Transmissão Multicast Confiável: Aplicação na Ferramenta Tangram Whiteboard e Experimentos na Internet

por

Jorge Allyson Azevedo

UFRJ

Tese submetida para a obtenção do título de

Mestre em Informática

ao Programa de Pós-Graduação do Instituto de Matemática e

Núcleo de Computação Eletrônica da UFRJ

por

Jorge Allyson Azevedo

Dezembro 2002

TRANSMISSÃO MULTICAST CONFIÁVEL: APLICAÇÃO NA FERRAMENTA TANGRAM WHITEBOARD E EXPERIMENTOS NA INTERNET

Jorge Allyson Azevedo

TESE SUBMETIDA AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO DO DEPARTAMENTO DE CIÊNCIAS DA COMPUTAÇÃO, INSTITUTO DE MATEMÁTICA E NÚCLEO DE COMPUTAÇÃO ELETRÔNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM INFORMÁTICA.

Aprovada por:

Prof. Edmundo Albuquerque de Souza e Silva, Ph.D.

Prof. Paulo Henrique de Aguiar Rodrigues, Ph.D.

Prof. Otto Carlos Muniz Bandeira Duarte, Dr.Ing.

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 2002

AZEVEDO, JORGE A.

Transmissão Multicast Confiável: Aplicação na Ferramenta Tangram Whiteboard e Experimentos na Internet [Rio de Janeiro] 2002

XIX, 96 p. 29,7 cm (IM/NCE UFRJ, M.Sc., Informática, 2002)

Tese - Universidade Federal do Rio de Janeiro, DCC/IM/NCE

1. Multicast Confiável
2. Whiteboard
3. Redes Multimídia
4. Ferramenta Tangram-II

I. IM/NCE UFRJ II. Título (Série)

Dedico este trabalho aos meus pais

João Batista e Maria Helena

Agradecimentos

Gostaria de agradecer, primeiramente, à minha família, que apesar da distância sempre me ajudou da melhor forma possível, com palavras ou com atitudes que no fim sempre me impulsionaram para o cumprimento de meus objetivos. Sem o apoio dos meus pais, João Batista e Maria Helena, e dos meus irmãos Simone, João Paulo e Suellen, a minha tarefa teria sido muito mais difícil. Agradeço também aos meus avós, que sempre torcem por mim e souberam me animar nos momentos mais complicados. Não posso esquecer também da minha tia Eva, cujo incentivo foi essencial em diversas situações.

Por estar longe da minha família, meus amigos tiveram papel muito importante durante o desenvolvimento deste trabalho. Por isso, tenho muito a agradecer a eles, principalmente ao Léo, pois sem a sua ajuda tudo teria sido muito mais difícil. No dia a dia, sempre presentes, o pessoal do LAND me ajudou muito e quero, sinceramente, agradecer todo o apoio que recebi de GD, Flávio, Drika, Kelvin, Magnos, Ana Paula, Fernando, Bernardo, Bruno e Guto. Além disso, devo agradecer especialmente à Milena e ao Daniel Sadoc, pois nosso trabalho em conjunto também foi uma experiência gratificante e ajuda do Daniel no início da implementação foi essencial. Não posso deixar de agradecer, com muito carinho, à Carolina, que desde que chegou ao LAND, foi muito mais do que uma secretária e sem a sua boa vontade, muitos problemas não teriam sido resolvidos.

Aos professores Edmundo e Rosa, o meu agradecimento e reconhecimento pelo trabalho dedicado que exercem, mantendo um grupo como o LAND e promovendo o desenvolvimento de tantas pessoas.

Resumo da Tese apresentada ao DCC/IM/NCE - UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

Transmissão Multicast Confiável: Aplicação na Ferramenta Tangram Whiteboard e Experimentos na Internet

Jorge Allyson Azevedo

Dezembro/2002

Orientador: Edmundo Albuquerque de Souza e Silva
Departamento: Ciência da Computação

A crescente popularização da Internet está abrindo espaço para novas aplicações. Dentre estas aplicações se destacam as ferramentas para trabalho cooperativo e ensino à distância. Essas ferramentas têm como requisito a interação, de forma rápida e transparente, entre os usuários. Uma forma de utilizar melhor os recursos da rede é realizar a transmissão através de IP Multicast. Entretanto, o IP Multicast não oferece garantia de entrega dos pacotes e, portanto, aplicações que necessitam de confiabilidade na transmissão dos dados devem implementar mecanismos para obtê-la. Este trabalho apresenta o desenvolvimento e a implementação de uma biblioteca de funções para transmissão multicast confiável denominada RML (*Reliable Multicast Library*), utilizada para completar o desenvolvimento do Tangram Whiteboard. O Tangram Whiteboard tem como principais vantagens o alto grau de interatividade e o mecanismo que mantém a consistência global, que é uma característica exclusiva desse *whiteboard*. Foram realizados experimentos na Internet que , em conjunto com um modelo do protocolo implementado mostraram a eficiência do algoritmo de transmissão confiável em função dos parâmetros do protocolo.

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

Reliable Multicast Transmission: The Tangram Whiteboard and Internet Experiments

Jorge Allyson Azevedo

December/2002

Advisor: Edmundo Albuquerque de Souza e Silva
Department: Computer Science

A broad range of new applications have been developed with the increasing dissemination of the Internet. Application examples include cooperative work and distance learning tools. Among the requirements for these applications are low delay to transfer information among users and reliability. For those applications, the network utilization can be improved using IP Multicast transmission. However IP Multicast does not offer reliable communication so applications must implement this service. This work presents the design and implementation of RML (*Reliable Multicast Library*), a library that offers reliable multicast transmission functions to applications. The RML was used for the Tangram Whiteboard. The main features of the Tangram Whiteboard are the high-level interactivity and the mechanism that keeps global consistency among users. This mechanism is a unique feature of the Tangram Whiteboard in comparison with other similar tools. In addition, several measurements have been conducted in the Internet with four participants sites, three in Brazil and one in the USA. These experiments, together with a simulation model of the protocol, showed the efficiency of the reliable multicast transmission algorithm implemented as function of the protocol parameters.

Palavras-chave

1. Multicast Confiável
2. Whiteboard
3. Redes Multimídia
4. Ferramenta Tangram-II

Glossário

WAN: redes de longa distância ou geograficamente distribuídas (*Wide Area Network*);

LAN: rede local (*Local Area Network*);

ACK: confirmação positiva (*positive ACKnowledgment*);

NAK: confirmação negativa (*Negative AcKnowledgment*);

QoS: qualidade de serviço (*Qualite of Service*);

Sumário

Resumo **i**

Abstract **ii**

Glossário **iv**

1 Introdução **1**

1.1 Motivação e Objetivo 1

1.2 Contribuição 3

1.3 Roteiro 4

2 Transmissão multicast confiável **5**

2.1 Transmissão Multicast 5

2.1.1 Suporte à Transmissão Multicast Confiável: Rede versus Aplicação 8

2.2 Multicast confiável usando IP Multicast 10

2.2.1 Protocolos baseados no emissor 11

2.2.2 Protocolos baseados no receptor 12

2.2.3 Protocolos baseados em árvore 14

2.2.4 Protocolos baseados em anel 16

2.2.5 Análise da vazão máxima 16

2.3 Multicast confiável na aplicação 18

3 RML: uma biblioteca para transmissão multicast confiável 22

3.1 Introdução 22

3.2 Identificação de membros e armazenamento de dados 24

3.2.1 Identificação de membros 24

3.2.2 A estrutura para armazenamento de dados 25

Recebendo e armazenando dados 26

3.3 Gerenciamento de membros na sessão multicast 28

3.4 Detecção e recuperação de perdas 30

3.4.1 Detecção de perdas 30

3.4.2 Envio de NAKs 30

3.4.3 Retransmissão dos dados 33

A lista de NAKs 35

3.4.4 Supressão de NAKs e retransmissões 35

3.5 A Lista de Eventos 36

3.5.1 Temporizadores 38

4 O TANGRAM-II Whiteboard 40

4.1 O Ambiente de Modelagem TANGRAM-II 40

4.2 Principais características do TGWB 43

4.3 O TGIF 44

4.4 Consistência Global e Segmentação 46

4.4.1 Análise do problema 46

4.4.2 Relógios Lógicos 50

4.4.3 Algoritmo para execução dos comandos no TGWB 51

4.4.4 Segmentação 54

4.5 As camadas RML e IP Multicast 54

5 Resultados de Experimentos e o modelo da RML 56

5.1 Ambiente de testes 56

5.1.1 O Cenário 1 57

5.1.2 O Cenário 2 59

5.1.3 Configuração da RML 59

O mcastproxy 61

5.2 Resultados dos experimentos 61

5.2.1 Relação entre os parâmetros dos temporizadores 62

5.2.2 Resultados referentes ao Cenário 1 64

5.2.3 Resultados referentes ao Cenário 2 69

5.3 O modelo da RML 70

6 Conclusão 74

6.1 Trabalhos Futuros 75

Referências Bibliográficas 77

A Pré-requisitos para o uso da RML 81

A.1 Endereços IP Multicast 81

A.2 Configurando o Multicast no GNU/Linux 82

A.3 Compilando a RML 83

B Utilização e configuração da RML **85**

B.1 Funções disponíveis na RML 85

B.2 Opções de configuração disponíveis na RML 87

B.3 Registro de pacotes recebidos e enviados 90

C Código fonte do rmchat, uma aplicação baseada na RML **93**

Lista de Figuras

2.1 (a) Transmissão Unicast (b) Transmissão Multicast 6

2.2 Um diagrama sobre protocolos baseados no emissor 11

2.3 Um diagrama sobre protocolos baseados no receptor 12

2.4 Um diagrama sobre protocolos RINA 13

2.5 Um diagrama sobre protocolos baseados em árvore 14

2.6 Um diagrama sobre protocolos baseados em anel 16

2.7 Representação das conexões físicas e retardos entre os sistemas . . . 19

2.8 *Mesh* criada pelo protocolo Narada 20

2.9 Árvore gerada pelo protocolo Narada 20

3.1 Esquema do uso da RML 23

3.2 Exemplo de lista de CACHE 27

3.3 Tipos de pacotes na RML 31

3.4 Retirando informações do pacote NAK 32

3.5 Algoritmo para determinação das requisições de um NAK 33

3.6 Recebimento de retransmissões 34

3.7 Exemplo de lista de eventos 37

4.1 Ferramenta TANGRAM-II 41

4.2 Ferramenta TANGRAM-II - Ambiente de Modelagem 42

4.3 Camadas componentes do TGWB 43

4.4 Um exemplo de modelo descrito no TGIF 45

4.5 Exemplo de um objeto TANGRAM-II e seus atributos 46

4.6 Grafo de precedência de uma computação distribuída 48

4.7 Exemplo de inconsistência 49

4.8 Lista de comandos e exemplo de *Rollback and Recovery* 53

5.1 Configuração da RML usada nos testes 60

5.2 Ambiente de testes considerando o mcastproxy 62

5.3 Relação entre os parâmetros 63

5.4 NAKs (ou requisições) enviados pela fenix com C=4 65

5.5 Tempo de recuperação dos dados pela fenix com C=4 65

5.6 NAKs (ou requisições) enviados pela fenix com C=5 65

5.7 Tempo de recuperação dos dados pela fenix com C=5 65

5.8 NAKs (ou requisições) enviados pela fenix com C=6 65

5.9 Tempo de recuperação dos dados pela fenix com C=6 65

5.10 Comportamento da supressão de NAKs em relação à variação dos parâmetros 66

5.11 Média de NAKs recebidos pela ilha para C=4, C=5 e C=6 68

5.12 Média de retransmissões recebidas pela ilha para C=4, C=5 e C=6 . 68

5.13 NAKs enviados pela newworld em relação ao emissor abacate 69

5.14 Tempo de recuperação da newworld em relação ao emissor abacate . . 69

5.15 NAKs enviados pela abacate em relação ao emissor fenix 69

5.16 Tempo de recuperação da abacate em relação ao emissor fenix 69

5.17 NAKs enviados pela newworld em relação ao emissor ilha 70

5.18 Tempo de recuperação da newworld em relação ao emissor ilha 70

5.19 NAKs enviados pela fenix em relação ao emissor newworld 70

5.20 Tempo de recuperação da fenix em relação ao emissor newworld . . . 70

5.21 Representação gráfica do modelo da RML 71

5.22 Representação dos pacotes no modelo da RML 72

5.23 NAKs enviados pela fenix: Resultados do experimento real x Resultados da simulação 72

5.24 Tempo de recuperação da fenix: Resultados do experimento real x Resultados da simulação 72

B.1 Exemplo de arquivo de configuração da RML 88

B.2 Exemplo de arquivo de registro da RML 92

Lista de Tabelas

2.1 Vazão máxima dos protocolos para transmissão multicast confiável . . 17

3.1 Temporizadores aleatórios usados na RML 39

5.1 Descrição da máquinas utilizadas nos experimentos 57

5.2 Número de roteadores e RTT médio entre os *hosts* 57

5.3 Distribuição dos *slides* conforme a freqüência e tamanho 58

5.4 Exemplo de estimativa de retardo para os temporizadores na máquina ilha 61

5.5 NAKs enviados e tempo de recuperação em relação à mudança dos parâmetros na fenix 64

5.6 Supressão de NAKs com C=4 67

5.7 Supressão de NAKs com C=5 67

5.8 Supressão de NAKs com C=6 67

5.9 Supressão de Retransmissões com C=4 67

5.10 Supressão de Retransmissões com C=5 68

5.11 Supressão de Retransmissões com C=6 68

A.1 Endereços multicast reservados 81

Capítulo 1

Introdução

ESTE trabalho abrange o projeto e implementação de um protocolo para transmissão multicast confiável em forma de uma biblioteca de funções. Ao integrar essa biblioteca ao TGIF (*Tangram Graphical Interface Facility*), interface gráfica do ambiente de modelagem e análise TANGRAM-II, foi criado o Tangram-II Whiteboard (TGWB), uma ferramenta para trabalho em grupo.

Nas seções seguintes serão apresentados: (1.1) a motivação e objetivo, que introduzem o trabalho como um todo e apresentam um resumo dos protocolos para transmissão multicast confiável; (1.2) a contribuição, onde a biblioteca é brevemente descrita e o resultado alcançado é apresentado; e (1.3) o roteiro que descreve a dissertação sob um ponto de vista macro.

1.1 Motivação e Objetivo

A crescente popularização dos computadores e a sua conexão através de redes IP como a Internet, está abrindo espaço para novas aplicações. Entre estas aplicações, as ferramentas para trabalho cooperativo e ensino à distância se destacam. Essas ferramentas têm como requisito a interação, de forma mais rápida e transparente possível, entre grupos de usuários. São exemplos dessas aplicações as ferramentas para videoconferência, transmissão de voz, servidores de vídeo sob demanda e os

whiteboards.

Porém, mesmo com o aumento gradual da oferta de banda de transmissão, para uma grande parte dos usuários existe sempre uma necessidade de se economizar tal banda, pois a demanda das aplicações têm aumentado com a disponibilidade dos recursos da rede. Some-se a isso o crescimento do número de usuários que contribui para uma maior utilização destes recursos.

Há algum tempo, o LAND [1], Laboratório para Análise e Modelagem de Sistemas de Computação e Comunicação, vem desenvolvendo pesquisas que resultam em ferramentas com aplicações tanto no ensino a distância quanto no trabalho cooperativo. Dentre estas aplicações, destacam-se o Servidor Multimídia [36, 32] e o VivaVoz [16, 15]. Para complementar as facilidades oferecidas por essas aplicações, desenvolveu-se uma ferramenta que permite uma interação entre os usuários através do compartilhamento de desenhos e pequenos textos. Esta ferramenta permite o desenvolvimento de um ambiente integrado de troca de informações e, além disso, a realização de testes de mecanismos de redes de forma a prover a QoS exigida pelas aplicações. Ferramentas para modelagem já existiam, como parte do ambiente de modelagem do TANGRAM-II [8, 23]. Portanto, faltava apenas uma ferramenta que integrasse o ambiente de modelagem e permitisse, além de outras coisas, que um grupo de usuários realizasse, de forma cooperativa, a criação de um modelo.

Existem ferramentas para trabalho cooperativo que possuem algumas das características procuradas. Porém, algumas destas aplicações são dependentes de um servidor central, outras não fazem uma utilização inteligente da rede por usarem simplesmente várias conexões ponto a ponto entre os usuários. Por fim, ferramentas como o *whiteboard* descrito em [39] não oferecem a interatividade necessária para o desenvolvimento, em conjunto, de um modelo de sistema.

Para o whiteboard apresentado neste trabalho, escolheu-se como ferramenta gráfica o TGIF [10](*Tangram Graphical Interface Facility*), por possuir uma grande variedade de funções para desenho vetorial e, além disso, ser a interface gráfica para a criação de modelos no TANGRAM-II. O TGIF foi adaptado para executar comandos recebidos via rede. Esses comandos inclusive podem ser enviados usando

transmissão multi-destino, ou multicast. O uso do multicast permite uma melhor utilização dos recursos da rede. Apesar da grande vantagem da melhor utilização da rede, a transmissão multicast disponível atualmente, por usar o protocolo UDP, não garante a entrega dos pacotes, que é um requisito essencial em aplicações como *whiteboards*.

O principal objetivo deste trabalho foi, portanto, o desenvolvimento de uma ferramenta para trabalho cooperativo que permitisse um alto grau de interatividade entre os usuários, e que, ao mesmo tempo, fizesse uso racional dos recursos da rede através da transmissão multicast confiável. Foi necessária a implementação de um protocolo que garantisse, para a aplicação, a entrega dos pacotes ordenados e sem perdas.

1.2 Contribuição

Considerando o problema da transmissão multicast, foi desenvolvida uma biblioteca de funções, denominada RML (*Reliable Multicast Library*), que implementa, no nível de aplicação, mecanismos que garantem a entrega dos pacotes ordenados e sem perdas. Por ser uma biblioteca independente, a RML pode ser usada por outras aplicações que necessitem de transmissão multicast confiável.

Após a implementação da RML, essa biblioteca foi utilizada para completar a adaptação do TGIF, iniciada em [12], para que este se tornasse um *whiteboard* distribuído, o TGWB (TANGRAM-II Whiteboard). A RML foi utilizada para permitir a transmissão multicast confiável, pois em [12] já haviam sido propostas e implementadas rotinas para a segmentação dos dados e para a garantia da consistência global entre os membros de uma sessão. A consistência global é a garantia de que todos os membros, que participam de uma mesma sessão de trabalho, terão em suas interfaces locais uma mesma apresentação dos dados. Essa característica, aliada à grande variedade de funções, que aumentam o nível de interatividade entre os usuários, são as principais vantagens do TGWB sobre outros *whiteboards*.

Durante o desenvolvimento deste trabalho, foi criado também, em conjunto com

Milena Scanferla, aluna de mestrado da Universidade Federal Fluminense, um modelo do protocolo implementado na RML. A partir desse modelo foi possível obter várias medidas, que, ao longo dos experimentos com a biblioteca e o TGWB, contribuíram de forma eficaz para a resolução de alguns problemas.

1.3 Roteiro

O capítulo 2 apresenta algumas propostas de multicast confiável encontradas na literatura. O capítulo 3 discute os principais pontos da implementação da RML e do protocolo utilizado no desenvolvimento desta biblioteca. As adaptações feitas no TGIF para criação do TGWB, como a inclusão das rotinas para garantia da consistência global, são mostradas no capítulo 4. O modelo desenvolvido, as medidas obtidas e os resultados dos experimentos com a RML são mostrados no capítulo 5. Por fim, o capítulo 6 traz as conclusões e sugestões para trabalhos futuros.

Capítulo 2

Transmissão multicast confiável

NESTE capítulo são discutidas algumas formas para transmissão multicast e as principais propostas de protocolos para transmissão multicast confiável. Entre estes estão os protocolos baseados nos receptores, que são a base para o desenvolvimento da biblioteca multicast confiável descrita no capítulo 3.

2.1 Transmissão Multicast

Durante os últimos anos, ocorreu um grande aumento no uso de aplicações que oferecem suporte a atividades interativas entre grupos de usuários na Internet. São exemplos as aplicações para videoconferência, atualização de aplicativos, compartilhamento de arquivos e alguns jogos, além de ferramentas para trabalho em grupo conhecidas como ferramentas para CSCW (*Computer Supported Cooperative Work*). Uma característica importante de algumas dessas aplicações é a necessidade de enviar um mesmo conjunto de dados para um determinado grupo de receptores. A transmissão eficiente deste conjunto de dados é uma das características de maior influência no desempenho destas aplicações. De acordo com as necessidades específicas da aplicação, a transmissão pode ser feita de duas formas: transmissão ponto a ponto ou *unicast* e transmissão *multicast*.

A transmissão de dados usando unicast é feita enviando-se uma cópia do pacote

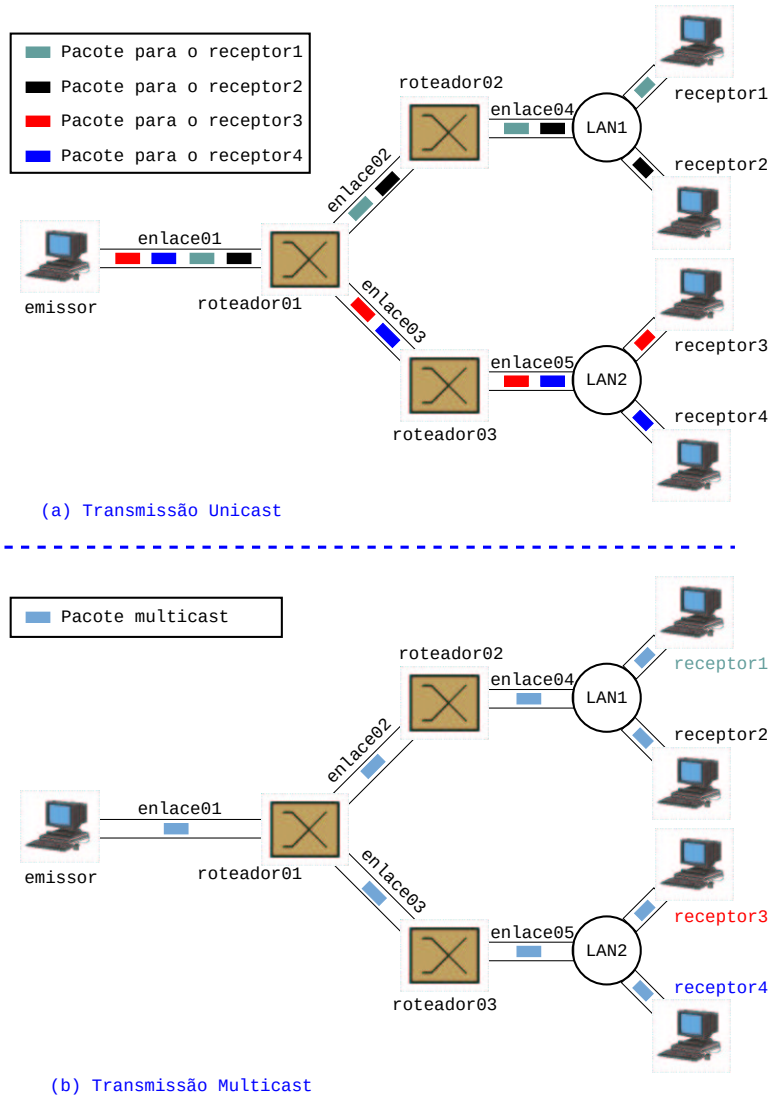

Figura 2.1: (a) Transmissão Unicast (b) Transmissão Multicast

de dados para cada membro do grupo de receptores. A Figura 2.1(a) ilustra este tipo de transmissão. Por ser uma transmissão ponto a ponto para cada receptor, quantidade de pacotes com o mesmo conteúdo que atravessa cada enlace da rede pode ser grande. Para transmitir dados para um grupo de receptores usando unicast, o número de pacotes que atravessa o primeiro enlace a partir do emissor é igual ao número de receptores. No exemplo da Figura 2.1(a) o emissor precisa enviar dados para quatro receptores, portanto são 4 pacotes passando pelo enlace01 e 2 pacotes passando em cada um dos enlaces restantes. A outra forma de transmissão é a transmissão multicast (Figura 2.1(b)), onde o número de pacotes atravessando um mesmo enlace é sempre igual a um, independentemente do número de recepto-

res. Essa melhor utilização dos recursos de rede é uma das principais vantagens da transmissão multicast. Outra vantagem é que o emissor não precisa necessariamente saber quantos e quem são os membros que formam o grupo de receptores, pois os dados são enviados para um endereço especial e não diretamente para cada membro do grupo.

Na Internet, o TCP e o UDP são os dois protocolos que podem ser usados para o transporte de pacotes. O TCP foi projetado para ser um protocolo de transporte fim a fim, confiável e orientado à conexão [20]. Fazer transporte confiável significa que o TCP garante a entrega de todos os pacotes, realizando retransmissões quando houver perdas. O projeto do TCP levou em consideração que a maior parte das perdas são causadas por estouro de *buffer* nos roteadores devido a congestionamento na rede. Para tentar diminuir essas perdas e evitar que o transmissor inunde o emissor com dados, o TCP utiliza mecanismos para controle de fluxo e congestionamento. O controle de fluxo estabelece a taxa máxima de envio de dados no emissor, baseado em informações da quantidade de dados que o receptor pode processar. O controle de congestionamento tenta reagir a possíveis congestionamentos na rede, diminuindo a taxa de envio caso ocorram perdas dos pacotes enviados. E finalmente, dizer que o TCP é orientado à conexão significa que antes de começar a transmitir os dados é necessário criar uma conexão entre os dois *hosts*. Esta conexão deve ser fechada ao término da transmissão.

Diferente do TCP, o UDP é um protocolo de transporte simples, que não garante a entrega dos pacotes e não cria conexões. O intervalo entre a geração dos dados e a sua transmissão efetiva na rede é chamado de latência. Por não fazer controle de congestionamento e de erros, a latência é menor no UDP, o que o torna mais indicado para protocolo de transporte em aplicações de tempo real, que dependem da entrega rápida dos pacotes. Outra característica importante do UDP é a possibilidade de enviar um pacote para um determinado grupo de *hosts*, ou seja, de realizar uma transmissão multicast. No contexto da transmissão multicast, o conjunto dos *hosts* formado pelo(s) emissor(es) e pelos receptores que trocam dados entre si é denominado grupo multicast. O período de tempo em que um determinado grupo multicast está ativo é chamado de sessão multicast.

Um requisito importante para que se possa usar o IP Multicast é que os roteadores localizados entre os *hosts* participantes de uma sessão sejam capazes de fazer o roteamento dos pacotes multicast. O roteamento multicast pode ser feito utilizando-se implementações de algoritmos como o DVMRP, o MOSPF, o PIM ou BGP. Infelizmente, mesmo existindo suporte a roteamento multicast na maioria dos roteadores atuais, esse serviço nem sempre é habilitado, o que dificulta uma maior disseminação do uso do IP Multicast. O cenário atual da Internet apresenta algumas "ilhas multicast", onde os roteadores estão aptos a fazer o roteamento multicast, cercadas por roteadores que não oferecem este suporte. Uma opção para interligar essas "ilhas" é o MBone, uma rede virtual sobre a Internet que utiliza túneis IP para interligar as áreas com suporte a multicast. No Brasil, a RNP (Rede Nacional de Pesquisa) [4], está implementando todo o suporte à transmissão multicast no seu *backbone*, usando roteadores com multicast nativo e túneis IP para criar um ambiente favorável ao uso desta tecnologia.

Apesar das claras vantagens da escalabilidade oferecida pelo IP Multicast em relação ao IP Unicast, questões como segurança e confiabilidade ainda precisam ser resolvidas. Não existe nenhum mecanismo no IP Multicast para impedir usuários maliciosos de enviar ou receber dados de um grupo multicast. Além disso, o IP Multicast não oferece qualquer garantia de entrega dos pacotes. Neste trabalho, a questão da transmissão multicast confiável é um dos fatores importantes e, por isso, a seguir são apresentadas algumas características e idéias propostas relativas a esse assunto.

2.1.1 Suporte à Transmissão Multicast Confiável: Rede versus Aplicação

Na arquitetura de protocolos usada na Internet, a camada de rede, ou camada IP, implementa a funcionalidade mínima necessária neste nível, ou seja, um serviço de datagrama baseado no melhor esforço. Desta forma, fica a cargo dos sistemas terminais a implementação de outras funcionalidades como correção de erros, controle de fluxo e controle de congestionamento. A simplicidade da camada IP é um dos

motivos que fizeram com que a Internet se disseminasse tão rapidamente. Porém este crescimento fez surgir aplicações com necessidades que a rede ainda não suportava. Entre essas novas necessidades estão a transmissão multicast e os mecanismos para garantir QoS. Surge então a questão sobre em que camada implementar tais funcionalidades.

De acordo com os argumentos de Saltzer *et al* [35] uma funcionalidade deve ser implementada em uma camada mais alta, a menos que a implementação em uma camada mais baixa se traduza em um ganho de desempenho que justifique a inclusão de complexidade neste nível. Embora as funcionalidades relativas a QoS dependam necessariamente de implementações na camada de rede, esse não é o caso do multicast. Algumas novas funcionalidades foram incluídas na camada IP, permitindo o envio de um mesmo pacote de dados para muitos destinos. Atualmente estas funcionalidades já estão incorporadas nas implementações da pilha de protocolos TCP/IP.

Porém, nem todos os problemas relativos à transmissão multicast foram resolvidos. O envio de dados através de IP Multicast não garante confiabilidade na transmissão, já que o protocolo usado para transporte dos pacotes é o UDP. Algumas aplicações, como por exemplo aplicações de vídeo (vic [28, 3]) e voz (vat [3]) podem suportar pequenas perdas, mas outras aplicações, como *whiteboards* compartilhados têm como requisito a transmissão confiável dos dados. Exemplos de *whiteboards* são o mb [39] e o TGWB, que será apresentado no capítulo 4.

Existem na literatura diversas propostas para solucionar a questão da garantia de confiabilidade. Algumas destas propostas, como por exemplo o trabalho de Lehman *at al* [22], sugerem a implementação de novas funcionalidades nos sistemas intermediários, ou seja, nos roteadores. Apesar destas propostas poderem oferecer aumento no desempenho, elas possuem alguns inconvenientes. Questões como o alto custo da atualização de *hardware* e *software* dos sistemas intermediários tornam estas propostas menos atrativas.

Outras soluções sugerem que os sistemas terminais, ou *hosts*, é que devem implementar as funcionalidades necessárias para a garantia de confiabilidade da transmis-

são. Pode-se agrupar estas soluções em dois conjuntos distintos: soluções baseadas em IP Multicast e soluções baseadas exclusivamente em IP Unicast. As propostas para transmissão multicast confiável baseadas em IP Multicast, como os trabalhos apresentados em [40, 17, 19, 41, 24], realizam o envio dos dados utilizando as funcionalidades multicast oferecidas pelo protocolo IP. Desta forma, os dados são enviados para o grupo multicast através do uso de um endereço IP multicast (mais detalhes no Apêndice A). Este é o caso da biblioteca apresentada no capítulo 3. Já as propostas baseadas em IP Unicast [11, 9, 30, 42], utilizam apenas transmissões unicast entre os membros do grupo.

Mesmo sendo pontos de vista distintos, nada impede a utilização conjunta das abordagens baseadas em IP Multicast e IP Unicast. Assim seria possível aproveitar as vantagens de cada uma delas, usando o IP Multicast quando este estiver disponível e utilizando o IP Unicast quando necessário. Pode-se, por exemplo, usar IP Multicast nas redes locais e usar o IP Unicast para interligar as redes onde os roteadores não implementam o multicast. A principal vantagem é que tudo isso pode ser feito no nível da aplicação, sem necessidade de modificar a estrutura ou atualizar o *software* dos roteadores.

Nas próximas seções são apresentadas algumas propostas para a transmissão multicast confiável. Na seção 2.2 são discutidas as propostas onde a transmissão é baseada em IP Multicast e, na seção 2.3, as propostas baseadas em IP Unicast.

2.2 Multicast confiável usando IP Multicast

O objetivo desta seção é apresentar os conceitos e as principais características das diversas propostas de transmissão multicast confiável que consideram o uso de IP Multicast.

2.2.1 Protocolos baseados no emissor

Um protocolo para transmissão multicast confiável é denominado baseado no emissor quando o emissor detém a responsabilidade de garantir a confiabilidade da transmissão. Cabe, portanto, ao emissor enviar os dados e verificar se todos os membros os receberam corretamente. Essa verificação pode ser feita criando-se uma lista de confirmações para cada pacote enviado. Após enviar um pacote o emissor aguarda a recepção de pacotes de confirmação ou ACKs (*positive ACKnowledgments*) de todos os receptores da sessão multicast, como pode ser visto na Figura 2.2. Uma perda é detectada quando o tempo de espera por ACKs de um determinado pacote termina e ACKs de um ou mais membros não são recebidos. Quando a perda é detectada, o emissor retransmite o pacote e espera novamente por um ACK dos receptores.

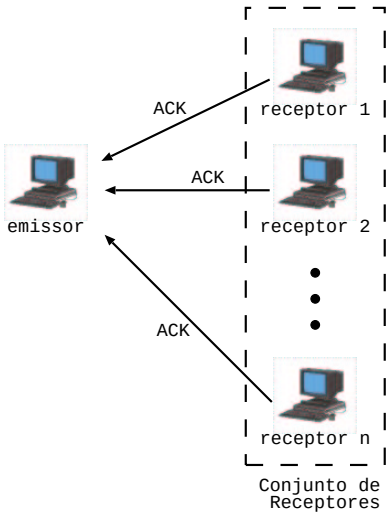

Figura 2.2: Um diagrama sobre protocolos baseados no emissor

Os principais problemas dos protocolos baseados no emissor são relativos à escalabilidade, como a necessidade de conhecer todos os membros do grupo multicast, que é requisito para utilização da lista de ACKs, e a alta carga de processamento imposta sobre o emissor. A carga de processamento no emissor e o número de ACKs trafegando na rede crescem linearmente com o número de receptores, podendo levar à chamada implosão por ACKs (*ACK implosion*). A implosão por ACKs pode fazer com que o emissor fique sobrecarregado e não consiga executar corretamente as suas funções.

2.2.2 Protocolos baseados no receptor

São denominados baseados no receptor os protocolos para transmissão multicast confiável, que delegam a cada receptor a responsabilidade da detecção de perdas e do pedido de retransmissão. Nestes protocolos, o emissor envia os dados até receber um pedido de retransmissão ou NAK (*Negative AcKnowledgement*). Ao receber um NAK, o emissor retransmite o pacote requisitado para o receptor que o enviou o NAK. Cada pacote de dados enviado recebe um número de seqüência. Os receptores detectam perdas quando recebem um pacote com o número de seqüência maior do que o esperado. Ao detectar a perda de um pacote, os receptores enviam um NAK ao emissor requisitando a retransmissão do pacote perdido. Após o envio do NAK, os receptores aguardam uma retransmissão por um determinado tempo. Caso o tempo de espera termine e nenhuma retransmissão tenha sido recebida, o receptor envia outro NAK ao emissor. A Figura 2.3 ilustra este tipo de protocolo.

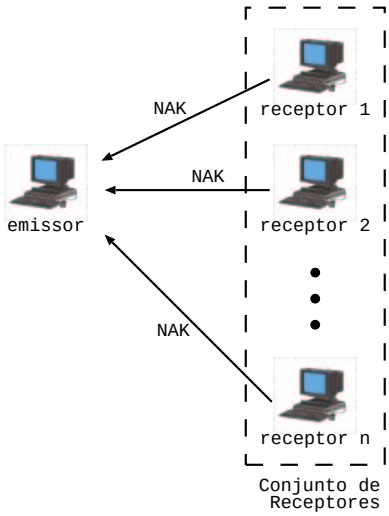

Figura 2.3: Um diagrama sobre protocolos baseados no receptor

Quando o número de membros que experimentam perdas de pacotes é muito grande, os protocolos baseados no receptor podem sofrer de implosão de NAKs, análoga à implosão de ACKs que ocorre com os protocolos baseados no emissor. Uma proposta para evitar a implosão de NAKs é alterar o modo como os receptores enviam NAKs ao emissor [33]. Nesta abordagem, ao detectar uma perda, o receptor espera por um tempo aleatório antes de enviar um NAK para o grupo. Se um NAK relativo ao mesmo pacote é recebido antes do tempo de espera expirar, então o

receptor se comporta como se ele tivesse enviado aquele NAK e espera um tempo pela retransmissão do pacote. Caso o tempo de espera para envio do NAK expire, o receptor envia o NAK para o grupo multicast. Com o uso de um temporizador aleatório antes do envio de NAKs, espera-se que apenas um NAK por pacote perdido seja enviado para o grupo e que os outros sejam suprimidos (Figura 2.4). Protocolos baseados no receptor que utilizam supressão de NAKs são denominados protocolos RINA (*Receiver Initiated with NAK Avoidance*) [25]. O protocolo SRM [17] é um exemplo de protocolo RINA e é a base do algoritmo implementado na biblioteca para transmissão multicast confiável apresentada no capítulo 3.

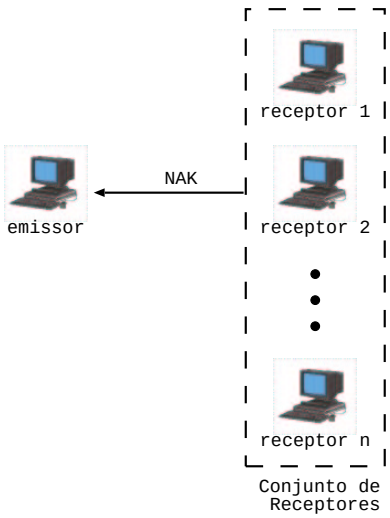

Figura 2.4: Um diagrama sobre protocolos RINA

A forma como os protocolos baseados no receptor detectam perdas pode, em alguns casos, causar inconsistência. Considerando que um receptor perca o último pacote enviado pelo emissor, se torna impossível detectar diretamente a perda, pois o receptor não vai receber nenhum pacote com número de seqüência maior. No SRM, esse problema é evitado com o uso de mensagens periódicas denominadas *session messages*. As *session messages* contém uma lista que indica o número de seqüência do último pacote de dados enviado por cada fonte da sessão multicast.

As principais vantagens dos protocolos baseados no receptor são: (a) o emissor não precisa conhecer todo o conjunto de receptores, (b) o emissor não precisa processar ACKs de cada receptor e (c) considerando a vazão como parâmetro, o desempenho é melhor do que os protocolos baseados no emissor, como pode ser visto

na seção 2.2.5. A principal desvantagem dos protocolos baseados no receptor é que não existe, na definição do protocolo, um mecanismo para decidir quando o emissor pode liberar os dados antigos da memória. Além disso, existem outras questões relativas à implementação deste tipo de protocolo que serão discutidas no capítulo 3.

2.2.3 Protocolos baseados em árvore

Trabalhos anteriores, como visto em Levine e Garcia-Luna-Aceves [25], definem que protocolos baseados em árvore são caracterizados pela divisão do conjunto de receptores em grupos, distribuindo a responsabilidade de transmissão em uma árvore de ACKs construída a partir dos grupos, tendo como raiz o emissor. Um exemplo deste tipo de árvore pode ser visto na Figura 2.5. A árvore de ACKs evita que os receptores enviem confirmações diretamente para o emissor, mantendo desta forma a escalabilidade do protocolo.

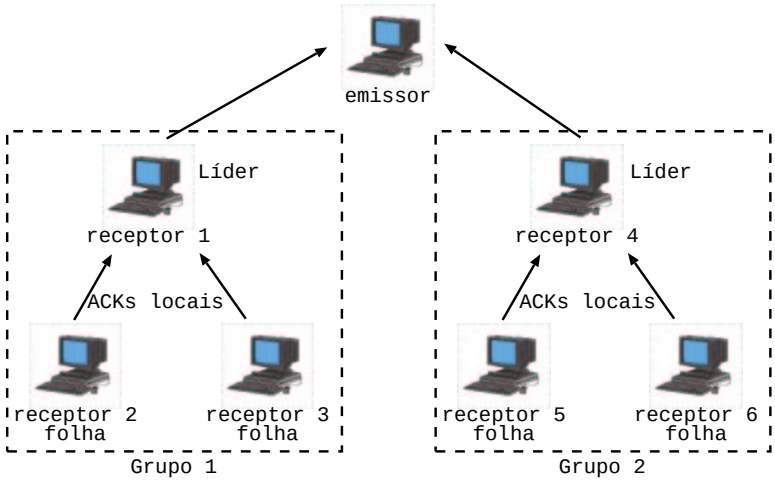

Figura 2.5: Um diagrama sobre protocolos baseados em árvore

A árvore de ACKs é formada pelos receptores e pelo emissor organizados em grupos locais, onde cada grupo local tem um líder responsável pelas retransmissões dentro do grupo. Cada líder de grupo, exceto o emissor, se comunica com outro grupo mais próximo do emissor para pedir retransmissões de pacotes que não foram recebidos corretamente. Quando existir mais de um emissor, cada grupo pode ter um líder relativo a cada emissor. Um *host* definido como nó terminal da árvore

é chamado de "folha". Uma árvore de ACKs formada apenas pelo emissor e por um conjunto de "folhas" corresponde exatamente ao caso dos protocolos baseados no emissor. A cada pacote recebido pelos membros de um grupo local, os ACKs são enviados exclusivamente para o líder deste grupo. Tais ACKs são chamados de ACKs locais.

Em protocolos baseados em árvore, os líderes também podem ser responsáveis por liberar dados da memória. Essa liberação só pode ser efetivada quando o líder recebe ACKs agregados de todos os membros do grupo. Estes ACKs agregados são enviados das folhas em direção ao emissor, um grupo local por vez. Um líder de grupo não pode enviar um ACK agregado antes de receber um ACK individual de todos os membros do seu grupo local. O uso de ACKs agregados é necessário para garantir que o protocolo funcione corretamente, mesmo se o líder de um grupo falhar ou se a árvore de ACKs permanecer particionada por longos períodos. Essa abordagem, denominada NAPP (*Nak Avoidance with Periodic Polling*), foi utilizada por Levine e Garcia-Luna-Aceves no desenvolvimento do protocolo Lorax [24]. Se os ACKs agregados não forem usados, a única forma de garantir o funcionamento correto do protocolo é não liberando os dados da memória. Esta é a abordagem usada por alguns protocolos baseados em árvore como o RMTP [26].

A principal vantagem dos protocolos baseados em árvore é a possibilidade da liberação eficiente dos dados armazenados na memória, mecanismo inexistente nos protocolos baseados no receptor. Apesar desta vantagem, o conjunto de operações, necessário para a criação e a manutenção da árvore de ACKs, aumenta consideravelmente a complexidade. Estão incluídas neste conjunto questões como identificação da raiz da árvore, quantidade máxima de membros em cada grupo, mecanismos para roteamento dos ACKs de acordo com cada fonte da árvore e rotinas para inserção e remoção de nós durante a sessão.

2.2.4 Protocolos baseados em anel

Os protocolos para transmissão multicast confiável baseados em anel foram criados, originalmente, para oferecer suporte a aplicações que tem como requisito principal ordenação total e atômica das transmissões em todos os receptores. O objetivo deste tipo de protocolo é combinar as vantagens da alta vazão dos protocolos baseados em receptores com a confiabilidade dos ACKs. A premissa básica de um protocolo baseado em anel é, para cada pacote enviado, ter apenas um membro receptor responsável por confirmar o recebimento dos pacotes ao emissor (Figura 2.6). O emissor retransmite um dado caso ele não receba um ACK deste membro dentro de um determinado tempo. Esse ACK também é usado por todos os receptores para garantir a ordenação global: um dado só é passado para a aplicação quando o ACK é recebido. O RMP [40] é um exemplo de protocolo baseado em anel.

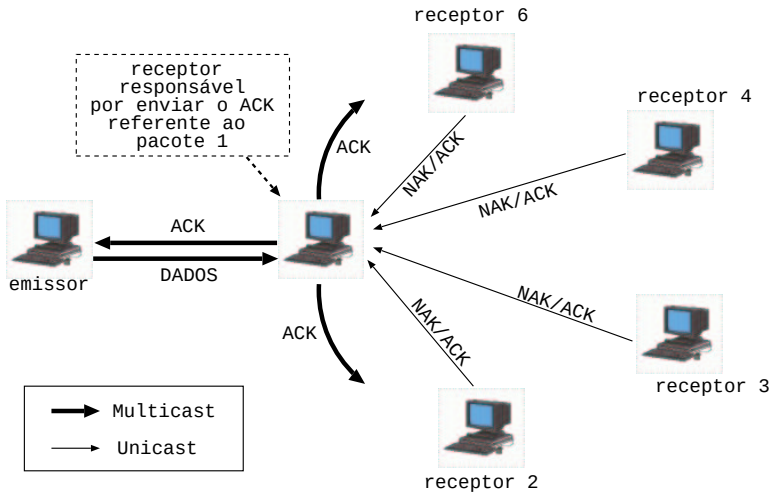

Figura 2.6: Um diagrama sobre protocolos baseados em anel

Os principais problemas de protocolos baseados em anel são a alta complexidade da implementação e a necessidade de se conhecer o grupo de receptores.

2.2.5 Análise da vazão máxima

Nesta seção são apresentados os resultados obtidos por Pingali *et al* [31] sobre a análise da vazão máxima dos protocolos baseados no emissor, baseados no receptor e baseados no receptor com supressão de NAKs. Esta análise foi mais tarde estendida

por Levine e Garcia-Luna-Aceves [25] aos protocolos baseados em árvores e aos baseados em anel. Nestes trabalhos, o modelo usado para os cálculos da vazão máxima têm foco nos requisitos de processamento dos protocolos descritos anteriormente. A vazão máxima de um dado protocolo é uma função da taxa de processamento por pacote nos emissores e receptores. A análise concentra-se em obter o tempo de processamento por pacote em um dado membro do grupo multicast.

A análise considera apenas um emissor X, enviando pacotes via multicast para R receptores idênticos. A probabilidade de perda de pacotes é p para qualquer membro. O tamanho dos grupos locais nos protocolos baseados em árvore é B. Para tornar mais clara a análise dos protocolos baseados em árvore, foi utilizada uma única árvore de ACKs tendo como raiz o emissor. A retransmissão é seletiva em todos os protocolos, ou seja, somente os pacotes perdidos são retransmitidos. Considera-se também que nenhum pacote de confirmação, ACK ou NAK, é perdido e que os eventos de perda em cada nó são mutuamente independentes.

Protocolo	Requisitos de processamento	p constante	$p \to 0$
Baseado no emissor	$O(R(1+\frac{plnR}{1-p})$	$O(RlnR)$	$O(R)$
Baseado no receptor	$O(1+\frac{pR}{1-p})$	$O(R)$	$O(1)$
Baseado no receptor com supressão de NAKs	$O(1+\frac{plnR}{1-p})$	$O(lnR)$	$O(1)$
Baseado em anel	$O(1+\frac{(R-1)p}{1-p})$	$O(R)$	$O(1)$
Baseado em árvore	$O(B(1-p)+pBlnB)$	$O(1)$	$O(1)$
Baseado em árvore com NAPP	$O(1+\frac{1-p+plnB+p^2(1-4p)}{1-p})$	$O(1)$	$O(1)$

Tabela 2.1: Vazão máxima dos protocolos para transmissão multicast confiável

A Tabela 2.1 apresenta os limites da vazão máxima para as classes de protocolos discutidas nas seções anteriores. É possível perceber que a vazão máxima dos protocolos baseados em árvore é a maior entre os protocolos estudados, mas o protocolo RINA vem logo em seguida, inclusive se equiparando aos protocolos baseados em árvore quando $p \to 0$.

2.3 Multicast confiável na aplicação

O IP Multicast oferece muitas vantagens para a transmissão de dados para um grupo de receptores, mas existem ainda alguns problemas que impedem sua utilização por um grande número de usuários e aplicações. Em primeiro lugar, os roteadores devem ser capazes de fazer o roteamento multicast. Muitas vezes os roteadores, em uma determinada rede, não oferecem esse serviço e a sua troca pode ser muito onerosa em relação ao preço do *hardware* e à nova configuração dos serviços. Em segundo lugar, os protocolos de roteamento guardam informações referentes a cada grupo multicast do qual ele roteia pacotes. Este comportamento dos roteadores pode quebrar a escalabilidade do multicast. Outra exigência do IP Multicast é que os endereços multicast sejam escolhidos para serem únicos em um escopo global, exigência difícil de se garantir de forma consistente e segura. Existe ainda a questão de que qualquer *host* pode transmitir dados para qualquer grupo multicast, o que deixa a rede vulnerável a ataques e complica o gerenciamento e a reserva de recursos. E, por fim, o IP Multicast é um serviço de melhor esforço e é mais difícil do que no IP Unicast oferecer confiabilidade, controle de fluxo, controle de congestionamento e segurança. Para tentar solucionar esses problemas surgiram propostas de implementação da transmissão multicast na aplicação, baseadas apenas em IP Unicast.

As vantagens de se usar transmissão unicast são muitas. Como os pacotes são transmitidos via IP Unicast, nenhuma alteração na estrutura da rede se faz necessária. Além disso, soluções para os problemas de confiabilidade, controle de fluxo, controle de congestionamento e segurança podem ser resolvidos utilizando-se mecanismos já conhecidos em transmissões unicast.

Existem também algumas desvantagens no uso do IP Unicast. Entre elas, podemos citar a maior necessidade de banda na rede e o maior atraso na entrega dos pacotes. O multicast na aplicação necessita de mais banda porque é difícil garantir que duas transmissões unicast não vão atravessar um mesmo enlace de dados num determinado ponto. A latência pode ser maior porque os pacotes podem passar por vários *hosts*, e não apenas por roteadores, antes de chegar ao destino.

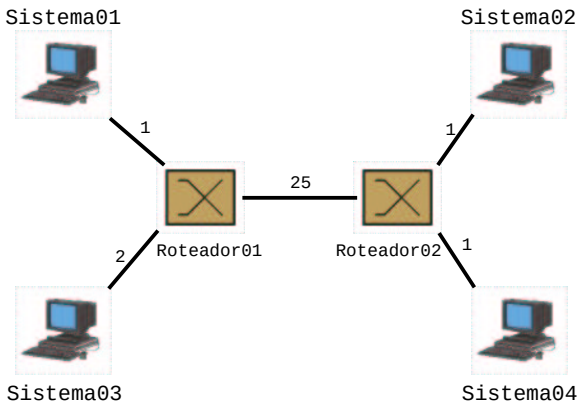

Figura 2.7: Representação das conexões físicas e retardos entre os sistemas

Um exemplo de multicast na aplicação é o trabalho de Chu *et al* [11], denominado *End System Multicast*, onde os autores propõem que os "sistemas terminais" sejam interligados por uma rede virtual montada sobre a rede existente, utilizando-se um protocolo chamado Narada. O Narada cria a rede para interligação dos sistemas em dois passos. Primeiro é gerada, a partir da rede descrita na Figura 2.7, uma estrutura denominada *mesh*, que interconecta cada sistema aos outros membros do grupo, como ilustra a Figura 2.8. Os números sobre os enlaces representam o retardo sofrido por um pacote ao atravessar aquele enlace. No segundo passo, algumas conexões são descartadas, seguindo regras pré-definidas, para formar uma árvore de transmissão entre o emissor e os outros membros do grupo. A Figura 2.9 mostra um exemplo da árvore criada a partir da *mesh* da Figura 2.8 onde o emissor é o Sistema01. O Narada implementa também mecanismos para que a rede possa se auto-organizar e se adaptar dinamicamente a mudanças no ambiente, como a entrada e/ou a saída de *hosts*, o congestionamento nos enlaces e a falha em *hosts* que pertencem à árvore.

Os principais pontos a serem considerados, na análise do desempenho da solução proposta, são o aumento da latência e o número de pacotes duplicados atravessando os enlaces da rede. O estudo da latência foi feito baseado em uma medida denominada RDP (*Relative Delay Penalty*), que representa a proporção do aumento da latência do envio direto, via unicast, entre dois membros e o envio através da rede criada pelo Narada. Comparando-se os retardos nas figuras 2.8 e 2.9, pode-se perceber que houve aumento no retardo do Sistema01 para o Sistema04. Assim, o RDP

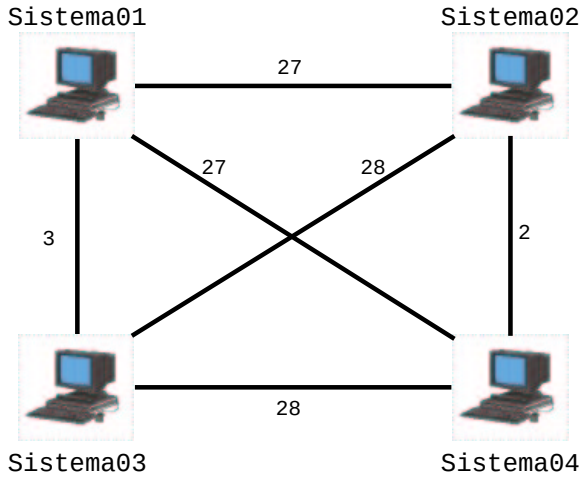

Figura 2.8: *Mesh* criada pelo protocolo Narada

entre o Sistema01 e o Sistema04 é $\frac{29}{27}$. Não houve aumento no retardo do Sistema01 para os outros sistemas e, portanto, o RDP em relação a eles é igual a 1. Em relação aos pacotes duplicados, considerando as figuras 2.7 e 2.9, a árvore de transmissão gerada pelo Narada vai fazer com que passem pacotes duplicados no Roteador02, um enviado do Sistema01 para o Sistema02 e outro enviado do Sistema02 para o Sistema04.

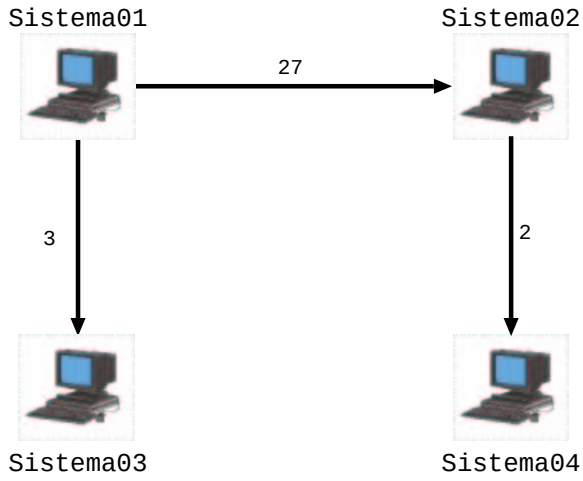

Figura 2.9: Árvore gerada pelo protocolo Narada

Os resultados de simulações e experimentos apresentados em [11] mostram que para grupos pequenos, por exemplo com 16 membros, na maioria dos casos obteve-se $RDP < 2,5$ e número de pacotes duplicados inferior a 6. A partir desses resultados, pode-se concluir que a proposta da transmissão multicast, implementada a nível

de aplicação e utilizando-se o Narada, pode ser interessante se aplicada a grupos pequenos e esparsos como conferências de áudio e vídeo, aulas virtuais e jogos que envolvem múltiplos jogadores. Porém, este tipo de transmissão pode não ser indicado para aplicações de tempo real, onde uma fonte envia muitos dados para um grande número de receptores. A desvantagem do uso do Narada em aplicações tempo real é o aumento da latência na transmissão dos dados, que ocorre cada vez que um dado passa por *hosts* intermediários.

Outros exemplos de multicast na aplicação são o Yoid [18], o Scattercast [9] e o Bayeux [42]. A principal diferença entre o Yoid e o protocolo baseado no Narada é que o primeiro constrói a árvore de transmissão diretamente, sem precisar de uma estrutura intermediária, como a *mesh* criada pelo Narada.

O Scattercast utiliza um protocolo chamado Gossamer, que também cria uma *mesh* antes de criar a árvore de transmissão final. A diferença em relação ao Narada é que o Scattercast é dependente de uma infraestrutura especial formada pelos SCXs (*ScatterCast proXies*).

Por fim, o Bayeux é um sistema multicast a nível de aplicação para transmissão confiável de dados para um grupo grande de receptores, baseado em um protocolo denominado Tapestry. O Tapestry é um protocolo de roteamento a nível de aplicação. Sobre o Tapestry, o Bayeux implementa um protocolo simples que organiza os receptores em uma árvore, tendo o emissor como raiz.

Capítulo 3

RML: uma biblioteca para transmissão multicast confiável

3.1 Introdução

Existem aplicações que podem conseguir grandes melhoras no desempenho, se utilizarem o IP Multicast para transmissão dos dados. Economia de banda na rede e escalabilidade são duas vantagens muito importantes oferecidas pelo IP Multicast. Porém, algumas aplicações, como compartilhamento de arquivos, simuladores distribuídos e os *whiteboards*, precisam de confiabilidade no envio dos dados, ou seja, os dados devem ser entregues sem perdas para todos os membros do grupo. Como no IP Multicast o protocolo de transporte utilizado é o UDP, não existe garantia da entrega dos pacotes. Portanto, as aplicações que necessitam de confiabilidade na transmissão multicast devem implementar mecanismos para garanti-la.

A RML (*Reliable Multicast Library*) desenvolvida no âmbito deste trabalho, é uma biblioteca implementada em C e criada para oferecer, de uma maneira simples e eficiente, um serviço de transmissão multicast confiável para as aplicações. Desta forma, as aplicações podem enviar dados através do IP Multicast de forma confiável, usando as funções oferecidas pela biblioteca. A Figura 3.1 ilustra como a biblioteca RML faz a interface entre a aplicação e a rede.

No projeto e desenvolvimento da RML, levou-se em consideração que esta deveria ser uma biblioteca que permitisse um alto grau de liberdade para as aplicações. Além disso, a implementação foi realizada de tal forma que a inclusão de novas funcionalidades não requer muito esforço do programador. O código-fonte da RML está disponível na página do projeto em [5].

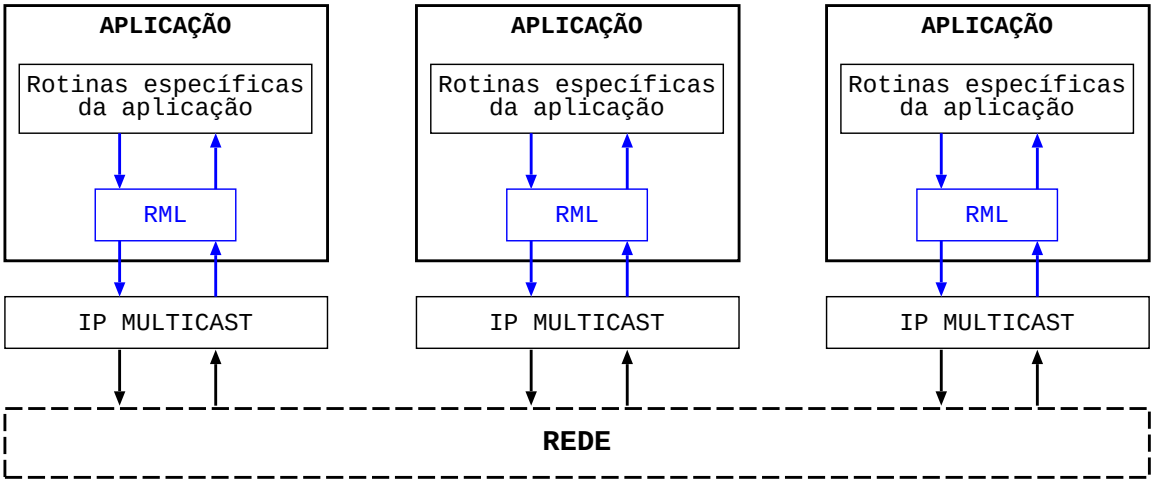

Figura 3.1: Esquema do uso da RML

Os requisitos cumpridos na implementação da RML tornaram-na uma opção interessante para diversos tipos de usuários. A RML pode ser usada, com pouca ou nenhuma modificação, por usuários que desejem apenas uma forma simples para transmissão multicast confiável. Porém, a biblioteca também pode ser utilizada por usuários que possuem necessidades mais específicas, pois como o código-fonte é livre, usuários avançados podem fazer otimizações e até mesmo inserir novas características, para que a biblioteca atenda às mais diversas necessidades.

O algoritmo para transmissão multicast confiável implementado na RML é do tipo RINA (*Receiver-Initiated with Nak Avoidance*) e foi baseado na proposta do SRM [17], apresentado na seção 2.2.2. Apesar do SRM ter sido a principal referência na implementação do algoritmo da RML, existem algumas diferenças que serão apresentadas no decorrer deste capítulo. Entre essas diferenças destaca-se o envio de NAKs, pois na RML cada NAK pode requisitar a retransmissão de até 64 pacotes diferentes. Para manter o tamanho do pacote NAK pequeno, foi proposta uma abordagem em que as requisições são representadas apenas por quatro números inteiros.

Para o tratamento de eventos concorrentes, como envio e recebimento de dados e tratamento dos pacotes, a RML utiliza *threads*. No GNU/Linux, plataforma utilizada para o desenvolvimento da RML, as *threads* de um aplicativo são como processos distintos, mas compartilham as mesmas variáveis e estruturas globais. Existem quatro *threads* na RML, que são descritas a seguir:

- ***Thread* Principal**: é responsável pela inicialização das estruturas da biblioteca e pelo envio de dados;
- ***Thread Manager***: é uma *thread* especial criada pelo sistema operacional para gerenciar as outras;
- ***Thread* para tratamento de sinais**: é responsável por capturar os sinais enviados para a RML. Cada sinal recebido pode requisitar um procedimento diferente, como fechar o programa ou enviar uma retransmissão;
- ***Thread* para Recebimento**: é responsável por receber, processar e armazenar os pacotes recebidos;

Na seção seguinte será apresentada a forma de identificação dos membros e a estrutura de armazenamento de dados utilizada pela RML. A seção 3.3 aborda a questão do gerenciamento de membros em uma sessão multicast. As rotinas para detecção e recuperação de perdas serão apresentadas na seção 3.4. A descrição e o funcionamento da lista de eventos são discutidos na seção 3.5. Mais informações técnicas sobre a RML podem ser encontradas nos apêndices A, B e C

3.2 Identificação de membros e armazenamento de dados

3.2.1 Identificação de membros

Para o correto funcionamento dos mecanismos de detecção e recuperação de perdas, é necessário que cada membro de uma sessão multicast tenha uma identificação

única. Na RML, cada membro é identificado por uma estrutura de dados, denominada MEMBER_ID, composta pelo endereço IP do membro e pela identificação do processo que está sendo executado por ele. Essa identificação de processo, conhecida como PID (*Process ID*), é um número inteiro seqüencial designado pelo sistema operacional. Como cada endereço IP só pode estar associado a um único *host* e cada processo tem sua identificação exclusiva em cada *host*, não podem existir dois membros com MEMBER_IDs iguais, garantindo portanto a sua unicidade.

É importante perceber que, apesar de todos os membros possuírem sua identificação, isto não implica, necessariamente, que todos os membros conheçam ou guardem informações de todos os outros participantes da sessão multicast. Em um cenário onde, por exemplo, existe apenas um membro gerando dados, este será o único membro conhecido por todos os outros participantes do grupo.

3.2.2 A estrutura para armazenamento de dados

A RML utiliza uma estrutura de dados, denominada CACHE, para armazenar os dados recebidos antes de enviá-los para o processamento na aplicação. Os dados são enviados para processamento somente quando estão ordenados e completos. Esta estrutura impede que pacotes que cheguem fora de seqüência sejam simplesmente descartados, o que provocaria, mais tarde, o envio de mais pedidos de retransmissão.

Durante uma sessão multicast, uma instância da estrutura CACHE é criada para cada membro gerador de dados. Membros que atuam apenas como receptores não precisam ser identificados desta forma. As instâncias da CACHE são organizadas em uma lista, como ilustra a Figura 3.2. A partir de agora, a lista de instâncias da estrutura CACHE será referenciada simplesmente como `cache`.

Cada nó da `cache` é formado por um conjunto de estruturas. Entre estas estruturas, a primeira componente é o SM_INFO, que por sua vez é composto de duas outras, denominadas MEMBER_ID e MEMBER_STATUS. O MEMBER_ID, como já foi descrito anteriormente, armazena a dupla IP e PID, usada para identificação do membro. A estrutura MEMBER_STATUS agrupa as informações referentes aos

dados recebidos do membro identificado pelo MEMBER_ID. Entre essas informações encontram-se:

- `first_rcv`: o número de seqüência do primeiro pacote recebido;
- `last_rcv`: o número de seqüência do último pacote recebido;
- `last_seq_rcv`: o número de seqüência do último pacote recebido que foi enviado para processamento na aplicação;
- `last_identified`: o número de seqüência do último pacote identificado;
- `window_size`: o tamanho da janela para envio de NAK;
- `window_mask`: vetor, com `window_size` posições, que identifica os pacotes perdidos;
- `window_ini`: indica a posição inicial do vetor `window_mask`.

Os campos `window_size`, `window_mask` e `window_ini` são utilizados para detecção e recuperação de perdas e serão discutidos com mais detalhes na seção 3.4. Os outros campos que compõem a estrutura CACHE são:

- `number_of_nodes`: quantidade de pacotes recebidos e que se encontram armazenados;
- `active`: indica se o membro ainda está ativo na sessão multicast;
- `first`: ponteiro para o primeiro nó da lista de pacotes de dados armazenados;
- `nak_list`: ponteiro para lista de controle de NAKs enviados.

Recebendo e armazenando dados

Ao receber um pacote de dados, a primeira operação a ser executada é verificar se o membro emissor destes dados já foi identificado e inserido na `cache`. Caso

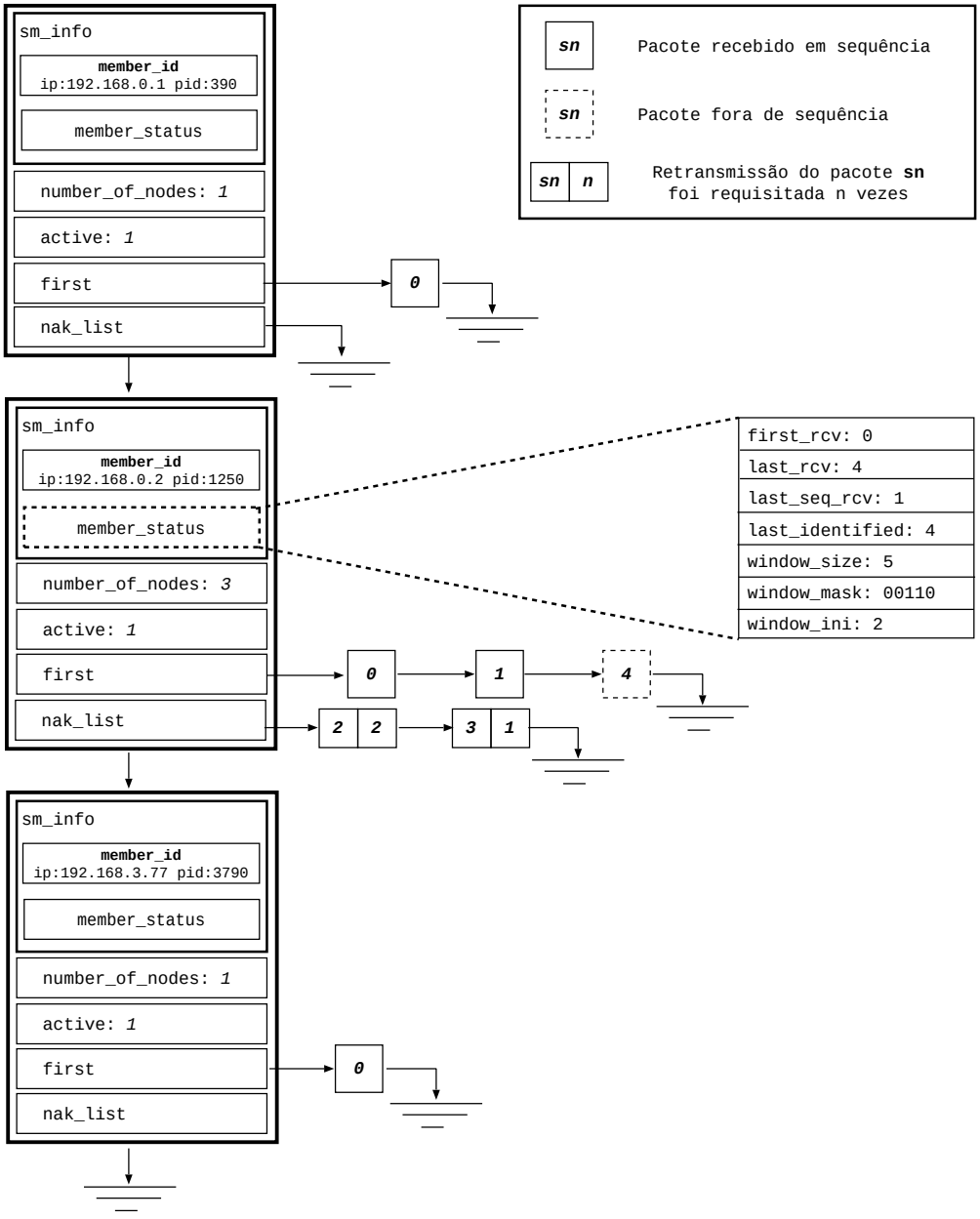

Figura 3.2: Exemplo de lista de CACHE

seja o primeiro dado recebido deste membro, é inserido um novo nó na `cache` e o dado recebido é colocado na lista apontada por `first`. Caso o membro já tenha sido identificado anteriormente, o pacote de dados é inserido na lista de pacotes correspondente. Nos dois casos, se o pacote recebido estiver em seqüência ele é enviado para processamento na aplicação.

Como o número de pacotes de dados que podem ser armazenados na `cache` é limitado, torna-se necessário fazer um controle dos pacotes recebidos. Quando um

novo pacote de dados é recebido, verifica-se o valor do campo `number_of_nodes`. Se este valor for menor que o número máximo de dados que podem ser armazenados por membro, ele é incrementado em uma unidade. Se o valor de `number_of_nodes` for igual ao máximo permitido, o novo pacote substitui o mais antigo entre os que estão armazenados e que já foram enviados para a aplicação.

Considerando o exemplo da Figura 3.2, é possível observar que foram registrados três membros emissores, identificados pelas duplas IP e PID: 192.168.0.1:390, 192.168.0.2:1250 e 192.168.3.77:3790. Somente os pacotes com número de seqüência 0 foram recebidos dos membros 192.168.0.1:390 e 192.168.3.77:3790. Do membro 192.168.0.2:1250 foram recebidos três pacotes de dados. Entre esses pacotes de dados, os pacotes 0 e 1, estão em seqüência, e o pacote 4, está fora de seqüência. O recebimento do pacote 4 permite detectar a perda dos pacotes 2 e 3. Os mecanismos para detecção e recuperação de perdas são descritos na seção 3.4

3.3 Gerenciamento de membros na sessão multicast

Os protocolos baseados no receptor, como apresentado na seção 2.2.2, não possuem, a princípio, mecanismos para indicar, durante uma sessão multicast, quando os dados armazenados podem ser descartados. Para algumas aplicações é possível calcular a capacidade de armazenamento necessária para evitar que, durante um sessão multicast, um pedido de retransmissão não seja atendido. Na RML, o tamanho da área de armazenamento pode ser determinado pela aplicação, permitindo assim que a biblioteca se adapte a diversos cenários. Porém, mesmo que seja possível calcular de forma satisfatória o tamanho do espaço de armazenamento necessário, problemas podem ocorrer.

Um exemplo simples é a questão do tempo de entrada na sessão multicast. Se um usuário tentar entrar na sessão muito tempo após o seu início, os atuais membros podem não ter mais os dados antigos armazenados para retransmissão. Neste caso, uma das saídas é requisitar os dados antigos diretamente para a aplicação. Na RML, foram criadas rotinas que permitem executar essa requisição e, utilizando

estas rotinas, foi desenvolvido um mecanismo para resolver o problema da entrada de usuários. É importante ressaltar que o uso deste mecanismo é opcional, pois ele não é necessário em aplicações em que todos os usuários se juntam ao grupo no início da sessão.

Quando um novo usuário deseja entrar na sessão multicast, ele envia uma mensagem do tipo JOIN REQUEST ao grupo e espera por uma resposta durante um determinado tempo. Qualquer membro do grupo multicast pode responder a uma mensagem JOIN REQUEST. Ao receber uma mensagem deste tipo, um membro responde com uma mensagem denominada JOIN ACCEPT, que contém o MEMBER_ID deste membro e o número de uma porta para conexão. Ao receber o primeiro JOIN ACCEPT, o novo usuário se conecta ao membro que enviou a mensagem e recebe o "estado atual" daquele membro. Esta conexão é feita via TCP e usando a porta indicada na mensagem JOIN ACCEPT. O "estado atual" de um membro é formado pelo conjunto de dados que está disponível na aplicação e pelas informações contidas na `cache` deste membro. Ao receber os dados de um membro do grupo, o novo usuário se torna também um membro, podendo inclusive responder às mensagens JOIN REQUEST de outros usuários. Se o novo usuário, após enviar uma mensagem JOIN REQUEST e esperar por um determinado tempo, não receber uma resposta, ele se considera como o primeiro membro da sessão multicast.

Outra questão importante é em relação à saída de membros em uma sessão multicast. Na RML, ao se executar a rotina para saída do grupo multicast, uma mensagem do tipo LEAVE GROUP é enviada e o emissor aguarda um determinado tempo antes de realmente sair do grupo. Os membros que recebem uma mensagem deste tipo alteram, em suas `caches`, o campo `active` relativo ao emissor da mensagem. Este procedimento serve para indicar que o membro não está mais ativo no grupo. O conhecimento da saída de membros do grupo pode ser usado para liberar memória, ou até mesmo alterar o cálculo dos temporizadores do protocolo.

3.4 Detecção e recuperação de perdas

3.4.1 Detecção de perdas

Na RML, as informações sobre os membros geradores de dados estão armazenadas na `cache`, como visto na seção 3.2.2. Quando um pacote de dados é recebido, o seu número de seqüência é comparado ao valor do campo `last_seq_rcv`, localizado no nó da `cache` referente ao emissor do pacote. Se o número de seqüência for menor ou igual ao `last_seq_rcv`, o pacote é ignorado, pois já foi recebido e enviado à aplicação. Se o número de seqüência for igual a `last_seq_rcv` + 1, o pacote é inserido na `cache` e enviado para a aplicação. Por fim, se o número de seqüência for maior que `last_seq_rcv` + 1, uma perda de pacotes é detectada. Neste caso, requisições referentes aos pacotes perdidos devem ser enviadas. Porém, o mecanismo de detecção de perdas baseado apenas na verificação de números de seqüência pode falhar.

A falha ocorre quando os últimos pacotes de dados enviados por um membro são perdidos. Nesta situação, nenhum pacote com número de seqüência maior vai ser recebido e a perda não poderá ser detectada. Para resolver este problema, os membros emissores de dados do grupo multicast enviam mensagens periódicas, chamadas REFRESH. Nesta mensagem, o emissor indica o número de seqüência do último pacote de dados enviado por ele. Assim, ao receber mensagens REFRESH dos emissores, os membros da sessão multicast podem detectar as perdas, mesmo quando os últimos pacotes de dados não são recebidos.

3.4.2 Envio de NAKs

Após a detecção de perdas, são iniciados os procedimentos para envio de NAKs. Para explicar estes procedimentos, será considerado o exemplo da Figura 3.2. Neste exemplo, ao receber o pacote com número de seqüência 4, relativo ao membro 192.168.0.2:1250, uma perda é detectada, pois 4 > `last_seq_rcv+1`. Os pacotes identificados como perdidos possuem números de seqüência no intervalo de

`last_seq_rcv+1` a `last_identified`, ou seja, os pacotes 2 e 3. O próximo passo a ser executado é o envio, para o grupo multicast, de requisições de retransmissão para estes pacotes.

Uma forma simples para enviar os pedidos de retransmissão seria enviar um NAK referente a cada pacote perdido. O problema desta abordagem é que se o número de pacotes perdidos for muito grande, muitas mensagens serão enviadas para o grupo, aumentando o tráfego na rede e, provavelmente, resultando em mais perdas de pacotes. Outra questão importante é se existe vantagem em enviar NAKs referentes a um conjunto muito grande de pacotes, pois, além da possibilidade de sobrecarregar os possíveis retransmissores, o receptor pode ser incapaz de processar um conjunto grande de retransmissões. Para resolver estas questões foi adotada uma outra abordagem, onde uma única mensagem NAK pode requisitar a retransmissão de vários pacotes perdidos. Além disso, esta requisição só é feita para os pacotes que pertencem a um determinado intervalo.

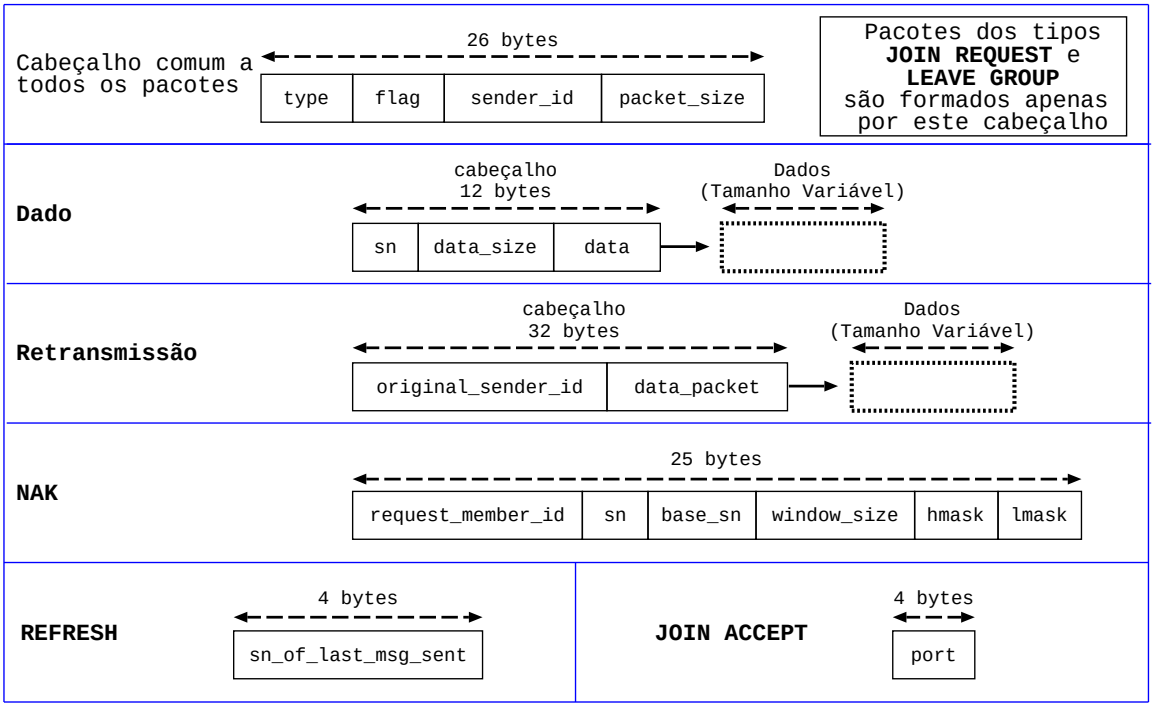

Figura 3.3: Tipos de pacotes na RML

No exemplo da Figura 3.2, cada mensagem NAK enviada pode requisitar até cinco retransmissões diferentes. O número máximo de requisições que podem ser enviadas em uma única mensagem NAK é determinado pelo campo `window_size`. As

requisições são referentes apenas a um intervalo, determinado entre `last_seq_rcv`+1 e `last_seq_rcv`+`window_size`. O vetor `window_mask` indica o número de seqüência dos pacotes que devem ter sua retransmissão requisitada, representados pelas posições de `window_mask` que possuem valor 1. Estes números de seqüência podem ser obtidos através dos valores de `last_seq_rcv`, `window_size`, `window_mask` e `window_ini`. No exemplo, a posição no vetor `window_mask` indicada por `window_ini`, representa o número de seqüência 2 (`last_seq_rcv`+1), a posição `window_ini`+1 indica o número de seqüência 3 (`last_seq_rcv`+2) e assim sucessivamente. Portanto, neste exemplo, o vetor `window_mask` representa os pacotes perdidos com números de seqüência 2 e 3.

Na implementação da RML, `window_size` tem valor 64. Este número foi escolhido de forma que, no pacote do tipo NAK mostrado na Figura 3.3, o vetor com os números de seqüência das requisições pudesse ser representado por dois inteiros, denominados `hmask` e `lmask`.

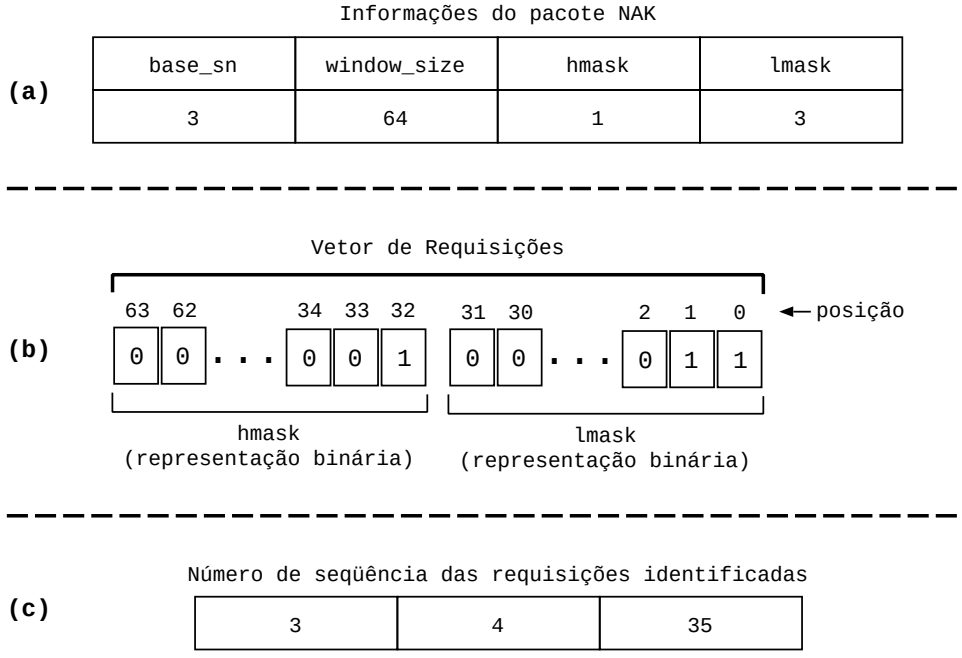

Figura 3.4: Retirando informações do pacote NAK

Quando um pacote NAK é recebido, as informações que ele contém são utilizadas para identificar as requisições. Utilizando as informações do NAK descrito na Figura 3.4(a) como exemplo, é possível construir o vetor de requisições identificadas. Este

vetor, que pode ser visto na Figura 3.4(c)), foi resultado da aplicação do algoritmo da Figura 3.5 ao vetor da Figura 3.4(b). No cenário descrito nesse exemplo, as requisições contidas no pacote NAK são referentes aos pacotes com números de seqüência 3, 4 e 35.

Utilizando-se a representação binária de `hmask` e `lmask` é possível referenciar 64 requisições em 8 bytes. Se cada requisição fosse representada separadamente, seriam necessários 256 bytes para representar as mesmas 64 requisições.

```
Algoritmo

    j=0
    Para i variando de 0 a window_size
       Se mask[i] = 1 então
          sn = i + base_sn
          req[j] = sn
          j++
       fim_se
    fim_para

Onde:

    sn     - número de seqüência da requisição
    mask[] - vetor contendo a união das representações binárias
             de hmask e lmask
    req[]  - vetor contendo os sn das requisições identificadas
```

Figura 3.5: Algoritmo para determinação das requisições de um NAK

3.4.3 Retransmissão dos dados

Ao receber um pacote NAK, a primeira providência tomada por um membro é verificar se os pacotes requisitados estão disponíveis na sua `cache`. Caso afirmativo o membro escalona as retransmissões correspondentes. Caso os pacotes requisitados não estejam disponíveis, existem duas possibilidades: os pacotes já foram removidos da `cache` ou também foram perdidos por este membro. Se os pacotes já foram removidos o membro ignora o NAK, pois não é capaz de satisfazer as requisições. Porém, se os pacotes também foram perdidos pelo membro que recebeu o NAK, é preciso verificar se existe um envio de NAK escalonado para estes pacotes. Caso exista, o envio é cancelado e o membro apenas espera pelas retransmissões. Se não existir um envio de NAK escalonado, o pacote é ignorado.

Um membro, quando recebe uma retransmissão, pode executar ações diferentes. A primeira situação a considerar é a de um membro que realmente perdeu o pacote referente a essa retransmissão. Neste caso, o membro cancela o envio de NAK, ou, caso o NAK já tenha sido enviado, o membro termina a fase de espera pela retransmissão. Após cancelar uma dessas ações, o membro insere o dado na `cache`. Se o número de seqüencia do pacote de dados recebido for igual a `last_seq_rcv` + 1, os dados contidos no pacote são enviados para a aplicação e o valor de `last_seq_rcv` é atualizado. Além disso, o recebimento da retransmissão pode liberar pacotes recebidos anteriormente e já armazenados na `cache`, que também podem ser enviados para a aplicação, como pode ser visto na Figura 3.6.

A segunda situação a considerar é quando o membro não esperava pela retransmissão. Neste caso, se o número de seqüência contido na retransmissão for maior que o valor do campo `last_seq_rcv`, referente ao emissor original do pacote, o dado é inserido na `cache`. Caso o número de seqüência seja menor que o valor de `last_seq_rcv`, o membro verifica se existe uma retransmissão escalonada referente ao mesmo dado. Se existir, o envio é cancelado, caso contrário a retransmissão recebida é ignorada.

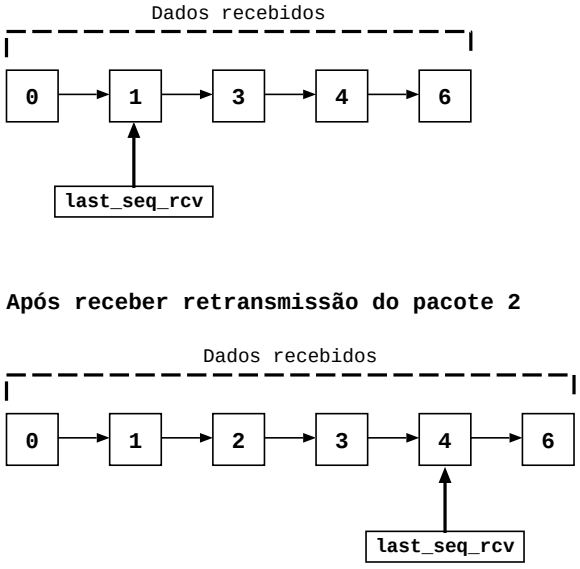

Figura 3.6: Recebimento de retransmissões

A lista de NAKs

Quando a perda de um pacote é detectada, uma requisição de retransmissão para este pacote deve ser enviada. Após enviar a requisição, o membro espera por um determinado tempo pela retransmissão. Se esse tempo de espera terminar antes do pacote requisitado ser recebido, uma nova requisição deve ser enviada. Para limitar o número de requisições enviadas referentes a um mesmo pacote, utiliza-se a lista de NAKs.

Ao detectar a perda de um pacote, é criado um novo nó na lista de NAKs referente ao emissor do pacote. Cada nó da lista contém o número de seqüência do pacote e um campo com o número de requisições que já foram enviadas. Este campo inicialmente tem o valor 1, e é incrementado a cada novo envio de requisição.

Se o número de requisições enviadas ultrapassar um determinado limite imposto pela aplicação, alguma ação pode ser executada. Se o número de requisições for grande, e mesmo assim não foi possível recuperar o dado perdido, a aplicação pode, por exemplo, ser finalizada.

3.4.4 Supressão de NAKs e retransmissões

Se todos os membros que detectassem perdas enviassem, imediatamente, um NAK relativo àquela perda, poderia haver, caso os pacotes perdidos fossem os mesmos, um grande número de requisições duplicadas na rede. Além de ocupar recursos, essas duplicatas também iriam gerar mais processamento para os membros que as recebessem. Para diminuir o número de mensagens duplicadas, uma abordagem que pode ser usada é a supressão de mensagens, utilizando-se temporizadores aleatórios como proposto em [33, 17].

Na RML, a supressão de mensagens é utilizada para diminuir o número de NAKs e retransmissões duplicados. Ao detectar uma perda, o membro escalona o envio do NAK usando um temporizador aleatório. Caso receba um NAK de outro membro do grupo antes do temporizador expirar, o envio é cancelado e o membro se comporta

como se ele tivesse enviado o NAK. De forma análoga, ao receber um NAK e escalonar uma retransmissão, os membros utilizam temporizadores aleatórios. O membro onde o temporizador expirar primeiro enviará a retransmissão que, possivelmente, irá suprimir as retransmissões dos outros membros.

3.5 A Lista de Eventos

O controle das principais ações, ou eventos, é feito através da lista de eventos. Cada nó desta lista é composto por uma instância da estrutura EVENT_LIST, que é formada basicamente pelos seguintes campos:

- `member_id`: ponteiro para a identificação de um membro na `cache`;
- `action`: indica o tipo de evento, ou seja, a ação a ser executada;
- `timer_value`: indica quando o evento deve ser executado;
- `sn`: indica, quando necessário, o número de seqüência de um pacote relacionado ao evento;

A lista de eventos é ordenada de acordo com o tempo de execução de cada evento, onde o primeiro nó é relativo ao primeiro evento a ser executado. O valor armazenado no campo `timer_value` é, na verdade, o intervalo de tempo entre o último evento e o disparo do evento em questão. No exemplo mostrado na Figura 3.7 existem três eventos escalonados. O primeiro será executado após 4 unidades de tempo. O segundo, será disparado 6 unidades de tempo após o primeiro evento. Por fim, o terceiro evento será executado 7 unidades de tempo após o segundo. Portanto, considerando o tempo absoluto, os eventos seriam executados, em 4, 10 e 17 unidades de tempo.

A ação que deve ser executada quando um evento dispara é indicada no campo `action`. Essa ação pode ser: enviar um NAK, enviar uma retrasmissão ou enviar uma mensagem REFRESH. Cada ação pode precisar de diferentes informações para

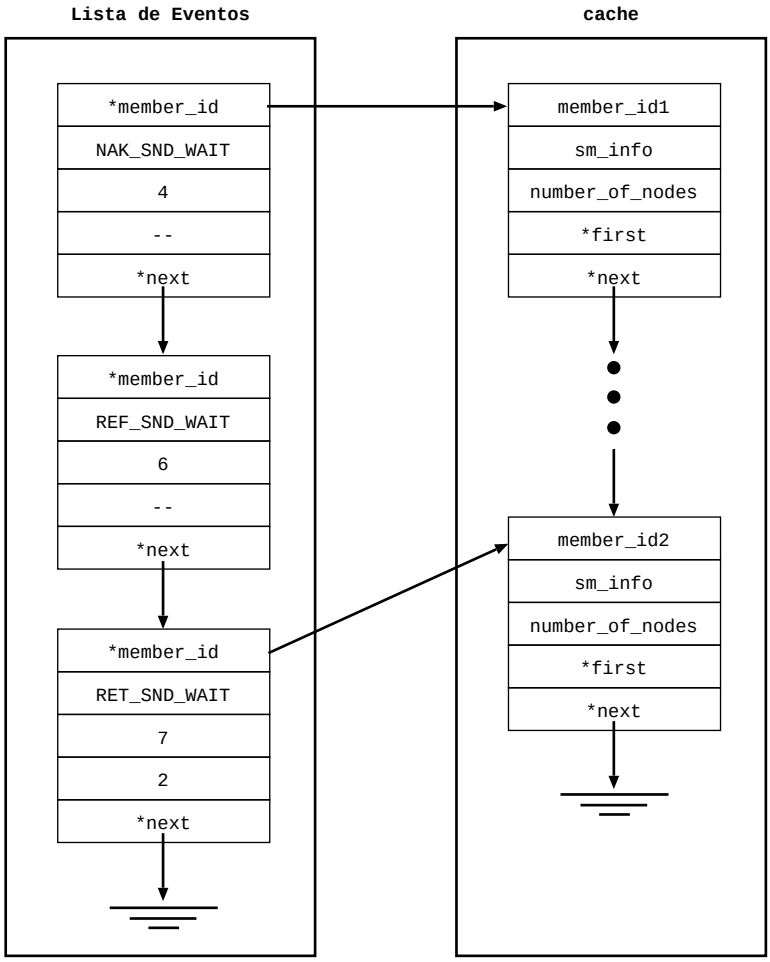

Figura 3.7: Exemplo de lista de eventos

ser executada, por isso, algumas destas informações são armazenadas na própria lista de eventos. Para enviar uma retransmissão, por exemplo, é necessário saber o `member_id` do emissor e o número de seqüência do pacote a ser retransmitido. A seguir, os tipos de eventos e suas ações correspondentes são apresentados:

- `NAK_SND_WAIT`: ao disparar este evento, um NAK referente ao membro indicado no campo `member_id` deve ser enviado e um evento do tipo `RET_RCV_WAIT` deve ser escalonado;
- `RET_RCV_WAIT`: ao disparar este evento, caso ainda não tenha sido atingido o número máximo de NAKs enviados, referente ao membro indicado no campo `member_id`, um evento `NAK_SND_WAIT` deve ser escalonado;
- `RET_SND_WAIT`: ao disparar este evento, caso o pacote com número de seqüência

`sn`, enviado pelo membro indicado no campo `member_id`, esteja disponível na `cache`, uma retransmissão deve ser enviada;

- `REF_SND_WAIT`: ao disparar este evento, uma mensagem do tipo REFRESH deve ser enviada, contendo o número de seqüencia do último pacote de dados enviado por este membro. Além disso, um novo evento REFRESH deve ser escalonado;

- `LEV_SND_WAIT`: ao disparar este evento, as rotinas para término da aplicação devem ser executadas.

3.5.1 Temporizadores

O desempenho da supressão de mensagens depende, principalmente, da escolha dos temporizadores utilizados para cada ação. Existem na literatura algumas propostas para determinar a distribuição e os valores para os temporizadores. Atualmente, na RML, o cálculo dos temporizadores é feito de forma similar ao proposto inicialmente para o SRM [17]. De acordo com esta proposta, cada membro calcula, usando uma distribuição uniforme, os valores para os temporizadores dentro de intervalos pré-definidos, baseados no retardo de propagação em relação aos emissores. Na RML atualmente é possível utilizar também uma distribuição exponencial no cálculo dos parâmetros.

Durante uma sessão, para cada fonte, três temporizadores podem ser necessários: T_{nak}, referente ao tempo de espera antes de enviar um NAK (evento `NAK_SND_WAIT`); T_{wait}, relativo ao tempo de espera por uma retransmissão (evento `RET_RCV_WAIT`) e T_{ret} relativo ao tempo que o membro deve aguardar antes de enviar uma retransmissão (evento `RET_SND_WAIT`). Esses temporizadores são calculados de acordo com a Tabela 3.1, onde R é uma estimativa do retardo de propagação do membro até a fonte emissora dos dados e A, B, C, D, E e F são constantes.

Na atual implementação da RML, os valores para R e para os parâmetros A, B, C, D, E e F são indicados no início da sessão. Porém, existem propostas, como as encontradas em [27, 29], onde o cálculo dos temporizadores é feito dinamicamente

Identificação do temporizador	Intervalo onde o temporizador é escolhido
T_{nak}	$(A \cdot R, (A + B) \cdot R)$
T_{wait}	$(C \cdot R, (C + D) \cdot R)$
T_{ret}	$(E \cdot R, (E + F) \cdot R)$

Tabela 3.1: Temporizadores aleatórios usados na RML

através de informações obtidas durante a sessão multicast. Estas propostas podem ser facilmente implementadas nas futuras versões da RML.

Capítulo 4

O TANGRAM-II Whiteboard

ESTE capítulo descreve as principais características do TANGRAM-II Whiteboard (TGWB), uma ferramenta gráfica para trabalho cooperativo entre grupos de usuários. O TGWB é um dos componentes do TANGRAM-II e utiliza a RML para fazer a transmissão multicast confiável dos dados entre os participantes, mas é totalmente independente do TANGRAM-II. Atualmente é distribuído com a ferramenta TGIF [10]

4.1 O Ambiente de Modelagem TANGRAM-II

O TANGRAM-II é uma ferramenta que possui um ambiente de modelagem desenvolvido para pesquisa e ensino, que permite a criação de diversos tipos de modelos [8, 14]. Entre esses modelos, o foco principal é desenvolver e resolver modelos de sistemas de computação e comunicação.

A primeira versão da ferramenta, que utilizava a linguagem Prolog, surgiu em 1991 [7]. Novas funcionalidades e a mudança da implementação de Prolog para C/C++, tornaram possível o desenvolvimento de uma nova versão do TANGRAM-II em 1997 [8]. Com uma nova interface gráfica, usando Java, iniciou-se, a partir de 1998, um novo período de desenvolvimento com a inclusão de novas características, que permitiram consolidar o ambiente de modelagem. Entre as novas características

estão incluídos novos métodos para solução de modelos [13, 38], um ambiente para engenharia de tráfego [23] e novos módulos para tornar possível o trabalho cooperativo entre os usuários da ferramenta. Os módulos que, atualmente, compõem o TANGRAM-II são o Ambiente de Modelagem, a ferramenta para geração de tráfego, a ferramenta para transmissão de voz sobre IP VivaVoz [16, 15] e o Whiteboard, como ilustra a Figura 4.1.

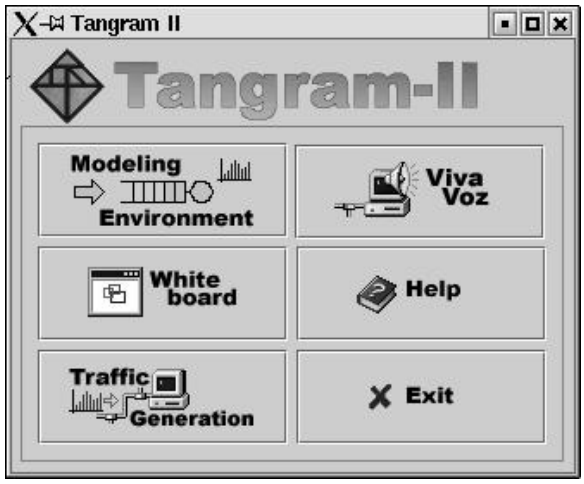

Figura 4.1: Ferramenta TANGRAM-II

A partir do ambiente de modelagem (*Modeling Environment*), apresentado na Figura 4.2 pode-se criar um modelo, resolvê-lo e retirar medidas de interesse. Os modelos no TANGRAM-II são descritos como proposto em [7], utilizando-se o paradigma de orientação a objetos. Através do TGIF (*Tangram Graphic Interface Facility*) [10], uma ferramenta de domínio público para desenho vetorial, é possível fazer a descrição gráfica dos modelos. Para modelar um sistema, o usuário pode utilizar objetos predefinidos, disponíveis na ferramenta, ou criar novos a partir de um objeto padrão. Após a criação do modelo, pode-se resolvê-lo através de métodos analíticos ou através de simulação. Por fim, depois de resolvido o modelo, a ferramenta oferece a possibilidade de obter algumas medidas.

A ferramenta para geração de tráfego disponível no TANGRAM-II, denominada *Traffgen*, permite gerar tráfego IP ou ATM e criar um arquivo de registro, ou *trace*, do qual é possível obter algumas medidas, como, por exemplo, probabilidade de perda.

Os outros dois módulos, o VivaVoz e o Whiteboard, podem ser usados para permitir o trabalho cooperativo entre usuários. O VivaVoz é uma ferramenta para transmissão de voz em uma rede IP, que utiliza mecanismos avançados para garantir uma boa qualidade de áudio [16, 15]. Usando o VivaVoz, dois usuários podem se comunicar via voz através da Internet. O TANGRAM-II Whiteboard, também chamado TGWB, é uma versão do TGIF, com a inclusão de novas funcionalidades, onde um grupo de usuários pode, entre outras tarefas, trabalhar cooperativamente no desenvolvimento de um mesmo modelo, desenho ou mesmo *slides* de uma apresentação.

Nas seções seguintes, são apresentadas as principais características do TGWB, como a transmissão multicast confiável, através do uso da RML, e o mecanismo utilizado para garantir a apresentação consistente dos dados para todos os participantes de uma sessão.

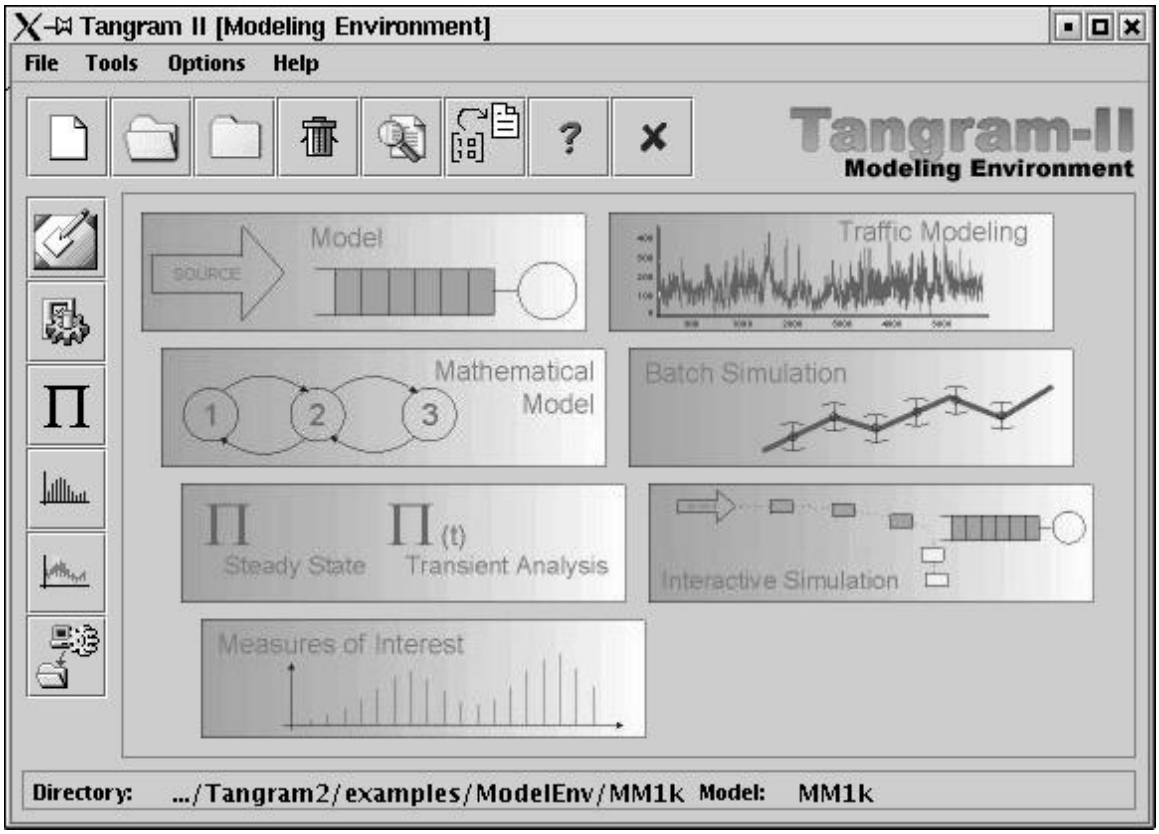

Figura 4.2: Ferramenta TANGRAM-II - Ambiente de Modelagem

4.2 Principais características do TGWB

O TGWB é uma ferramenta para trabalho em grupo, desenvolvida a partir do aplicativo para desenhos vetoriais TGIF e da biblioteca RML. Além de poder utilizar as funcionalidades para desenho vetorial do TGIF, os usuários podem construir, de forma cooperativa, modelos que mais tarde podem ser resolvidos usando o ambiente de modelagem do TANGRAM-II.

Utilizando as funcionalidades oferecidas pelo TGIF, o usuário tem acesso a todas as operações de uma ferramenta de desenho vetorial. A capacidade do TGIF foi estendida para que ele se tornasse um *whiteboard* distribuído e pudesse desempenhar outras funções, como se tornar uma ferramenta para trabalho cooperativo ou para ensino a distância.

Um dos principais problemas no desenvolvimento de uma aplicação distribuída é garantir a consistência dos dados entre todos os participantes e, além disso, oferecer um bom desempenho da aplicação. No caso de um *whiteboard* distribuído, a questão é como garantir que os dados apresentados para todos os membros de uma sessão sejam os mesmos. Existem basicamente dois problemas que podem causar inconsistência: perda de pacotes e execução de comandos em ordens diferentes para cada membro de uma sessão. O primeiro problema é resolvido utilizando uma forma confiável para transmissão dos dados. O segundo problema depende da implementação de um esquema de controle para garantir a consistência da apresentação dos dados em cada membro.

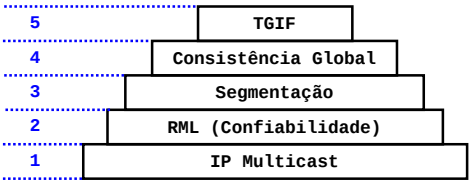

Figura 4.3: Camadas componentes do TGWB

Para transformar o TGIF em um *whiteboard* distribuído e resolver os problemas de perda e inconsistência, foram implementadas novas rotinas no TGIF. Estas rotinas foram implementadas em camadas, como mostra a Figura 4.3. As camadas

referentes à consistência global e a segmentação foram implementadas em trabalhos anteriores, como pode ser visto em [12]. Porém, como a camada relativa à transmissão multicast foi completamente reescrita, foram feitos alguns ajustes para melhorar a interatividade da ferramenta. A ferramenta resultante foi denominada TGWB e suas camadas serão descritas a seguir, começando das camadas mais altas.

4.3 O TGIF

O TGIF é uma ferramenta de desenho muito completa, e através dela os usuários podem criar desenhos vetoriais e realizar várias operações sobre eles. Os desenhos criados pelos usuários, como círculos, quadrados e polígonos em geral, são representados por objetos. Diversas operações podem ser executadas sobre estes objetos, como mudar o tamanho, cor e preenchimento, realizar rotações, agrupar um conjunto de objetos, entre outras.

Outra característica interessante do TGIF é a possibilidade de criar várias páginas dentro de um mesmo arquivo. Esta característica permite que o TGIF seja usado como uma ferramenta de apresentação de *slides*. O modo *Slide Show*, onde cada página representa um *slide* que é apresentado em tela cheia, tem sido usado em palestras e apresentações em geral.

Uma das funcionalidades principais do TGIF é a descrição de modelos para o TANGRAM-II. Os modelos são compostos de objetos, como mostra o exemplo da Figura 4.4. Neste exemplo, está representado um sistema $M/M/1/k$, composto por dois objetos: uma fonte Poisson e um servidor com taxa de serviço exponencial. A fonte Poisson envia pacotes para o servidor, que armazena os pacotes em uma fila antes de processá-los. Os objetos do TANGRAM-II possuem diversos atributos, que são editados pelo usuário. A Figura 4.5 exibe o objeto Servidor e seus principais atributos.

Apesar das diversas características oferecidas pelo TGIF, ele não supria uma necessidade importante do TANGRAM-II: o trabalho cooperativo. Para suprir essa necessidade, o TGIF deveria poder ser usado de forma distribuída em uma sessão de

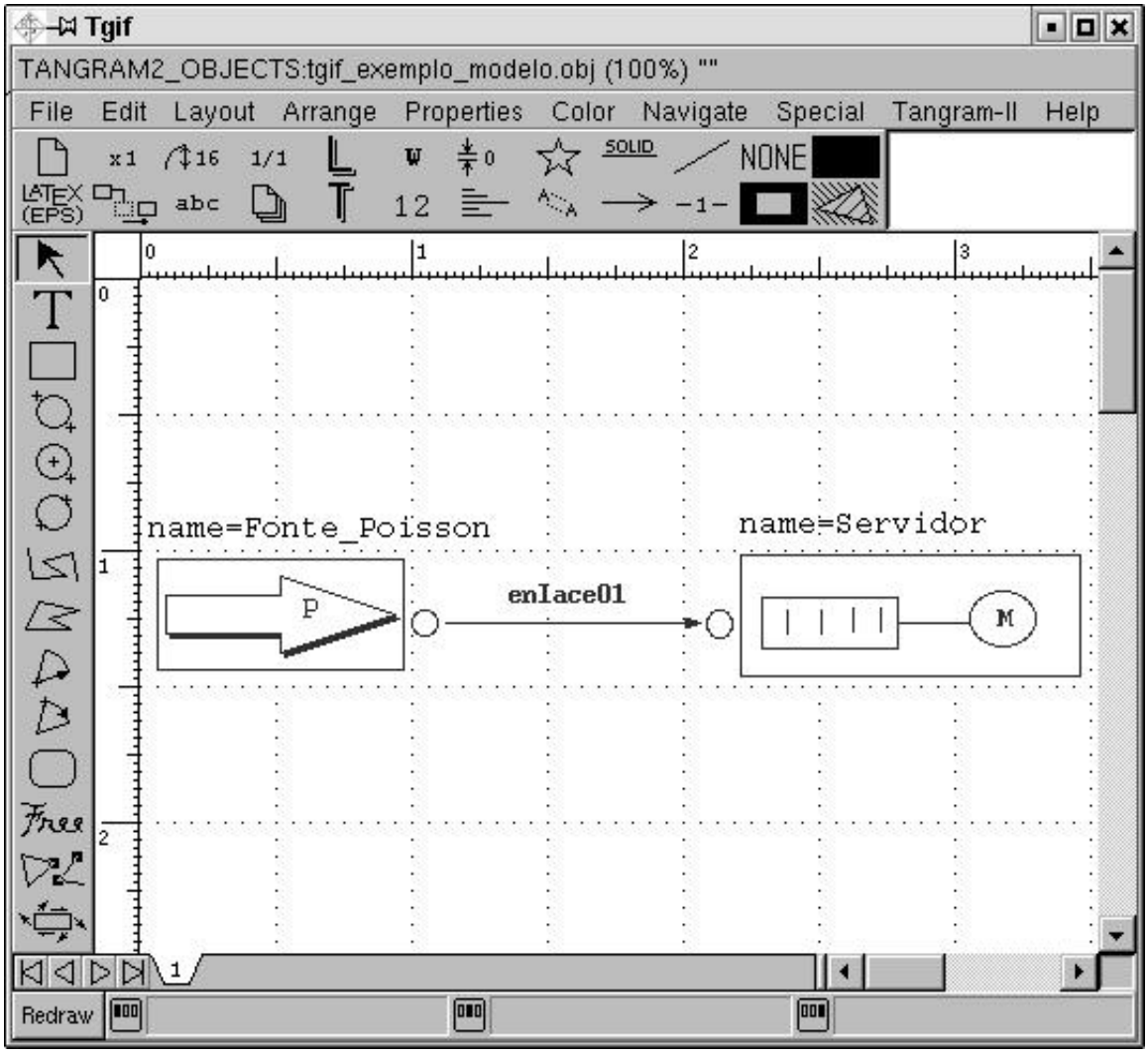

Figura 4.4: Um exemplo de modelo descrito no TGIF

trabalho de um grupo de usuários. Com este objetivo, foram implementadas rotinas que, após o início de uma sessão, permitem que os comandos gerados em cada interface local, sejam enviados para todos os outros membros e executados em suas interfaces locais. Entre as novas funcionalidades, necessárias para o uso distribuído do TGIF, encontram-se a garantia da consistência global e do recebimento confiável dos dados. Essas funcionalidades são apresentadas nas próximas seções.

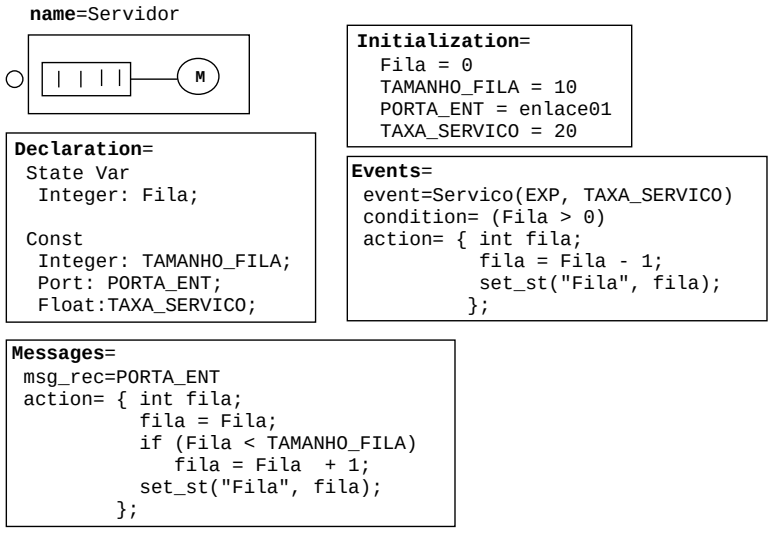

Figura 4.5: Exemplo de um objeto TANGRAM-II e seus atributos

4.4 Consistência Global e Segmentação

Nesta seção, o problema de estabelecer uma ordem única para execução dos comandos é apresentado. Em seguida, será discutida a solução proposta. É importante ressaltar que a análise considera que os dados são recebidos em ordem e sem perdas. Esta consideração é garantida pelo uso das rotinas da RML para transmissão confiável dos dados.

4.4.1 Análise do problema

Como descrito anteriormente, quando um membro qualquer executa um comando no TGWB, este comando é enviado para ser executado também nas interfaces locais dos outros membros. A questão é como garantir que estes comandos sejam executados na mesma ordem por todos os membros da sessão.

Para resolver este problema, foi utilizada a abordagem descrita, por exemplo em [6] para análise de uma computação distribuída genérica. Os nós desta computação representam os membros da sessão do TGWB e os eventos são associados com os comandos gerados por estes nós. Quando um nó gera um evento, uma mensagem é enviada para cada um dos outros nós. Portanto, a questão é determinar uma ordem única para os eventos distribuídos.

Um evento é uma unidade fundamental de uma computação distribuída. Na situação discutida nesta seção, um evento é a geração de um comando em uma interface local. A análise de uma computação distribuída é feita sobre o conjunto de eventos, denominado $\mathcal{E}$. Cada evento ϵ de $\mathcal{E}$ é definido pela tupla:

$$\epsilon = (n_i, \tau, \Lambda, m)$$

Onde

- n_i: é a identificação do nó que gerou o evento;
- τ: é o tempo no qual o evento ocorreu, dado pelo relógio local de n_i;
- Λ: é o conjunto de mensagens, ou seja, o conjunto dos eventos gerados pelos outros membros da sessão, que influenciam ϵ;
- m: é a mensagem enviada para todos os outros membros em conseqüência do evento ϵ.

Essa definição de evento é geral, pois o critério para definição do conjunto de mensagens é deixado em aberto e, conseqüentemente, deve ser definido pela aplicação. No caso de um *whiteboard*, pode-se considerar que um evento influencia outro se, por exemplo, ambos atuam sobre um mesmo objeto ou sobre uma mesma área da interface.

Os eventos que compõem o conjunto $\mathcal{E}$ são fortemente correlacionados, já que uma mensagem enviada por um nó pode estar no conjunto de mensagens que influenciam eventos em outros nós. Portanto, é possível definir uma relação binária, identificada por $\rightarrow$, sobre o conjunto de eventos $\mathcal{E}$:

Definição 1 *Se ϵ_1 e ϵ_2 são eventos, então $\epsilon_1 \rightarrow \epsilon_2$ se, e somente se, uma das seguintes condições se aplica [6]:*

i) ϵ_1 e ϵ_2 são gerados no mesmo nó, nos tempos τ_1 e τ_2 respectivamente, tal que

$\tau_1 < \tau_2$. *Além disso, nenhum outro evento ocorre no tempo* τ, *tal que* $\tau_1 < \tau < \tau_2$

ii) ϵ_1 *e* ϵ_2 *são gerados em nós distintos e a mensagem* m, *que é enviada como resultado do evento* ϵ_1, *está no conjunto de mensagens* Λ *de* ϵ_2

A condição i) expressa o conceito intuitivo de causa-efeito que existe entre eventos gerados em um mesmo nó. A condição ii) representa a relação causa-efeito que existe entre nós distintos. A Figura 4.6 mostra o grafo de precedência $G = (\mathcal{E} , \rightarrow)$, que é um grafo direcionado e acíclico, onde o conjunto de nós é dado pelo conjunto de eventos $\mathcal{E}$, e o conjunto de arestas é determinado pelos pares de eventos relacionados por $\rightarrow$.

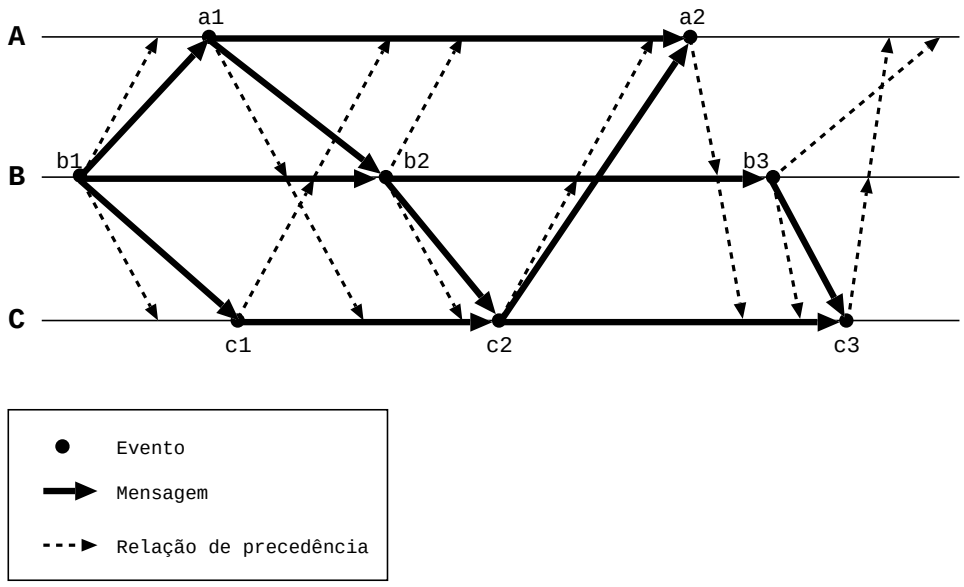

Figura 4.6: Grafo de precedência de uma computação distribuída

O fecho transitivo da relação $\rightarrow$, denominado $\rightarrow^+$, é irreflexivo e transitivo e, portanto, estabelece uma ordem parcial no conjunto de eventos $\mathcal{E}$. Dois eventos ϵ_1 e ϵ_2 que não são relacionados por $\rightarrow^+$ são denominados eventos concorrentes. Na Figura 4.6, os pares de eventos $(a1, c3)$ e $(b1, a2)$ são relacionados por $\rightarrow^+$, enquanto $a1$ e $c1$ são eventos concorrentes.

A relação $\rightarrow^+$ pode ser utilizada para definir os conceitos de passado e futuro de um evento. Estes conceitos serão usados para explicar o modelo de execução do TGWB.

Definição 2 *Para qualquer evento ϵ, define-se [6]:*
$Passado(\epsilon) = \{\epsilon' \mid \epsilon' \rightarrow^+ \epsilon, \epsilon' \in \mathcal{E}\}$
$Futuro(\epsilon) = \{\epsilon' \mid \epsilon \rightarrow^+ \epsilon', \epsilon' \in \mathcal{E}\}$

À primeira vista, pode-se pensar que, para garantir a consistência entre os membros de uma sessão do TGWB, bastaria executar os comandos de acordo com os conjuntos *Passado* e *Futuro*, usando a seguinte regra:

Regra 1 *O comando relativo ao evento ϵ deve ser executado depois de todos os comandos do conjunto $Passado(\epsilon)$ e antes de todos os comandos no $Futuro(\epsilon)$.*

Porém, os conjuntos $Passado(\epsilon)$ e $Futuro(\epsilon)$ foram definidos baseados em uma relação de ordem parcial $\rightarrow^+$ e, por isso, podem existir eventos que não pertencem a nenhum desses conjuntos. Portanto, se uma ordem única não for definida, eventos concorrentes podem ser executados em ordens diferentes em cada nó, levando a um estado de inconsistência. O exemplo seguinte, ilustrado na Figura 4.7, mostra uma situação onde um estado de inconsistência é alcançado.

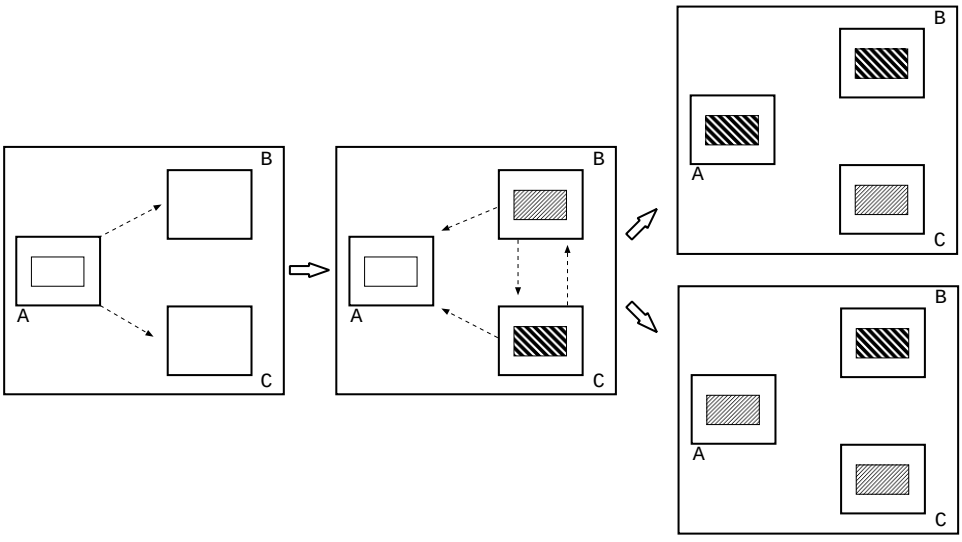

Figura 4.7: Exemplo de inconsistência

Neste exemplo, considera-se uma sessão do TGWB com três membros: A, B e C. Em um determinado momento, A constrói, em sua interface local, um retângulo sem preenchimento. Este evento dispara o envio de mensagens para os outros participantes da sessão. Após receberem e executarem o comando de A, B e C modificam

o preenchimento do retângulo aproximadamente ao mesmo tempo. Como B e C executaram a mudança do preenchimento antes de receberem a mensagem associada ao outro, os eventos gerados são considerados concorrentes. Se o critério para a execução de eventos concorrentes for, simplesmente, a ordem de chegada destes eventos, um estado de inconsistência é alcançado. Neste estado, B possui um retângulo com o preenchimento escolhido por C e C possui um retângulo com o preenchimento escolhido por B. O membro A vai ter o retângulo com o preenchimento escolhido por B ou C, dependendo da ordem de chegada das mensagens.

Como a relação de ordem parcial $\rightarrow^+$ não é suficiente para garantir a consistência entre os usuários, é necessário obter uma relação de ordem total, denominada $\rightarrow_t^+$. Uma relação de ordem total é uma relação de ordem parcial que inclui exatamente (ϵ_1, ϵ_2) ou (ϵ_2, ϵ_1) para todos $\epsilon_1, \epsilon_2 \in \mathcal{E}$. Esta relação de ordem total pode ser obtida de $\rightarrow^+$ pela inclusão dos pares de eventos concorrentes.

4.4.2 Relógios Lógicos

A ordenação total dos eventos é obtida através do uso de relógios lógicos, usando o algoritmo proposto por Lamport em [21]. Um relógio lógico é apenas uma maneira de associar um número a um evento, definido com base na relação de ordem entre os eventos de acordo com a seguinte condição:

Condição 1 *Para quaisquer eventos a e b, se $a \rightarrow^+ b$ então $C(a) < C(b)$, onde $C(\epsilon)$ é o valor do relógio lógico associado ao evento ϵ.*

Para implementar o mecanismo de relógios lógicos, cada nó n_i mantém um contador C_i. Uma identificação de tempo $C_i(\epsilon)$ é associada ao evento ϵ gerado no nó n_i. Esta identificação é dada pelo valor de C_i no momento em que foi gerado o evento ϵ. Os contadores C_i são atualizados segundo as seguintes regras:

1. Imediatamente após a geração de um evento, o nó n_i incrementa seu contador C_I em uma unidade;

2. Quando o nó n_i recebe uma mensagem, ele ajusta seu contador C_i para o valor $max\{C_i, C(msg) + 1\}$

Embora este mecanismo estabeleça apenas uma relação de ordem parcial entre os eventos, pode-se utilizar a identificação única dos nós, $Id(n_i)$, para definição de uma relação de ordem total.

Definição 3 *Sejam a e b eventos gerados, respectivamente, pelos nós n_i e n_j. Então $a \rightarrow_t^+ b$ se, e somente se, uma das seguintes condições se aplica [21]:*
i) $C(a) < C(b)$
ii) $C(a) = C(b)$ *e* $Id(n_i) < Id(n_j)$

No TGWB, a identificação dos nós consiste de duas partes: o endereço IP do *host* onde o TGWB está sendo executado e a identificação do processo, ou PID. O uso do PID permite que mais de uma instância do TGWB possa ser executada em um mesmo *host*. A ordem total dos nós é definida pela avaliação lexicográfica da identificação destes nós.

4.4.3 Algoritmo para execução dos comandos no TGWB

A Regra 1, apresentada anteriormente, sugere um algoritmo simples para a execução dos comandos em uma sessão do TGWB. Neste algoritmo, quando um novo comando é gerado ou recebido através de uma mensagem, ele é armazenado até que cada um dos comandos pertencentes ao seu passado tenha sido recebido e executado. Entretanto, este esquema possui dois problemas: (1) a identificação de todo o conjunto de eventos relativos ao passado de um comando pode ser difícil e (2) não oferece o nível desejado de interação entre o usuário e a aplicação.

O primeiro problema consiste em determinar se todos os eventos, pertencentes ao passado de um dado comando c, foram recebidos. A questão é detectar se algum membro gerou um comando no passado de c que ainda não foi recebido. Como visto anteriormente, este problema é resolvido com a utilização de relógios lógicos. O

segundo problema acontece pois os comandos não são executados imediatamente. Neste caso, um comando gerado na interface local poderia não ser executado no momento em que foi gerado, o que não parecia intuitivo para os usuários.

Uma importante observação deve ser feita sobre a relação de ordem total. Essa relação, necessária para garantir a consistência entre os usuários, é muito mais forte do que a relação causa-efeito da semântica do TGWB. Um exemplo é a situação onde dois comandos, gerados por dois usuários diferentes, referem-se a objetos em áreas distintas da interface. Neste caso, a ordem de execução de um comando não afeta a execução do outro. Esta situação é comum em uma sessão do TGWB.

A maioria das aplicações para desenho, como o TGIF, oferecem a possibilidade de "desfazer" e "refazer" um comando, embora a implementação destas funções possa ser bem diferente entre as aplicações. Utilizando a função "desfazer", pode-se reverter os efeitos de um conjunto de comandos. Desta forma é possível "voltar ao passado" de uma sessão do TGWB e, utilizando o comando "refazer", executar os comandos numa ordem diferente. Este mecanismo é denominado *Rollback and Recovery* (algo como Retroceder e Recuperar) e é usado em outras aplicações, como simuladores distribuídos [34]. A principal desvantagem deste mecanismo em um *whiteboard* é que desfazer e refazer certas operações, em tempo real, pode parecer estranho para os usuários. Porém, esta é uma questão menor, visto que os usuários podem se acostumar a este comportamento incomum para poder utilizar uma abordagem que garante a consistência global.

A partir dessas observações foi desenvolvido o algoritmo para execução dos comandos em uma sessão do TGWB. Segundo este algoritmo, quando um novo comando é gerado ou recebido da rede, o primeiro passo é tentar executá-lo. Entretanto, em alguns casos, a execução imediata de um comando não é possível. Por exemplo, quando o comando se refere a um objeto que não existe na interface local. A idéia então é identificar o comando como inconsistente e armazená-lo para ser executado mais tarde. Pode ocorrer também a situação onde o comando recebido viola a relação $\rightarrow_t^+$. Nesta situação, o mecanismo de *Rollback and Recovery* é aplicado e os comandos são executados na ordem correta.

Para implementar este mecanismo, cada membro da sessão mantém uma lista dos comandos gerados ou recebidos por ele. Esta lista é ordenada de acordo com a relação $\rightarrow_t^+$. Cada comando da lista pode estar em um de dois estados: `executado` ou `inconsistente`.

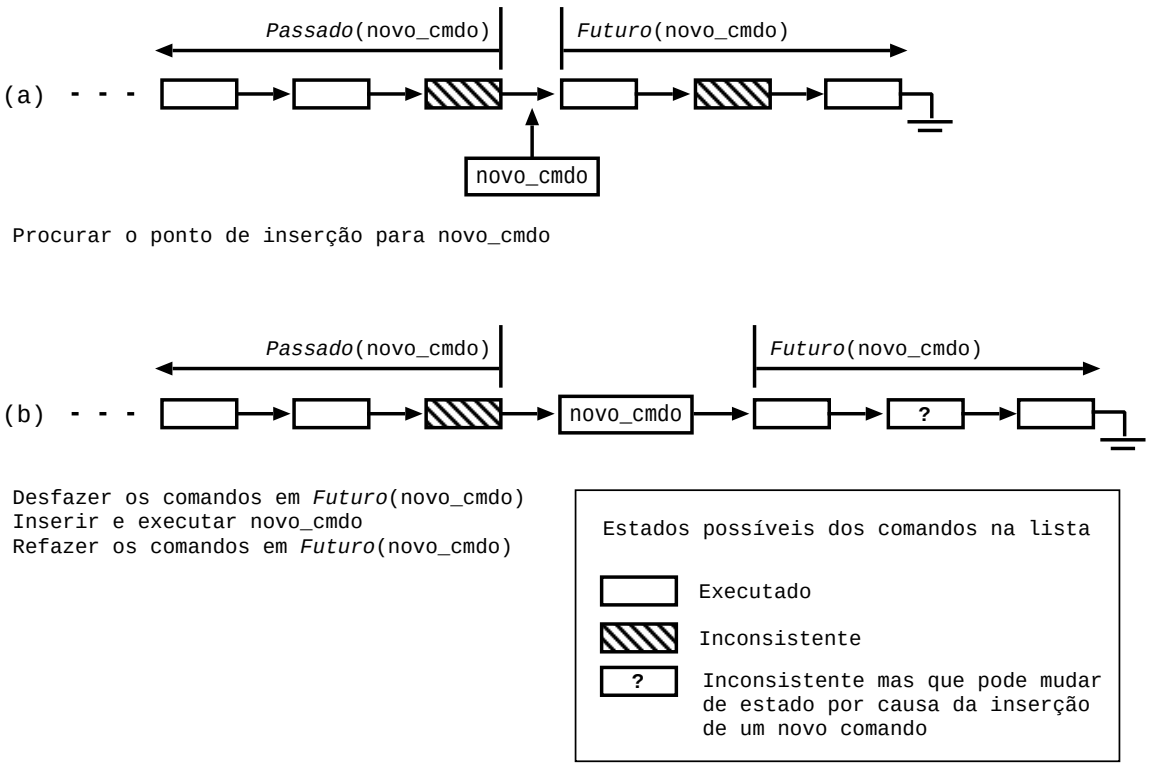

Figura 4.8: Lista de comandos e exemplo de *Rollback and Recovery*

Quando um novo comando (`novo_cmdo`) é recebido da rede, seu ponto de inserção na lista é identificado. Os comandos que se encontram antes deste ponto pertencem ao *Passado*(`novo_cmdo`), enquanto os comandos posteriores constituem o *Futuro*(`novo_cmdo`), como pode ser visto na Figura 4.8(a). De acordo com o a Regra 1, o comando `novo_cmdo` deve ser executado antes de qualquer comando em *Futuro*(`novo_cmdo`). Portanto, é necessário aplicar a função "desfazer" aos comandos do *Futuro*(`novo_cmdo`), que estão no estado `executado`. Assim, `novo_cmdo` pode ser inserido na lista e executado. Por fim, utilizando a função "refazer", os comandos do *Futuro*(`novo_cmdo`) são executados de acordo com a ordem da lista, como mostra a Figura 4.8(b). É possível perceber que um comando que pertence ao *Futuro*(`novo_cmdo`) e que estava no estado `inconsistente`, pode se tornar consistente após a execução de `novo_cmdo`.

Para evitar que a lista de comandos cresça indefinidamente, um mecanismo é executado periodicamente para retirar comandos da lista. Um membro pode retirar um comando c da lista se todos os comandos do $Passado(c)$ já foram recebidos e executados.

4.4.4 Segmentação

Alguns comandos do TGWB podem gerar mensagens grandes, dificultando a sua transmissão. A importação de figuras nos formatos GIF ou JPEG e a operação de copiar e colar um conjunto de objetos são exemplos típicos. Se estes comandos fossem colocados em um único pacote de dados, seria muito provável que em algum ponto da rede esse pacote teria que ser fragmentado, provocando, desta forma um atraso na transmissão dos dados. Pacotes grandes também podem influenciar no desempenho da recuperação, pois a retransmissão desses pacotes é mais custosa e, novamente, pode ser necessário que eles sejam fragmentados na rede.

Para evitar essa situação, o TGWB utiliza rotinas para segmentar os comandos, caso eles sejam muito grandes. Assim, um comando que, por exemplo, representa uma imagem no formato JPEG, vai ser segmentado em vários pacotes de tamanho fixo, que atualmente é de 783 bytes sem o cabeçalho da RML, e só depois será enviado pela rede. Ao receber um desses pacotes, os outros membros podem identificar que estão segmentados. Depois é necessário aguardar a chegada de todos os pacotes referentes ao comando para, só então, poder verificar se o comando pode ser executado.

4.5 As camadas RML e IP Multicast

As camadas 1 e 2 do TGWB (Figura 4.3), implementam as funções para envio e recebimento dos dados. De fato, essas funções são cumpridas pela biblioteca RML, descrita no capítulo 3.

Através das funções da RML, os dados são transmitidos de forma confiável.

Quando um dado é recebido ele é passado para a camada de segmentação. Se um dado é recebido da camada de segmentação, ele é enviado para o grupo multicast.

Capítulo 5

Resultados de Experimentos e o modelo da RML

NESTE capítulo serão apresentados os testes realizados com a biblioteca RML e em seguida os resultados obtidos. Também será feita uma breve apresentação de resultados obtidos a partir de um modelo desenvolvido utilizando-se a ferramenta TANGRAM-II.

5.1 Ambiente de testes

Os experimentos realizados têm como objetivo mostrar a viabilidade do uso da RML em aplicações como o *whiteboard* TGWB. Para isso, o ambiente montado para os testes inclui quatro universidades: a UFRJ, do Rio de Janeiro; a UFF, de Niterói; a UFJF de Juiz de Fora e, finalmente, a UMASS de Amherst nos Estados Unidos. Estas universidades foram escolhidas por possuírem máquinas rodando o sistema operacional GNU/Linux e pela facilidade de acesso a elas.

As máquinas utilizadas têm configurações diferentes entre si, como quantidade de memória e processamento disponíveis, como pode ser visto na Tabela 5.1. Além disso, o número de roteadores e o RTT (*Round-Trip Time*) entre cada uma das máquinas varia conforme a Tabela 5.2.

Nome da Máquina	Localização	Processadores (Quant./Clock)	Memória RAM	Tempo médio de processamento para os pacotes da RML
abacate	UFF	2/1GHz	1GB	300 μs
fenix	UFJF	1/200MHz	64MB	700 μs
ilha	UFRJ	1/1GHz	512MB	500 μs
newworld	UMASS	2/1,2GHz	2GB	300 μs

Tabela 5.1: Descrição da máquinas utilizadas nos experimentos

	abacate	**fenix**	**ilha**	**newworld**
abacate	0/0	11/300ms	8/47ms	17/164ms
fenix	11/258ms	0/0	12/260ms	18/444ms
ilha	9/51ms	12/304ms	0/0	18/237ms
newworld	17/164ms	24/462ms	18/208ms	0/0

Tabela 5.2: Número de roteadores e RTT médio entre os *hosts*

Para realizar os testes foram considerados dois cenários, denominados Cenário 1 e Cenário 2. O Cenário 1 representa uma sessão multicast que possui apenas um emissor, enquanto o Cenário 2 representa uma sessão multicast onde todos os membros podem atuar simultaneamente como emissores e receptores de dados. A seguir serão discutidos os detalhes referentes a esses cenários.

5.1.1 O Cenário 1

O Cenário 1 considera uma sessão multicast com um único emissor de dados. Um exemplo típico deste cenário, considerando o uso do TGWB, é uma apresentação ou aula, onde o apresentador ou professor exibe alguns *slides* e os espectadores apenas assistem à apresentação. Neste caso, não existe uma maior interação entre os membros da sessão multicast.

A partir dos dados obtidos da análise de algumas apresentações feitas no LAND, foi possível perceber que, durante uma apresentação, ocorre a troca de *slides* em intervalos que variam de 30 a 60 segundos. Ao observar cada um dos *slides*, foi constatado que a maioria deles, aproximadamente 70%, continha texto e desenhos vetoriais. Outros *slides*, cerca de 25%, continham algumas pequenas figuras e ra-

ramente, somando apenas 5%, apareciam *slides* com figuras grandes. Considerando esta distribuição, os *slides* foram divididos em três tipos, conforme pode ser visto na Tabela 5.3.

O número médio de pacotes gerados por cada tipo de *slide* depende diretamente do seu conteúdo. Pequenas quantidades de texto e desenhos vetoriais produzem um menor número de pacotes, pois podem ser representados por comandos simples do TGIF. Por outro lado, a transmissão de figuras no formato de mapa de bits, ou *bitmap*, produz um número maior de pacotes, já que todo o mapa de bits que representa a figura deve ser enviado. Como o TGWB tem foco em desenho vetorial, não é comum utilizar grandes imagens em formato *bitmap* nos *slides*.

Tipo	Conteúdo	Freqüência em uma apresentação típica	Pacotes gerados (média)	Tamanho da rajada (bytes)
A	Texto e desenhos vetoriais	70%	25	20575
B	Texto e pequenas figuras em *bitmap*	25%	50	41150
C	Outras figuras em *bitmap*	5%	100	82300

Tabela 5.3: Distribuição dos *slides* conforme a freqüência e tamanho

Com as informações da distribuição da freqüência e do número médio de pacotes gerados para cada grupo de *slides*, foi construída uma aplicação para os experimentos. Essa aplicação pode ser executada em dois modos: o modo emissor e o modo receptor. No modo emissor, a aplicação envia uma rajada de dados, que representa a transmissão de um *slide*, em intervalos que variam aleatoriamente de 30 a 60 segundos. A probabilidade de que uma rajada de dados represente um determinado tipo é igual à freqüência relativa ao tipo, como visto na Tabela 5.3. Rajadas do tipo A, por exemplo, ocorrem em 70% das transmissões. No modo receptor, a aplicação apenas aguarda a recepção dos pacotes.

Nos experimentos realizados, um segmento de dados tem 783 bytes, que é o

tamanho recebido da camada de segmentação do TGWB (seção 4.4.4). Ao enviar os dados, é adicionado o cabeçalho da RML e, portanto, os pacotes de dados que trafegam na rede possuem 823 bytes. Com estes valores, é possível perceber que para transmitir um *slide* do tipo A, serão enviados 22395 bytes através da rede. *Slides* dos tipos B e C geram, respectivamente, 82300 e 166400 bytes.

5.1.2 O Cenário 2

O segundo cenário, que também foi considerado nos testes, consiste de uma sessão onde os usuários realizam um trabalho cooperativo, como por exemplo a construção de um modelo do TANGRAM-II. As principais diferenças deste cenário, em relação ao cenário de apresentação discutido anteriormente, são o número de emissores, o intervalo entre as transmissões de dados e a quantidade de pacotes enviados em cada transmissão.

No Cenário 2, ao invés de enviar grandes rajadas, os comandos são enviados em pequenas rajadas em intervalos que variam de 1 a 60 segundos. Além disso, a probabilidade de inserção de figuras no formato *bitmap* e, portanto, o envio de grandes rajadas de pacotes é bem pequena visto que esta não é uma operação que se repete muitas vezes na construção de um modelo.

5.1.3 Configuração da RML

Um exemplo típico de configuração da RML, utilizada nos testes, é mostrado na Figura 5.1. Em cada um dos experimentos alguns parâmetros foram modificados para observar seu impacto no desempenho da aplicação e tais modificações serão discutidas posteriormente. Os retardos utilizados para o cáculo dos temporizadores, conforme mostrado na seção 3.5.1, são calculados no início de cada rodada de testes, utilizando o RTT como estimativa. O RTT consiste no tempo entre o envio de um pacote ICMP e o recebimento de uma resposta relativa a esse pacote. Portanto, antes de iniciar uma sessão, os *hosts* participantes trocam pacotes ICMP entre si para calcular o RTT. São enviados 50 pacotes ICMP para cada *host* e depois retirada

a média (RTT_{med}) do RTT relativo a esses pacotes.

O valor colocado no arquivo de configuração deve ser relativo ao retardo em uma direção, do receptor ao emissor. Porém, o cálculo desse retardo não é trivial e, por isso, foi usado como estimativa o valor de $RTT_{med}/2$. Para calcular os temporizadores para os *hosts* que não foram explicitamente identificados no arquivo de configuração, é utilizado o valor indicado como DEFAULT, que é uma média simples dos valores dos temporizadores dos *hosts* identificados. Os valores da Tabela 5.4 constituem um exemplo, apresentando a estimativa da máquina `ilha` em um dos experimentos.

```
#RML Configuration File
VERSION=1.0
TRANSMISSION_MODE=0
DEST_IP=226.2.2.2
DEST_PORT=15151
TTL=0
MICROSLEEP=2000
LOG_FILE=log
TIMER_DISTRIBUTION=0
TIMER_PARAM_A=2
TIMER_PARAM_B=2
TIMER_PARAM_C=5
TIMER_PARAM_D=2
TIMER_PARAM_E=2
TIMER_PARAM_F=2

HOSTS_IDENTIFIED=3
DEFAULT 98
abacate 25
fenix 152
newworld 118

MAX_NAK=100
MAX_MEMBER_CACHE_SIZE=1000
NEW_USER_SUPPORT=0
STATISTICS=0
REFRESH_TIMER=10
LOSS_PROB=0
LEAVE_GROUP_WAIT_TIME=1000000
RCV_BUFFER_SIZE=10000
```

Figura 5.1: Configuração da RML usada nos testes

É importante ressaltar que os valores da Tabela 5.4 apresentam apenas uma média dos valores do RTT. Nos testes esses valores podem variar bastante, tanto para cima quanto para baixo. De acordo com os experimentos realizados, somente o RTT com relação à `newworld` não sofre grandes variações. Já da máquina `fenix` para a máquina `ilha`, o valor do RTT esteve no intervalo entre 32 e 1225 milissegundos.

Hosts identificados	RTT_{med} em relação ao *host*	Estimativa do retardo ($\frac{RTT_{med}}{2}$)	Valor DEFAULT
abacate	50ms	25ms	
fenix	304ms	152ms	98ms
newworld	236ms	118ms	

Tabela 5.4: Exemplo de estimativa de retardo para os temporizadores na máquina ilha

O mcastproxy

Uma última, mas não menos importante, observação deve ser feita com relação ao ambiente utilizado para os experimentos. No período de realização dos testes não foi possível estabelecer uma transmissão multicast nativa entre as universdidades participantes. Para contornar esse problema, foi criado um pequeno programa, denominado `mcastproxy`, que se encarregou de fazer a ligação entre as universidades, como ilustra a Figura 5.2.

O objetivo é que dentro de cada universidade a transmissão seja feita usando multicast. Cada instância do `mcastproxy` participa do grupo multicast referente à universidade a que ela pertence. Todos os pacotes recebidos dos usuários "locais" via multicast são encaminhados, usando UDP unicast, para as instâncias do `mcastproxy` das outras universidades. Os pacotes recebidos via unicast são encaminhados para o grupo multicast, completando assim a comunicação entre as universidades. Dessa forma, fica transparente para a aplicação o envio dos pacotes entre as universidades participantes, visto que o `mcastproxy` não modifica o conteúdo dos pacotes que ele encaminha.

5.2 Resultados dos experimentos

O objetivo principal dos experimentos realizados com a RML é verificar se a biblioteca cumpre seu papel principal: entregar os pacotes ordenados e sem perdas para a aplicação. Além disso, os experimentos também mostram o comportamento do mecanismo de recuperação, com relação ao tempo médio de recuperação de per-

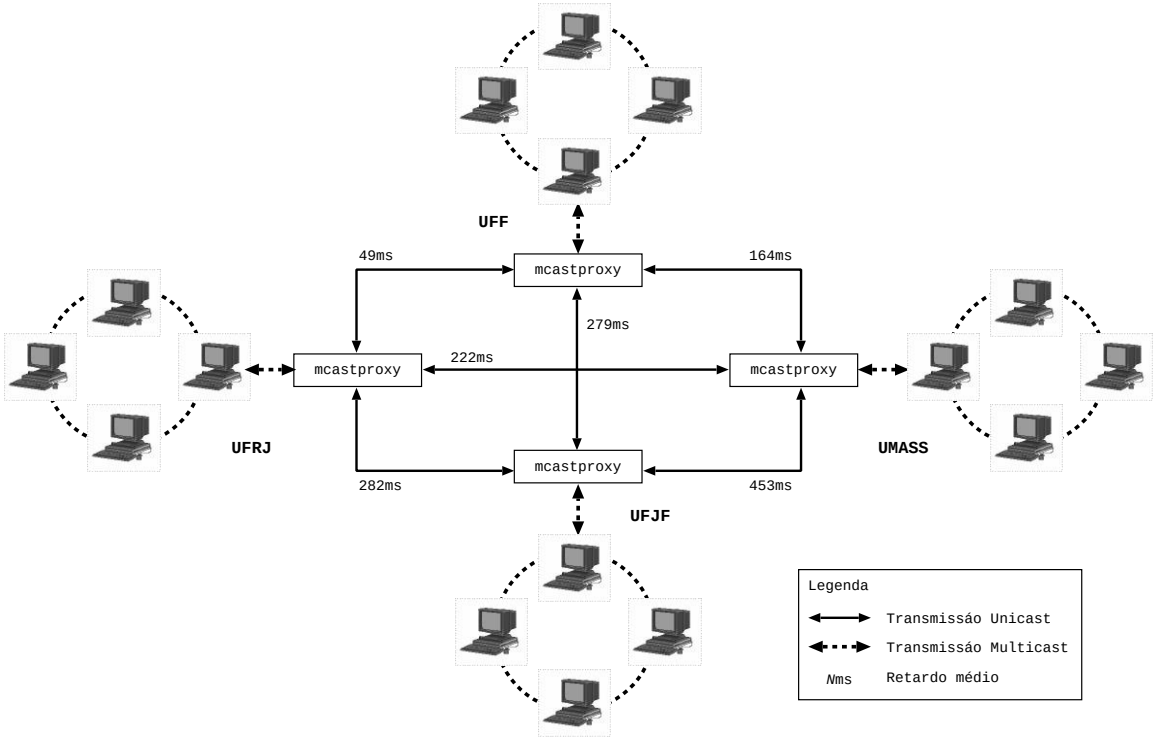

Figura 5.2: Ambiente de testes considerando o mcastproxy

das, ao número médio de NAKs e retransmissões enviados e à supressão de NAKs e retransmissões.

Em todos os testes, todos os pacotes foram recebidos corretamente e em tempo satisfatório pela aplicação. No caso de um *whiteboard*, não é recomendável que o tempo de recuperação passe de alguns poucos segundos, pois caso contrário pode complicar o trabalho dos usuários. A maior influência do tempo de recuperação é sobre o mecanismo de *Rollback and Recovery* (seção 4.4), pois dados que chegam "atrasados" podem provocar muitas operações de desfazer/refazer, o que poderia criar situações incômodas para os usuários.

5.2.1 Relação entre os parâmetros dos temporizadores

O primeiro passo foi definir os valores dos parâmetros A,B,C,D,E e F, de maneira que o tempo de espera por retransmissão fosse o mínimo suficiente pra receber uma retransmissão do emissor. O cálculo é feito baseado no emissor porque apesar de que qualquer membro da sessão que possua o pacote requisitado seja capaz de realizar

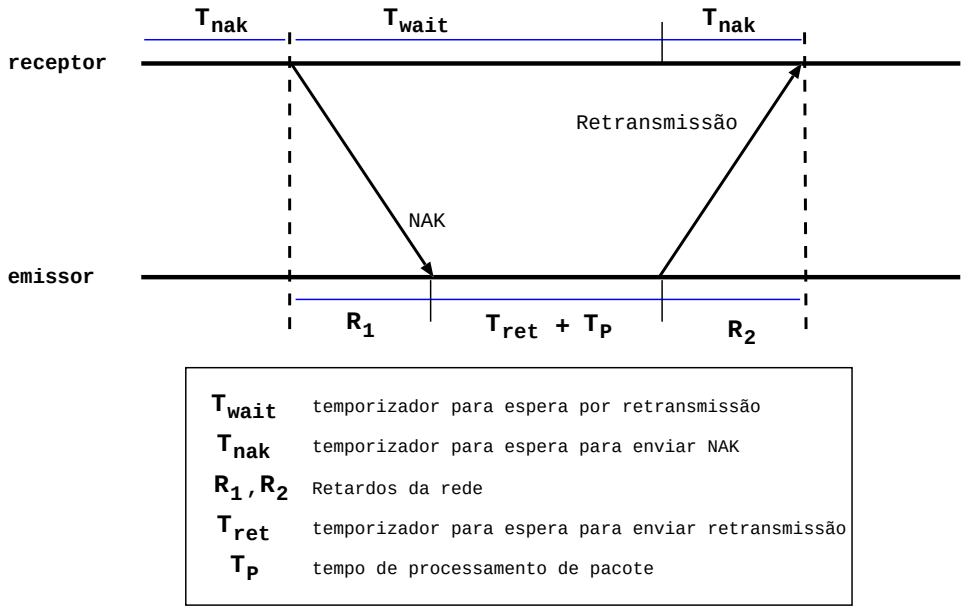

Figura 5.3: Relação entre os parâmetros

a retransmissão, o emissor é o único membro que, com certeza, possui o pacote requerido. A Figura 5.3 ilustra o ambiente usado no cáculo da relação entre os temporizadores.

De acordo com a Figura 5.3, o tempo de espera será suficiente para que o emissor atenda a uma requisição se a seguinte inequação for satisfeita:

$$min\{T_{wait}\} + min\{T_{nak}\} > R_1 + max\{T_{ret}\} + T_P + R_2 \tag{5.1}$$

É importante notar que uma retransmissão tem o mesmo efeito de supressão se chegar no período de T_{wait} ou de T_{nak}. Assumindo os valores dos intervalos para cálculo dos temporizadores mostrados na Tabela 3.1 da seção 3.5.1 e considerando $R_1 = R_2 = R$ e $T_P \approx 0$ a inequação 5.1 pode ser escrita da seguinte forma:

$$C \cdot R + A \cdot R > 2 \cdot R + (E + F) \cdot R \tag{5.2}$$

Como a intenção é descobrir qual a relação que garante que o tempo de espera por uma retransmissão vai ser suficiente, o parâmetro C deve ser isolado na inequação 5.2, obtendo a seguinte relação:

$$C > 2 + E + F - A \tag{5.3}$$

De acordo com a inequação 5.3, se escolhermos $A = B = D = E = F = 1$ então $C = 4$. Como poderá ser verificado nos resultados mostrados a seguir, a relação da inequação 5.3 só garante o envio de apenas um NAK por pacote perdido se não houver perda de NAKs e se o retardo da rede não variar muito.

5.2.2 Resultados referentes ao Cenário 1

Para a representação do Cenário 1, a máquina `ilha` foi escolhida para ser o *host* emissor dos dados. Com os parâmetros ajustados de acordo com a inequação 5.3, o esperado era que fosse enviado apenas um NAK relativo a cada pacote perdido. Porém, como o retardo na rede pode variar durante a sessão multicast, nem sempre o valor dos parâmetros é suficiente para garantir a eficiência do envio de NAKs.

	NAKs enviados			Tempo de recuperação (ms)		
Parâmetros	Min	Média	Max	Min	Média	Max
$A = B = D = E = F = 1$ $C = 4$	1	1,61	3	60,142	197,036	350,189
$A = B = D = E = F = 2$ $C = 5$	1	1,09	2	70,124	254,255	544,520
$A = B = D = E = F = 3$ $C = 6$	1	1,00	1	90,139	484,580	860,162

Tabela 5.5: NAKs enviados e tempo de recuperação em relação à mudança dos parâmetros na fenix

Para verificar essa situação foram realizados experimentos onde os parâmetros dos temporizadores assumiram três valores distintos, mas sempre mantendo a relação da inequação 5.3. A Tabela 5.5 apresenta os resultados obtidos considerando o *host* receptor `fenix`. De acordo com estes resultados, é possível perceber que o número médio de NAKs enviados por pacote perdido diminui com o aumento do valor dos parâmetros. Isso acontece porque esse aumento acaba por compensar a variação do retardo da rede e a perda de NAKs. Portanto, para este experimento, a melhor configuração da RML foi a que utilizou $A = B = D = E = F = 3$ e $C = 6$.

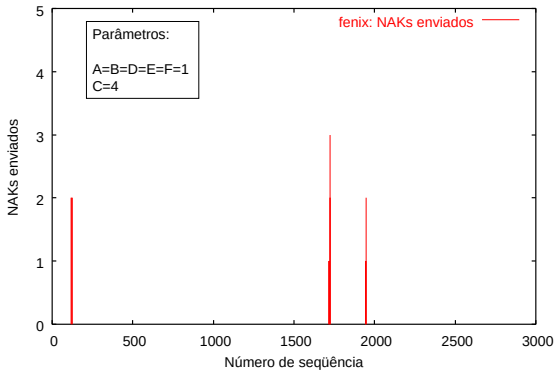

Figura 5.4: NAKs (ou requisições) enviados pela fenix com C=4

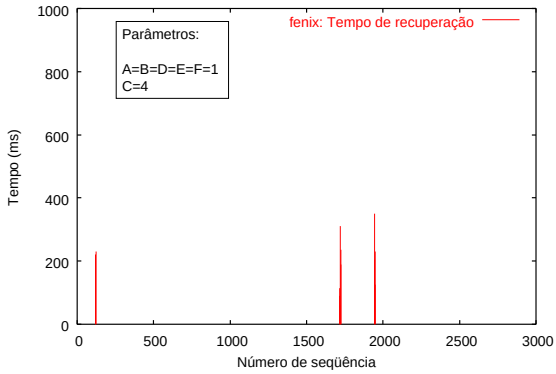

Figura 5.5: Tempo de recuperação dos dados pela fenix com C=4

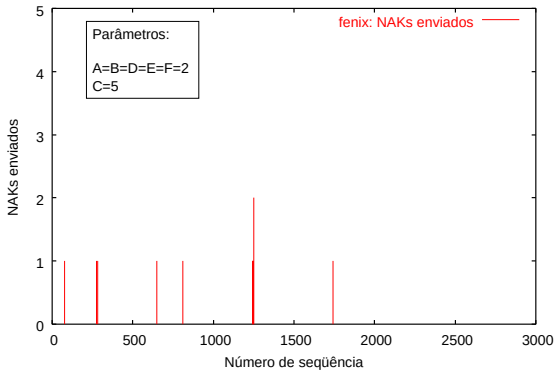

Figura 5.6: NAKs (ou requisições) enviados pela fenix com C=5

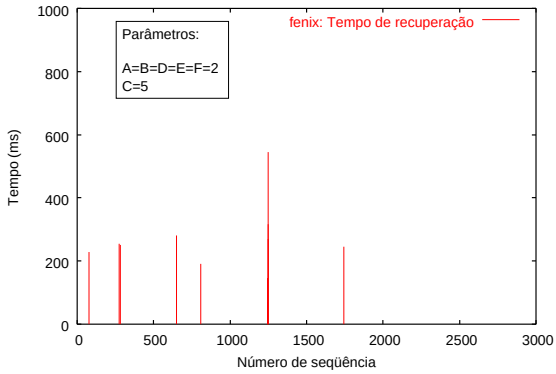

Figura 5.7: Tempo de recuperação dos dados pela fenix com C=5

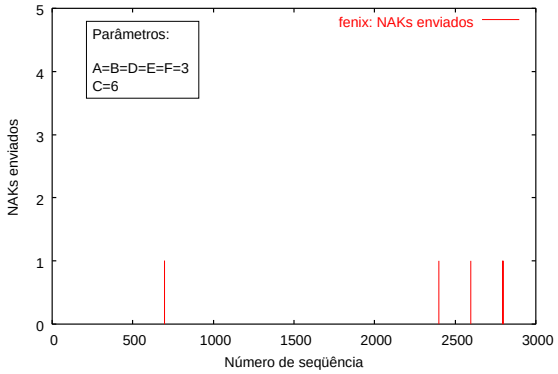

Figura 5.8: NAKs (ou requisições) enviados pela fenix com C=6

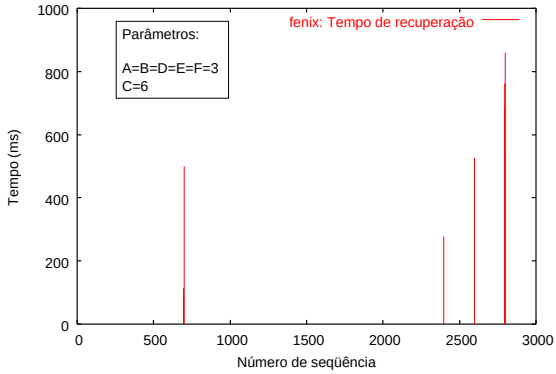

Figura 5.9: Tempo de recuperação dos dados pela fenix com C=6

Entretanto, o aumento no valor dos parâmetros influencia diretamente no tempo de recuperação, como pode ser visto na Tabela 5.5. O tempo de recuperação é o intervalo entre o envio de um NAK e o recebimento da primeira retransmissão referente a esse NAK. As figuras 5.4 a 5.9 mostram o número de NAKs enviados por

pacote perdido e o tempo de recuperação, considerando a variação dos parâmetros dos temporizadores na `fenix`.

Outra medida que é afetada pelo valor dos parâmetros é a supressão de NAKs e retransmissões. De acordo com o protocolo da RML, como foi apresentado na seção 3.4.4, os *hosts* que tiverem o menor retardo com relação ao emissor devem enviar NAKs ou retransmissões antes e suprimir os *hosts* com retardo maior. Portanto, quanto mais distante um *host* estiver do emissor e dos outros membros, maior a chance de supressão para este *host*. O gráfico da Figura 5.10 ilustra o comportamento da supressão de NAKs conforme a variação dos parâmetros dos temporizadores. É possível perceber que a `newworld` é o *host* que mais se beneficia da supressão, chegando a ter 80,78% dos NAKs escalonados suprimidos antes de serem enviados. Isso acontece porque a `newworld` é o *host* que possui o maior retardo em relação ao emissor. Por outro lado, a `abacate` quase não sofre supressão por ter o menor retardo em relação ao emissor. As tabelas 5.6, 5.7 e 5.8 trazem mais detalhes sobre a supressão de NAKs, onde pode-se verificar que a `abacate` é o *host* que suprime mais os outros. A `ilha` não aparece nos resultados porque é o emissor dos dados e, portanto, não envia NAKs.

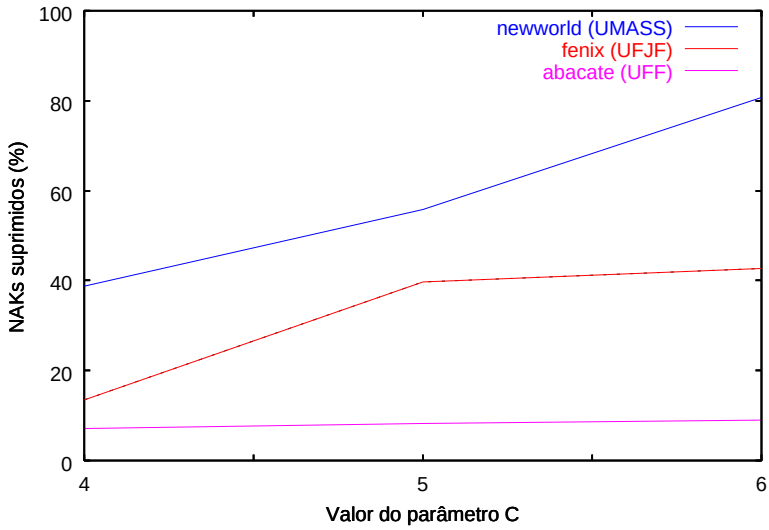

Figura 5.10: Comportamento da supressão de NAKs em relação à variação dos parâmetros

De forma análoga à supressão de NAKs, ocorre a supressão de retransmissões. As tabelas 5.9, 5.10 e 5.11 contém os valores referentes à este tipo de supressão.

Host	Suprimido pela abacate	Suprimido pela fenix	Suprimido pela newworld	Supressão em relação aos NAKs escalonados
abacate	0%	58,00%	42,00%	7,14%
fenix	90,29%	0%	9,71%	13,43%
newworld	75,11%	24,89%	0%	38,78%

Tabela 5.6: Supressão de NAKs com C=4

Host	Suprimido pela abacate	Suprimido pela fenix	Suprimido pela newworld	Supressão em relação aos NAKs escalonados
abacate	0%	55,43%	44,57%	8,29%
fenix	98,58%	0%	1,42%	39,75%
newworld	94,83%	5,17%	0%	55,75%

Tabela 5.7: Supressão de NAKs com C=5

Host	Suprimido pela abacate	Suprimido pela fenix	Suprimido pela newworld	Supressão em relação aos NAKs escalonados
abacate	0%	90,00%	10,00%	9,0%
fenix	99,22%	0%	0,78%	42,78%
newworld	78,89%	21,11%	0%	80,78%

Tabela 5.8: Supressão de NAKs com C=6

Neste caso a `ilha` também participa e é responsável pela maior parte das supressões de retransmissões na `abacate` e na `newworld`. Novamente a `newworld` é a mais suprimida, chegando a ter até $73,89\%$ das retransmissões escalonadas suprimidas.

Host	Suprimida pela abacate	Suprimida pela fenix	Suprimida pela ilha	Suprimida pela newworld	Supressão em relação às Ret. escalonadas
abacate	0%	19,25%	56,62%	24,12%	17,25%
fenix	43,62%	0%	35,00%	21,50%	17,12%
ilha	43,50%	26,00%	0%	30,38%	10,00%
newworld	28,33%	30,44%	41,43%	0%	41,44%

Tabela 5.9: Supressão de Retransmissões com C=4

É importante mostrar também que nos experimentos realizados, a média de retransmissões e NAKs recebidos para um mesmo pacote decaiu com o aumento do

Host	Suprimida pela abacate	Suprimida pela fenix	Suprimida pela ilha	Suprimida pela newworld	Supressão em relação às Ret. escalonadas
abacate	0%	9,86%	81,14%	9,00%	22,43%
fenix	49,25%	0%	38,58%	12,17%	43,58%
ilha	66,50%	22,92%	0%	10,58%	11,00%
newworld	24,58%	26,17%	49,17%	0%	64,33%

Tabela 5.10: Supressão de Retransmissões com C=5

Host	Suprimida pela abacate	Suprimida pela fenix	Suprimida pela ilha	Suprimida pela newworld	Supressão em relação às Ret. escalonadas
abacate	0%	33,33%	66,67%	0%	13,00%
fenix	59,12%	0%	33,75%	7,12%	34,75%
ilha	56,29%	36,29%	0%	7,29%	16,00%
newworld	21,89%	34,33%	43,67%	0%	73,89%

Tabela 5.11: Supressão de Retransmissões com C=6

valor dos parâmetros como pode ser visto nas figuras 5.11 e 5.12. O ideal seria que apenas um pacote de NAK ou retransmissão fosse enviado para o grupo multicast, mas considerando que a atual implementação da RML não está preparada para grandes variações no retardo da rede durante as sessões, os valores obtidos são satisfatórios. Neste ponto deve-se enfatizar que, no futuro, o cálculo dos temporizadores na RML pode ser facilmente alterado para se adaptar, dinamicamente, às variações de retardo e perda da rede. Essa alteração pode, inclusive, se basear nas propostas apresentadas em [27] e [29].

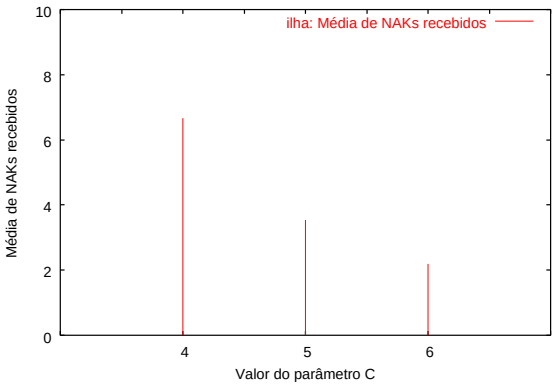

Figura 5.11: Média de NAKs recebidos pela ilha para C=4, C=5 e C=6

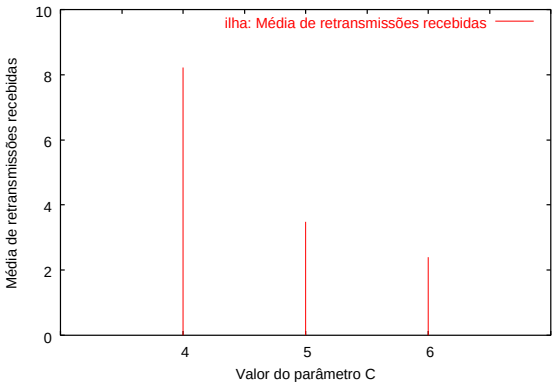

Figura 5.12: Média de retransmissões recebidas pela ilha para C=4, C=5 e C=6

5.2.3 Resultados referentes ao Cenário 2

Nos experimentos referentes ao Cenário 2, todos os quatro *hosts* atuam na sessão multicast como membros receptores e emissores de dados. Apesar deste cenário possuir quatro emissores, a taxa de transmissão de dados é menor quando comparada ao Cenário 1. No Cenário 2 são enviadas rajadas de 5, 10 e 50 pacotes, em intervalos que variam de 1 a 60 segundos. Portanto, esperava-se que os resultados dos testes fossem parecidos com os resultados obtidos para o Cenário 1.

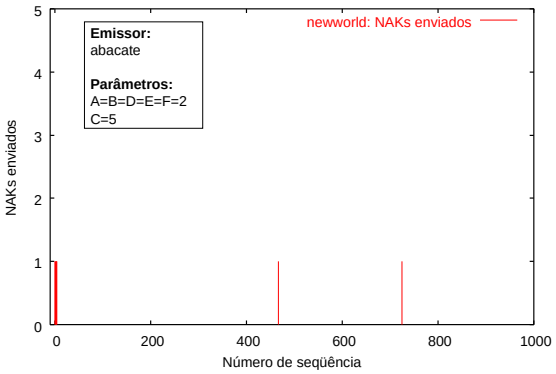

Figura 5.13: NAKs enviados pela newworld em relação ao emissor abacate

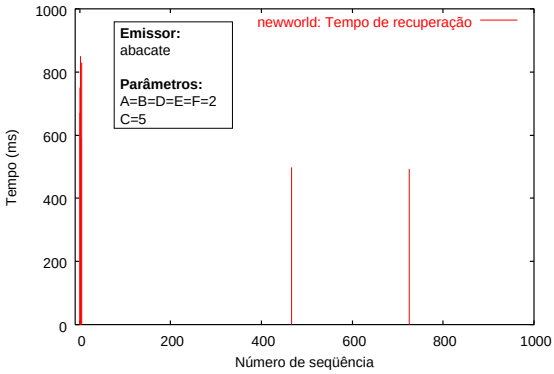

Figura 5.14: Tempo de recuperação da newworld em relação ao emissor abacate

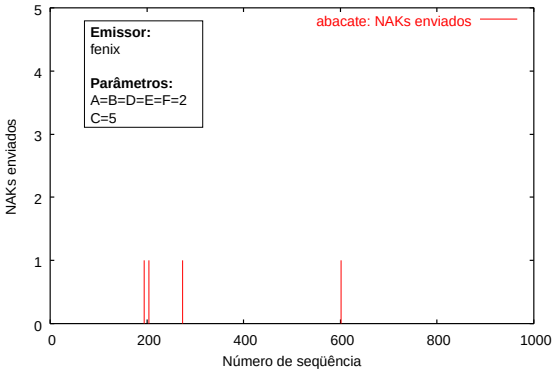

Figura 5.15: NAKs enviados pela abacate em relação ao emissor fenix

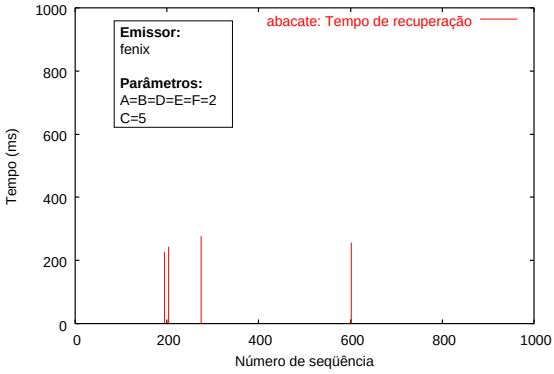

Figura 5.16: Tempo de recuperação da abacate em relação ao emissor fenix

Após obter os resultados do experimento com o Cenário 2 foi verificado que o resultado foi muito bom. O número de NAKs enviados por pacote se manteve baixo durante os testes, como pode ser visto no experimento representado pelas figuras 5.13, 5.15, 5.17 e 5.19, onde o número de NAKs enviados se manteve igual a 1. O tempo de recuperação também se manteve baixo, conforme pode ser observado nos

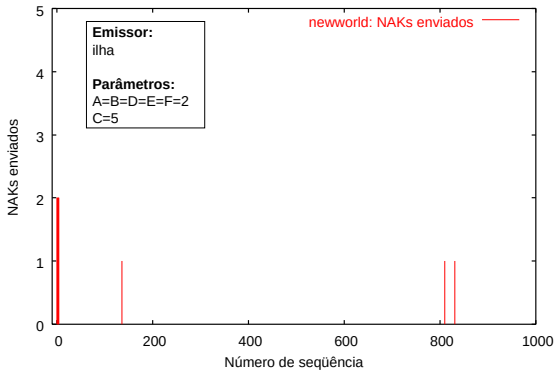

Figura 5.17: NAKs enviados pela newworld em relação ao emissor ilha

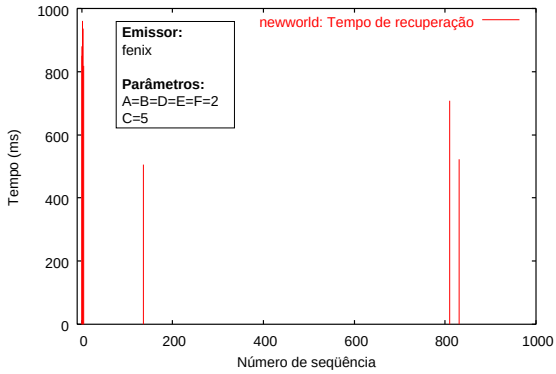

Figura 5.18: Tempo de recuperação da newworld em relação ao emissor ilha

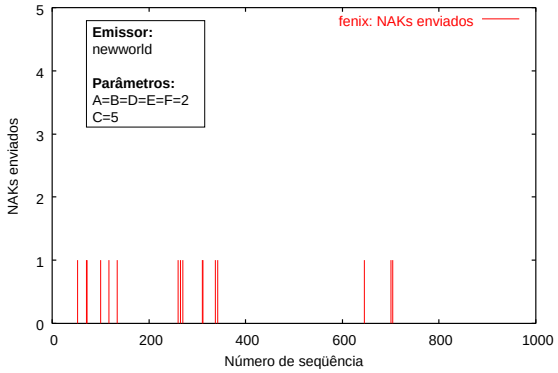

Figura 5.19: NAKs enviados pela fenix em relação ao emissor newworld

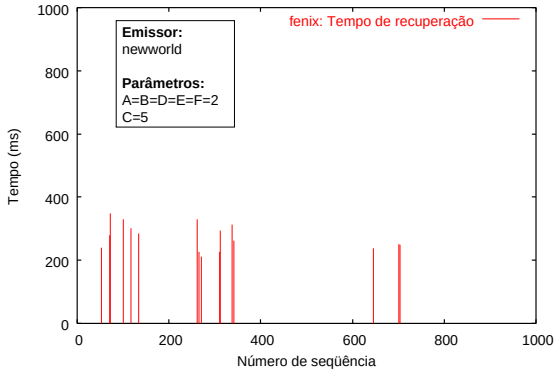

Figura 5.20: Tempo de recuperação da fenix em relação ao emissor newworld

gráficos das figuras 5.14, 5.16, 5.18 e 5.20.

Os experimentos mostraram que a RML se comportou de maneira satisfatória também no Cenário 2, pois o número de NAKs enviados e o tempo de recuperação se mantiveram dentro do esperado.

5.3 O modelo da RML

Em paralelo à implementação da RML, foi desenvolvido, em conjunto com Milena Scanferla, um modelo desta biblioteca. Esse modelo é assunto de sua dissertação de mestrado [37] e contribuiu para fazer alguns ajustes na RML. Nesta seção será apresentada, de forma resumida, como o modelo foi desenvolvido e quais simplificações foram adotadas. Além disso, são discutidas algumas medidas retiradas no modelo

que contribuíram para ajustar alguns parâmetros da RML.

O modelo foi construído, em primeiro lugar, para representar o funcionamento da RML em um ambiente como descrito na seção 5.1.1. No cenário escolhido existe apenas um emissor. O modelo foi desenvolvido no ambiente de modelagem da ferramenta TANGRAM-II, descrita na seção 4.1. A Figura 5.21 contém a representação gráfica do modelo, onde pode-se observar os objetos que o compõem. Existem quatro objetos representando os *hosts*: `UFRJ`, `UFF`, `UFJF` e `UMASS`. Nesses objetos estão implementados os mecanismos de funcionamento da RML. Foram desenvolvidos também outros quatro objetos para representar as instâncias do `mcastproxy`, visto na seção 5.1.3. Por fim, foram criados seis objetos para simular o retardo entre as universidades: `rede01`, `rede02`, `rede03`, `rede04`, `rede05` e `rede06`.

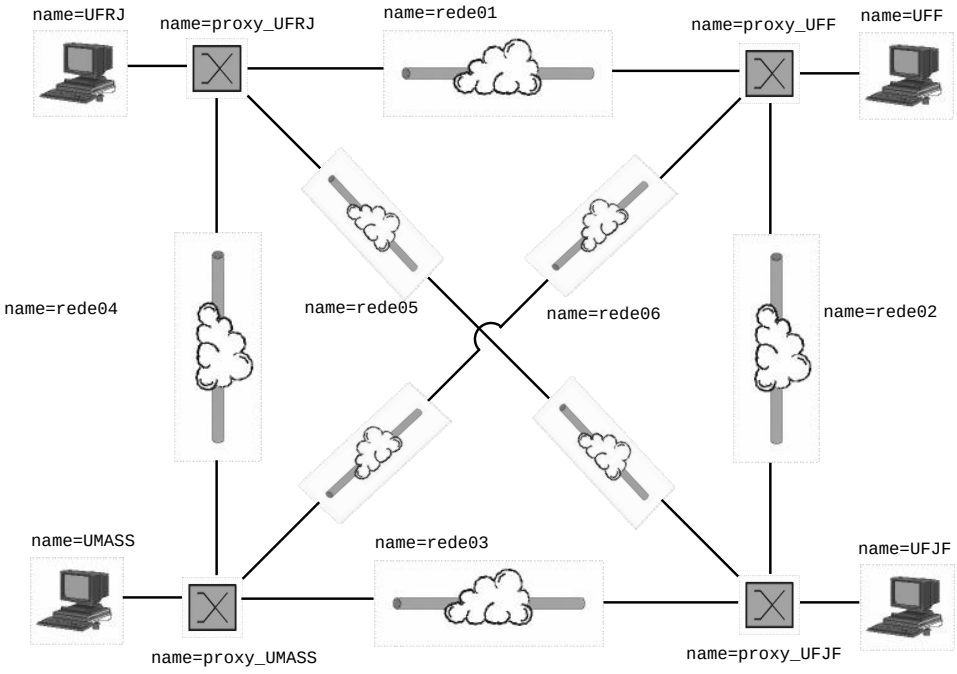

Figura 5.21: Representação gráfica do modelo da RML

Entre as principais simplificações feitas para a modelagem da RML, pode-se citar a representação da `cache` e do conteúdo dos pacotes enviados. Com relação à `cache`, o modelo representa apenas uma parte dos pacotes armazenados, referente aos pacotes com número de seqüência, ou `sn`, entre o `sn` do último pacote recebido em seqüência e o `sn` do último pacote identificado. Portanto, alguns pacotes

armazenados na `cache` não são explicitamente representados, o que permite que o vetor que representa a `cache` possa ter um tamanho menor. Quanto ao conteúdo dos pacotes, os pacotes no modelo são formados por um vetor, contendo algumas informações, como mostra a Figura 5.22. Além disso, no modelo foram suprimidos os tipos JOIN_GROUP, JOIN_ACCEPT e LEAVE_GROUP, pois estes pacotes praticamente não têm influência na geração das medidas estudadas.

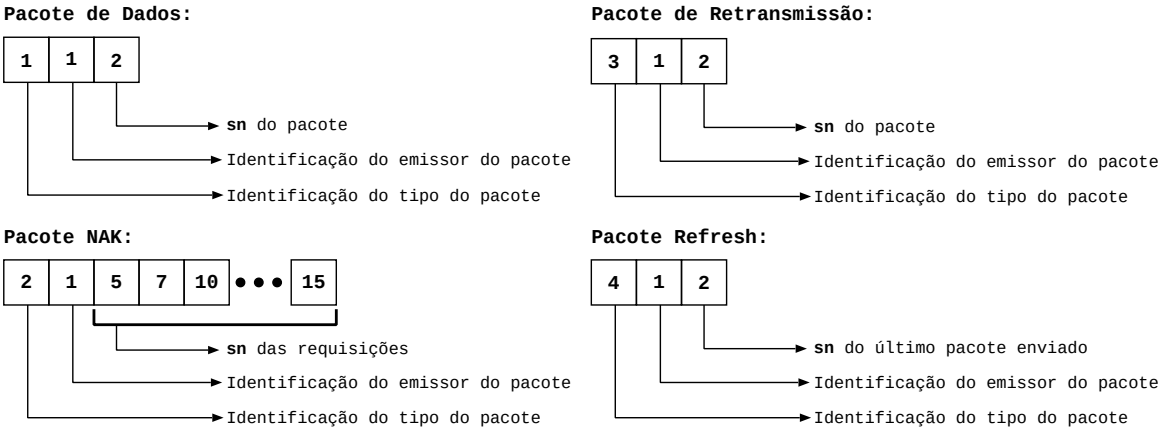

Figura 5.22: Representação dos pacotes no modelo da RML

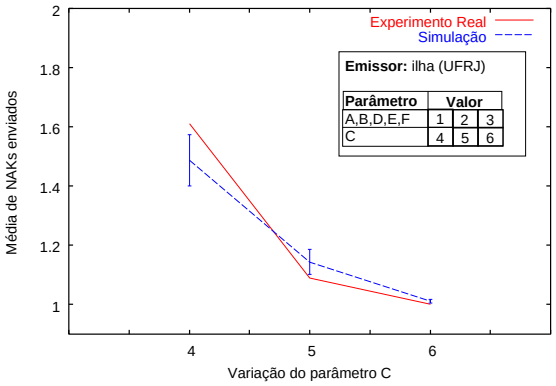

Figura 5.23: NAKs enviados pela fenix: Resultados do experimento real x Resultados da simulação

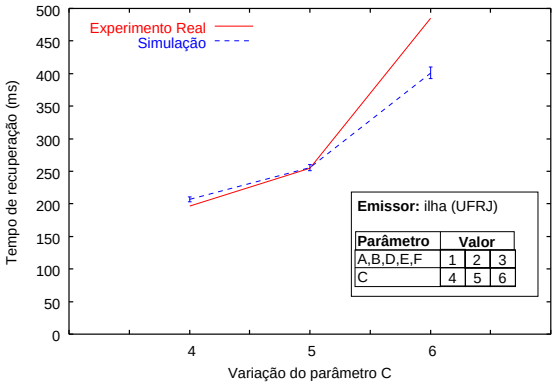

Figura 5.24: Tempo de recuperação da fenix: Resultados do experimento real x Resultados da simulação

Utilizando o modelo foi possível verificar algumas tendências que, em alguns casos, ajudaram a entender melhor o comportamento da RML e detectar certos problemas. Um exemplo foi a detecção de um erro de implementação que provocava um aumento na média do tempo de recuperação dos pacotes. Sem a ajuda das simulações seria muito mais difícil identificar e corrigir esse tipo de erro, pois no

ambiente real não é possível controlar todos os parâmetros que influenciam no tempo de recuperação.

Os gráficos das figuras 5.23 e 5.24 contém a comparação entre os resultados dos experimentos reais e os resultados obtidos com a simulação do modelo. Esses resultados são relativos à variação dos parâmetros dos temporizadores. Na Figura 5.23, o resultado da simulação mostrou a tendência do decrescimento da média de NAKs enviados, apesar de que em números absolutos os valores do modelo foram menores. No que se refere ao tempo de recuperação (Figura 5.24), o modelo apresentou uma tendência ainda mais clara, apesar de, novamente, os valores absolutos não corresponderem exatamente ao resultado do experimento real.

Capítulo 6

Conclusão

Considerando as novas exigências de interatividade impostas atualmente, além da necessidade do uso racional dos recursos das redes de computadores, buscou-se, neste trabalho, desenvolver mecanismos para atingir tais objetivos. Para isso, foi criado um *whiteboard* distribuído, que oferece as características de uma ferramenta completa para desenho vetorial e utiliza transmissão multicast para melhor aproveitamento dos recursos da rede.

A primeira questão a ser resolvida foi referente à transmissão multicast confiável, requisito indispensável para o funcionamento correto da aplicação. No sentido de resolver esta questão, foi criada a RML, uma biblioteca que oferece, para as aplicações, funções para transmissão multicast confiável.

Depois da implementação da RML, o próximo passo foi adaptar a ferramenta para desenho vetorial TGIF, para que ela pudesse executar comandos recebidos através da rede e funcionar como um *whiteboard*. A transmissão multicast dos comandos, de forma confiável, foi garantida pelo uso da RML. Foi também implementado um mecanismo para garantir a consistência global entre os participantes de uma sessão.

Em paralelo ao desenvolvimento da biblioteca de do TGWB, foi criado um modelo da RML, que foi utilizado para obter algumas medidas de interesse. As tendências mostradas pelos resultados do modelo foram confirmadas com os experimentos realizados com a implementação da biblioteca RML.

A montagem do ambiente de testes, o conjunto de experimentos realizados e o modelo da RML mostraram a viabilidade do uso do Tangram Whiteboard, tanto para o desenvolvimento de modelos, quanto para o ensino à distância. De acordo com os resultados obtidos nos experimentos, foi possível perceber a influência do intervalo utilizado para o cálculo dos temporizadores tanto sobre o número de NAKs e retransmissões enviados, quanto sobre o tempo de recuperação e a supressão de NAKs e retransmissões. De acordo com esta relação, o aumento dos parâmetros utilizados para o cálculo dos intervalos dos temporizadores provoca a diminuição do envio de duplicatas de NAKs e retransmissões. Por outro lado, esse mesmo aumento no valor dos parâmetros provoca um aumento no tempo de recuperação.

Por oferecer diversas opções de configuração, a RML oferece grande liberdade para que os usuários ajustem os parâmetros da biblioteca de acordo com as suas necessidades.

6.1 Trabalhos Futuros

Apesar da implementação da RML desenvolvida e apresentada neste trabalho estar totalmente funcional, existem ainda pontos que podem ser melhorados. A principal questão a ser abordada é o estudo e implementação de temporizadores com ajuste dinâmico, como os sugeridos em [29] e [27]. Além disso, podem ser implementadas novas características na interface para melhorar a interatividade com o usuário, como por exemplo um *chat*.

Outra questão, que pode ser trabalhada, é a implementação de multicast no nível da aplicação, para desta forma permitir, por exemplo, o uso do TGWB entre redes que não possuem multicast nativo. Um primeiro passo nesse sentido foi o `mcastproxy` utilizado nos experimentos realizados neste trabalho.

Com relação ao modelo, é possível utilizá-lo para verificar outras características do protocolo como, por exemplo, a questão da escalabilidade, ou seja, qual a influência do número de membros na sessão multicast em relação ao desempenho do protocolo.

Por fim, é necessário fazer ainda uma maior integração entre as ferramentas do TANGRAM-II, principalmente entre o Servidor Multimídia, o VivaVoz e o TGWB, com o objetivo de aumentar a produtividade dos usuários destas ferramentas.

Referências Bibliográficas

[1] URL http://www.land.ufrj.br/. Site do LAND.

[2] URL http://www.linuxdoc.org/howto/multicast-howto.html. Multicast HOWTO.

[3] URL http://www.openmash.org. Site do Projeto Open Mash.

[4] URL http://www.rnp.br. Rede Nacional de Pesquisa.

[5] Azevedo, J. A. URL http://www.land.ufrj.br/tools/rmcast. Site da RML.

[6] Barbosa, V. C. *An Introduction to Distributed Algorithms*. The MIT Press, 1996.

[7] Berson, S., de Souza Silva, E., e Muntz, R. R. *Numerical Solution of Markov Chains*. Marcel Dekker, Inc., 1991, ch. An Object Oriented Methodology for the Specification of Markov Models, pp. 11–36.

[8] Carmo, R., de Carvalho, L., de Souza e Silva, E., Diniz, M., e Muntz, R. Performance/Availability Modeling with the TANGRAM-II Modeling Environment. *Performance Evaluation 33* (1998), 45–65.

[9] Chawathe, Y., McCanne, S., e Brewer, E. A. RMX: Reliable multicast for heterogeneous networks. In *Proceedings of IEEE INFOCOM* (Tel Aviv, Israel, março de 2000), IEEE, pp. 795–804.

[10] Cheng, W. C. URL http://bourbon.cs.umd.edu:8001/tgif/. TANGRAM Graphic Interface Facility.

[11] Chu, Y.-H., Rao, S. G., e Zhang, H. A case for end system multicast. In *Proceedings of ACM SIGMETRICS* (Santa Clara, CA, junho de 2000), ACM, pp. 1–12.

[12] de Brito, C. E. F., de Moraes, R. S., de Oliveira, D. J., e de Souza e Silva, E. Comunicação Multicast Confiável na Implementação de uma Ferramenta Whiteboard. In *XVII Simpósio Brasileiro de Redes de Computadores* (1999), pp. 222–237.

[13] de Freitas Reinhardt, K. Um Ambiente de Simulação de Fluido com Aplicações a Redes Multimídia. Tese de Mestrado, COPPE/UFRJ, 2002.

[14] de Souza e Silva, E., e Leão, R. M. M. The Tangram-II Environment. In *TOOLS2000* (Schaumburg, IL, USA, março de 2000), pp. 366–369.

[15] Duarte, F. P. Previsão de Perdas utilizando Cadeias de Markov Escondidas. Tese de Mestrado, COPPE/UFRJ, 2002.

[16] Figueiredo, D. R., e de Souza e Silva, E. Efficient mechanisms for recovering voice packets in the internet. In *Proceedings of IEEE/Globecom* (1999), pp. 1830–1837.

[17] Floyd, S., Jacobson, V., Liu, C.-G., McCanne, S., e Zhang, L. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking 5*, 6 (dezembro de 1997), 784–803.

[18] Francis, P. Yoid: Extending the internet multicast architecture, abril de 2000. http://www.icir.org/yoid/docs/index.html.

[19] Holbrook, H., Singhal, S. K., e Cheriton, D. R. Log-based receiver-reliable multicast for distributed interactive simulation. In *Proceedings of ACM SIGCOMM* (agosto de 1995), ACM, pp. 328–341.

[20] Kurose, J. F., e Ross, K. W. *Computer Networking: a top-down approach feturing the Internet (ISBN 0-201-47711-4)*. Addison-Wesley, 2001.

[21] Lamport, L. Time, clocks and ordering of events in a distributed system. *Commun. ACM 21*, 7 (1978), 558–565.

[22] Lehman, L.-W. H., Garland, S. J., e Tennenhouse, D. L. Active reliable multicast. In *INFOCOM (2)* (1998), pp. 581–589.

[23] Leão, R. M., de Souza e Silva, E., e de Lucena, S. C. A Set of tools for Traffic Modelling, Analysis and Experimentation. In *Proceedings of TOOLS* (2000).

[24] Levine, B. N., e Garcia-Luna-Aceves, J. J. A case for reliable concurrent multicasting using shared ack trees. In *Proceedings of ACM Multimedia* (novembro de 1996), ACM, pp. 365–376.

[25] Levine, B. N., e Garcia-Luna-Aceves, J. J. A comparison of reliable multicast protocols. *Multimedia Systems 6*, 5 (1998), 334–348.

[26] Lin, J. C., e Paul, S. RMTP: A reliable multicast transport protocol. In *Proceedings of IEEE INFOCOM* (março de 1996), IEEE, pp. 1414–1425.

[27] Liu, C., Estrin, D., Shenker, S., e Zhang, L. Timer adjustment in SRM, 1997.

[28] McCanne, S., e Jacobson, V. vic: a flexible framework for packet video. In *Proceedings of ACM Multimedia* (1995), pp. 511–522.

[29] Nonnenmacher, J., e Biersack, E. Optimal multicast feedback. In *Proceedings of IEEE INFOCOM (3)* (1998), pp. 964–971.

[30] Pendarakis, D., Shi, S., Verma, D., e Waldvogel, M. ALMI: An application level multicast infrastructure. In *3rd USENIX Symposium on Internet Technologies and Systems (USITS)* (2001), pp. 49–60.

[31] Pingali, S., Towsley, D., e Kurose, J. F. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceedings of the Sigmetrics Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, 1994), ACM Press, pp. 221–230.

[32] Quevedo, A. Mecanismos para Garantir Qualidade de Serviço de Aplicações de Vídeo sob Demanda. Tese de Mestrado, COPPE/UFRJ, 2002.

[33] Ramakrishnan, S., e Jain, B. N. A negative acknowledgement with periodic polling protocol for multicast over lans. In *Proceedings of Infocom'87* (1987), pp. 502–511.

[34] Rigther, R., e Walrand, J. Distributed simulation of discrete event systems. In *Proceedings of IEEE* (janeiro de 1989), IEEE, pp. 99–113.

[35] Saltzer, J. H., Reed, D. P., e Clark, D. D. End-to-end arguments in system design. *ACM Transactions on Computer Systems 2*, 4 (novembro de 1984), 277–288.

[36] Santos, J., Muntz, R., e Ribeiro-Neto, B. Comparing Random Data Allocation and Data Striping in Multimedia Storage Servers. In *Proceedings of ACM SIGMETRICS* (2000), pp. 44–55.

[37] Scanferla, M. Transmissão Multicast Confiável: a Implementação no TANGRAM II Whiteboard e a Análise de Desempenho. Tese de Mestrado, UFF, 2002.

[38] Silva, A. P. C. Métodos de Solução para Modelos Markovianos com Recompensa. Tese de Mestrado, COPPE/UFRJ, 2001.

[39] Tung, T. Mediaboard: A Shared Whiteboard Application for the MBone. Tese de Mestrado, University of California, Berkeley, 1997, 1997.

[40] Whetten, B., Montgomery, T., e Kaplan, S. M. A high performance totally ordered multicast protocol. In *Dagstuhl Seminar on Distributed Systems* (1994), pp. 33–57.

[41] Yavatkar, R., Griffioen, J., e Sudan, M. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of ACM Multimedia* (novembro de 1995), ACM, pp. 333–344.

[42] Zhuang, S., Zhao, B., Joseph, A., Katz, R., e Kubiatowicz, J. Bayeux: An architecture for scalable and fault-tolerant widearea data dissemination. In *Proceedings of NOSSDAV 2001.*

Apêndice A

Pré-requisitos para o uso da RML

A.1 Endereços IP Multicast

O primeiro passo para se utilizar a transmissão multicast é conhecer quais endereços podem ser utilizados. Para realizar uma transmissão multicast via UDP são utilizados endereços IP especiais, chamados de endereços IP Multicast [2]. Os endereços IP Multicast, ou endereços IP classe D como eram denominados antigamente, variam de 224.0.0.0 a 239.255.255.255. Entre estes endereços, os listados na Tabela A.1 são reservados e não devem ser usados.

Endereços	Função
224.0.0.1	Todos os hosts na rede local
224.0.0.2	Todos os roteadores na rede local
224.0.0.4	Todos os roteadores DVMRP na rede local
224.0.0.5	Todos os roteadores OSPF na rede local
224.0.0.6	Todos os roteadores OSPF "designated"na rede local
224.0.0.13	Todos os roteadores PIM na rede local

Tabela A.1: Endereços multicast reservados

A.2 Configurando o Multicast no GNU/Linux

Esta seção não pretende explicar detalhadamente a implementação e configuração do Multicast no GNU/Linux, mas apenas dar algumas dicas de como configurar um sistema para permitir o uso do IP Multicast em uma rede local. A página do Multicast HOWTO [2] contém mais detalhes. Para habilitar o IP Multicast entre redes diferentes é necessário que os roteadores entre as redes suportem o roteamento multicast ou que exista um túnel IP para que os pacotes atravessem os roteadores que não implementam o roteamento multicast.

É preciso fazer algumas verificações para saber se um sistema GNU/Linux está habilitado para enviar e receber pacotes multicast. Em primeiro lugar, o módulo (ou *driver*) da interface de rede deve ter o multicast habilitado. A maioria dos módulos vêm com o multicast habilitado. Para verificar essa informação é usado o comando *ifconfig*. A seguir são mostradas a linha de comando e a sua saída.

```
ifconfig -a

eth0
Link encap:Ethernet HWaddr 00:50:BF:06:89:47
inet addr:192.168.1.2 Bcast:192.168.1.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:12438583 errors:0 dropped:0 overruns:0 frame:0
TX packets:6498370 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
RX bytes:1100375580 (1049.3 Mb)
TX bytes:2158372342 (2058.3 Mb)
Interrupt:10 Base address:0x7000

lo
Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:8361666 errors:0 dropped:0 overruns:0 frame:0
TX packets:8361666 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:1830657956 (1745.8 Mb)
TX bytes:1830657956 (1745.8 Mb)
```

O resultado do comando *ifconfig* mostra que duas interfaces de rede estão habi-

litadas: *eth0*, que representa uma placa de rede ethernet e *lo* que representa uma interface local, usada como rede virtual. É possível perceber que a interface *lo* não tem o multicast habilitado pois está faltando a *flag* MULTICAST. Para habilitar o multicast em uma interface de rede, o seguinte comando é usado:

```
ifconfig <nome_interface> multicast
```

O parâmetro *nome_interface* deve ser substituído pelo nome da interface onde o multicast vai ser habilitado. O próximo passo é configurar a rota que os pacotes multicast devem seguir. Para configurar essa rota pode ser usado o comando *route*, como mostrado no exemplo a seguir.

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev <nome_interface>
```

O parâmetro *nome_interface* deve ser substituído pelo nome da interface da rede por onde os pacotes multicast devem ser enviados. Depois de configuradas as interfaces e as rotas, o comando *ping* pode ser usado para testar a configuração.

```
ping 224.0.0.1
```

Após este comando, cada máquina da rede local que estiver com o multicast habilitado deve responder ao *ping*.

A.3 Compilando a RML

Para compilar a biblioteca RML deve-se seguir os seguintes passos:

1. Verificar a existência de um ambiente completo para programação usando a linguagem C. Isso implica na disponibilidade de ferramentas como o compilador gcc, gmake, bibliotecas C, entre outras.
2. Copiar o código-fonte da RML a partir da página indicada em [5];

3. Descompactar o arquivo que contém o código-fonte. Ao descompactá-lo, um diretório chamado RelMulticast será criado;

4. Entrar no diretório RelMulticast e executar o comando *make*. O resultado do *make*, caso não ocorra erro, é a geração da biblioteca estática RML, contida no arquivo librmcast.a.

Os programas que utilizam a RML devem ser compilados indicando o local onde se encontram os arquivo 'rmcast.h' e 'librmcast.a'. Considerando que os arquivos da RML estão em '/src/RelMulticast', para compilar um programa rmX a seguinte linha de comando é utilizada:

```
gcc -I/src/RelMulticast -L/src/RelMulticast -lrmcast -o rmX rmX.c
```

Informações mais atualizadas e completas a respeito da RML podem ser obtidas nos arquivos README e INSTALL, presentes no diretório RelMulticast.

Apêndice B

Utilização e configuração da RML

O objetivo desta seção é fornecer as informações necessárias para a utilização da RML. Na seção B.1 são apresentadas as funções disponíveis na biblioteca. A seção B.2 apresenta as opções de configuração que podem ser usadas pela aplicação e, por fim, a seção B.3 apresenta a estrutura do arquivo que contém o registro dos pacotes enviados e recebidos.

B.1 Funções disponíveis na RML

A RML disponibiliza um conjunto de funções para a aplicação, uma breve descrição destas funções é apresentada a seguir:

- `RM_initialize(void)`:
  Inicializa as estruturas de dados da RML, como a `cache` e a lista de eventos.
- `RM_joinGroup(char *group, int port)`:
  Executa as rotinas para entrada no grupo multicast indicado por `group`. O parâmetro `port` é utilizado para identificar a porta usada pelo processo. A função retorna a identificação de um *socket* que será usada para envio e recebimento dos dados.
- `RM_leaveGroup(int sock, char *group)`:

Escalona o evento `LEV_GRP_WAIT`, que ao disparar executa as rotinas para término da aplicação.

- `RM_sendto(int socket, void *buffer, int buffsize)`:
  Envia `buffsize` bytes dos dados contidos em `buffer` para o grupo multicast. O parâmetro `socket` deve ser a identificação do *socket* retornada pela função `RM_joinGroup`.

- `RM_recv(int socket, void *buffer, int buffsize)`:
  Recebe dados do grupo multicast e armazena `buffsize` bytes em `buffer`. O parâmetro `socket` deve ser a identificação do *socket* retornada pela função `RM_joinGroup`.

- `RM_getCurStatus(char *group, int port, CurStatus *c)`:
  Executa as rotinas para obtenção do "estado atual" do grupo multicast.

- `RM_sendCurStatus(int connfd, char *buff, int buffsize)`:
  Envia, usando a conexão TCP identificada por `connfd`, o "estado atual" do membro.

- `RM_readConfigFile(char *filename, char show_config_file)`:
  Lê as configurações da RML do arquivo identificado por `filename`. Exibe as configurações lidas caso o valor do parâmetro `show_config_file` seja 1.

- `RM_getOption(int opt, void *return_value)`:
  Retorna em `return_value` o valor da opção identificada por `opt`.

- `RM_setOption(int opt, void *optvalue)`:
  Atribui o conteúdo de `optvalue` à opção identificada por `opt`.

- `RM_setHostTimers(char *hostname, int ltimer,int htimer )`:
  Ajusta os limites do intervalo para cálculo dos temporizadores do *host* identificado por `hostname`. Onde `ltimer` é o limite inferior e `htimer` é o limite superior do novo intervalo.

- `RM_getHostTimers(char *hostname, int *ltimer,int *htimer )`: Obtém os limites inferior e superior do intervalo usado para cálculo dos temporizadores do *host* identificado por `hostname`.

B.2 Opções de configuração disponíveis na RML

As opções de configuração da biblioteca para transmissão multicast confiável, a RML, podem ser alteradas de duas formas. A primeira é lendo os valores para as opções a partir de um arquivo de configuração. A Figura B.1 exibe o exemplo de um arquivo de configuração. Este arquivo deve ser passado como parâmetro para a função `RM_readConfigFile`. A outra forma disponível para uma aplicação alterar as configurações da RML é usar a função `RM_setOption`. A seguir são apresentadas as opções de configuração disponíveis na RML:

- `VERSION`: indica a versão da RML;
- `TRANSMISSION_MODE`: indica o modo de transmissão. 0 (zero) para multicast, 1 para unicast;
- `DEST_IP`: indica o endereço IP de destino. Deve ser um endereço IP Multicast ou IP Unicast, de acordo com a opção de transmissão escolhida para `TRANSMISSION_MODE`;
- `DEST_PORT`: indica o número da porta que será usada na transmissão;
- `TTL`: *time to live*, indica por quantos roteadores os pacotes enviados podem passar antes de expirar;
- `MICROSLEEP`: indica o intervalo de tempo, em microssegundos, entre o envio de cada pacote. Essa opção se tornou necessária pois o *buffer* UDP de algumas máquinas pode ser insuficiente para suportar altas taxas de envio de dados, o que poderia provocar perdas;
- `LOG_FILE`: indica o nome base do arquivo onde as informações sobre os pacotes recebidos e enviados serão gravadas. Caso o usuário não queira registrar

essas informações, basta atribuir 'NULL' a essa opção. Ao nome base serão adicionados o nome do *host* e a identificação do processo (PID). Por exemplo, se o nome base for 'log', o nome do *host* 'host1' e o PID da aplicação for 1500, o arquivo de registro gerado será 'log.host1.1500';

```
#RML Configuration File
VERSION=1.0

TRANSMISSION_MODE=0

DEST_IP=226.2.2.2

DEST_PORT=15151

TTL=0

MICROSLEEP=2000

LOG_FILE=log

TIMER_DISTRIBUTION=0

TIMER_PARAM_A=2
TIMER_PARAM_B=2
TIMER_PARAM_C=2
TIMER_PARAM_D=2
TIMER_PARAM_E=2
TIMER_PARAM_F=2
HOSTS_IDENTIFIED=1
DEFAULT 300
host1 200

MAX_NAK=100

MAX_MEMBER_CACHE_SIZE=1000

NEW_USER_SUPPORT=0

STATISTICS=0

REFRESH_TIMER=10

LOSS_PROB=0

LEAVE_GROUP_WAIT_TIME=1000000

RCV_BUFFER_SIZE=10000
```

Figura B.1: Exemplo de arquivo de configuração da RML

- `TIMER_DISTRIBUTION`: indica a distribuição das variáveis aleatórias que serão geradas para cálculo dos temporizadores. O valor 0 (zero) indica distribuição uniforme e o valor 1 indica distribuição exponencial;

- `TIMER_PARAM_A, TIMER_PARAM_B`: indicam os valores dos parâmetros para o cálculo do intervalo onde serão gerados os temporizadores relativos ao envio de NAKs. Esse intervalo é calculado segundo a fórmula: $(\texttt{TIMER_PARAM_A} \cdot R, (\texttt{TIMER_PARAM_A} + \texttt{TIMER_PARAM_B}) \cdot R)$, onde R representa a estimativa do retardo de propagação até o emissor;

- `TIMER_PARAM_C, TIMER_PARAM_D`: indicam os valores dos parâmetros para o cálculo do intervalo onde serão gerados os temporizadores relativos à espera por retransmissões. Esse intervalo é calculado segundo a fórmula: $(\texttt{TIMER_PARAM_C} \cdot R, (\texttt{TIMER_PARAM_C} + \texttt{TIMER_PARAM_D}) \cdot R)$, onde R representa a estimativa do retardo de propagação até o emissor;

- `TIMER_PARAM_E, TIMER_PARAM_F`: indicam os valores dos parâmetros para o cálculo do intervalo onde serão gerados os temporizadores relativos ao envio de retransmissões. Esse intervalo é calculado segundo a fórmula: $(\texttt{TIMER_PARAM_E} \cdot R, (\texttt{TIMER_PARAM_E} + \texttt{TIMER_PARAM_F}) \cdot R)$, onde R representa a estimativa do retardo de propagação até o emissor;

- `HOSTS_IDENTIFIED`: indica quantos *hosts* terão temporizadores especiais. Se o valor for 0 (zero), apenas um valor padrão para os temporizadores será lido na linha seguinte. Se o valor atribuído a essa opção for N, onde $N > 0$, então N valores, para N *hosts* serão lidos. No exemplo da Figura B.1, o valor desta opção é 1, portanto serão lidos os valores para o temporizador DEFAULT (300) e e para o 'host1' (200). Os valores correspondem à estimativa de retardo de propagação em relação aos *hosts* e devem estar em milissegundos;

- `MAX_NAK`: indica o número máximo de pedidos de retransmissão que poderão ser feitos para cada pacote perdido;

- `MAX_MEMBER_CACHE_SIZE`: indica o número máximo de pacotes de dados que poderão ser armazenados na `cache`;

- `NEW_MEMBER_SUPPORT`: habilita o suporte a novos membros quando o valor 1 é atribuído, ou desabilita esse suporte caso o valor atribuído seja 0 (zero);

- `STATISTICS`: habilita o cálculo de estatísticas durante a sessão multicast. Essa opção só será reconhecida na versão 2 da RML;

- `REFRESH_TIMER`: indica o tempo, em segundos, entre o envio de mensagens de atualização;

- `LOSS_PROB`: habilita a simulação de perdas. Se `LOSS_PROB` = N, então $N\%$ dos pacotes serão perdidos.
- `LEAVE_GROUP_WAIT_TIME`: indica o tempo de espera, em milissegundos, antes de realmente sair do grupo multicast, após a chamada da função `RM_leaveGroup`;
- `RCV_BUFFER_SIZE`: indica o tamanho, em bytes, do *buffer* de recepção da RML;

O formato do arquivo de configuração da RML pode ser visto na Figura B.1. As linhas que iniciam com '#' são comentários. Através das opções desse exemplo podemos perceber que a versão da RML utilizada será a 1.0, o modo de transmissão será multicast e o endereço IP será 226.2.2.2, porta 15151. Os pacotes serão enviados apenas para a rede local, pois `TTL` = 0, com intervalo mínimo de 2000 microssegundos entre cada pacote. O arquivo de registro será gerado com o nome base 'log'. Os temporizadores serão gerados, usando distribuição uniforme, no intervalo entre 600 e 1200 milissegundos, exceto para o 'host1', para o qual será utilizado o intervalo entre 400 e 800 milissegundos. O número máximo de pedidos de retransmissão para um mesmo pacote será 100. A cache armazenará no máximo 4000 pacotes de dados. O suporte a novos membros estará desabilitado e não serão geradas estatísticas durante a sessão multicast. O tempo entre o envio de mensagens de atualização será de 10 segundos. A simulação de perda estará desabilitada. O tempo de espera antes de sair do grupo multicast será de 1 segundo. Por fim, o *buffer* de recebimento da RML será de 10000 bytes.

B.3 Registro de pacotes recebidos e enviados

De acordo com o exemplo da Figura B.2, o arquivo de registro da RML é formado pelos seguintes campos:

- `time`: indica o momento em que o pacote foi recebido ou enviado.
- `snd/rcv/loss`: indica se o pacote registrado foi enviado (S), recebido (R), ou perdido usando simulação de perdas (L).

- `type`: indica o tipo do pacote, isto é, NAK (NK), dados (DT), retransmissão (RT), REFRESH (RF), JOIN REQUEST (JR), JOIN ACCEPT (JA), LEAVE GROUP (LG) e, caso o tipo do pacote não seja identificado, ele é registrado como desconhecido (UN).

- `sender_ip`: indica o endereço IP do emissor do pacote.

- `sender_pid`: indica o PID do emissor do pacote.

- `requested_ip`: este campo somente é usado para pacotes NAK ou retransmissões. Indica o endereço IP do emissor do dado original, requisitado pelo pacote NAK, ou contido no pacote de retransmissão.

- `requested_pid`: este campo somente é usado para pacotes NAK ou retransmissões. Indica o PID do emissor do dado original, requisitado pelo pacote NAK, ou contido no pacote de retransmissão.

- `sn`: esse campo tem diferentes significados, dependendo do tipo do pacote. Para os tipos dados ou retransmissão indica o número de seqüência do pacote. Quando o tipo é REFRESH, este campo indica o número de seqüência do último pacote de dados enviado pelo membro. Este campo não tem significado nos outros pacotes.

- `base_sn`: indica o número de seqüência da primeira retransmissão requisitada em um pacote do tipo NAK.

- `win_size`: indica o tamanho da janela usada no pacote NAK, normalmente esse valor é 64.

- `hmask`: um inteiro que representa a parte superior do vetor de requisições contido em um pacote do tipo NAK.

- `lmask`: um inteiro que representa a parte inferior do vetor de requisições contido em um pacote do tipo NAK.

```
host: receiverhost ip: 192.168.1.2
pid: 18348
-------------------------------------------------------------------------------------------------------------------
time snd/rcv/loss type sender_ip sender_pid requested_ip requested_pid sn [{base_sn} {win_size} {hmask} {lmask}]
-------------------------------------------------------------------------------------------------------------------
51800783466 L     RF   192.168.1.1 13893                              -1
51808642569 L     DT   192.168.1.1 13893                               0
51810314729 S     RF   192.168.1.2 18348                              -1
51829942926 R     DT   192.168.1.1 13893                              48
51829947209 S     NK   192.168.1.2 18348    192.168.1.1  13893              -1          64     29280  235372671
```

Figura B.2: Exemplo de arquivo de registro da RML

Apêndice C

Código fonte do rmchat, uma aplicação baseada na RML

```
/***************************************************************************
                            rmchat.c
                            --------------------
    Início                : Outubro 2002
    copyright             : (C) 2002 by Jorge Allyson Azevedo
    email                 : allyson@land.ufrj.br

    Este programa é um simples 'chat' que utiliza a RML para transmissão
    multicast confiável.

 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <unistd.h>
```

```
#include "rmcast.h" /* Cabeçalho da RML */

#define BUFFSIZE 200 /* Tamanho máximo do buffer de mensagens */

#define TRUE 1
#define FALSE 0


/* Variáveis Globais */
int chatSock;
int end = 0;
int not_send_first = 1;


/*************************************************************************
*   ReceivePacket                                                        *
*                                                                        *
*   Função (thread) para receber dados da rede                           *
*                                                                        *
*************************************************************************/
void *ReceivePacket( void *arg ){

    char receive_buffer[BUFFSIZE];

    int bytes_read;

    while( !end ){

        if( (bytes_read = RM_recv(chatSock,receive_buffer, BUFFSIZE)) > 0 ){

             /* Dado recebido da rede, mostrar ao usuário */
             fprintf(stdout,"\n%s>",receive_buffer);
             fflush(stdout);
        }
        else{

             fprintf(stderr,"[ReceivePacket]: Erro ao receber dados.\n");
             break;
        }

    }

    return 0;
}


int main(int argc, char **argv){

    pthread_t tid=0; /* Identificação da thread */
    int port=0;
```

```
    char buffer[BUFFSIZE],aux[16],user_name[20],msg_buff[BUFFSIZE-20],ip[16];

    /* Verificando os parâmetros de entrada */
    if ( argc != 2 ){
        fprintf(stderr,"Uso:\n\t%s <config_file>\n",argv[0]);
        exit ( -1 );
    }

    /* Lendo arquivo de configuração */
    if ( !RM_readConfigFile(argv[1], TRUE) ){
        fprintf(stderr,"[rmchat]: Erro arquivo de configuração (%s)\n",argv[1]);
        exit( -1 );
    }

    /* Inicializando as funções da RML */
    RM_initialize();

/*
 *     Entrando para o grupo multicast usando o IP e porta indicados no
 *     arquivo de configuração
 */

    chatSock= RM_joinGroup((char*)RM_USE_CURRENT_CONFIG, RM_USE_CURRENT_CONFIG);

    fprintf(stdout,"\nEntre um nome de usuário (max 16 caracteres):\n");
    scanf("%s",user_name);
    aux[0]='\0';
    strcat(aux,"[");
    strcat(aux,user_name);
    strcat(aux,"]>");

    /* Obtendo o IP e a porta do grupo multicast da configuração da RML */
    RM_getOption(DEST_IP, (void *)ip);
    RM_getOption(DEST_PORT, (void *)&port);

    fprintf(stdout,"\n*************\nOlá %s!\n",user_name);
    fprintf(stdout,"Agora você pode enviar mensagens para o grupo multicast\n");
    fprintf(stdout," IP:%s Porta:%d\n",ip,port);
    fprintf(stdout,"Digite \"exit\" quando você quiser terminar o programa\n");
    fprintf(stdout,"*************\n>");

    /* Criando a thread responsável por receber os dados da rede */
    if( pthread_create( &tid, NULL, ReceivePacket, NULL ) != 0 ){

        /* Error, terminate the application */
        fprintf( stderr,"[rmchat]: Erro ao criar thread recepção de dados\n" );
        RM_leaveGroup( chatSock, (char*)RM_USE_CURRENT_CONFIG );
    }
```

```
    while( TRUE ){

        buffer[0]='\0';

        /* Lendo as mensagens a partir da entrada padrão */
        fgets(msg_buff,BUFFSIZE-21,stdin);

        if (strcmp(msg_buff,"exit\n")==0){

            /* O usuário quer terminar a aplicação */
            fprintf(stdout,"\n*************\n");
            fprintf(stdout,"Terminando rmchat ...\n");
            break;
        }
        else{

                strcat(buffer,aux);
                strcat(buffer,msg_buff);

                if ( !not_send_first )
                    /* Enviando a mensagem para o grupo Multicast */
                    RM_sendto(chatSock,buffer,BUFFSIZE);
                else
                    not_send_first=0;

                fprintf(stdout,"\n%s>",buffer);
        }
    }

    /* Parando a thread de recepção*/
    pthread_cancel(tid);

    /* Saindo do grupo multicast */
    fprintf(stdout,"Saindo do grupo multicast...");
    RM_leaveGroup( chatSock, (char*)RM_USE_CURRENT_CONFIG );

    return TRUE;
}
```