-lo, pois não produz nenhum novo tráfego. Mas, esse tipo de ataque é tão prejudicial quanto os ataques ativos, dependendo do tipo de aplicação a que se propõe a rede. Recentemente, as pesquisas começaram a estudar mais sobre o anonimato e privacidade em redes ad hoc.

Prover anonimato em redes ad hoc é complicado devido à natureza dessas redes. O objetivo dos pesquisadores é tentar transportar e adaptar conceitos conhecidos e utilizados na Internet na provisão do anonimato. Este trabalho apresentou o protocolo de roteamento anônimo ANDSR (*Anonymous Dynamic Source Routing*) como forma de criar dificuldades à análise de tráfego. O ANDSR esconde o endereço de origem e destino da comunicação e as informações trocadas através de criptografia, assim é preservada a privacidade das partes envolvidas na comunicação. Além disso, o ANDSR usa o conceito de nós misturadores [1] que são nós confiáveis e responsáveis pelo encaminhamento dos pacotes de forma anônima na rede. Seu algoritmo é baseado no protocolo DSR (*Dynamic Source Routing*), ou seja, seu algoritmo é uma alteração do DSR que proporciona o estabelecimento de caminhos ou rotas anônimas.

Para a validação do algoritmo proposto optou-se pelo uso de técnicas de descrição formal, as FDTs, pois essas técnicas permitem a modelagem, verificação e análise de sistemas distribuídos de forma precisa e não ambígua. Um protocolo que esteja especificado formalmente numa FDT pode ser verificado para correção e contra uma operação ruim, além de ser simulado para propósitos experimentais. Adicionalmente, as FDTs tornam o trabalho de alteração ou inserção de novas funcionalidades ao algoritmo uma tarefa mais fácil, se comparada a uma linguagem de programação como C, Java, entre outras.

No presente projeto utilizou-se a linguagem LOTOS desenvolvida pela ISO (*International Organization for Standardization*) para a especificação formal de sistemas distribuídos e concorrentes em geral. Optou-se pela linguagem LOTOS devido, sobretudo, ao seu rigor matemático. O software de análise usado foi o CADP, pois esse pacote de ferramentas é adequado à engenharia de protocolos e, além disso, o grupo que o desenvolveu oferece suporte. O CADP dedica-se a eficiente compilação, simulação, verificação formal e testes de descrições escritas em LOTOS.

O projeto do protocolo ANDSR pode ser dividido em duas fases. Na fase inicial do projeto, o desenvolvimento do protocolo foi baseado em estudos de casos. Nesses estudos de casos cada nó da rede representa um comportamento do protocolo. A adição de mais nós na rede é a adição de mais comportamentos. Porém, isso aumentou bastante a complexidade do algoritmo em desenvolvimento. Após análise de vários estudos de casos, havia a necessidade de um modelo que fosse mais abrangente. Então, optou-se pela adoção da metodologia sugerida por Dos Santos et. al. [11] numa fase posterior. Segundo essa metodologia a modelagem dos protocolos em redes ad hoc deveria ser independente da topologia e do número de nós considerados. A partir disso foi desenvolvido o modelo genérico do protocolo ANDSR. Esse modelo baseia-se somente no comportamento dos nós na rede e não considera topologia, posição de nós e quantidade de nós na rede. O objetivo dele é separar os nós em grupos, o grupo de nós fonte, o grupo de nós misturadores, o grupo de nós destino, o grupo de nós intrusos e o grupo de nós comuns. Cada grupo desse corresponde a uma nuvem de nós do mesmo tipo que executam o algoritmo da mesma maneira. Além disso, cada grupo é representado por um processo escrito em LOTOS. Nos estudos de casos, cada nó da rede era representado em LOTOS, então, caso a rede possuísse 14 nós existiriam 14 processos em LOTOS representando esses nós. Com a nova metodologia sempre existirão cinco processos em LOTOS representando os cinco tipos de nós da rede (nós fonte, misturadores, destino, intrusos e comuns), independente do número de nós na rede e de sua topologia. Isso diminui o código, o tempo de processamento, elimina as ambigüidades e facilita a alteração ou expansão do algoritmo.

No processo de verificação da especificação do protocolos ANDSR, o resultado mais importante é comprovar que o serviço foi oferecido, ou seja, comprovar que o protocolo é capaz de estabelecer rotas anônimas e fazer sua devida manutenção. De acordo com os resultados apresentados no capítulo anterior, os modelo de protocolo e serviço descritos em LOTOS apresentam equivalência observacional, demonstrando que o protocolo especificado atende aos requisitos do serviço. Este resultado comprova a exatidão do modelo proposto. Comprovou-se também que o protocolo ANDSR é vivo, ou seja, não há presença de bloqueios (*deadlocks*), é reinicializável, visto que, de qualquer estado, é possível alcançar o estado inicial e também não possui *livelocks*. Adicionalmente, o modelo do ANDSR desenvolvido neste projeto possui um número considerável de situações e comportamentos modelados caracterizando o funcionamento do protocolo.

Como contribuição cita-se o algoritmo ANDSR que trata de anonimato em redes ad hoc, assunto ainda pouco explorado, além de ser uma proposta que sugere alterações no algoritmo do protocolo bastante conhecido em redes ad hoc, o DSR; e a utilização de técnicas de descrição formal para validação de protocolos que apesar de seu rigor matemático na validação de sistemas distribuídos, ainda não é muito utilizada em redes ad hoc.

Para o futuro, seria interessante analisar o desempenho do protocolo proposto e avaliar o custo de seu funcionamento. Para essa análise seria necessário utilizar algum simulador, como o do próprio pacote CADP (ainda em desenvolvimento) ou o simulador de redes ns-2 [38], bastante difundido na comunidade científica, para fazer uma comparação com outros trabalhos da área. Outra proposta seria desenvolver um algoritmo que fosse resistente à análise de tráfego e ao mesmo tempo também prevenisse contra alguns ataques ativos.

Referências Bibliográficas

[1] CHAUM, D. L. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM 24*, 2 (1981), 84–90.

[2] BERTHOLD, O., FEDERRATH, H., E KÖHNTOPP, M. Project "Anonymity and Unobservability in the Internet".

[3] KESDOGAN, D., EGNER, J., E BUSCHKES, R. Stop-and-Go-Mixes Providing Probabilistic Security in an Open System. *Information Hiding: Second International Workshop 1525* (1998), 83–98.

[4] REED, M. G., SYVERSON, P. F., E GOLDSCHLAG, D. M. Anonymous Connections and Onion Routing. *IEEE Journal on Special Areas in Communications 16*, 4 (1998), 482–494.

[5] JIANG, S., VAIDYA, N., E ZHAO, W. A Dynamic Mix Method for Wireless Ad-Hoc Networks. *Military Communications Conference, MILCOM 2001* (2001).

[6] EL-KHATIB, K., KORBA, L., SONG, R., E YEE, G. Secure Dynamic Distributed Routing Algorithm for Wireless Ad Hoc Networks. *Proceedings in International Conference on Parallel Processing Workshops* (2003).

[7] KONG, J., E HONG, X. ANODR: ANonymous on Demand Routing with Untraceable Routes for Mobile Ad-hoc Networks. *Proceedings of the 4th ACM International Symposium on Mobile ad hoc Networking Computing* (2003).

[8] PFITZMANN, A., E KÖHNTOPP, M. Anonymity, Unobservability and Pseudonymity - A Proposal of Terminology. *Workshop on Design Issues in Anonymity and Unobservability* (2000).

[9] LEDUC, G., E GERMEAU, F. Verification of Security Protocols using LOTOS-Method and Application. *Computer Communications 23* (2000), 1089–1103.

[10] ISO/IFC. *Is 8807: Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Techniques Based on The Temporal Ordering of Observacional Behavior.* 1988.

[11] DOS SANTOS, C. F., CÂMARA, D., E LOUREIRO, A. A. F. Uma Metodologia para Verificação Formal de Protocolos de Roteamento para Redes Móveis Ad Hoc. *19°. SBRC - Simpósio Brasileiro de Redes de Computadores* (2001).

[12] BAGATELLI, R., MOURA, D. F. C., E PEDROZA, A. C. P. Especificação Formal de uma Arquitetura de Suporte à Descoberta de Serviços em Redes Móveis Ad Hoc. *V Workshop de Métodos Formais, WMF'2002* (2002).

[13] FERNANDEZ, J. C., GARAVEL, H., KERBRAT, A., MOUNIER, L., MATEESCU, R., E SIGHIREANU, M. CADP: A Protocol Validation and Verification Toolbox. *Proceedings of the Eight International Conference on Computer Aided Verification CAV 1102* (1996), 437–440.

[14] MESHNETWORKS. *http://www.meshnetworks.com.*

[15] TANENBAUM, A. S. *Redes de Computadores.* Editora Campus, 2003.

[16] PEREIRA, I. C. M., E C.P.P., A. Análise de Redes Móveis Ad Hoc para Cenários de Operações Militares. *I Workshop de Computação da Região Sul, I WORKCOMP - SUL* (2004).

[17] JOHANSSON, P., LARSSON, T., HEDMAN, N., MIELCZAREK, B., E DEGERMARK, M. Scenario-Based Performance Analysis of Routing Protocols for Mobile Ad-Hoc Networks. *5th ACM/IEEE International Conference on Mobile Computing and Networking - MOBICOM'99* (1999), 195–206.

[18] VILLELA, B. A. M., E DUARTE, O. C. M. B. Uma Análise de Protocolos de Roteamento Sob Demanda em Redes Ad Hoc. *Simpósio Brasileiro de Telecomunicações - SBT'2003* (2003).

[19] PERKINS, C. E., ROYER, E. M., E DAS, S. R. Ad Hoc On Demand Distance Vector (AODV) Routing. *Internet draft, draft-ietf-manet-aodv-13.txt* (2003).

[20] PERKINS, C. E., E ROYER, E. M. Ad Hoc On Demand Distance Vector Routing. *Second IEEE Worshop on Mobile Computing Sysytems and Applications - WMCSA'99* (February de 1999), 90–100.

[21] PERKINS, C. E., E BAHAGWAT, P. Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for Mobile Computers. *ACM Conference on Communication Architetures, Protocols and Applications - SIGCOMM'94* (1994), 234–244.

[22] JOHNSON, D. B., MALTZ, D. A., E HU, Y.-C. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). *Internet Draft, draft-ietf-manet-dsr-10.txt* (2004).

[23] MOHAPRATA, P., GUI, C., E LI, J. Group Communications in Mobile Ad Hoc Networks. *IEEE Computer Society* (2004), 52–59.

[24] TEIXEIRA, I., DE REZENDE, J. F., E PEDROZA, A. C. P. Wireless Sensor Network: Improving the Network Energy Consumption. *XXI Simpósio Brasileiro de Telecomunicações - SBT'2004* (2004).

[25] CHEN, S., E NAHRSTEDT, K. Distributed Quality-of-Service Routing in Ad Hoc Networks. *IEEE Journal on Selected Areas in Communication 17*, 8 (1999).

[26] SIVAKUMAR, R., E SINHA, P. CEDAR: A Core-Extration Distributed Ad Hoc Routing Algorithm. *IEEE Journal on Selected Areas in Communication 17*, 8 (1999).

[27] ROCHA, L. G. S., COSTA, L. H. M. K., E DUARTE, O. C. M. B. Avaliação de Impacto da Ação Maliciosa de Nós no Roteamento em Redes Ad Hoc. *III Workshop em Segurança em Sistemas Computacionais, III WSEG* (2003), 69–76.

[28] GUPTE, S., E SINGHAL, M. Secure Routing in Mobile Wireless Ad Hoc Networks. *Ad Hoc Networks 1*, 1 (2003), 151–174.

[29] HU, Y.-C., PERRIG, A., E JOHNSON, D. B. Ariadne: A Secure On Demand Routing Protocol for Ad Hoc Networks. *Proceedings of the 8th ACM/IEEE International Conference on Mobile Computing and Networking - MOBICOM'02* (2002), 12–23.

[30] BROCH, J., MALTZ, D. A., JOHNSON, D. B., HU, Y.-C., E JETCHEVA, J. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. *4th ACM/IEEE International Conference on Mobile Computing and Networking - MOBICOM'98* (1998), 85–97.

[31] R. L. RIVEST, A. S., E ADLEAMN, L. A Method Obtaining Sigital Signatures and Public-key Criptosystems. *Communication ACM 21*, 2 (1977), 120–126.

[32] DIETRICH, F., E HUBAUX, J.-P. Formal Methods for Communication Services: Meeting the Industrial Expectations. *Computer Networks 38* (2002), 99–120.

[33] MILNER, R. *Communication and Concurrency*. Prentice Hall International, 1989.

[34] ARAUJO, A. M., E PEDROZA, A. C. P. Um Protocolo de Roteamento Anônimo para Redes Móveis Ad Hoc. *I Workshop de Computação da Região Sul, I WORKCOMP - SUL* (2004).

[35] ARAUJO, A. M., E PEDROZA, A. C. P. Especificação Formal de um Protocolo de Roteamento Anônimo para Redes Ad Hoc. *IV Workshop em Segurança em Sistemas Computacionais, IV WSEG* (2004), 151–162.

[36] FERNANDEZ, J.-C., JARD, C., JÉRON, T., NEDELKA, L., E VIHO, C. An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology. *Sci. Comput. Program. 29*, 1-2 (1997), 123–146.

[37] ARAUJO, A. M., E PEDROZA, A. C. P. Um Protocolo de Roteamento Anônimo para Redes Ad Hoc. *XV Congresso Brasileiro de Automática, CBA 2004* (2004).

[38] NS 2. *http://www.isi.edu/nsnam/ns/*.

Apêndice A

Biblioteca ANDSR_LIB.lib

O CÓDIGO apresentado aqui é a biblioteca ANDSR_LIB.lib desenvolvida em LOTOS. Nela estão declaradas todas as funções e tipos criados pelo projetista.

```
library BOOLEAN, NATURALNUMBER endlib


type MSG is Boolean


sorts MSG (*! implementedby MSG comparedby CMP_MSG enumeratedby
      ENUM_MSG printedby PRINT_MSG *)

opns

Rota_para_Destino  (*! implementedby Rota_para_Destino constructor *),
Iniciar_Route_Discovery (*! implementedby Iniciar_Route_Discovery
            constructor *),
Buscar_Misturador (*! implementedby Buscar_Misturador constructor *),
Nao_existe_rota   (*! implementedby Nao_existe_rota constructor *),
Consulta_Tabela   (*! implementedby Consulta_Tabela constructor *),
Existe_rota    (*! implementedby Existe_rota constructor *),
Nao_conhece_T  (*! implementedby Nao_conhece_T constructor *),
Conhece_T      (*! implementedby Conhece_T constructor *),
RotaEncontrada (*! implementedby RotaEncontrada constructor *),
Descartar      (*! implementedby Descartar constructor *),
Continua       (*! implementedby Continua constructor *),
MixEncontrado  (*! implementedby MixEncontrado constructor *),
Iniciar_Transmissao (*! implementedby Iniciar_Transmissao
constructor *),
```

```
Descobri_S_quer_Rota (*! implementedby Descobri_S_quer_Rota
 constructor *),
Observa (*! implementedby Observa  constructor *),
msgRecebida  (*! implementedby msgRecebida constructor *),
Timeout      (*! implementedby Timeout constructor *),
Sincronismo  (*! implementedby Sincronismo constructor *),
Pacote_Duplicado (*! implementedby Pacote_Duplicado constructor *),
Nothing  (*! implementedby Nothing  constructor *),
Link_Error  (*! implementedby Link_Error constructor *),
NaoRecebeu  (*! implementedby NaoRecebeu constructor *),
RREQ     (*! implementedby RREQ constructor *),
RREP        (*! implementedby RREP constructor *),
ACK         (*! implementedby ACK constructor *),
msg         (*! implementedby msg constructor *),
msg_resp    (*! implementedby msg_resp constructor *),
eq     (*! implementedby Seq constructor *): -> MSG
_eq_     (*! implementedby EQ_MSG *),
_ne_       (*! implementedby NEQ_MSG *): MSG, MSG -> Bool

eqns  forall X,Y : MSG

ofsort Bool

X=Y => X eq Y = true;
X eq Y = false;  (* para os outros casos *)

X=Y => X ne Y = false;
X ne Y = true;

endtype


type Public_key is

sorts Public_key (*! implementedby Public_key comparedby
     CMP_Public_key enumeratedby
     ENUM_Public_key printedby
     PRINT_Public_key *)

opns Ks (*! implementedby  Ks constructor *),
     Km (*! implementedby  Km constructor *),
     Kc (*! implementedby  Kc constructor *),
     Ki (*! implementedby  Ki constructor *),
     Kt (*! implementedby  Kt constructor *): -> Public_key

endtype

type Private_key is
```

```
sorts Private_key (*! implementedby Private_key comparedby
      CMP_Private_key enumeratedby
      ENUM_Private_key printedby
      PRINT_Private_key *)

opns Ss (*! implementedby  Ss constructor *),
     Sc (*! implementedby  Sc constructor *),
     Sm (*! implementedby  Sm constructor *),
     Si (*! implementedby  Si constructor *),
     St (*! implementedby  St constructor *): -> Private_key

endtype

type Random_number is

sorts Random_number (*! implementedby Randon_number comparedby
CMP_Randon_number enumeratedby ENUM_Randon_number
printedby PRINT_Randon_number *)

opns  R (*! implementedby  R constructor *): -> Random_number


endtype



type Address is

sorts Address (*! implementedby Address comparedby CMP_Address
  enumeratedby ENUM_Address printedby PRINT_Address *)

opns {} (*! implementedby  vazio constructor *),
     Ab (*! implementedby  Ab constructor *),
     As (*! implementedby  As constructor *),
     Am (*! implementedby  Am constructor *),
     Am1 (*! implementedby  Am1 constructor *),
     Ai (*! implementedby  Ai constructor *),
     Ac (*! implementedby  Ac constructor *),
     At (*! implementedby  At constructor *): -> Address

endtype

type Encrypt is Address, Random_number,Public_key

sorts Encrypt (*! implementedby Encrypt comparedby CMP_Encrypt
  enumeratedby ENUM_Encrypt printedby PRINT_Encrypt *)

opns
```

```
E (*! implementedby E constructor *): Address, Random_number,
              Public_key -> Encrypt

endtype


type Encrypt_Dados is MSG, Random_number, Public_key

sorts Encrypt_Dados (*! implementedby Encrypt_Dados comparedby
CMP_Encrypt_Dados enumeratedby
ENUM_Encrypt_Dados printedby
PRINT_Encrypt_Dados *)

opns

Ed (*! implementedby Ed constructor *): MSG, Random_number,
Public_key -> Encrypt_Dados

endtype



type Path is Encrypt, Address, Address, Address, Address

sorts Path (*! implementedby Path comparedby CMP_Path
       enumeratedby ENUM_Path printedby PRINT_Path *)

opns
      path (*! implementedby path constructor *): Encrypt, Address,
  Address, Address,
  Address -> Path

endtype

type Path_R is Encrypt, Address, Address, Address, Encrypt

sorts Path_R (*! implementedby Path_R comparedby CMP_Path_R
 enumeratedby ENUM_Path_R printedby PRINT_Path_R *)

opns

   pathR (*! implementedby pathR constructor *): Encrypt, Address,
 Address, Address,
 Encrypt -> Path_R

   null  (*! implementedby null constructor *) : -> Path_R
```

```
endtype


type ident is

sorts ident  (*! implementedby ident comparedby CMP_ident
 enumeratedby ENUM_ident printedby PRINT_ident *)

opns

 r1 (*! implementedby r1 constructor *),
 r2 (*! implementedby r2 constructor *),
 r3 (*! implementedby r3 constructor *):  -> ident

endtype



type RouteRequest is Encrypt, Encrypt, Address, Path, ident

sorts RouteRequest (*! implementedby RouteRequest comparedby
       CMP_RouteRequest enumeratedby
       ENUM_RouteRequest printedby
       PRINT_RouteRequest *)

opns

Route_Request (*! implementedby Route_Request constructor *):
       Encrypt, Encrypt,
Address, Path,
ident -> RouteRequest


endtype


type RouteReply is Encrypt, Encrypt, Address, Path_R, ident

sorts RouteReply (*! implementedby RouteReply comparedby
     CMP_RouteReply enumeratedby
     ENUM_RouteReply printedby
     PRINT_RouteReply *)

opns

Route_Reply (*! implementedby Route_Reply constructor *):
Encrypt, Encrypt,
Address, Path_R,
```

```
ident -> RouteReply


endtype


type MixRequest is Address, Address, MSG

sorts MixRequest (*! implementedby MixRequest comparedby
     CMP_MixRequest enumeratedby
     ENUM_MixRequest printedby
     PRINT_MixRequest *)

opns

Mix_Request (*! implementedby Mix_Request constructor *):
 Address,
 Address,
 MSG -> MixRequest


endtype




type MixReplay is Address, Address, MSG

sorts MixReplay (*! implementedby MixReplay comparedby
    CMP_MixReplay enumeratedby ENUM_MixReplay
    printedby PRINT_MixReplay *)

opns

Mix_Replay (*! implementedby Mix_Replay constructor *):
Address,
Address,
MSG -> MixReplay


endtype

type Pacote_dados is Encrypt, Encrypt, Address, MSG, Encrypt_Dados

sorts Pacote_dados (*! implementedby Pacote_dados comparedby
       CMP_Pacote_dados enumeratedby
       ENUM_Pacote_dados printedby
       PRINT_Pacote_dados *)
```

```
opns

Pkt_d (*! implementedby Pkt_d constructor *): Encrypt,
Encrypt,
Address,
Encrypt_Dados -> Pacote_dados



endtype


type Rota is Address, Address, Address, Address, Address,
     Address

sorts Rota (*! implementedby Rota comparedby CMP_Rota
       enumeratedby ENUM_Rota printedby PRINT_Rota *)

opns

rota (*! implementedby rota constructor *):
Address, Address, Address, Address, Address,
Address -> Rota


endtype


type CacheRota is Rota, Rota, Rota

sorts CacheRota (*! implementedby CacheRota comparedby CMP_CacheRota
    enumeratedby ENUM_CacheRota printedby
    PRINT_CacheRota *)

opns

Route_cache (*! implementedby Route_cache constructor *):
    Rota, Rota,
    Rota -> CacheRota


endtype

type Buffer is MSG

sorts Buffer (*! implementedby  Buffer comparedby CMP_Buffer
 enumeratedby ENUM_Buffer printedby
 PRINT_Buffer *)
```

```
opns

Send_Buffer (*! implementedby Send_Buffer constructor *):
           MSG -> Buffer


endtype



type Menor_Rota is Address, Address, Address, Address,
   Address, Address

sorts Menor_Rota (*! implementedby Menor_Rota comparedby
     CMP_Menor_Rota enumeratedby ENUM_Menor_Rota
     printedby PRINT_Menor_Rota *)
opns

Menor_rota (*! implementedby Menor_rota constructor *):
Address, Address, Address, Address, Address,
Address -> Menor_Rota

endtype
```

Apêndice B

ANDSR.lotos

O CÓDIGO apresentado aqui é a especificação do protocolo ANDSR em LOTOS.

```
specification ANDSR_V11 [Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                         Qi, Ut, Pt, Qt ]: noexit

(* Modelo Geral *)
library ANDSR_LIB10 endlib

behaviour


      Rede[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
           Pt, Qt ]

where

process Rede[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
             Pt, Qt ]: noexit :=

(
 Busca_Misturador_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                             Ui, Pi, Qi, Ut, Pt, Qt ]
         >>(
 TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                     Qi, Ut, Pt, Qt ]
 []
 TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                     Qi, Ut, Pt, Qt ]
 []
```

```
 Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc,
                               Qc, Ui, Pi, Qi, Ut, Pt, Qt ]
 []
 Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc,
                               Qc, Ui, Pi, Qi, Ut, Pt, Qt ]
 []
 (TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                     Qi, Ut, Pt, Qt ]
  >> Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc,
                                   Qc, Ui, Pi, Qi, Ut, Pt, Qt ]
 )
 []
 (TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                     Qi, Ut, Pt, Qt ]
  >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc,
                                   Qc, Ui, Pi, Qi, Ut, Pt, Qt ]
 )
    ) >> Rede[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                     Qi, Ut, Pt, Qt ]

)
[]
        (
Busca_Misturador_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui,
                           Pi, Qi, Ut, Pt, Qt ]
>>(
(Busca_Rota_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]
>> TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                      Qi, Ut, Pt, Qt ]
)

[] (Busca_Rota_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]
>> TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                      Qi, Ut, Pt, Qt ]
)

[]
(Busca_Rota_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]
               >> TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                      Qi, Ut, Pt, Qt ]
  >> Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                                  Ui, Pi, Qi, Ut, Pt, Qt ]
               )

[]
```

```
(Busca_Rota_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]
                 >> Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc,
                                  Qc, Ui, Pi, Qi, Ut, Pt, Qt ]
                )

[]

(Busca_Rota_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui,
                              Pi, Qi, Ut, Pt, Qt ]
                 >> TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                        Ut, Pt, Qt ]
              >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                                  Ui, Pi, Qi, Ut, Pt, Qt ]
)

[]

(Busca_Rota_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                              Qi, Ut, Pt, Qt ]
                 >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                                  Ui, Pi, Qi, Ut, Pt, Qt ]
                )

[]

(Busca_Rota_c3[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]
)


        []      (Busca_Rota_c4[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]
                >> TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                       Qi, Ut, Pt, Qt ]
                )


        []

                (Busca_Rota_c4[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]
                 >> TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                        Qi, Ut, Pt, Qt ]
                 >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc,
                                  Qc, Ui, Pi, Qi, Ut, Pt, Qt ]
                )
```

```
        []

                (Busca_Rota_c4[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                               Ui, Pi, Qi, Ut, Pt, Qt ]
                 >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc,
                                   Qc, Ui, Pi, Qi, Ut, Pt, Qt ]
                )

        []

                (Busca_Rota_c5[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui,
                               Pi, Qi, Ut, Pt, Qt ]
                 >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                                   Ui, Pi, Qi, Ut, Pt, Qt ]
                )




  ) >> Rede[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                    Qi, Ut, Pt, Qt ]
)
[]
(
 Busca_Misturador_c3[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                             Ui, Pi, Qi, Ut, Pt, Qt ]
 >> Rede[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                 Ut, Pt, Qt ]
)


where

(* Caso em q o noh fonte nao conhece a rota p/ o destino  *)
process Busca_misturador_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                            Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
             Mi_in, Mi_out, Mt_in, Mt_out in

(
Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks,
                                    R, As, At, Ab, Ai, Ac, Am)
|||
Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km,
                                Ks, R, As, At, Ab, Ai, Ac, Am)
|||
```

```
Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km, Ks, R,
                                        As, At, Ab, Ai, Ac, Am)
|||
Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km, Ks,
                                    R, As, At, Ab, Ai, Ac, Am)
|||
Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt, Km, Ks,
                                    R, As, At, Ab, Ai, Ac, Am)
)

|[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                  Mi_in, Mi_out, Mt_in, Mt_out]|

(
Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks,
                                      R, As, At, Ab, Ai, Ac, Am)
)


where

process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks :
                       Public_key, R : Random_number, As, At, Ab,
                       Ai, Ac, Am : Address) : exit :=

Us ! Rota_para_Destino;
Ps ! Consulta_Tabela;
Ps ! Nao_existe_rota;
Us ! Nao_existe_rota;
Us ! Buscar_Misturador;

Ms_out ! Mix_Request(As, Ab, Seq);

Ms_in ! Sincronismo;
Ms_in ! Mix_Replay(Am, As, Seq);
Us ! MixEncontrado;

exit
endproc

process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km,
                           Ks : Public_key, R : Random_number,
                           As, At, Ab, Ai, Ac, Am : Address)
                           : exit :=

Mm_in ! Sincronismo;
  Mm_in ! Mix_Request(As, Ab, Seq);
```

```
Mm_out ! Mix_Replay(Am, As, Seq);

exit

endproc

process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt, Km, Ks :
                      Public_key, R : Random_number, As, At,
                      Ab, Ai, Ac, Am : Address) : exit :=

Mc_in ! Sincronismo;
Mc_in ! Mix_Request(As, Ab, Seq);
Uc ! Descartar;
Mc_out ! Sincronismo;

Mc_in ! Sincronismo;
Mc_in ! Mix_Replay(Am, As, Seq);
Uc ! Descartar;
Mc_out ! Sincronismo;

exit


endproc

process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km, Ks :
                        Public_key, R : Random_number, As, At,
                        Ab, Ai, Ac, Am : Address) : exit :=



Mi_in ! Sincronismo;
Mi_in ! Mix_Request(As, Ab, Seq);
Qi ! Mix_Request(As, Ab, Seq);
Ui ! Descobri_S_quer_Rota;
Mi_out ! Sincronismo;

Mi_in ! Sincronismo;
Mi_in ! Mix_Replay(Am, As, Seq);
  Qi ! Mix_Replay(Am, As, Seq);
Mi_out ! Sincronismo;

exit


endproc


process Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt, Km, Ks :
```

```
                       Public_key, R : Random_number, As, At,
                       Ab, Ai, Ac, Am : Address) : exit :=


Mt_in ! Mix_Request(As, Ab, Seq);
Ut ! Descartar;
Mt_out ! Sincronismo;

Mt_in ! Mix_Replay(Am, As, Seq);
Ut ! Descartar;
Mt_out ! Sincronismo;

exit


endproc


process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in , Mt_out](Kt, Km, Ks
                     : Public_key, R : Random_number, As, At, Ab,
                     Ai, Ac, Am : Address) : exit :=


Ms_out ! Mix_Request(As, Ab, Seq);
Mt_in ! Mix_Request(As, Ab, Seq);
Mt_out ! Sincronismo;
Mc_in ! Sincronismo;
Mc_in ! Mix_Request(As, Ab, Seq);
Mc_out ! Sincronismo;
Mi_in ! Sincronismo;
Mi_in ! Mix_Request(As, Ab, Seq);
Mi_out ! Sincronismo;
Mm_in ! Sincronismo;
Mm_in ! Mix_Request(As, Ab, Seq);
Mm_out ! Mix_Replay(Am, As, Seq);
Mt_in ! Mix_Replay(Am, As, Seq);
Mt_out ! Sincronismo;
Mc_in ! Sincronismo;
Mc_in ! Mix_Replay(Am, As, Seq);
Mc_out ! Sincronismo;
Mi_in ! Sincronismo;
Mi_in ! Mix_Replay(Am, As, Seq);
Mi_out ! Sincronismo;
Ms_in ! Sincronismo;
Ms_in ! Mix_Replay(Am, As, Seq);

exit
```

```
endproc

endproc
(*******************************************************)
(* Caso em que o no fonte conhece a rota para o destino *)
process Busca_misturador_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc,
                            Qc, Ui, Pi, Qi, Ut, Pt,
                            Qt ]: exit :=
        Us ! Rota_para_Destino;
        Ps ! Consulta_Tabela;
      Ps ! Existe_rota;
Us ! Existe_rota;
exit
endproc

(*******************************************************)
(* Nao ha misturadores no raio de alcance da fonte  *)
process Busca_misturador_c3[Us, Ps, Qs, Um, Pm, Qm, Uc,
                            Pc, Qc, Ui, Pi, Qi, Ut, Pt,
                            Qt ]: exit :=

hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
             Mi_in, Mi_out, Mt_in, Mt_out in

        (
                Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks,
    R, As, At, Ab, Ai, Ac, Am)

                |[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                  Mi_in, Mi_out, Mt_in, Mt_out]|

                Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks,
                                      R, As, At, Ab, Ai, Ac, Am)
        )

where

process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks
                       : Public_key, R : Random_number, As, At,
                       Ab, Ai, Ac, Am : Address) : exit :=

Us ! Rota_para_Destino;
                Ps ! Consulta_Tabela;
                Ps ! Nao_existe_rota;
                Us ! Nao_existe_rota;
                Us ! Buscar_Misturador;
```

```
                Ms_out ! Mix_Request(As, Ab, Seq);
         Ms_in ! Timeout;
Us ! Timeout;

exit

endproc

        process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks :
                     Public_key, R : Random_number, As, At, Ab,
                     Ai, Ac, Am : Address) : exit :=

Ms_out ! Mix_Request(As, Ab, Seq);
Ms_in ! Timeout;
exit

endproc


endproc


(***************************************************************)
(* Caso em que so existe um misturador na rota *)

process Busca_Rota_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                      Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
             Mi_in, Mi_out, Mt_in, Mt_out in

(
Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks,
                                    R, As, At, Ab, Ai, Ac, Am)
|||
Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt,
                         Km, Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km,
                            Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km,
                             Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
Target[Ut, Pt, Qt, Mt_in, Mt_out](Kt, Km,
                            Ks, R, As, At, Ab, Ai, Ac, Am)
```

```
)

|[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                 Mi_in, Mi_out, Mt_in, Mt_out]|
(
Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in,
                     Mc_out, Mi_in, Mi_out, Mt_in, Mt_out]
                     (Kt, Km, Ks, R, As, At, Ab, Ai, Ac, Am)
)


where

process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km,
                       Ks : Public_key, R : Random_number,
                       As, At, Ab, Ai, Ac, Am : Address)
                       : exit :=

Us ! Iniciar_Route_Discovery;

Ms_out ! Route_Request(E(As, R, Kt),
                         E(At, R, Km), Am,
                         path(E(As, R, Ks), {}, {},
                               {}, {}), r1);
Ms_in ! Route_Reply(E(At, R, Ks),
                        E(As, R, Ks), Ab, pathR(E(At, R, Ks),
                              Am, {}, {}, E(As, R, Ks)), r1);
Us ! RotaEncontrada;

exit

endproc

process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km,
                           Ks : Public_key, R : Random_number,
                           As, At, Ab, Ai, Ac, Am : Address)
                           : exit :=

Mm_in ! Sincronismo;
Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                        Am, path(E(As, R, Ks), {}, {}, {},
                                 {}), r1);

Pm ! Consulta_Tabela;
Pm ! Conhece_T;

Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                         Ab, path(E(As, R, Ks), Am, {}, {},
```

```
                                  {}), r1);



Mm_in ! Sincronismo;
Mm_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Am, pathR(E(At, R, Ks), Am, {}, {},
                                  E(As, R, Ks)), r1);



Mm_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                         Ab, pathR(E(At, R, Ks), Am, {}, {},
                                   E(As, R, Ks)), r1);
exit

endproc

process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt, Km,
                      Ks : Public_key, R : Random_number, As,
                      At, Ab, Ai, Ac, Am : Address) : exit :=



Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                        Ab, path(E(As, R, Ks), Am, {}, {},
                                 {}), r1);
Uc ! Descartar;
Mc_out ! Sincronismo;

exit


endproc


process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km,
                        Ks : Public_key, R : Random_number,
                        As, At, Ab, Ai, Ac, Am : Address)
                        : exit :=



Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                        Am, path(E(As, R, Ks), {}, {}, {},
                                   {}), r1);
Qi ! Route_Request(E(As, R, Kt), E(At, R, Km),
                     Am, path(E(As, R, Ks), {}, {}, {},
                              {}), r1);
Mi_out ! Sincronismo;

Mi_in ! sincronismo;
```

```
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                        Ab, path(E(As, R, Ks), Am, {}, {},
                                 {}), r1);
Qi ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                          path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_out ! Sincronismo;



Mi_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Am, pathR(E(At, R, Ks), Am, {}, {},
                                  E(As, R, Ks)), r1);
Qi ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                     Am, pathR(E(At, R, Ks), Am, {}, {},
                               E(As, R, Ks)), r1);
Mi_out ! Sincronismo;

exit


endproc





process Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt, Km, Ks :
                       Public_key, R : Random_number, As, At, Ab,
                       Ai, Ac, Am : Address) : exit :=



Mt_in ! Sincronismo;


Mt_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                        Ab, path(E(As, R, Ks), Am, {}, {},
                                  {}), r1);


Mt_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                         Am, pathR(E(At, R, Ks), Am, {}, {},
                                   E(As, R, Ks)), r1);



exit



endproc

process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks :
```

```
                     Public_key, R : Random_number, As, At, Ab, Ai,
                     Ac, Am : Address) : exit :=


Ms_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                         Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                        Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
                Mi_out ! Sincronismo;



Mm_in ! Sincronismo;
Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                        Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                         Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                        Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
                Mc_out ! Sincronismo;

Mi_in ! Sincronismo;
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                        Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_out ! Sincronismo;

Mt_in ! Sincronismo;

Mt_in ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mt_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
                         pathR(E(At, R, Ks), Am, {}, {},
                               E(As, R, Ks)), r1);
Mi_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Am, pathR(E(At, R, Ks), Am, {}, {},
                                  E(As, R, Ks)), r1);
Mi_out ! Sincronismo;

Mm_in ! Sincronismo;

Mm_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Am, pathR(E(At, R, Ks), Am, {}, {},
                                  E(As, R, Ks)), r1);

Mm_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                         Ab, pathR(E(At, R, Ks), Am, {}, {},
                                   E(As, R, Ks)), r1);
```

```
Ms_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Ab, pathR(E(At, R, Ks), Am, {}, {},
                                  E(As, R, Ks)), r1);

exit


endproc


endproc

(************************************************************)
(* Existe varios misturadores na rota *)

process Busca_Rota_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui,
                      Pi, Qi, Ut, Pt, Qt ]: exit :=

hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out, Mi_in,
             Mi_out, Mt_in, Mt_out in

(
Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks,
                                     R, As, At, Ab, Ai, Ac, Am)
|||
Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km,
                                 Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km, Ks, R,
                                         As, At, Ab, Ai, Ac, Am)
|||
Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km, Ks,
                                      R, As, At, Ab, Ai, Ac, Am)
                |||
Target[Ut, Pt, Qt, Mt_in, Mt_out](Kt, Km, Ks, R,
                                         As, At, Ab, Ai, Ac, Am)
)

|[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                                  Mi_in, Mi_out, Mt_in, Mt_out]|

(
Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks,
                                       R, As, At, Ab, Ai, Ac, Am)
)
```

```
where

process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks :
                       Public_key, R : Random_number, As, At, Ab,
                       Ai, Ac, Am : Address) : exit :=

Us ! Iniciar_Route_Discovery;
Ms_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                         Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
Ms_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Ab,
                        pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                              Ks)), r1);
Us ! RotaEncontrada;

exit

endproc

process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km, Ks :
                           Public_key, R : Random_number, As, At, Ab,
                           Ai, Ac, Am : Address) : exit :=

Mm_in !Sincronismo;

Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Am,
                        path(E(As, R, Ks), {}, {}, {}, {}), r1);

Pm ! Consulta_Tabela;
Pm ! Nao_conhece_T;

Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                         path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mm_in ! Sincronismo;

Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);

Pm ! Consulta_Tabela;
Pm ! Conhece_T;

Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                         path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mm_in ! Sincronismo;

Mm_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
```

```
                        pathR(E(At, R, Ks), Am, {}, {}, E(As, R, Ks))
                              , r1);

Mm_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Ab,
                         pathR(E(At, R, Ks), Am, {}, {}, E(As, R, Ks)
                               ), r1);

exit

endproc

process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt, Km, Ks :
                      Public_key, R : Random_number, As, At, Ab, Ai,
                      Ac, Am : Address) : exit :=



Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);
Uc ! Descartar;
Mc_out ! Sincronismo;

Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);
Uc ! Descartar;
Mc_out ! Sincronismo;


exit


endproc


process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km, Ks : Public_key,
                        R : Random_number, As, At, Ab, Ai, Ac, Am : Address)
                        : exit :=




Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Am,
                        path(E(As, R, Ks), {}, {}, {}, {}), r1);
Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Am,
                     path(E(As, R, Ks), {}, {}, {}, {}), r1);
Mi_out ! Sincronismo;

Mi_in ! Sincronismo;
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);
```

```
Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                     path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_out ! Sincronismo;


Mi_in ! sincronismo;
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);
Qi ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                     path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_out ! Sincronismo;



Mi_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
                        pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                              Ks)), r1);
Qi ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                     Am, pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                               Ks)), r1);
Mi_out ! Sincronismo;

exit


endproc




process Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt, Km, Ks :
                       Public_key, R : Random_number, As, At, Ab,
                       Ai, Ac, Am : Address) : exit :=


Mt_in !Sincronismo;

Mt_in ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mt_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
                         pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                               Ks)), r1);


exit
```

```
endproc

process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                    Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks :
                    Public_key, R : Random_number, As, At, Ab, Ai,
                    Ac, Am : Address) : exit :=



Ms_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                         Am, path(E(As, R, Ks), {}, {}, {}, {}),
                                    r1);
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                        Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
Mi_out ! Sincronismo;

Mm_in ! Sincronismo;

Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                        Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);

Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                         Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mc_out ! Sincronismo;

Mi_in ! Sincronismo;
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_out ! Sincronismo;

Mm_in ! Sincronismo;

Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                         path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mc_out ! Sincronismo;

Mi_in ! sincronismo;
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
```

```
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_out ! Sincronismo;

Mt_in ! Sincronismo;

Mt_in ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);

Mt_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
                         pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                                 Ks)), r1);
Mi_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
                        pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                              Ks)), r1);
Mi_out ! Sincronismo;

Mm_in ! Sincronismo;

Mm_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
                        pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                        Ks)), r1);

Mm_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Ab,
                         pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                         Ks)), r1);

Ms_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Ab,
                        pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                                Ks)), r1);

exit



endproc


endproc

(**************************************************************)
(* Caso em q nao existe misturadores no alcance de um misturador
   e ele nao eh vizinho do destino *)

process Busca_Rota_c3[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui,
                      Pi, Qi, Ut, Pt, Qt ]: exit :=

        hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out, Mi_in,
             Mi_out, Mt_in, Mt_out in
```

```
        (
                Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks, R,
                                          As, At, Ab, Ai, Ac, Am)
                |||
                Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km, Ks,
                                        R, As, At, Ab, Ai, Ac, Am)
                |||
                Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km, Ks, R, As,
                                                At, Ab, Ai, Ac, Am)
                |||
                Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km, Ks, R,
                                           As, At, Ab, Ai, Ac, Am)

)

                |[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out, Mi_in,
                  Mi_out, Mt_in, Mt_out]|
(
                Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks, R,
                                            As, At, Ab, Ai, Ac, Am)
        )


        where

Process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks :
                       Public_key, R : Random_number, As, At, Ab,
                                 Ai, Ac, Am : Address) : exit :=

                Us ! Iniciar_Route_Discovery;
                Ms_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                         Am, path(E(As, R, Ks), {}, {}, {}, {}),
                                     r1);
                Ms_in ! Link_Error;
Us ! Link_Error;
                exit

        endproc

      process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km, Ks
                         : Public_key, R : Random_number, As, At,
                              Ab, Ai, Ac, Am : Address) : exit :=


                Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                       Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
```

```
            Pm ! Consulta_Tabela;
            Pm ! Nao_conhece_T;

            Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                     Ab, path(E(As, R, Ks), Am, {}, {}, {})
                     , r1);

            Mm_in ! Sincronismo;
            Mm_in ! Timeout;

            Mm_out! Link_Error;
            exit

        endproc

        process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt, Km, Ks :
                  Public_key, R : Random_number, As, At, Ab,
                  Ai, Ac, Am : Address) : exit :=

            Mc_in ! Sincronismo;
            Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                    Ab, path(E(As, R, Ks), Am, {}, {}, {}),
                    r1);
            Uc ! Descartar;
            Mc_out ! Sincronismo;

            exit
endproc
        process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km, Ks :
                    Public_key, R : Random_number, As, At, Ab,
                    Ai, Ac, Am : Address) : exit :=


            Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                    Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
            Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                       path(E(As, R, Ks), Am, {}, {}, {}), r1);
            Mi_out ! Sincronismo;

exit

endproc

        process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                 Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks :
                 Public_key, R : Random_number, As, At, Ab, Ai,
                 Ac, Am : Address) : exit :=
```

```
                Ms_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                         Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Am,
                        path(E(As, R, Ks), {}, {}, {}, {}), r1);
Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                         path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_out ! Sincronismo;
Mc_in ! Sincronismo;
Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                        path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mc_out ! Sincronismo;
Mm_in ! Sincronismo;
Mm_in ! Timeout;
Mm_out ! Link_Error;
Ms_in ! Link_Error;
exit
endproc

endproc

(*******************************************************)
(* Caso em que um misturador recebe pacotes duplicados  *)

process Busca_Rota_c4[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                      Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

        hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
             Mi_in, Mi_out, Mt_in, Mt_out in

        (
                Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km,
                       Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt,
                           Km, Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km, Ks,
                      R, As, At, Ab, Ai, Ac, Am)
                |||
                Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km,
                        Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Target[Ut, Pt, Qt, Mt_in, Mt_out](Kt, Km,
                       Ks, R, As, At, Ab, Ai, Ac, Am)
```

```
)

          |[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in,
            Mc_out, Mi_in, Mi_out, Mt_in, Mt_out]|

        (
                Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in,
                     Mc_out, Mi_in, Mi_out, Mt_in, Mt_out]
                    (Kt, Km, Ks, R, As, At, Ab, Ai, Ac, Am)
        )


        where

        process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km,
                       Ks : Public_key, R : Random_number, As,
                       At, Ab, Ai, Ac, Am : Address) : exit :=

                Us ! Iniciar_Route_Discovery;
                Ms_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                       Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
                Ms_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Ab,
                        pathR(E(At, R, Ks), Am, {}, {}, E(As, R,
                               Ks)), r1);
                Us ! RotaEncontrada;

                exit

        endproc

        process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt,
                           Km, Ks : Public_key, R :
                           Random_number, As, At, Ab, Ai,
                           Ac, Am : Address) : exit :=

                Mm_in !Sincronismo;

                Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                     Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);

                Pm ! Consulta_Tabela;
                Pm ! Nao_conhece_T;

                Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                      Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

                Mm_in ! Sincronismo;
```

```
                Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                      Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
Um ! Pacote_Duplicado;
                Um ! Descartar;

Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                      Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

                Pm ! Consulta_Tabela;
                Pm ! Conhece_T;

                Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                      Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

                Mm_in ! Sincronismo;

                Mm_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Am, pathR(E(At, R, Ks), Am, {}, {},
                                  E(As, R, Ks)), r1);
                Mm_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                         Ab, pathR(E(At, R, Ks), Am, {}, {},
                                   E(As, R, Ks)), r1);

                exit

        endproc

        process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt, Km,
                      Ks : Public_key, R : Random_number,
                      As, At, Ab, Ai, Ac, Am : Address)
                      : exit :=



                Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                     Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
                Uc ! Descartar;
                Mc_out ! Sincronismo;

                Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                      Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
                Uc ! Descartar;
                Mc_out ! Sincronismo;

                exit


        endproc
```

```
process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km,
                Ks : Public_key, R : Random_number,
                As, At, Ab, Ai, Ac, Am : Address) : exit :=



        Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
              Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
        Qi ! Route_Request(E(As, R, Kt), E(At, R, Km),
             Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
        Mi_out ! Sincronismo;

        Mi_in ! Sincronismo;
        Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
             Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Qi ! Route_Request(E(As, R, Kt), E(At, R, Km),
             Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Mi_out ! Sincronismo;


        Mi_in ! sincronismo;
        Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
              Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Qi ! Route_Request(E(As, R, Kt), E(At, R, Kt),
              Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Mi_out ! Sincronismo;



        Mi_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                Am, pathR(E(At, R, Ks), Am, {}, {},
                          E(As, R, Ks)), r1);
        Qi ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
             pathR(E(At, R, Ks), Am, {}, {}, E(As, R, Ks)), r1);
        Mi_out ! Sincronismo;

        exit


endproc



process Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt, Km, Ks
               : Public_key, R : Random_number, As, At,
               Ab, Ai, Ac, Am : Address) : exit :=
```

```
            Mt_in !Sincronismo;

            Mt_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                 Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

            Mt_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                  Am, pathR(E(At, R, Ks), Am, {}, {},
                  E(As, R, Ks)), r1);


         exit


   endproc

   process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks :
                Public_key, R : Random_number, As, At, Ab,
                Ai, Ac, Am : Address) : exit :=

         Ms_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
               Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
         Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
               Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
         Mi_out ! Sincronismo;

         Mm_in ! Sincronismo;

         Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
               Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);

         Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
               Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

         Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
               Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

         Mc_out ! Sincronismo;

         Mi_in ! Sincronismo;
         Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
              Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
         Mi_out ! Sincronismo;

         Mm_in ! Sincronismo;

Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                       Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
```

```
            Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                    Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

            Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                     Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

            Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                    Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

            Mc_out ! Sincronismo;

            Mi_in ! sincronismo;
            Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                 Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
            Mi_out ! Sincronismo;

            Mt_in ! Sincronismo;
                 Mt_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                   Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);

            Mt_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                    Am, pathR(E(At, R, Ks), Am, {}, {},
                              E(As, R, Ks)), r1);
            Mi_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                    Am, pathR(E(At, R, Ks), Am, {}, {},
                              E(As, R, Ks)), r1);
            Mi_out ! Sincronismo;

            Mm_in ! Sincronismo;

            Mm_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                    Am, pathR(E(At, R, Ks), Am, {}, {},
                              E(As, R, Ks)), r1);

            Mm_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                     Ab, pathR(E(At, R, Ks), Am, {}, {},
                               E(As, R, Ks)), r1);

            Ms_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                    Ab, pathR(E(At, R, Ks), Am, {}, {}, E(As, R, Ks)), r1);

            exit

endproc


endproc
```

```
(***************************************************)
(* Intruso tentando se passar por misturador *)

process Busca_Rota_c5[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                      Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

        hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
             Mi_in, Mi_out, Mt_in, Mt_out in

        (
                Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks,
                                   R, As, At, Ab, Ai, Ac, Am)
                |||
                Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt,
                        Km, Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km,
                            Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km,
                            Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Target[Ut, Pt, Qt, Mt_in, Mt_out](Kt, Km,
                             Ks, R, As, At, Ab, Ai, Ac, Am)
)

                |[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                  Mi_in, Mi_out, Mt_in, Mt_out]|

        (
                Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks,
                                       R, As, At, Ab, Ai, Ac, Am)
        )


        where

        process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km,
                       Ks : Public_key, R : Random_number, As,
                       At, Ab, Ai, Ac, Am : Address) : exit :=

                Us ! Iniciar_Route_Discovery;
                Ms_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                      Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
                Ms_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                       Ab, pathR(E(At, R, Ks), Ai, Am, {},
                                 E(As, R, Ks)), r1);
```

```
        Us ! RotaEncontrada;

        exit

endproc

process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km,
                   Ks : Public_key, R : Random_number,
                   As, At, Ab, Ai, Ac, Am : Address) :
                   exit :=

        Mm_in !Sincronismo;

        Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);

        Pm ! Consulta_Tabela;
        Pm ! Nao_conhece_T;

        Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                Ab, path(E(As, R, Ks), Am, Ai, {}, {}), r1);

        Pm ! Consulta_Tabela;
        Pm ! Conhece_T;

        Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                Ab, path(E(As, R, Ks), Am, Ai, {}, {}), r1);

        Mm_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                Am, pathR(E(At, R, Ks), Ai, Am, {},
                          E(As, R, Ks)), r1);

        Mm_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                 Ab, pathR(E(At, R, Ks), Ai, Am, {},
                           E(As, R, Ks)), r1);

        exit

endproc

process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt, Km, Ks :
              Public_key, R : Random_number, As, At, Ab,
              Ai, Ac, Am : Address) : exit :=
       Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
               Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
       Uc ! Descartar;
```

```
               Mc_out ! Sincronismo;
               exit

endproc


        process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km,
                        Ks : Public_key, R : Random_number, As,
                        At, Ab, Ai, Ac, Am : Address) : exit :=

Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                     Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                     path(E(As, R, Ks), Am, {}, {}, {}), r1);
Ui ! Descobri_destino_no_Target;
Mi_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                       Ab, path(E(As, R, Ks), Am, Ai, {}, {}), r1);
                Mi_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Ai,
                        pathR(E(At, R, Ks), Ai, Am, {}, E(As, R,
                               Ks)), r1);
                Qi ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Ai,
                     pathR(E(At, R, Ks), Ai, Am, {}, E(As, R,
                             Ks)), r1);
Mi_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                         Am, pathR(E(At, R, Ks), Ai, Am, {},
                                   E(As, R, Ks)), r1);
exit
endproc


        process Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt, Km,
                       Ks : Public_key, R : Random_number, As,
                       At, Ab, Ai, Ac, Am : Address) : exit :=

Mt_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                      Ab, path(E(As, R, Ks), Am, Ai, {}, {}), r1);
Mt_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Ai, pathR(E(At, R, Ks), Ai, Am, {},
                                  E(As, R, Ks)), r1);
exit
endproc


        process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks :
                     Public_key, R : Random_number, As, At, Ab,
                     Ai, Ac, Am : Address) : exit :=
```

```
Ms_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                       Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
Mc_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                       Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
Mc_out ! Sincronismo;
Mm_in ! Sincronismo;
Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                       Am, path(E(As, R, Ks), {}, {}, {}, {}), r1);
Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                        Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                        Ab, path(E(As, R, Ks), Am, {}, {}, {}), r1);
Mi_out ! Route_Request(E(As, R, Kt), E(At, R, Km),
                        Ab, path(E(As, R, Ks), Am, Ai, {}, {}), r1);
Mm_in ! Route_Request(E(As, R, Kt), E(At, R, Km),
                         Ab, path(E(As, R, Ks), Am, Ai, {}, {}), r1);
Mm_out ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                         Ab, path(E(As, R, Ks), Am, Ai, {}, {}), r1);
Mt_in ! Route_Request(E(As, R, Kt), E(At, R, Kt),
                        Ab, path(E(As, R, Ks), Am, Ai, {}, {}), r1);
Mt_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Ai, pathR(E(At, R, Ks), Ai, Am, {},
                                  E(As, R, Ks)), r1);
Mi_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Ai, pathR(E(At, R, Ks), Ai, Am, {},
                        E(As, R, Ks)), r1);
Mi_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                         Am, pathR(E(At, R, Ks), Ai, Am, {},
                                   E(As, R, Ks)), r1);
Mm_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Am, pathR(E(At, R, Ks), Ai, Am, {},
                                  E(As, R, Ks)), r1);
Mm_out ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                         Ab, pathR(E(At, R, Ks), Ai, Am, {},
                         E(As, R, Ks)), r1);
Ms_in ! Route_Reply(E(At, R, Ks), E(As, R, Ks),
                        Ab, pathR(E(At, R, Ks), Ai, Am, {},
                        E(As, R, Ks)), r1);
exit
endproc

endproc


(****************************************************)
(* Caso em que so existe um misturador entre fonte e destino *)
process TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
            Ut, Pt, Qt ]: exit :=
```

```
hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
             Mi_in, Mi_out, Mt_in, Mt_out in

(
Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km,
                            Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km,
                                Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km, Ks,
                                  R, As, At, Ab, Ai, Ac, Am)
|||
Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km,
                              Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt, Km, Ks,
                                   R, As, At, Ab, Ai, Ac, Am)
)

|[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                  Mi_in, Mi_out, Mt_in, Mt_out]|

(
Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks, R,
                                          As, At, Ab, Ai, Ac, Am)
)


where

process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks
                       : Public_key, R : Random_number, As,
                       At, Ab, Ai, Ac, Am : Address) : exit :=

Us ! Iniciar_Transmissao;
Ms_out ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Ab, Ed(msg, R, Kt));
Ms_in ! ack;
Ms_out ! Sincronismo;

exit

endproc

process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt,
```

```
                           Km, Ks : Public_key, R :
                           Random_number, As, At, Ab, Ai,
                           Ac, Am : Address) : exit :=


Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                        Ab, Ed(msg, R, Kt));
Mm_out ! ack;
Mm_in ! Sincronismo;
Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                         Ab, Ed(msg, R, Kt));
Mm_in ! ack;




exit

endproc

process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt, Km,
                      Ks : Public_key, R : Random_number,
                      As, At, Ab, Ai, Ac, Am :
                      Address) : exit :=

Mc_in ! Sincronismo;
Mc_in ! ack;
Uc ! Descartar;
Mc_out ! Sincronismo;

exit

endproc

        process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km,
                        Ks : Public_key, R : Random_number,
                        As, At, Ab, Ai, Ac, Am : Address) : exit :=
  Mi_in ! Sincronismo;
Mi_in ! ack;
Qi ! ack;
Mi_out ! Sincronismo;
exit

endproc


process Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt,
                       Km, Ks : Public_key, R :
```

```
                       Random_number, As, At, Ab, Ai,
                       Ac, Am : Address) : exit :=

Mt_in ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                        Ab, Ed(msg, R, Kt));
Ut ! msgRecebida;
Mt_out ! ack;


exit

endproc

process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks :
                     Public_key, R : Random_number, As, At, Ab,
                     Ai, Ac, Am : Address) : exit :=



Ms_out ! Pkt_d(E(As, R, Kt), E(At, R, Km), Ab,
                         Ed(msg, R, Kt));
Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km), Ab,
                        Ed(msg, R, Kt));
Mm_out ! ack;
Ms_in ! ack;
Ms_out ! Sincronismo;
Mc_in ! Sincronismo;
Mc_in ! ack;
Mc_out ! Sincronismo;
Mi_in ! Sincronismo;
Mi_in ! ack;
Mi_out ! Sincronismo;
Mm_in ! Sincronismo;
Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                         Ab, Ed(msg, R, Kt));
Mt_in ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                         Ab, Ed(msg, R, Kt));
Mt_out ! ack;
Mm_in ! ack;

exit

endproc
endproc
```

```
(***********************************************************)
(* Caso em que exite dois ou mais misturadores
   entre fonte e destino *)
process TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
            Qi, Ut, Pt, Qt ]: exit :=

hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
             Mi_in, Mi_out, Mt_in, Mt_out in

(
Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt,
                      Km, Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt,
                       Km, Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km,
                           Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km,
                            Ks, R, As, At, Ab, Ai, Ac, Am)
|||
Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt, Km,
                            Ks, R, As, At, Ab, Ai, Ac, Am)
)

|[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                  Mi_in, Mi_out, Mt_in, Mt_out]|

(
Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks,
                                      R, As, At, Ab, Ai, Ac, Am)
)


where

process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks
                       : Public_key, R : Random_number, As, At,
                       Ab, Ai, Ac, Am : Address) : exit :=

Us ! Iniciar_Transmissao;
Ms_out ! Pkt_d(E(As, R, Kt), E(At, R, Km), Ab,
                               Ed(msg, R, Kt));
Ms_in ! ack;
Ms_out ! Sincronismo;
```

```
exit

endproc

process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt, Km,
                           Ks : Public_key, R : Random_number,
                           As, At, Ab, Ai, Ac, Am : Address)
                           : exit :=



Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                        Ab, Ed(msg, R, Kt));
Mm_out ! ack;
Mm_in ! Sincronismo;

Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Km), Am,
                               Ed(msg, R, Kt));
Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km), Am,
                              Ed(msg, R, Kt));
Mm_out ! ack;
Mm_in ! ack;

Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Kt), Ab,
                               Ed(msg, R, Kt));
Mm_in ! ack;


exit

endproc

process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt, Km, Ks
                      : Public_key, R : Random_number, As, At,
                       Ab, Ai, Ac, Am : Address) : exit :=

Mc_in ! Sincronismo;
Mc_in ! ack;
Uc ! Descartar;
Mc_out ! Sincronismo;

exit

endproc

        process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km,
                        Ks : Public_key, R : Random_number, As,
                        At, Ab, Ai, Ac, Am : Address) : exit :=
                Mi_in ! Sincronismo;
```

```
                Mi_in ! ack;
                Qi ! ack;
                Mi_out ! Sincronismo;
                exit

        endproc

process Target[Ut, Pt, Qt, Mt_in, Mt_out] (Kt, Km,
                       Ks : Public_key, R : Random_number,
                       As, At, Ab, Ai, Ac, Am : Address)
                       : exit :=

Mt_in ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                        Ab, Ed(msg, R, Kt));
Ut ! msgRecebida;
Mt_out ! ack;

exit

endproc

process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in,
                     Mc_out, Mi_in, Mi_out, Mt_in, Mt_out]
                    (Kt, Km, Ks : Public_key, R : Random_number,
                     As, At, Ab, Ai, Ac, Am : Address) : exit :=



Ms_out ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Ab, Ed(msg, R, Kt));
Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                        Ab, Ed(msg, R, Kt));
Mm_out ! ack;
Ms_in ! ack;
Ms_out ! Sincronismo;
Mc_in ! Sincronismo;
Mc_in ! ack;
Mc_out ! Sincronismo;
Mi_in ! Sincronismo;
                Mi_in ! ack;
                Mi_out ! Sincronismo;
Mm_in ! Sincronismo;

Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Am, Ed(msg, R, Kt));
Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Am, Ed(msg, R, Kt));
Mm_out ! ack;
Mm_in ! ack;
```

```
Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                         Ab, Ed(msg, R, Kt));
Mt_in ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                        Ab, Ed(msg, R, Kt));
Mt_out ! ack;
Mm_in ! ack;

exit

endproc
endproc

(*****************************************************************)
(* Quebra de link quando existe somente um misturador
   entre fonte e destino *)

process Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc,
                       Qc, Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

        hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
             Mi_in, Mi_out, Mt_in, Mt_out in

        (
                Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt,
                     Km, Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt,
                       Km, Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km,
                           Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km,
                            Ks, R, As, At, Ab, Ai, Ac, Am)

         )

                |[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in,
                  Mc_out, Mi_in, Mi_out, Mt_in, Mt_out]|

        (
                Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in,
                     Mc_out, Mi_in, Mi_out, Mt_in, Mt_out]
                   (Kt, Km, Ks, R, As, At, Ab, Ai, Ac, Am)
        )
```

```
        where

        process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km,
                       Ks : Public_key, R : Random_number,
                       As, At, Ab, Ai, Ac, Am : Address)
                       : exit :=



                Us ! Iniciar_Transmissao;
                Ms_out ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Ab, Ed(msg, R, Kt));
                Ms_in ! ack;
                Ms_out ! Sincronismo;

Ms_in ! Sincronismo;
Ms_in ! Link_Error;
Us ! Link_Error;

                exit

        endproc

        process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt,
                           Km, Ks : Public_key, R : Random_number,
                           As, At, Ab, Ai, Ac, Am : Address)
                           : exit :=



                Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                        Ab, Ed(msg, R, Kt));
                Mm_out ! ack;
                Mm_in ! Sincronismo;

                Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                         Ab, Ed(msg, R, Kt));
Mm_in ! Timeout;

Mm_out ! Link_Error;

         exit

endproc

        process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt,
                      Km, Ks : Public_key, R :
                      Random_number, As, At, Ab, Ai,
                      Ac, Am : Address) : exit :=
```

```
                Mc_in ! Sincronismo;
                Mc_in ! ack;
                Uc ! Descartar;
                Mc_out ! Sincronismo;

Mc_in ! Link_Error;
                Uc ! Descartar;
                Mc_out ! Sincronismo;

                exit

        endproc

        process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km,
                        Ks : Public_key, R : Random_number,
                        As, At, Ab, Ai, Ac, Am : Address)
                        : exit :=
                Mi_in ! Sincronismo;
                Mi_in ! ack;
                Qi ! ack;
                Mi_out ! Sincronismo;

Mi_in ! Sincronismo;
                Mi_in ! Link_Error;
                Qi ! Link_Error;
                Mi_out ! Sincronismo;

                exit

        endproc



        process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks :
                     Public_key, R : Random_number, As, At, Ab,
                     Ai, Ac, Am : Address) : exit :=



                Ms_out ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Ab, Ed(msg, R, Kt));
                Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                        Ab, Ed(msg, R, Kt));
                Mm_out ! ack;
                Ms_in ! ack;
                Ms_out ! Sincronismo;
Mc_in ! Sincronismo;
Mc_in ! ack;
Mc_out ! Sincronismo;
```

```
Mi_in ! Sincronismo;
                Mi_in ! ack;
                Mi_out ! Sincronismo;

                Mm_in ! Sincronismo;
                Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                         Ab, Ed(msg, R, Kt));
Mm_in ! Timeout;
Mm_out ! Link_Error;
Mc_in ! Link_Error;
                Mc_out ! Sincronismo;
                Mi_in ! Sincronismo;
                Mi_in ! Link_Error;
                Mi_out ! Sincronismo;
                Ms_in ! Sincronismo;
Ms_in ! Link_Error;



                exit

        endproc
endproc


(*************************************************************)
(* Caso em que existe dois ou mais misturadores entre
   fonte e destino *)

process Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui,
                       Pi, Qi, Ut, Pt, Qt ]: exit :=

        hide Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
             Mi_in, Mi_out, Mt_in, Mt_out in

        (
                Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt,
                      Km, Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Misturador[Um, Pm, Qm, Mm_in, Mm_out]
                    (Kt, Km, Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Comum[Uc, Pc, Qc, Mc_in, Mc_out](Kt, Km,
                            Ks, R, As, At, Ab, Ai, Ac, Am)
                |||
                Intruso[Ui, Pi, Qi, Mi_in, Mi_out](Kt, Km,
                             Ks, R, As, At, Ab, Ai, Ac, Am)
```

```
        )

                |[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                  Mi_in, Mi_out, Mt_in, Mt_out]|

        (
                Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km, Ks, R,
                     As, At, Ab, Ai, Ac, Am)
        )

where

        process Source[Us, Ps, Qs, Ms_in, Ms_out] (Kt, Km, Ks :
                       Public_key, R : Random_number, As,
                       At, Ab, Ai, Ac, Am : Address) : exit :=


                Us ! Iniciar_Transmissao;
                Ms_out ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Ab, Ed(msg, R, Kt));
                Ms_in ! ack;
                Ms_out ! Sincronismo;

Ms_in ! Sincronismo;
                Ms_in ! Link_Error;
                Us ! Link_Error;

                exit

        endproc

        process Misturador[Um, Pm, Qm, Mm_in, Mm_out] (Kt,
                           Km, Ks : Public_key, R :
                           Random_number, As, At, Ab, Ai,
                           Ac, Am : Address) : exit :=


                Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                        Ab, Ed(msg, R, Kt));
                Mm_out ! ack;
                Mm_in ! Sincronismo;

                Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Am, Ed(msg, R, Kt));
Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                        Am, Ed(msg, R, Kt));
Mm_out ! ack;
```

```
Mm_in ! ack;
Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                         Ab, Ed(msg, R, Kt));
                Mm_in ! Timeout;

                Mm_out ! Link_Error;

                exit

endproc

        process Comum[Uc, Pc, Qc, Mc_in, Mc_out] (Kt, Km,
                      Ks : Public_key, R : Random_number,
                      As, At, Ab, Ai, Ac, Am : Address) : exit :=

                Mc_in ! Sincronismo;
                Mc_in ! ack;
                Uc ! Descartar;
                Mc_out ! Sincronismo;

Mc_in ! Link_Error;
Uc ! Descartar;
                Mc_out ! Sincronismo;

                exit

        endproc

        process Intruso[Ui, Pi, Qi, Mi_in, Mi_out] (Kt, Km,
                        Ks : Public_key, R : Random_number,
                        As, At, Ab, Ai, Ac, Am : Address) : exit :=
                Mi_in ! Sincronismo;
                Mi_in ! ack;
                Qi ! ack;
                Mi_out ! Sincronismo;

Mi_in ! Sincronismo;
Mi_in ! Link_Error;
Qi ! Link_Error;
Mi_out ! Sincronismo;

                exit

        endproc

        process Meio[Ms_in, Ms_out, Mm_in, Mm_out, Mc_in, Mc_out,
                     Mi_in, Mi_out, Mt_in, Mt_out](Kt, Km,
                     Ks : Public_key, R : Random_number, As,
```

```
                     At, Ab, Ai, Ac, Am : Address) : exit :=


                Ms_out ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Ab, Ed(msg, R, Kt));
                Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Ab, Ed(msg, R, Kt));
                Mm_out ! ack;
                Ms_in ! ack;
                Ms_out ! Sincronismo;
Mc_in ! Sincronismo;
                Mc_in ! ack;
                Mc_out ! Sincronismo;
                Mi_in ! Sincronismo;
                Mi_in ! ack;
                Mi_out ! Sincronismo;
                Mm_in ! Sincronismo;
Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Am, Ed(msg, R, Kt));
                Mm_in ! Pkt_d(E(As, R, Kt), E(At, R, Km),
                         Am, Ed(msg, R, Kt));
                Mm_out ! ack;
                Mm_in ! ack;
                Mm_out ! Pkt_d(E(As, R, Kt), E(At, R, Kt),
                         Ab, Ed(msg, R, Kt));
                Mm_in ! Timeout;

                Mm_out ! Link_Error;
Mc_in ! Link_Error;
Mc_out ! Sincronismo;
Mi_in ! Sincronismo;
Mi_in ! Link_Error;
Mi_out ! Sincronismo;
        Ms_in ! Sincronismo;
Ms_in ! Link_Error;

exit

endproc

endproc

endproc
endspec
```

Apêndice C

ANDSR_serv.lotos

O CÓDIGO apresentado aqui é a especificação do serviço oferecido pelo protocolo ANDSR em LOTOS.

```
Specification ANDSR11_SERVICE [Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                               Ut, Pt, Qt ] : noexit


library ANDSR_LIB10 endlib

behaviour

        SERVICE [Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut, Pt, Qt]

where

process SERVICE [Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut, Pt, Qt] : noexit :=

(
         Busca_Misturador_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                             Ut, Pt, Qt ]

         >>(
                 TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut, Pt, Qt]
                 []
                 TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut, Pt, Qt]
                 []
                 Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                                Ut, Pt, Qt]
                 []
                 Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
```

```
                        Ut, Pt, Qt ]
             []
             (TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut, Pt, Qt]
              >> Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                            Qi, Ut, Pt, Qt ]
             )
             []
             (TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut, Pt, Qt]
              >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                            Qi, Ut, Pt, Qt ]
             )
         ) >> SERVICE[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
                   Pt, Qt]

      )
      []
     (

Busca_Misturador_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
                     Pt, Qt ]
      >>(
            (Busca_Rota_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                       Ut, Pt, Qt ]
            >> TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
                   Pt, Qt ]
            )

      []    (Busca_Rota_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                       Ut, Pt, Qt ]
            >> TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
                   Pt, Qt ]
            )

      []
            (Busca_Rota_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                       Ut, Pt, Qt ]
            >> TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
                   Pt, Qt ]
            >> Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                          Qi, Ut, Pt, Qt ]
            )

      []

            (Busca_Rota_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                       Ut, Pt, Qt ]
             >> Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                           Qi, Ut, Pt, Qt ]
```

```
                )

        []

                (Busca_Rota_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                               Ut, Pt, Qt ]
                >> TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
                       Pt, Qt ]
                >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                                  Qi, Ut, Pt, Qt ]
                )

        []

                (Busca_Rota_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                               Ut, Pt, Qt ]
                 >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                                   Qi, Ut, Pt, Qt ]
                )

[]

(Busca_Rota_c3[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                               Ut, Pt, Qt ]
)


        []      (Busca_Rota_c4[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                               Ut, Pt, Qt ]
                >> TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
                       Pt, Qt ]
                )


[]
                (Busca_Rota_c4[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                               Ut, Pt, Qt ]
                >> TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
                       Pt, Qt ]
                >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                                  Qi, Ut, Pt, Qt ]
                )

        []

                (Busca_Rota_c4[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                               Qi, Ut, Pt, Qt ]
                 >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
```

```
                              Qi, Ut, Pt, Qt ]
                )

        []

                (Busca_Rota_c5[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui,
                              Pi, Qi, Ut, Pt, Qt ]
                 >> Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui,
                                  Pi, Qi, Ut, Pt, Qt ]
                )

          ) >> SERVICE[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi,
                      Ut, Pt, Qt ]
)
[]
(
 Busca_Misturador_c3[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                            Qi, Ut, Pt, Qt ]
         >>SERVICE[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi, Qi, Ut,
                  Pt, Qt ]
)

        where



        process Busca_misturador_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                                    Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

                Us ! Rota_para_Destino;
                Ps ! Consulta_Tabela;
                Ps ! Nao_existe_rota;
                Us ! Nao_existe_rota;
                Us ! Buscar_Misturador;
                Ut ! Descartar;
                Uc ! Descartar;
                Qi ! Mix_Request(As, Ab, Seq);
                Ui ! Descobri_S_quer_Rota;
                Ut ! Descartar;
                Uc ! Descartar;
                Qi ! Mix_Replay(Am, As, Seq);
                Us ! MixEncontrado;
                exit
        endproc

        process Busca_misturador_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                                    Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=
```

```
                Us ! Rota_para_Destino;
                Ps ! Consulta_Tabela;
Ps ! Existe_rota;
      Us ! Existe_rota;
exit
endproc

process Busca_misturador_c3[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                                    Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

                Us ! Rota_para_Destino;
                Ps ! Consulta_Tabela;
Ps ! Nao_existe_rota;
                Us ! Nao_existe_rota;
                Us ! Buscar_Misturador;
Us ! Timeout;
exit

endproc

process Busca_Rota_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

                Us ! Iniciar_Route_Discovery;
                Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Am,
                                   path(E(As, R, Ks), {}, {}, {}, {}), r1);
                Pm ! Consulta_Tabela;
                Pm ! Conhece_T;
                Uc ! Descartar;
                Qi ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                                   path(E(As, R, Ks), Am, {}, {}, {}), r1);
                Qi ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
                                 pathR(E(At, R, Ks), Am, {}, {},
                                 E(As, R, Ks)), r1);
                Us ! RotaEncontrada;
                exit
        endproc

        process Busca_Rota_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

                Us ! Iniciar_Route_Discovery;
                Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Am,
                                   path(E(As, R, Ks), {}, {}, {}, {}), r1);
                Pm ! Consulta_Tabela;
                Pm ! Nao_conhece_T;
                Uc ! Descartar;
                Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
```

```
                  path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Pm ! Consulta_Tabela;
        Pm ! Conhece_T;
        Uc ! Descartar;
        Qi ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                  path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Qi ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
                pathR(E(At, R, Ks), Am, {}, {},
                E(As, R, Ks)), r1);
        Us ! RotaEncontrada;
        exit
endproc

process Busca_Rota_c3[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                      Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

        Us ! Iniciar_Route_Discovery;
        Pm ! Consulta_Tabela;
        Pm ! Nao_conhece_T;
        Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                  path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Uc ! Descartar;
        Us ! Link_Error;
        exit
endproc

process Busca_Rota_c4[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                      Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

        Us ! Iniciar_Route_Discovery;
        Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Am,
                  path(E(As, R, Ks), {}, {}, {}, {}), r1);
        Pm ! Consulta_Tabela;
        Pm ! Nao_conhece_T;
        Uc ! Descartar;
        Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                  path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Um ! Pacote_Duplicado;
        Um ! Descartar;
        Pm ! Consulta_Tabela;
        Pm ! Conhece_T;
        Uc ! Descartar;
        Qi ! Route_Request(E(As, R, Kt), E(At, R, Kt), Ab,
                  path(E(As, R, Ks), Am, {}, {}, {}), r1);
        Qi ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Am,
                pathR(E(At, R, Ks), Am, {}, {},
                E(As, R, Ks)), r1);
        Us ! RotaEncontrada;
```

```
                exit
        endproc

        process Busca_Rota_c5[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                              Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

                Us ! Iniciar_Route_Discovery;
Uc ! Descartar;
                Pm ! Consulta_Tabela;
                Pm ! Nao_conhece_T;
                Qi ! Route_Request(E(As, R, Kt), E(At, R, Km), Ab,
                                   path(E(As, R, Ks), Am, {}, {}, {}), r1);
                Ui ! Descobri_destino_no_Target;
Pm ! Consulta_Tabela;
                Pm ! Conhece_T;
                Qi ! Route_Reply(E(At, R, Ks), E(As, R, Ks), Ai,
                                 pathR(E(At, R, Ks), Ai, Am, {},
                                 E(As, R, Ks)), r1);
                Us ! RotaEncontrada;
                exit
        endproc


process TD1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui, Pi,
                    Qi, Ut, Pt, Qt ]: exit :=

                Us ! Iniciar_Transmissao;
Uc ! Descartar;
Qi ! ack;
                Ut ! msgRecebida;

                exit
        endproc

        process TD2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc, Ui,
                    Pi, Qi, Ut, Pt, Qt ]: exit :=

                Us ! Iniciar_Transmissao;
                Uc ! Descartar;
Qi ! ack;
Ut ! msgRecebida;

                exit
        endproc

process Quebra_link_c1[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                               Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=
```

```
                Us ! Iniciar_Transmissao;
Uc ! Descartar;
Qi ! ack;
Uc ! Descartar;
Qi ! Link_Error;
                Us ! Link_Error;

                exit
        endproc

process Quebra_link_c2[Us, Ps, Qs, Um, Pm, Qm, Uc, Pc, Qc,
                               Ui, Pi, Qi, Ut, Pt, Qt ]: exit :=

                Us ! Iniciar_Transmissao;
Uc ! Descartar;
Qi ! ack;
                Uc ! Descartar;
                Qi ! Link_Error;
                Us ! Link_Error;

                exit
        endproc



endproc
endspec
```