UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

SAM: um sistema adaptativo para transmissão e recepção de sinais multimídia em redes de computadores

por

VALTER ROESLER

Tese submetida à avaliação,
como requisito parcial para a obtenção do grau de
Doutor em Ciência da Computação

Prof. Dr. José Valdeni de Lima
Orientador

Prof. Dra. Liane Margarida Rockenbach Tarouco
Co-orientadora

Porto Alegre, dezembro de 2003

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Roesler, Valter

SAM: um sistema adaptativo para transmissão e recepção de sinais multimídia em redes de computadores / por Valter Roesler. – Porto Alegre: PPGC da UFRGS, 2003.

271 f.: il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2003. Orientador: Lima, José Valdeni de; Co-orientadora: Tarouco, Liane Margarida Rockenbach.

1. Adaptabilidade. 2. Multimídia. 3. Multicast. 4. Transmissão em Camadas. 5. Controle de Congestionamento. I. Lima, José Valdeni de. II. Tarouco, Liane Margarida Rockenbach. III. Título.

Dedicatória

Aos meus pais, Ilgo e Telva, pelo exemplo de trabalho e de vida...

À Eliane e à Helena, pelo exemplo de amor...

Agradecimentos

Após o trabalho concluído, gostaria de agradecer a uma série de pessoas que auxiliaram, tornando a jornada muito mais agradável do que normalmente seria.

Primeiramente, gostaria de agradecer à Unisinos, representada pela pessoa da Sílvia Dutra, que me liberou 20 horas semanais para fazer o doutorado (e, o melhor de tudo, remunerado).

Ainda no alto escalão (e com alto grau de amizade), gostaria de agradecer ao incentivo do Candido.

Também ao pessoal do PRAV, que prestou um auxílio muito grande, como apoio na confecção de programas, implementações e testes. Em especial, o Maiko, o Hades, o Gaspare, o João, o Peter e o Oldoni. Com o Gaspare a coisa rendeu também uma série de publicações conjuntas e sua dissertação de mestrado, na qual ele implementou o VEBIT.

A Eliane teve muita paciência e quase sempre apoiou, principalmente nos fins de semana e madrugadas que eu passei trabalhando. Além disso, ela leu e corrigiu alguns capítulos da Tese. A Helena não leu nada, mas soube aceitar quando eu tinha que trabalhar.

Ao Luciano e ao Marinho, pelo incentivo e apoio, dando várias sugestões úteis ao longo do trabalho.

Ao Valdeni e à Liane Tarouco, que não se mostraram somente bons orientadores, mas também bons amigos.

Ao CNPq, que patrocinou alguns bolsistas para auxiliarem na implementação e simulações do ALM, dentro do projeto Metropoa.

Peço desculpas se esqueci alguém...

Sumário

Lista de Abreviaturas **9**
Lista de Figuras **12**
Lista de Tabelas **17**
Resumo **18**
Abstract **19**
1 Introdução **20**
2 Conceitos Básicos **24**
2.1 Tipos de tráfego nas redes de computadores **25**
2.2 Transmissão multimídia multicast e unicast **27**
2.3 Ambientes heterogêneos e codificação multi-taxas **29**
2.3.1 Transmissão multicast com codificação em camadas cumulativas 30
2.3.2 Transmissão multicast com codificação em camadas não cumulativas 32
2.3.3 Transmissão multicast usando codificação com replicação 33
2.3.4 Transmissão multicast usando transcodificadores 34
2.4 Estabilidade de tráfego **34**
2.5 Eqüidade de tráfego **35**
2.6 Funcionamento do TCP **38**
2.7 Congestionamento em redes de computadores **41**
2.8 Esquemas de controle de congestionamento **44**
2.8.1 Baseado em janela e baseado em taxa 44
2.8.2 Unicast e multicast 45
2.8.3 Taxa única e multi-taxa 47
2.8.4 Apoiado pelo roteador e fim-a-fim 48
2.8.5 Orientados a transmissor e orientados a receptor 49
2.9 Mecanismos de realimentação do nível de congestionamento **49**
2.9.1 Descarte de pacotes 50
2.9.2 ECN: Explicit Congestion Notification 50
2.9.3 ICMP Source Quench 51
2.10 Políticas de gerência de filas nos roteadores **51**
2.11 Políticas de descarte de pacotes nas filas dos roteadores **52**
2.12 Tráfego CBR e VBR **52**
2.13 Protocolos de tempo real RTP e RTCP **53**
3 Trabalhos Relacionados **55**
3.1 RLM (Receiver-driven Layered Multicast) **56**
3.1.1 Pontos negativos do RLM 58
3.2 RLC (Receiver-driven Layered Congestion Control) **58**
3.2.1 Pontos negativos do RLC 61
3.3 Thin Streams **61**

3.4 Transcodificadores Intermediários ... 63
3.5 PLM ... 64
3.5.1 Pontos negativos do PLM ... 66
3.6 Redes Ativas ... 66
3.7 FLID-DL e WEBRC ... 67
3.7.1 Funcionamento do FLID ... 68
3.8 LTS ... 70
3.9 TFMCC e TFRC ... 71
3.10 MLDA e LDA+ ... 73
3.11 HALM: Hybrid Adaptation Layered Multicast ... 76
3.12 TCP Vegas ... 77
3.13 TEAR ... 80
3.14 RAP ... 82

4 O SAM: Sistema Adaptativo para Multimídia ... 84

4.1 Fonte do sinal multimídia para entrada no SAM ... 85
4.2 Módulo "descrição das taxas" ... 86
4.3 Módulos "codificador e decodificador em camadas" ... 87
4.4 Módulo "Transmissão" ... 87
4.5 Módulo "controle de congestionamento e camadas (ALM)" ... 88
4.6 Controle de limitações da máquina ... 89
4.6.1 Limitação na capacidade de processamento do receptor ... 90
4.6.2 Limitações em outros aspectos do receptor ... 90
4.7 Metodologia para as simulações dos algoritmos ... 90
4.7.1 Métricas estabelecidas para as simulações ... 91
4.7.2 Principais parâmetros relacionados à topologia e ao tráfego na rede ... 92
4.7.3 Topologia utilizada para simulações ... 94
4.7.4 Metodologia para variação dos parâmetros no simulador ... 95
4.7.5 Metodologia para análise de resultados no simulador ... 97
4.8 Definição das simulações ... 103
4.8.1 Adaptabilidade em ambientes heterogêneos ... 103
4.8.2 Escalabilidade ... 104
4.8.3 Eqüidade de tráfego ... 104
4.8.4 Estabilidade ... 105
4.8.5 Tempo de adaptação ... 105
4.8.6 Taxa de perdas ... 106

5 ALMP: ALM for Private Networks ... 107

5.1 Detalhamento do algoritmo ALMP ... 108
5.1.1 Tempo de estabilização ... 112
5.1.2 Receptores inscritos na mesma sessão ... 112
5.1.3 Receptores inscritos em sessões diferentes ... 113
5.2 Simulações do ALMP ... 114
5.2.1 Análise da adaptabilidade do ALMP em ambientes heterogêneos ... 114
5.2.2 Análise de escalabilidade do ALMP ... 121
5.2.3 Análise de eqüidade de tráfego do ALMP ... 127
5.2.4 Análise da estabilidade do ALMP ... 138

5.2.5 Análise do tempo de adaptação do ALMP 140
5.2.6 Análise da taxa de perdas do ALMP 142
5.3 Implementação do algoritmo ALMP 143
5.3.1 Modelo proposto 145
5.3.2 Problemas enfrentados 146
5.4 Conclusões sobre o ALMP 146

6 ALMTF: ALM TCP-Friendly 149

6.1 Detalhamento do algoritmo ALMTF 150
6.1.1 Detalhe do algoritmo e controle de congestionamento baseado em janela 152
6.1.2 Controle de congestionamento baseado na equação do TCP 156
6.1.3 Cálculo do RTT 157
6.1.4 Intervalo entre *feedbacks* 161
6.1.5 Cálculo de perdas 162
6.1.6 Identificação de *timeout* 162
6.1.7 Uso de pares de pacotes 163
6.1.8 Sincronização entre receptores da mesma sessão 164
6.1.9 Dessincronização de receptores de sessões diferentes 166
6.2 Simulações do ALMTF 167
6.2.1 Análise da adaptabilidade do ALMTF em ambientes heterogêneos 167
6.2.2 Análise de escalabilidade do ALMTF 176
6.2.3 Análise de eqüidade de tráfego do ALMTF 181
6.2.4 Análise da estabilidade do ALMTF 190
6.2.5 Análise do tempo de adaptação do ALMTF 192
6.2.6 Análise da taxa de perdas do ALMTF 193
6.3 Conclusões sobre o ALMTF 194

7 Comparações com outros algoritmos 196

7.1 Comparações dos algoritmos em termos de adaptação em ambientes heterogêneos e estabilidade 197
7.2 Comparações de escalabilidade dos algoritmos 200
7.3 Comparações do tempo de adaptação inicial dos algoritmos 201
7.4 Comparações de eqüidade com o próprio tráfego, estabilidade e tempo de adaptação em regime permanente 203
7.5 Comparações de eqüidade com tráfego TCP concorrente 207
7.6 Fluxo do algoritmo iniciando antes e depois do TCP 209
7.7 Conclusões sobre as comparações efetuadas 211

8 Codificação de Vídeo Multi-Taxas 214

8.1 Conceitos sobre codificação de vídeo multi-taxa 215
8.1.1 Escalabilidade temporal 215
8.1.2 Escalabilidade espacial 216
8.1.3 Escalabilidade de qualidade ou SNR 217
8.1.4 Escalabilidade por freqüência ou particionamento de dados 218
8.1.5 Escalabilidade granular fina por bitplanes 218
8.2 Trabalhos relacionados 219
8.3 Detalhamento do VEBIT 220

8.3.1 Os arquivos de entrada e saída de vídeo 222
8.3.2 A transformada 3D-DCT 223
8.3.3 Quantização 224
8.3.4 Codificação em bitplanes 224
8.4 Resultados obtidos com o codificador VEBIT 226
8.4.1 Rapidez de codificação e decodificação do VEBIT 227
8.4.2 Taxa de compressão do VEBIT 228
8.4.3 Taxa de transmissão por camada 228
8.4.4 Qualidade via PSNR 230
8.5 Extensão do VEBIT para suporte à transmissão ao vivo 231
8.6 Integração no SAM 232
8.6.1 Conseqüência dos erros 232
8.7 Conclusões e melhorias futuras 233

9 Conclusões 234

9.1 Contribuições desta Tese 236
9.2 Trabalhos futuros 237

Anexo A Análise do mecanismo de pares de pacotes 239

Anexo B Descrição do conteúdo do CD 258

Referências 262

Lista de Abreviaturas

AC	Alternate Current
ACK	Acknowledge
ADPCM	Adaptive Differential Pulse Code Modulation
ADSL	Asymmetric Digital Subscriber Line
AIAD	Additive Increase Adaptive Decrease
AIMD	Additive Increase Multiplicative Decrease
ALM	Adaptive Layered Multicast
API	Application Program Interface
BF	Bounded Fairness
CBQ	Class-Based Queuing
CBR	Constant Bit Rate
CD	Compact Disk
CE	Congestion Experienced
CIF	Common Intermediate Format
CLR	Current Limiting Receiver
CODEC	Coder-Decoder
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSRC	Contributing Source
cwnd	Congestion Window
DC	Direct Current
DCT	Discrete Cosine Transform
DS	Differentiated Services
ECN	Explicit Congestion Notification
ECT	ECN-Capable Transport
FCFS	First Come First Served
FGS	Fine Granular Scalability
FIFO	First-In-First-Out
FLID-DL	Fair Layered Increase / Decrease with Dynamic Layering
FQ	Fair Queuing
FTP	File Transfer Protocol
GOP	Group of Pictures
GPS	Generalized Processor Sharing
GPS	Global Positioning System
HALM	Hybrid Adaptation Layered Multicast
HTTP	Hypertext Transfer Protocol
H-GPS	Hierarchical Generalized Processor Sharing
ICMP	Internet Control Message Protocol
IFG	Inter Frame Gap
IGMP	Internet Group Management Protocol
IP	Internet Protocol
LDA+	Enhanced Loss-Delay based Adaptation Algorithm
LTS	Layered Transmission Scheme

LVMR	Layered Video Multicast with Retransmissions
MBONE	Multicast Backbone
MDC	Multiple De*script*ion Coding
MDMC	Multiple De*script*ion Motion Compensation
MPEG	Movie Pictures Expert Group
MIT	Massachussetts Institute of Technology
MLDA	Multicast Enhanced Loss-delay based Adaptation Algorithm
MSB	Most Significant Bit
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NAM	Network Animator
NS	Network Simulator
NTP	Network Time Protocol
OC	Optical Carrier
OSI	Open Systems Interconnection
PBM	Packet Bunch Modes
PCM	Pulse Code Modulation
Ping	Packet Internet Groper
PLM	packet Pair receiver-driven cumulative Layered Multicast
POP3	Post Office Protocol version 3
PP	Packet-Pair ou Pares de Pacotes
PPM	Portable Pixmap
PRAV	Pesquisa em Redes de Alta Velocidade
PSNR	Peak Signal to Noise Ratio
PT	Payload Type
QCIF	Quarter CIF
QoS	Quality of Service
RAP	Rate Adaptation Protocol
RED	Random Early Detection
RGB	Red Green Blue
RLC	Receiver-Driven Layered Congestion Control
RLE	Run Length Encoding
RLM	Receiver-Driven Layered Multicast
RMTP	Reliable Multicast Transport Protocol
RR	Round Robin
RSVP	Resource Reservation Protocol
RTCP	RTP Control Protocol
RTP	Real-time Transport Protocol
RTT	Round Trip Time
SAM	Sistema Adaptativo para Multimídia
SFQ	Stochastic Fair Queuing
SMTP	Simple Mail Transfer Protocol
SNR	Signal to Noise Ratio
SR	Sender Report
SSRC	Synchronization Source
ssthresh	*slow-start* Threshold
TCP	Transmission Control Protocol

TCPW	TCP Westwood
TEAR	TCP Emulation At Receivers
TFMCC	TCP-Friendly Multicast Congestion Control
TFRC	TCP-Friendly Rate Control Protocol
TFRCP	TCP-Friendly Rate Control Protocol
TOS	Type of Service
TTL	Time To Live
UCL	University College London
UDP	User Datagram Protocol
UFRGS	Universidade Federal do Rio Grande do Sul
Unisinos	Universidade do Vale do Rio dos Sinos
UTS	University of Technology in Australia
VBR	Variable Bit Rate
VCM	Variações de Camada por Minuto
VEBIT	Vídeo Escalável por Bitplanes
VoD	Video on Demand
VPN	Virtual Private Network
WEBRC	Wave and Equation Based Rate Control
WFQ	Weighted Fair Queuing
WF^2Q	Worst Case Fair Weighted Fair Queuing
WRR	Weighted Round-Robin

Lista de Figuras

FIGURA 2.1 – Diferentes aplicações em redes de computadores 26
FIGURA 2.2 – Comparação entre multicast e unicast 28
FIGURA 2.3 – Transmissão multimídia sobre multicast usando camadas 31
FIGURA 2.4 – Largura de banda para os receptores na transmissão em camadas 31
FIGURA 2.5 – Exemplo possível de transmissão e recepção de vídeo em camadas 32
FIGURA 2.6 – Transmissão multicast em camadas não cumulativas 33
FIGURA 2.7 – Comparação de uso de banda com taxas replicadas e em camadas 34
FIGURA 2.8 – *Slow-start* e *congestion avoidance* no TCP 39
FIGURA 2.9 – Introduzir um enlace de alta velocidade pode diminuir o desempenho 43
FIGURA 2.10 – Configuração balanceada e sujeita a congestionamento 43
FIGURA 2.11 – Necessidade de sincronização entre receptores da mesma sessão 46
FIGURA 2.12 – Necessidade de evitar sincronização entre sessões diferentes 47
FIGURA 2.13 – Comparação entre CBR e VBR numa transmissão de um segundo 53

FIGURA 3.1 – Classificação de esquemas de controle de congestionamento 55
FIGURA 3.2 – Funcionamento do RLM 56
FIGURA 3.3 – Topologias utilizadas para simulação do RLM 57
FIGURA 3.4 – Tempo para RLM adaptar-se 58
FIGURA 3.5 – Algoritmo do RLC 59
FIGURA 3.6 – Pontos de sincronização no RLC 60
FIGURA 3.7 – Topologias utilizadas para simulação do RLC 60
FIGURA 3.8 – Topologias usadas para simular o protocolo *Thin Streams* 62
FIGURA 3.9 – Transcodificador adaptando para sub-rede lenta 63
FIGURA 3.10 – Topologias de simulação no PLM 65
FIGURA 3.11 – Criação das tabelas de disponibilidade 67
FIGURA 3.12 – Topologia utilizada para simulação do FLID com tráfego TCP 69
FIGURA 3.13 – Momentos diferentes para receptores adaptarem-se 74
FIGURA 3.14 – Topologia utilizada na simulação do protocolo HALM 77
FIGURA 3.15 – Mecanismo de retransmissão do TCP Vegas 78
FIGURA 3.16 – Ajuste de janela no TCP Vegas 79
FIGURA 3.17 – Definição de "*round*" no TEAR 81
FIGURA 3.18 – Topologia utilizada na simulação do protocolo TEAR 82
FIGURA 3.19 – Transmissão unicast em camadas com *bufferização* no receptor 83
FIGURA 3.20 – Topologia utilizada na simulação do protocolo RAP 83

FIGURA 4.1 – Modelo do SAM 85
FIGURA 4.2 – Randomização do tráfego transmitido 94
FIGURA 4.3 – Topologias utilizadas para as simulações 94
FIGURA 4.4 – Exemplo de gráfico de camadas 97
FIGURA 4.5 – Exemplo de gráfico de variação de banda 99
FIGURA 4.6 – Exemplo do *Network Animator* 103

FIGURA 5.1 – Visão geral do algoritmo ALMP em termos da variável *bwshare* 112

FIGURA 5.2 – Receptores em sub-redes diferentes compartilhando um gargalo 113
FIGURA 5.3 – Topologia para simulações de adaptabilidade 114
FIGURA 5.4 – ALMP com camadas exponenciais padrão sem utilizar pares de pacotes 115
FIGURA 5.5 – Comparação entre a vazão obtida e ideal 116
FIGURA 5.6 – ALMP com camadas exponenciais padrão utilizando pares de pacotes... 116
FIGURA 5.7 – Comparação entre a vazão obtida e ideal 117
FIGURA 5.8 – Camadas iguais ao VEBIT sem pares de pacotes 117
FIGURA 5.9 – Comparação entre a vazão obtida e ideal 118
FIGURA 5.10 – Camadas iguais ao VEBIT com pares de pacotes 118
FIGURA 5.11 – ALMP com camadas de 50 kbit/s sem pares de pacotes 119
FIGURA 5.12 – Camadas de 50 kbit/s sem pares de pacotes 119
FIGURA 5.13 – Camadas de 100 kbit/s com pares de pacotes 120
FIGURA 5.14 – ALMP com fila RED 10/20 121
FIGURA 5.15 – Topologia para simulações de escalabilidade 122
FIGURA 5.16 – Escalabilidade do ALMP - 20 receptores sem pares de pacotes 123
FIGURA 5.17 – Comparação entre vazão obtida e vazão ideal para 20 receptores..... 123
FIGURA 5.18 – Escalabilidade do ALMP – 100 receptores sem pares de pacotes..... 124
FIGURA 5.19 – Comparação entre vazão obtida e vazão ideal para 100 receptores..... 124
FIGURA 5.20 – Escalabilidade do ALMP – 100 receptores com pares de pacotes 125
FIGURA 5.21 – 100 receptores ALMP: camadas de 100 kbit/s 127
FIGURA 5.22 – 100 receptores ALMP: camadas exponenciais, RED 10/20 127
FIGURA 5.23 – Topologia para simulações de eqüidade 128
FIGURA 5.24 – Eqüidade do ALMP para 2 fluxos concorrentes 129
FIGURA 5.25 – Eqüidade para 10 fluxos concorrentes com camadas exponenciais 129
FIGURA 5.26 –20 fluxos concorrentes com fila *droptail* de 20 130
FIGURA 5.27 –20 fluxos concorrentes com fila *droptail* de 60 130
FIGURA 5.28 – Camadas iguais ao VEBIT – enlace de 5 Mbit/s 131
FIGURA 5.29 – Camadas iguais ao VEBIT – enlace de 3,5Mbit/s 131
FIGURA 5.30 – 10 fluxos e camadas de 100 kbit/s 132
FIGURA 5.31 – 10 fluxos e camadas de 100 kbit/s (utilização da banda)..... 132
FIGURA 5.32 – Eqüidade do ALMP para dois fluxos, um iniciando 100s após o outro . 133
FIGURA 5.33 – Cinco fluxos exponenciais, cada um iniciando 100s após o outro 133
FIGURA 5.34 – Camadas de 100 kbit/s, cada fluxo iniciando 100s após o outro 134
FIGURA 5.35 – Tráfego VBR Exponencial. Banda nos receptores 135
FIGURA 5.36 – Tráfego VBR Exponencial. Camadas nos receptores..... 135
FIGURA 5.37 – Tráfego VBR Pareto. Banda nos receptores 136
FIGURA 5.38 – Tráfego VBR Pareto. Camadas nos receptores..... 136
FIGURA 5.39 – Eqüidade de tráfego do ALMP com tráfego CBR..... 137
FIGURA 5.40 – Fase *start-state* do ALMP com camadas exponenciais iguais ao VEBIT140
FIGURA 5.41 – Fase *start-state* do ALMP com camadas de 50 kbit/s 141
FIGURA 5.42 – Interface com o usuário – *receiver1* associado ao *sender1* 144
FIGURA 5.43 – Ambiente de teste 144
FIGURA 5.44 – Gráfico da adaptação do ALMP num ambiente real controlado 145
FIGURA 5.45 – Modelo da implementação..... 146

FIGURA 6.1 – Cabeçalho dos pacotes ALMTF 150
FIGURA 6.2 – Controle de fluxo do ALMTF (janela e equação) 152

FIGURA 6.3 – Criação do vetor "evento de perda" para cálculo da variável *p* 156
FIGURA 6.4 – Cálculo do RTT do ALMTF no receptor 158
FIGURA 6.5 – Cálculo do RTT em redes simétricas com relógio sincronizado 159
FIGURA 6.6 – Cálculo do RTT em redes simétricas com relógio dessincronizado 160
FIGURA 6.7 – Cálculo do RTT em redes assimétricas com relógio dessincronizado 160
FIGURA 6.8 – Cálculo de *timeout* no ALMTF 162
FIGURA 6.9 – Filtro de pares de pacotes para cálculo de banda máxima no ALMTF 164
FIGURA 6.10 – Pontos de sincronização no ALMTF 165
FIGURA 6.11 – Topologia para simulações de adaptabilidade 167
FIGURA 6.12 – Camadas exponenciais padrão sem utilizar pares de pacotes 168
FIGURA 6.13 – Comparação entre vazão obtida e ideal para 4 receptores 169
FIGURA 6.14 – Camadas exponenciais padrão utilizando pares de pacotes 169
FIGURA 6.15 – Comparação entre vazão obtida e ideal para 4 receptores 170
FIGURA 6.16 – Camadas exponenciais, sem pares de pacotes e atraso 1ms 170
FIGURA 6.17 – Camadas exponenciais iguais ao VEBIT sem pares de pacotes 171
FIGURA 6.18 – Comparação entre vazão obtida e ideal para 4 receptores 171
FIGURA 6.19 – Camadas exponenciais iguais ao VEBIT com pares de pacotes 172
FIGURA 6.20 – Camadas de 50 kbit/s sem pares de pacotes 173
FIGURA 6.21 – Camadas de 50 kbit/s com pares de pacotes 173
FIGURA 6.22 – ALMTF com camadas de 100 kbit/s 174
FIGURA 6.23 – Camadas exponenciais e pacote de 250 bytes 174
FIGURA 6.24 – ALMTF com fila RED 10/20 sem e com pares de pacotes 175
FIGURA 6.25 – Topologia para simulações de escalabilidade 176
FIGURA 6.26 – Escalabilidade do ALMTF - 20 receptores sem pares de pacotes 177
FIGURA 6.27 – Comparação entre vazão obtida e ideal para 20 receptores 177
FIGURA 6.28 – Escalabilidade do ALMTF – 100 receptores sem pares de pacotes 178
FIGURA 6.29 – Comparação entre vazão obtida e ideal para 100 receptores 179
FIGURA 6.30 – Escalabilidade do ALMTF – 100 receptores com pares de pacotes 179
FIGURA 6.31 – 100 receptores: camadas de 100 kbit/s 181
FIGURA 6.32 – 100 receptores: camadas exponenciais, RED 10/20 181
FIGURA 6.33 – Topologia para simulações de eqüidade 182
FIGURA 6.34 – Eqüidade do ALMTF para 2 fluxos concorrentes 183
FIGURA 6.35 – Eqüidade para 2 fluxos ALMTF e dois fluxos TCP concorrentes 183
FIGURA 6.36 – Eqüidade para 10 fluxos ALMTF concorrentes e camadas exponenciais 184
FIGURA 6.37 – Eqüidade para 10 fluxos ALMTF e 10 fluxos TCP concorrentes 184
FIGURA 6.38 – Eqüidade para camadas iguais ao VEBIT – enlace de 1,75 Mbit/s 185
FIGURA 6.39 – 5 ALMTF com camadas iguais ao VEBIT e 5 TCP concorrentes 185
FIGURA 6.40 – Eqüidade do ALMTF para 5 fluxos e camadas de 100 kbit/s 186
FIGURA 6.41 – 10 fluxos, sendo 5 ALMTF com camadas de 100kbit/s e 5 TCP 186
FIGURA 6.42 – Eqüidade do ALMTF para dois fluxos, um iniciando 100s após o outro 187
FIGURA 6.43 – Eqüidade para cinco fluxos, cada um iniciando 100s após o outro 188
FIGURA 6.44 – 10 fluxos (5 ALMTF, e 5 TCP) iniciando a cada 100s 188
FIGURA 6.45 – 10 fluxos (5 ALMTF, camadas 100k, e 5 TCP), iniciando a cada 100s. 189
FIGURA 6.46 – Eqüidade de tráfego do ALMTF com CBR 190
FIGURA 6.47 – Fase *start-state* do ALMTF com camadas exponenciais padrão 192
FIGURA 6.48 – Fase *start-state* do ALMTF com camadas de 50 kbit/s com PP 193

FIGURA 7.1 – Topologia para as comparações 197
FIGURA 7.2 – Adaptação dos algoritmos em redes heterogêneas 198
FIGURA 7.3 – Comparação de escalabilidade para 100 receptores 200
FIGURA 7.4 – Tempo de adaptação inicial do ALMTF, ALMP e RLM 202
FIGURA 7.5 – Tempo de adaptação inicial do RLC e do TFMCC 202
FIGURA 7.6 – Topologia para simulações de eqüidade 203
FIGURA 7.7 – Tempo de adaptação em regime permanente e eqüidade para dois fluxos 204
FIGURA 7.8 – Eqüidade entre 10 fluxos iniciando ao mesmo tempo 206
FIGURA 7.9 – Comparação com dois fluxos TCP concorrentes 208
FIGURA 7.10 – Eqüidade do ALMP começando antes e depois do TCP 209
FIGURA 7.11 – Eqüidade do ALMTF começando antes e depois do TCP 210
FIGURA 7.12 – Eqüidade do RLM começando antes e depois do TCP 210
FIGURA 7.13 – Eqüidade do RLC começando antes e depois do TCP 211
FIGURA 7.14 – Eqüidade do TFMCC começando antes e depois do TCP 211

FIGURA 8.1 – Modelo do SAM 214
FIGURA 8.2 – Redução espacial baseada em pirâmide laplaciana 217
FIGURA 8.3 – Redução espacial baseada em Wavelet 217
FIGURA 8.4 – Ponto de particionamento na escalabilidade por freqüência 218
FIGURA 8.5 – Divisão de vetor DCT em *bitplanes* 219
FIGURA 8.6 – Modelo de codificação utilizado no VEBIT 221
FIGURA 8.7 – Modelo de decodificação utilizado no VEBIT 222
FIGURA 8.8 – Modelo de decodificação utilizado no VEBIT 222
FIGURA 8.9 – Arquivo de saída gerado pelo VEBIT 223
FIGURA 8.10 – Transformada 3D-DCT no VEBIT 223
FIGURA 8.11 – Qualidade do vídeo *Carphone* do ponto de vista dos receptores 227
FIGURA 8.12 – Banda por camada do vídeo *Claire* 229
FIGURA 8.13 – Banda por camada do vídeo *Night* 229
FIGURA 8.14 – Divisão de banda por camada em todos vídeos 230
FIGURA 8.15 – PSNR do vídeo *Forest* 230
FIGURA 8.16 – Extensão ao VEBIT para suporte de vídeo ao vivo 231
FIGURA 8.17 – Diferença de padrão RGB para PPM 231
FIGURA 8.18 – Processo de obtenção dos vídeos e conversão para PPM 232

FIGURA A.1 – Funcionamento básico do mecanismo de pares de pacotes 242
FIGURA A.2 – Exemplo de inferência de banda por pares de pacotes 242
FIGURA A.3 – Problemas no uso de pares de pacotes em redes reais 243
FIGURA A.4 – Topologia da rede simulada 245
FIGURA A.5 – Inferência da banda do receptor de pares de pacotes 246
FIGURA A.6 – Pares de pacote com 64 bytes, 500 e 1000 bytes 246
FIGURA A.7 – Gargalo de 100 kbit/s e 10Mbit/s 247
FIGURA A.8 – Impacto da variação do RTT na precisão de inferência de banda 248
FIGURA A.9 – Ambiente controlado de teste 251
FIGURA A.10 – Espaçamento devido ao sistema operacional 251
FIGURA A.11 – Pacotes de 64 bytes e 1000 bytes, rajadas de 2 e 8 pacotes 252
FIGURA A.12 – Ambiente controlado, 100 kbit/s e 10 Mbit/s 253
FIGURA A.13 – Conexão Internet via ADSL e modem de 56 kbit/s 254

FIGURA A.14 – Filtro de pares de pacotes no ALMTF 255
FIGURA A.15 – Simulação 1M2M4M8M com e sem o filtro de pares de pacotes 255
FIGURA A.16 – Análise do filtro para redes de 100 kbit/s com 40 FTPs concorrentes .. 256
FIGURA A.17 – Análise do filtro para RTT de 1ms 256

Lista de Tabelas

TABELA 4.1 – Exemplo hipotético de camadas transmitidas 87

TABELA 5.1 – Análise de estabilidade, perdas e vazão para o algoritmo ALMP 126
TABELA 5.2 – Estabilidade – número de variações de camada por minuto (VCM) 139
TABELA 5.3 – Taxa de perdas para simulações do ALMP 142

TABELA 6.1 – Análise de estabilidade, perdas e vazão para o algoritmo ALMTF 180
TABELA 6.2 – Estabilidade do ALMTF – variações de camada por minuto (VCM)..... 191
TABELA 6.3 – Taxa de perdas para simulações do ALMTF 194

TABELA 7.1 – Estabilidade dos algoritmos para a simulação de adaptabilidade 199
TABELA 7.2 – Estabilidade e vazão para dez fluxos concorrendo entre si 205
TABELA 7.3 – Detalhe da estabilidade para os dez fluxos 207
TABELA 7.4 – Vazão para dois fluxos TCP concorrendo com dois do algoritmo 208
TABELA 7.5 – Estabilidade para dois fluxos TCP concorrendo com dois do algoritmo. 209
TABELA 7.6 – Resumo das comparações entre os algoritmos 213

TABELA 8.1 – Comparação entre MPEG-4 e diferentes codificadores em camada..... 220
TABELA 8.2 – Distribuição dos bits DC nas diferentes camadas, sem contar o MSB 225
TABELA 8.3 – Distribuição dos bits AC nas diferentes camadas..... 226
TABELA 8.4 – Desempenho da rotina de codificação para os vídeos Forest e Night 227
TABELA 8.5 – Desempenho da rotina de decodificação para o vídeo Night..... 228
TABELA 8.6 – Compressão do Vídeo Carphone 228

Resumo

Esta Tese apresenta o SAM (Sistema Adaptativo para Multimídia), que consiste numa ferramenta de transmissão e recepção de sinais multimídia através de redes de computadores. A ferramenta pode ser utilizada para transmissões multimídia gravadas (vídeo sob demanda) ou ao vivo, como aulas a distância síncronas, *shows* e canais de TV. Seu maior benefício é aumentar o desempenho e a acessibilidade da transmissão, utilizando para isso um sinal codificado em camadas, onde cada camada é transmitida como um grupo multicast separado. O receptor, utilizando a ferramenta, adapta-se de acordo com a sua capacidade de rede e máquina no momento. Assim, por exemplo, um receptor com acesso via modem e outro via rede local podem assistir à transmissão na melhor qualidade possível para os mesmos.

O principal foco da Tese é no algoritmo de controle de congestionamento do SAM, que foi denominado ALM (*Adaptive Layered Multicast*). O algoritmo ALM tem como objetivo inferir o nível de congestionamento existente na rede, determinando a quantidade de camadas que o receptor pode suportar, de forma que a quantidade de banda recebida gere um tráfego na rede que seja eqüitativo com outros tráfegos ALM concorrentes, ou outros tráfegos TCP concorrentes.

Como se trata de transmissões multimídia, é desejável que a recepção do sinal seja estável, ou seja, sem muitas variações de qualidade, entretanto, o tráfego TCP concorrente é muito agressivo, dificultando a criação de um algoritmo estável. Dessa forma, desenvolveu-se dois algoritmos que formam o núcleo desta Tese: o ALMP (voltado para redes privativas), e o ALMTF (destinado a concorrer com tráfego TCP).

Os elementos internos da rede, tais como os roteadores, não necessitam quaisquer modificações, e o protocolo funciona sobre a Internet atual. Para validar o método, se utilizou o software NS2 (*Network Simulator*), com modificações no código onde requerido. Além disso, efetuou-se uma implementação inicial para comparar os resultados das simulações com os da implementação real.

Em http://www.inf.unisinos.br/~roesler/tese e também no CD anexo a esta Tese, cuja descrição encontra-se no Anexo B, podem ser encontrados todos os programas de apoio desenvolvidos para esta Tese, bem como a maior parte da bibliografia utilizada, o resultado das simulações, o código dos algoritmos no simulador e o código do algoritmo na implementação real.

Palavras-chave: Adaptabilidade, Multimídia, Multicast, Transmissão em Camadas, Controle de Congestionamento.

TITLE: “SAM: AN ADAPTIVE SYSTEM FOR MULTIMEDIA TRANSMISSIONS IN COMPUTER NETWORKS”

Abstract

This Thesis presents SAM (Adaptive System for Multimedia), a tool for transmission and reception of multimedia signals through computer networks. The tool can be used in recorded multimedia transmissions (video on demand) or in live video transmissions, like synchronous distance classes, shows and TV channels. Its greatest benefit is to increase transmission accessibility and performance, using a layered coded signal where each layer is transmitted as a separate multicast group. The receiver, using this tool, adapts according to his/her network condition and machine at that specific moment. So, for example, if one receiver is connected through a modem and another via local network, both of them can watch the transmission using the best possible quality for each one.

The main focus of this Thesis is in SAM’s congestion control algorithm, named ALM (Adaptive Layered Multicast). ALM’s algorithm infers the network congestion level, defining the number of layers which the receiver can afford, in a way that the amount of received bandwidth is friendly with other TCP flows, as well as fair with other flows of its own algorithm.

In multimedia transmissions, it is important a stable signal reception, i.e. without many quality changes. However, TCP traffic is very aggressive, complicating the creation of a stable algorithm. So, it was developed two algorithms which are the Thesis’ core: the ALMP (ALM for Private networks) and the ALMTF (ALM TCP Friendly).

The internal elements of the network, such as routers, do not need any modification. To validate the method, the NS2 (Network Simulator) software was used, with modifications where required. Various simulations were undertaken to prove the algorithm’s functionality.

In http://www.inf.unisinos.br/~roesler/tese and also in the CD annex to this Thesis, whose description is in Anexo B, are all the programs, simulations, source-code, implementation, most of references, and so on.

Keywords: Adaptability, Multimedia, Multicast, Layered Transmission, Congestion Control.

1 Introdução

A utilização de transmissões multimídia em redes de computadores está se tornando comum nos últimos anos, pois som e vídeo se integram harmoniosamente às páginas estáticas de texto e imagens existentes na Internet, acrescentando dinamismo e tendo um papel importante para a assimilação das informações existentes. Segundo Bo Li [LI 2003], a distribuição de vídeo em tempo real está emergindo como uma das aplicações mais importantes em IP multicast, sendo um componente essencial de muitas necessidades atuais, como videoconferência e ensino a distância. Entretanto, transmissões em tempo real exigem certas garantias da rede, como, por exemplo, um tempo máximo de latência em todos os pacotes, estabilidade na transmissão e uma largura de banda mínima para essa aplicação. Nesse contexto se encontra o tema desta Tese, que é: *acessibilidade ao conteúdo de transmissões multimídia via redes de computadores*.

Apesar da transmissão de vídeos na rede ser de uso corriqueiro atualmente, alguns problemas ainda são tópicos de pesquisa, e um deles é a transmissão de vídeo multicast em multi-taxa através de redes *best-effort* [LI 2003], pois deve ser considerado, além da topologia da rede e seu nível de congestionamento, também a vontade do usuário e capacidade da sua máquina. O principal problema que esta Tese busca abordar no contexto do tema proposto é: *como obter acesso universal à transmissão multimídia sendo efetuada?* Isso quer dizer que todos usuários devem conseguir acesso a determinada transmissão, independente de estarem em redes congestionadas, enlaces lentos, possuírem máquinas com baixo poder de processamento, resolução de tela, e assim por diante. Dentro desse problema abordado, surgem outras questões essenciais relacionadas, descritas a seguir:

1. *Como fazer para que o sistema se adapte automaticamente às condições de rede e máquina do usuário?* Isso quer dizer que o sistema deve ser adaptativo e automático, ou seja, o algoritmo a ser desenvolvido deve reconhecer a capacidade individual do seu receptor e adaptar-se para que o mesmo obtenha a melhor qualidade possível, dentro de suas limitações;
2. *Como fazer para que o sistema utilize uma parcela eqüitativa de tráfego na Internet atual?* Isso significa que o algoritmo a ser desenvolvido deve funcionar de forma "imparcial"[1] (divisão de banda eqüitativa com seu próprio tráfego) e "amigável"[2] (divisão de banda eqüitativa com tráfegos de outros tipos). Além disso, como a Internet atual não possui mecanismos de garantia de qualidade de serviço largamente difundidos, o algoritmo deve funcionar numa rede *best-effort*. Mesmo com garantias de qualidade de serviço, tráfegos com a mesma prioridade concorrem entre si, necessitando mecanismos de compartilhamento de banda;
3. *Como fazer para que o sistema se mantenha estável ao longo do tempo?* Isso é necessário para evitar que o usuário perceba mudanças significativas no sinal recebido, o que torna a transmissão desagradável de assistir.

[1] Na língua inglesa, o termo utilizado é "*fair*", e o objetivo dos algoritmos é atingir "*fairness*".
[2] Na língua inglesa, o termo utilizado é "*friendly*", e o objetivo dos algoritmos é atingir "*friendliness*".

Buscando aprimorar o conhecimento existente em relação às questões acima citadas, se utilizou como base resultados científicos de trabalhos anteriores, que serão abordados no capítulo 3. Em função disso, algumas hipóteses foram elaboradas, pesquisadas e validadas através de simulações e implementação de um aplicativo destinado a transmissões adaptativas. As principais hipóteses abordadas neste trabalho foram as seguintes:

- Se o transmissor dividir o sinal a ser transmitido em camadas, conforme definição no capítulo 2, então dará condições ao receptor adaptar-se às suas condições de tráfego e de máquina, garantindo acesso universal à transmissão. O diferencial em relação aos trabalhos anteriores é a existência das camadas cumulativas e opcionais simultaneamente;
- Se os receptores utilizarem o algoritmo de adaptação ao congestionamento na rede denominado ALM (*Adaptive Layered Multicast*), que é parte integrante do sistema SAM (Sistema Adaptativo para Multimídia), ambos propostos nesta Tese (capítulos 4, 5 e 6), então eles devem receber a informação de forma estável, imparcial, amigável e adaptável às condições da rede;
- É possível efetuar transmissões multimídia estáveis em redes de computadores sem a necessidade de QoS (*Quality of Service*) ou quaisquer alterações nos roteadores intermediários.

Assim, o restante deste trabalho explica o sistema proposto e mostra a validação das hipóteses citadas acima. A base para isso é o sistema SAM, descrito no capítulo 4. A metodologia seguida para validar o método proposto foi inicialmente através de simulações, utilizando o simulador NS2 (*Network Simulator*[1]) [MCC 2003]. Após a fase de simulações, efetuou-se a implementação do algoritmo, e diversos testes foram realizados buscando analisar o desempenho do sistema para cada um dos problemas citados acima.

O objetivo final deste trabalho é obter um novo método para controle de congestionamento e adaptação automática na rede, visando sua utilização em uma ferramenta de transmissão de vídeo em redes de computadores. Tal ferramenta pode ser utilizada em diversas aplicações onde é necessária a transmissão de vídeo de forma adaptativa, como ensino a distância, transmissão de *shows*, conferências, canais de TV, e assim por diante.

O restante deste trabalho está estruturado da seguinte forma: o capítulo 2 (Conceitos Básicos) trata de alguns conceitos necessários ao entendimento dos outros capítulos, como multicast, controle de congestionamento e transmissão em camadas. No capítulo 3 (Trabalhos Relacionados) é feito um estudo aprofundado de alguns trabalhos relacionados ao desenvolvido nesta Tese.

Como modelo de utilização dos algoritmos propostos nesta Tese, desenvolveu-se uma proposta de arquitetura para transmissão e recepção de sinais multimídia. Essa arquitetura foi denominada SAM, e será mostrada no capítulo 4 (O SAM: Sistema Adaptativo para Multimídia), bem como sua aplicabilidade e forma de funcionamento.

[1] Projeto VINT – *Virtual InterNetwork Testbed*. Em http://www.isi.edu/nsnam/vint (20/10/2003)

O módulo principal do SAM está localizado no receptor, sendo responsável pela determinação do número de camadas que o receptor deve se inscrever. Esse módulo é a principal contribuição desta Tese, e o algoritmo responsável pela adaptação foi denominado, de uma forma geral, ALM (*Adaptive Layered Multicast*). Existem duas variantes deste algoritmo, que foram analisadas via simulação. A metodologia de simulação é vista no item 4.7 (Metodologia para as simulações dos algoritmos).

A primeira variante do ALM é mais adequada ao uso em redes privativas, sendo detalhada no capítulo 5 (ALMP: ALM for Private Networks). Esse algoritmo possui parâmetros flexíveis de forma a possibilitar sua concorrência com o TCP, entretanto, é mais adequado ao uso em redes privativas, pois os parâmetros podem ser adaptados a fim de que se obtenha uma maior estabilidade.

A segunda variante do ALM foi denominada ALMTF (*ALM TCP Friendly*), sendo descrita no capítulo 6 (ALMTF: ALM TCP-Friendly). Seu principal objetivo é a transmissão de tráfego de forma eqüitativa como o TCP buscando manter a estabilidade na transmissão, ou seja, seu uso é voltado para a Internet atual.

Para uma melhor visualização dos pontos fortes e fracos dos algoritmos, o capítulo 7 (Comparações com outros algoritmos) apresenta uma comparação entre o ALM e outros protocolos analisados em detalhes no capítulo 3. Os algoritmos são comparados principalmente em termos de adaptabilidade, eqüidade e estabilidade.

Dois importantes módulos do SAM formam o codificador e o decodificador em camadas, portanto, o capítulo 8 (Codificação de Vídeo Multi-Taxas) tem como objetivo mostrar a realidade atual nesta área e descrever a implementação do VEBIT (Vídeo Escalável por Bitplanes), que foi desenvolvido em paralelo com esta Tese como uma dissertação de mestrado [BRU 2003], visando sua integração no SAM. O VEBIT efetua codificação e decodificação de vídeo em até 5 camadas.

O capítulo 9 (Conclusões) trata das conclusões do trabalho, enquanto o item 9.2 (Trabalhos futuros) oferece algumas sugestões para continuidade do mesmo.

O Anexo A (Análise do mecanismo de pares de pacotes) mostra um estudo aprofundado sobre a possibilidade de estimar a banda máxima na rede através do método de pares de pacotes, utilizado extensivamente em outros trabalhos, bem como no módulo transmissor do SAM. O estudo foi baseado em simulações e na implementação de uma ferramenta específica para a medição de banda utilizando o método em questão. Esta análise ficou em um anexo separado, pois é um estudo completo somente do mecanismo de pares de pacotes. A visão geral da integração do método no SAM é definida nos capítulos 4, 5 e 6.

Após o estudo aprofundado dos mecanismos de controle de congestionamento e a análise da característica da transmissão multimídia, chegou-se à conclusão que é praticamente impossível manter a estabilidade quando existe tráfego TCP concorrente, pois este é muito agressivo. Assim, em [ROE 2003j], sugere-se a implementação de um novo protocolo, que utiliza o apoio dos roteadores intermediários e possui uma série de vantagens em relação ao TCP tradicional, como a possibilidade de transmissão confiável e também não-confiável, de forma eqüitativa entre todas as sessões envolvidas e sem geração de erros para determinação da fatia de banda a ser utilizada pelo transmissor.

O Anexo B (Descrição do conteúdo do CD) descreve o conteúdo do CD anexo a esta Tese, que contém todos os programas de apoio desenvolvidos para este trabalho, bem como a maior parte da bibliografia utilizada, o resultado das simulações, o código dos algoritmos no simulador e o código do algoritmo na implementação real. Um outro local com a mesma informação é http://www.inf.unisinos.br/~roesler/tese.

2 Conceitos Básicos

O objetivo deste capítulo é definir alguns conceitos que serão utilizados no decorrer desta Tese, e aos quais o leitor deve estar familiarizado a fim de compreender o trabalho desenvolvido.

A Internet atual possui uma grande diversidade de aplicações funcionando concomitantemente, algumas de tempo real e outras sem limites rígidos para a transmissão, porém, não existem mecanismos de QoS largamente difundidos de forma a garantir a transmissão de tráfego de tempo real em meio a diferentes tipos de aplicações, ou seja, aplicações com objetivos diversos. O item 2.1 (Tipos de tráfego nas redes de computadores) detalha as diferentes necessidades e características das aplicações existentes.

Como o objetivo desta Tese é abordar especificamente o problema de aplicações multimídia que geram tráfego de tempo real e necessitam uma grande largura de banda, como por exemplo a transmissão de uma aula remota, o item 2.2 (Transmissão multimídia multicast e unicast) analisa o funcionamento de transmissões multimídia, tanto em unicast como em multicast.

Além disso, existe uma grande heterogeneidade de equipamentos, enlaces e condições de acesso. As topologias são muito variadas, e onde de um lado existem usuários com acesso rápido, de outro estão os usuários localizados em redes congestionadas, ou com conexões via linha discada, ADSL e *Cable Modem*. Isso torna extremamente difícil, se não impossível, haver um acordo entre os receptores sobre a melhor taxa de codificação a ser utilizada. Caso a qualidade de codificação do vídeo no transmissor seja muito baixa, os usuários com mais largura de banda e/ou poder de processamento estarão subutilizando seus recursos. Por outro lado, caso a qualidade de codificação seja muito alta, muitos usuários ficarão impossibilitados de acessar a informação.

Existem várias técnicas propostas para transmitir sinais multimídia nesse ambiente, e essas técnicas normalmente envolvem adaptação de forma a atender as características de cada receptor. Uma das formas de resolver o problema da heterogeneidade é a utilização de codificação em camadas, onde cada camada é transmitida em uma qualidade complementar, de forma que a soma de todas as camadas forme a melhor transmissão. O receptor se adapta para receber a quantidade de camadas que a sua rede e máquina permitir. O funcionamento da transmissão em camadas é descrito em detalhes no item 2.3 (Ambientes heterogêneos e codificação multi-taxas), bem como sua comparação em relação a transmissão com taxas replicadas.

Um cuidado que toda aplicação de transmissão de áudio e vídeo deve ter é a estabilidade da transmissão, ou seja, qual nível de variação na qualidade recebida é aceitável do ponto de vista do usuário. Esse assunto é analisado no item 2.4 (Estabilidade de tráfego).

Para que o receptor se adapte corretamente, é necessário saber qual a largura de banda disponível até o transmissor, mantendo a justiça com o tráfego TCP. É importante

entender o funcionamento do TCP, pois ele é o protocolo "modelo" na Internet, ou seja, todos os outros algoritmos devem adaptar-se a ele de forma cooperativa. Isso é necessário, pois o TCP corresponde a mais de 90% do tráfego da Internet [HUS 2000b], [THO 97], e a criação de protocolos que não sigam suas características provocaria um impacto muito grande, prejudicando as aplicações existentes. O item 2.5 (Eqüidade de tráfego) trata da obrigatoriedade de manter a eqüidade de tráfego com fluxos TCP concorrentes.

Para que se consiga criar um novo algoritmo que seja compatível com o TCP, é necessário entender bem seu funcionamento, e é esse o objetivo do item 2.6 (Funcionamento do TCP). Além disso, é necessário compreender que a Internet é um ambiente bastante variado, onde existem congestionamentos causados por diversos fatores, que são detalhados no item 2.7 (Congestionamento em redes de computadores). Para lidar com o congestionamento, é necessário efetuar o controle da rede, seja com adaptação por taxa, janela ou outro método, e o estudo desses métodos é o objetivo do item 2.8 (Esquemas de controle de congestionamento). Para que o esquema de controle de congestionamento funcione, ele necessita realimentação da rede visando inferir o nível de tráfego existente no momento, e o estudo desses mecanismos é o objetivo do item 2.9 (Mecanismos de realimentação do nível de congestionamento).

Dependendo do nível de congestionamento, os roteadores tomam determinadas atitudes de acordo com suas políticas, e essas atitudes são importantes para a implementação de qualquer novo algoritmo. Essas políticas são detalhadas nos itens 2.10 (Políticas de gerência de filas nos roteadores) e 2.11 (Políticas de descarte de pacotes nas filas dos roteadores).

Outro conceito importante na transmissão de dados é o tipo de tráfego que vai ser compartilhado na rede, e também o tipo que o algoritmo vai utilizar, ou seja, se o tráfego possui uma taxa constante ou variável. Essas conceituações podem ser vistas no item 2.12 (Tráfego CBR e VBR).

Em transmissões de tempo real, principalmente utilizando UDP (*User Datagram Protocol*), o receptor deve possuir certas informações, como o número de seqüência do pacote, a fim de poder detectar perdas, ou o instante de tempo no qual o pacote deixou o transmissor, a fim de reproduzi-lo corretamente, e para isso existem os protocolos analisados no item 2.13 (Protocolos de tempo real RTP e RTCP).

A seguir encontra-se o detalhamento dos vários itens descritos acima.

2.1 Tipos de tráfego nas redes de computadores

As aplicações transmitidas em redes de computadores se encontram subdivididas em dois tipos principais, como ilustra a figura 2.1: as que geram tráfego elástico e as que geram tráfego inelástico.

Segundo Croll [CRO 2000], as aplicações que geram tráfego elástico necessitam confiabilidade de entrega, dando mais valor à recepção do tráfego na ordem correta e sem perdas, sem necessidade de apresentar para o usuário os dados à medida que eles vão chegando. Isso faz com que esse tipo de aplicação suporte atrasos na recepção sem prejudicar o usuário. Na figura, esse tipo de tráfego está associado a "Dados / Texto /

Imagem", pois são as classes de conteúdo mais utilizadas com tráfego elástico. Quando se transmite vídeo através de aplicações desse tipo, o vídeo vem codificado como dados e é transmitido de forma assíncrona e com confiabilidade de entrega, tornando problemática sua apresentação em tempo real.

As aplicações que geram tráfego elástico constituem a classe mais utilizada atualmente na Internet, e exemplos desse tipo de aplicações são POP3 (*Post Office Protocol* versão 3), SMTP (*Simple Mail Transfer Protocol*), HTTP (*Hypertext Transfer Protocol*), telnet, FTP (*File Transfer Protocol*), e assim por diante. Normalmente esse tipo de aplicação utiliza o TCP no nível 4 do modelo OSI, devido à sua confiabilidade na entrega dos pacotes, entretanto, em certas situações, o TCP pode não ser utilizado, como em alguns protocolos de multicast confiável.

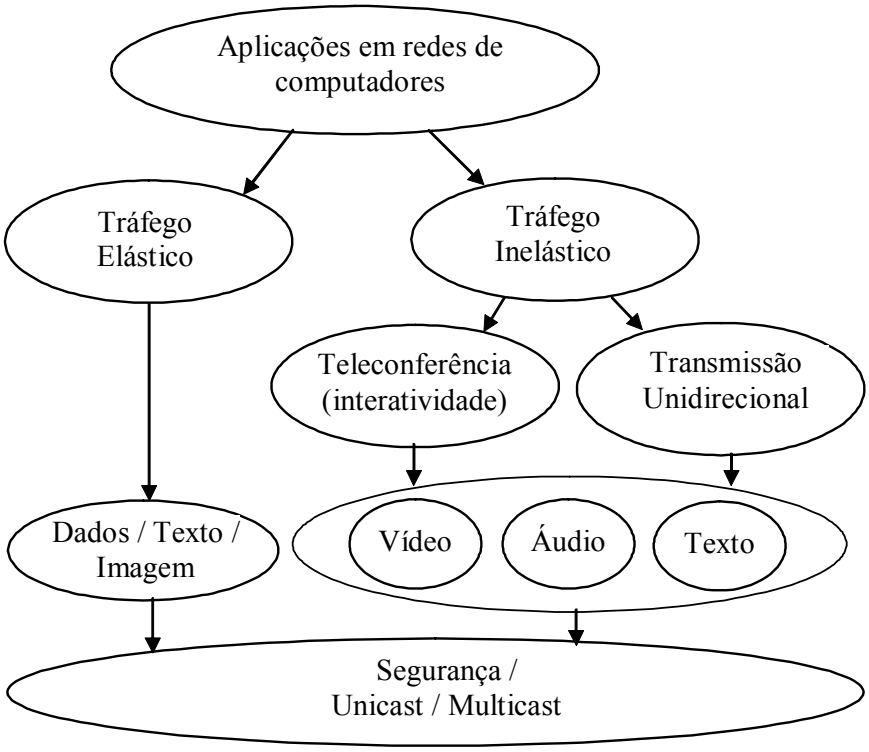


FIGURA 2.1 – Diferentes aplicações em redes de computadores

Já as aplicações que geram tráfego inelástico possuem restrições de tempo, pois devem apresentar o resultado ao usuário dentro de um certo período, caso contrário o dado não tem mais valor, podendo ser descartado. Esse tipo de tráfego utiliza normalmente o protocolo UDP no nível 4 do modelo OSI, não requerendo retransmissões, pois as mesmas poderiam fazer o dado atrasar demasiadamente, se tornando inútil. Como pode ser visto na figura 2.1, é possível dividir esse tipo de aplicação em dois outros tipos de acordo com sua rigidez em relação ao tempo: teleconferência (que requer interatividade) e transmissão unidirecional (que envolve apenas um lado transmitindo e um ou mais clientes recebendo). Exemplos de aplicações desse tipo são:

- Áudio: com interatividade (conversa telefônica na Internet) ou sem interatividade (transmissão de rádio);
- Vídeo: com interatividade (videoconferência) ou sem interatividade (transmissão de

TV);
- Texto: com interatividade (*Chat*) ou sem interatividade (cotação da bolsa).

As aplicações que requerem interatividade possuem necessidades mais rígidas em relação à latência[1] e *jitter*[2] do que aplicações de transmissão unidirecional, como rádio ou TV. Isso se deve justamente ao contexto de teleconferência, que exige rapidez na entrega dos pacotes para viabilizar a interação confortável entre os participantes. Segundo Passmore [PAS 97], um atraso de 100 ms é aceitável para voz sobre IP, e acima disso começa a se tornar desconfortável para o usuário. Xue Li [LI 99] relata que o atraso máximo para aplicações interativas deve ser inferior a 200 ms.

Já no caso de transmissões unidirecionais, basta que o receptor armazene em memória alguns pacotes antes de começar a mostrar o sinal ao usuário, pois as restrições de latência são menores, como mostra o exemplo detalhado em [ROE 2003g], onde o usuário começa a assistir uma palestra com 20 s de atraso em relação à palestra ao vivo, sem prejuízo algum.

Associado a isso, a figura 2.1 mostra que cada classe de aplicação pode necessitar segurança (dados criptografados), transmissão unicast (serviço mais personalizado) ou multicast (buscando economia de banda e processamento). O uso dessas formas de transmissão vai depender da aplicação em questão.

Apesar das aplicações possuírem necessidades diferentes, existe uma tendência atualmente para sua convergência em um único meio físico, criando uma estrutura conhecida como redes multiserviço [CRO 2000]. O objetivo das redes multiserviço é unificar o meio físico, a fim de que ele possa compartilhar a transmissão das várias mídias, como texto, voz, vídeo, imagens, músicas, e tudo que possa ser transformado em bits.

Detalhes mais aprofundados sobre o significado de latência, *jitter*, tipos de atraso nas redes e estudos de caso sobre transmissões multimídia sobre unicast e multicast podem ser vistos em [CRO 2000], [PAU 98] ou [BAR 2000]. Um relatório técnico sobre o assunto foi desenvolvido e está disponível em [ROE 2003g].

2.2 Transmissão multimídia multicast e unicast

Dentro deste trabalho foi desenvolvido um estudo detalhado sobre o funcionamento da transmissão multicast, bem como sua forma de endereçamento, uso do protocolo IGMP (*Internet Group Management Protocol*) e protocolos de roteamento multicast. Além disso, foi feito um estudo de caso onde se efetuou codificação e transmissão de vídeo em multicast. Este estudo está disponível em [ROE 2003d], e não foi incluído nesta Tese a fim de que se mantenha o foco nas hipóteses lançadas e solução dos problemas descritos no capítulo anterior, entretanto, aconselha-se a leitura do texto para quem não está familiarizado com tais conceitos.

O grande ponto a favor do multicast é que ele permite uma distribuição

[1] Tempo de atraso de um pacote, da origem até o destino.
[2] Variação estatística da latência.

simultânea para um grande número de clientes, sem exigir muito dos recursos do servidor e sem gerar muito tráfego na rede. Assim, essa tecnologia facilita a existência de uma nova geração de aplicações "um para muitos" em rede, como tráfego de vídeo, áudio, trabalho colaborativo, tecnologia "*push*", transmissão de arquivos simultaneamente para muitos usuários, *shows* ao vivo, palestras de conferências ao vivo, aulas a distância e canais de TV. As novas aplicações podem ou não exigir confiabilidade (transmissão livre de erros), entretanto, para transmissões multicast confiáveis (como transmissão de arquivos), existe uma complexidade adicional, que é controlar o fluxo de retorno por parte dos receptores, evitando o problema conhecido como "*implosão de feedback*".

A figura 2.2 mostra uma comparação entre multicast e unicast em um ambiente com três *switchs* ligados através de um roteador que suporta multicast. No caso, a aplicação que roda no transmissor (vídeo, por exemplo) está habilitada a gerar tráfego multicast na rede, e também tráfego unicast para cada cliente que solicita. As duas situações são comparadas a seguir:

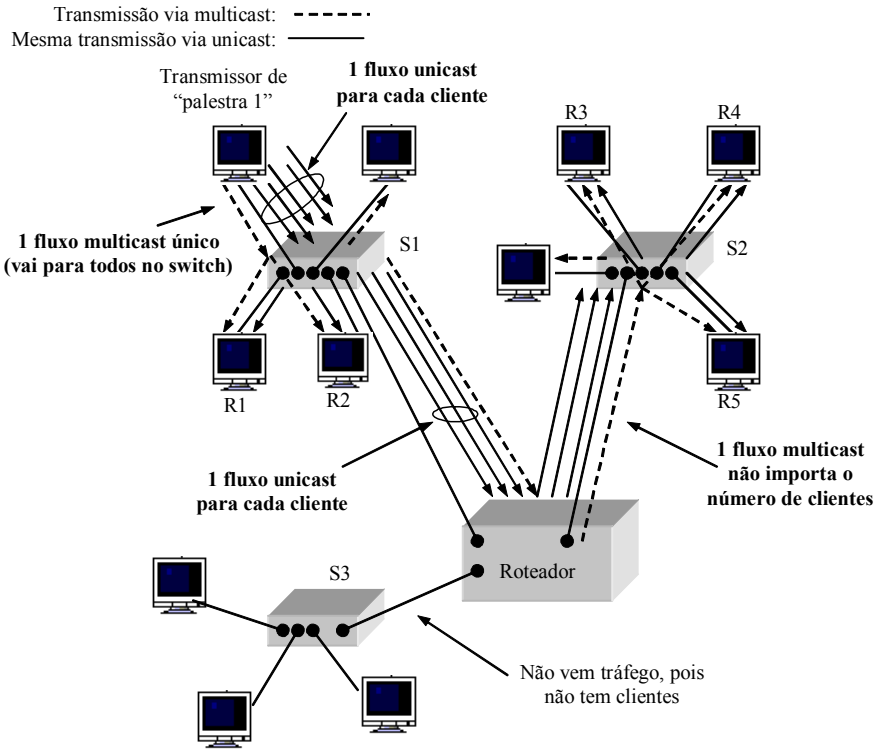


FIGURA 2.2 – Comparação entre multicast e unicast

- **Tráfego multicast**: o transmissor gera um fluxo de pacotes IP multicast no *switch* S1, que distribui para todas suas portas (inclusive a do roteador). Os clientes cadastrados no grupo multicast recebem a informação e a apresentam na tela. O roteador, por sua vez, verifica se alguma de suas portas possui clientes cadastrados no grupo multicast. Caso positivo (*switch* S2), envia um fluxo multicast para essa porta. O *switch* o distribui em todas suas portas via IP multicast. Como não existem clientes cadastrados no *switch* S3, não é gerado tráfego para essa porta. Existem *switchs* mais modernos que suportam multicast e compreendem o protocolo IGMP, enviando fluxo multicast somente para os receptores cadastrados no grupo;

- **Tráfego unicast**: para cada cliente, o transmissor deve gerar um fluxo de pacotes IP unicast. Assim, na figura 2.2, pode-se ver que o transmissor deve gerar cinco vezes o mesmo tráfego, uma vez para cada cliente que o solicitou, e a rede deve suportar cinco vezes mais largura de banda, o que não acontece no multicast. Entretanto, isso permite uma maior individualidade aos usuários, pois cada um pode assistir a transmissão num instante de tempo diferente, além de poder avançar rapidamente o vídeo. Exemplos típicos de aplicações de transmissão multimídia em unicast são *shows* disponibilizados através de um servidor de Vídeo sob Demanda (VoD), servidor de notícias, palestras, aulas pré-gravadas e programas de TV criados para acesso aleatório por parte dos usuários.

Como foi visto, na transmissão multicast todos os receptores cadastrados num determinado grupo recebem a mesma informação simultaneamente, economizando tráfego na rede e processamento no transmissor. Entretanto, o uso do multicast só se justifica quando existir mais de um receptor querendo receber a transmissão, e quanto mais receptores estiverem assistindo, melhor o custo-benefício em relação ao unicast.

Os clientes que desejam receber determinado tráfego de IP multicast, como a *palestra 1*, vista na figura 2.2, devem se inscrever no grupo multicast em que está sendo gerada essa informação. Isso é feito através da mensagem de IGMP *join* no grupo multicast de "*palestra 1*". Os grupos IP multicast ficam no intervalo entre 224.0.0.0 a 239.255.255.255, e qualquer mensagem com IP dentro desse intervalo corresponde a uma transmissão multicast. Quando o receptor quer deixar o grupo multicast e parar de receber a transmissão, deve avisar através do envio da mensagem IGMP *leave*.

2.3 Ambientes heterogêneos e codificação multi-taxas

Um dos grandes desafios atualmente na Internet é permitir acesso universal às transmissões multimídia, mesmo para receptores localizados em ambientes heterogêneos, que são definidos nesta Tese como os seguintes:

- Enlaces físicos com diferentes características topológicas, com larguras de banda que podem variar em várias ordens de magnitude. Por exemplo, modem de 56 kbit/s, modem a cabo, ADSL, acesso via rede local de 10 Mbit/s, 100 Mbit/s ou 1Gbit/s, acesso via Internet2, e assim por diante;
- Enlaces livres e congestionados, onde o receptor pode utilizar mais ou menos banda dinamicamente, dependendo do nível de congestionamento atual;
- Receptores com características diversas, tais como resolução da tela e capacidade de processamento.

O multicast permite o acesso a um grande número de usuários, mas a diversidade da rede exige que o transmissor utilize algum método para permitir acesso tanto para os receptores localizados em enlaces rápidos como aos localizados em enlaces lentos, caso contrário, a taxa única a ser transmitida teria que ser equivalente à do receptor mais lento, podendo subutilizar a possibilidade de recepção dos usuários localizados em enlaces rápidos.

Uma solução para esse problema é a transmissão do sinal em multi-taxas, ou

seja, com diversos níveis de codificação. Isso pode ser conseguido por três formas [KIM 2001]: a) através da codificação em camadas cumulativas; b) através da codificação em camadas não cumulativas; c) através da codificação com replicação. Além dessas técnicas, ainda existe a possibilidade da taxa adicional ser obtida num agente intermediário entre o transmissor e o receptor, através do uso de transcodificadores intermediários. Esses itens serão detalhados a seguir.

2.3.1 Transmissão multicast com codificação em camadas cumulativas

Na codificação e transmissão multicast em camadas[1], o transmissor envia um sinal multicast básico e diversas camadas (grupos multicast) de refinamento, onde cada uma delas adiciona qualidade à anterior. A equação a seguir mostra matematicamente o resultado, e pode-se ver que a banda necessária para uma determinada camada *i* é a soma (união) das camadas iguais e menores a ela [MCC 96]. Entretanto, segundo [LI 2003], o *overhead* causado por codificações reais para todas as camadas chega a 20%, ou seja, caso um receptor se cadastre em todas as camadas, ele vai exigir um tráfego na rede 20% maior do que se fosse em taxa única, isso sem considerar o tráfego das mensagens de controle multicast.

$$Bi = \bigcup_{j=0}^{j=i} Cj$$

Neste documento, essa estrutura de codificação é definida como "**camadas cumulativas**", pois a camada superior complementa as anteriores, adicionando qualidade. Em linhas gerais, caso o receptor esteja num enlace lento, ou tenha uma máquina lenta, ele se cadastra somente na camada básica, ou nas primeiras camadas, de acordo com sua capacidade. Caso o receptor se encontre num enlace rápido com uma máquina rápida, ele pode se cadastrar em mais camadas.

Existem diversos exemplos de codificação em camadas, como o utilizado pelo LVMR (*Layered Video Multicast with Retransmissions*) [LI 98], onde uma camada é composta dos quadros tipo I do MPEG, outra é composta pelos quadros tipo P e outra pelos quadros tipo B. Outro exemplo é o utilizado por [ACH 2003], onde os diferentes objetos do MPEG-4 são enviados como camadas separadas, assim, uma camada poderia ser a voz, enquanto outra poderia ser o palestrante e outra o fundo. Conjuntamente com esta Tese desenvolveu-se uma dissertação de mestrado onde foi efetuada a implementação de um codificador em camadas [BRU 2003], e o mesmo encontra-se descrito no capítulo 8.

A figura 2.3 ilustra um modelo com três camadas (C1 a C3) transmitindo em grupos multicast diferentes. No caso, a taxa de transmissão da camada C1 é 50 kbit/s e este sinal é transmitido através do seu grupo multicast G1, que possui o número IP 239.1.1.1; a taxa de C2 é 150 kbit/s, sendo transmitido no grupo multicast 239.1.1.2; a taxa de C3 é 500 kbit/s, e seu grupo multicast é 239.1.1.3. Os receptores R3 e R4 se associaram somente à camada mais básica, recebendo uma qualidade baixa e onerando pouco a rede, que suporta, no máximo, 64 kbit/s. Já os receptores R1 e R2 detectaram que a rede suporta 2 Mbit/s e se associaram a todos os fluxos, recebendo uma qualidade teórica igual à soma de todas as larguras de banda utilizadas, ou seja, 700 kbit/s.

[1] No inglês, o termo utilizado é *layers*

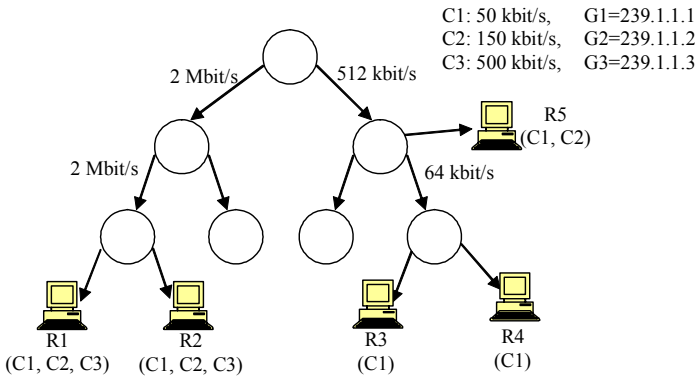


FIGURA 2.3 – Transmissão multimídia sobre multicast usando camadas

Uma outra forma de visualizar a mesma transmissão é ilustrada na figura 2.4, onde se enfatiza a questão da banda e do tipo das camadas. Pode-se ver que, quanto mais camadas são recebidas por um determinado receptor, mais largura de banda é exigida, e conseqüentemente se obtém mais qualidade. Na figura, os receptores R1 e R2 recebem a mesma qualidade de vídeo, porém, o receptor R1 vai assistir ao vídeo com som original, enquanto R2 vai receber o som dublado em português.

Cada camada é um grupo multicast com determinada função, e o receptor escolhe as camadas de acordo com suas necessidades. Por exemplo, o receptor R4 escolheu somente áudio, como mostra a figura 2.4. Entretanto, no caso das camadas cumulativas, existe uma ordem de precedência, ou seja, um receptor não pode se cadastrar na camada C2 sem estar cadastrado na camada C1. No caso da utilização de legenda, se utiliza transmissão em unicast para cada receptor, por se tratar de texto puro, com poucos kbytes a transmitir. Assim, caso um receptor queira receber uma determinada legenda, ele busca o texto completo da legenda no servidor, sincronizando apropriadamente conforme recebe o vídeo em multicast. Essa metodologia diminui o tráfego e a complexidade necessária para manter um grupo multicast adicional. Outro exemplo de camada opcional é a transmissão de cores (que não faz sentido se o receptor possui um monitor monocromático).

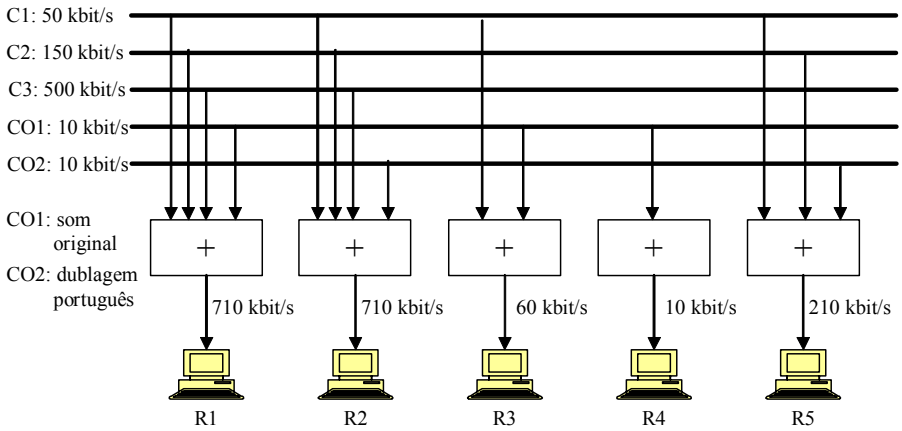


FIGURA 2.4 – Largura de banda para os receptores na transmissão em camadas

A figura 2.5 [HER 2003] mostra um exemplo de resultado possível com a utilização da transmissão em camadas, através do vídeo "*apocalypse now*". Observa-se

que, quanto mais camadas são recebidas, maior a qualidade.

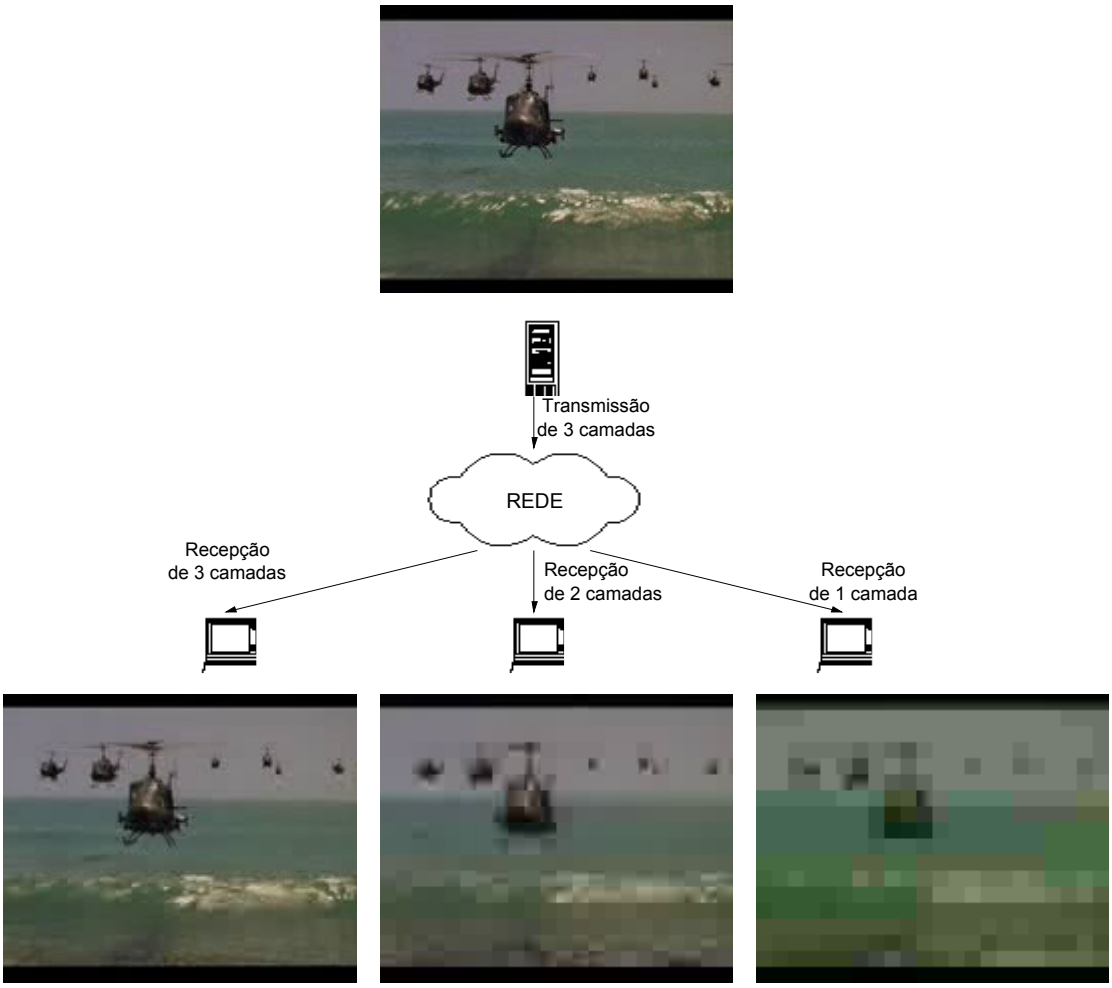


FIGURA 2.5 – Exemplo possível de transmissão e recepção de vídeo em camadas

Um dos focos do sistema proposto nesta Tese visa adaptação automática no receptor ao número de camadas que ele pode se inscrever, visto que existem diversos fatores limitantes, como o congestionamento na rede, a velocidade máxima do enlace ao qual o receptor está conectado, o tipo de processador que o usuário está utilizando, e assim por diante.

A transmissão em camadas não necessita obrigatoriamente ser transmitida em multicast. Na verdade, o uso de multicast ou unicast depende do tipo de aplicação desejado, como detalhado no item 2.2 (Transmissão multimídia multicast e unicast), e existe uma classe de aplicações onde o multicast é mais adequado, e outra onde o unicast é mais apropriado.

2.3.2 Transmissão multicast com codificação em camadas não cumulativas

Na transmissão multicast com codificação em camadas não cumulativas, o vídeo é codificado em várias camadas independentes entre si, e o receptor não precisa necessariamente se inscrever na camada básica para receber o vídeo. Um exemplo desse tipo de codificação é a divisão de um vídeo em quadros pares e ímpares, sendo que cada

um deles é transmitido numa camada separada, conforme ilustra a figura 2.6.

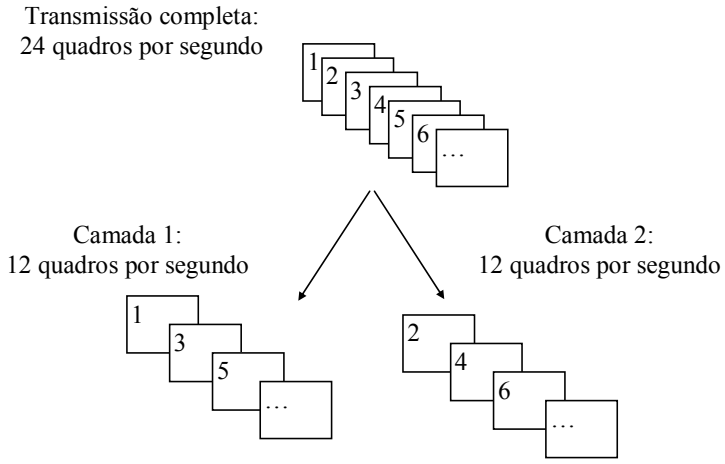


FIGURA 2.6 – Transmissão multicast em camadas não cumulativas

2.3.3 Transmissão multicast usando codificação com replicação

A codificação do sinal com replicação é feita no lado do transmissor, e o mesmo deve codificar o sinal a ser transmitido em diferentes taxas de compressão, enviando cada uma em um grupo multicast separado.

A diferença em relação ao uso de camadas é que cada grupo multicast contém a informação completa a ser decodificada, não necessitando CODECs que trabalhem com camadas. Como pontos positivos, pode-se dizer que é possível a utilização de CODECs largamente difundidos atualmente, e o receptor deve se cadastrar em apenas um grupo multicast, reduzindo a complexidade e mensagens de controle multicast. Além disso, a decodificação exige menor poder de processamento por parte do receptor, e não consome o *overhead* de 20% quando o receptor está cadastrado em todas as camadas, conforme visto no item 2.3.1.

Entretanto, esse algoritmo provoca um aumento na largura de banda em relação ao uso de camadas, pois, como cada taxa é independente, possui uma codificação completa. Uma fórmula matemática para mostrar a largura de banda total a ser transmitida pode ser vista na equação a seguir, onde n é o número de taxas de compressão utilizadas, Bt é a banda total, e Bi é a banda de cada taxa de compressão utilizada.

$$Bt = \sum_{i=0}^{i=n} Bi$$

Por exemplo, se no método de camadas fosse utilizado 64 kbit/s, 256 kbit/s e 1 Mbit/s, seria utilizada na rede uma banda de 1.320 kbit/s. No método de taxas replicadas seria necessário 64 kbit/s, 320 kbit/s (64+256) e 1.320 kbit/s (64+256+1000), totalizando uma banda de 1.704 kbit/s, ou seja, quase 30% superior para obter a mesma qualidade, conforme ilustra a figura 2.7.

Em [KIM 2001] é feita uma comparação mais detalhada em relação à transmissão em camadas e com replicação. Kim argumenta que é melhor a utilização de transmissão multi-taxa com replicação, principalmente devido à simplicidade obtida.

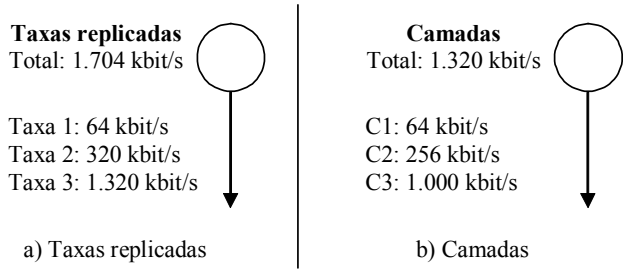


FIGURA 2.7 – Comparação de uso de banda com taxas replicadas e em camadas

2.3.4 Transmissão multicast usando transcodificadores

Outro método de transmissão multi-taxa é conhecido como "transcodificação", e consiste em uma máquina que converte um fluxo de vídeo num outro fluxo com uma taxa ou formato diferente. Um protocolo que utiliza tal sistema será analisado no item 3.4 (Transcodificadores Intermediários). Segundo Li [LI 2003], uma transcodificação rápida pode ser obtida através da manipulação dos dados diretamente na forma comprimida, como por exemplo filtragem na freqüência ou ajuste na escala de quantização. Isso seria possível, já que os padrões atuais de codificação de vídeo, como o MPEG e as famílias H.261/H.263, são todos baseados em DCT (*Discrete Cosine Transform*). Além disso, o padrão MPEG-7 definiu um conjunto de meta-dados a serem inseridos no fluxo de vídeo denominados *transcoding hints*, que efetivamente ajudam no processo de transcodificação, adaptando a velocidade buscando preservar ao máximo a qualidade do sinal.

2.4 Estabilidade de tráfego

Uma propriedade importante no contexto deste trabalho é a estabilidade da transmissão. Uma boa estabilidade pode ser definida como a habilidade do receptor manter a transmissão sem muitas variações na qualidade da imagem ou mudanças de resolução, pois isso pode perturbar o usuário. Entretanto, é possível que um determinado tráfego esteja adequado num determinado momento e no outro esteja acima ou abaixo da sua fatia eqüitativa de banda, devido a mudanças nas condições da rede. Isso acarreta um comportamento oscilatório e instável, o que é negativo para transmissões multimídia, que são o foco deste trabalho.

Nos capítulos 5 (*ALMP: ALM for Private Networks*) e 6 (*ALMTF: ALM TCP-Friendly*) são descritos os métodos utilizados para minimizar a característica oscilatória do sistema, mesmo na presença de tráfego concorrente, buscando manter a divisão eqüitativa da banda. Alguns algoritmos apresentados no capítulo 3 (Trabalhos Relacionados) mostram alternativas propostas por outros autores tentando resolver o mesmo problema.

Neste trabalho, é considerado um distúrbio na estabilidade quando acontecem variações na qualidade percebida pelo usuário devido à *joins* ou *leaves* efetuados pelo algoritmo no receptor. Essa variação pode acontecer devido a mudanças no tráfego (como o início ou o término de sessões concorrentes) ou à características do algoritmo.

A aplicação também pode minimizar até certo ponto a instabilidade recebida.

Por exemplo, existe um tipo de instabilidade ocasionada por tentativas frustradas de *join*, ou seja, o receptor acha que tem condições de aumentar o seu tráfego e se inscreve em um novo grupo multicast, porém, esse *join* provoca congestionamento e perda de pacotes, fazendo com que o receptor efetue *leave* rapidamente do grupo. Para evitar os efeitos desse tipo de situação, o decodificador no receptor deve esperar um período de estabilização antes de modificar a qualidade mostrada ao usuário. Essa precaução pode impedir mudanças temporárias e potencialmente irritantes na qualidade percebida pelo usuário no caso de um *join* sem sucesso.

A menos que se utilize um mecanismo de QoS, como o descarte por prioridades, descrito em [ROE 2003h], as perdas devido à tentativa de *join* descrita acima ocorrem em todas as camadas. Isso pode gerar perda na qualidade da imagem, caso o decodificador não tenha informações redundantes suficientes para recuperar o sinal perdido.

2.5 Eqüidade de tráfego

Todo novo algoritmo gerador de tráfego, quando utilizado em redes de computadores compartilhadas por outros tráfegos, deve seguir certas regras a fim de não prejudicar as aplicações existentes. Assim, esse novo tráfego deve dividir sua banda de forma eqüitativa[1] entre sessões similares (de mesmo algoritmo) e também em relação a sessões estabelecidas com o TCP.

É importante entender o funcionamento do TCP, pois ele é o protocolo "modelo" na Internet, ou seja, todos os outros algoritmos devem adaptar-se a ele de forma cooperativa. Isso é necessário, pois o TCP responde por 90% do tráfego da Internet [HUS 2000b], e a criação de protocolos que não sigam suas características provocaria um impacto muito grande na rede. Thompson [THO 97] efetuou uma pesquisa detalhada num enlace OC-3 da MCI, e ressalta que o TCP é responsável por 95% do tráfego de bytes, 90% do tráfego de pacotes e representa, no mínimo, 75% dos fluxos existentes neste enlace.

Sendo assim, o comportamento do algoritmo proposto nesta Tese deve ser *TCP-friendly*, e determinado fluxo é considerado *TCP-friendly* se "a quantidade de dados transferida em longo prazo não exceder a quantidade de dados transferida por um fluxo TCP sob as mesmas circunstâncias" [FLO 99]. A mesma definição é descrita na RFC 2309 [BRA 98], introduzindo o termo de fluxo "compatível com TCP", que é equivalente a "*TCP-friendly*".

Entretanto, segundo [GEV 2001], essa definição é fraca, pois existem várias variantes do TCP com características de desempenho variadas. Além disso, a mesma taxa de perdas pode afetar o desempenho de forma diferente, dependendo do momento em que a perda ocorre.

Uma definição largamente aceita pela comunidade científica é conhecida como "*max-min fairness*", e significa que "a quantidade de dados transferida por usuário é no mínimo tão grande quanto a de todos os outros usuários que compartilham o mesmo

[1] A palavra *eqüitativa* será utilizada ao longo deste trabalho para representar um tráfego cuja divisão de banda é igual entre sessões com o mesmo algoritmo (*fair*) e também com tráfego TCP (*TCP-friendly*).

gargalo[1]" [GEV 2001].

Outra variante sugerida por Widmer [WID 2001] é: "um fluxo unicast é considerado *TCP-friendly* quando ele não reduz o desempenho em longo prazo de uma conexão TCP mais do que outro fluxo TCP reduziria sob as mesmas condições de rede". Essa definição permite a existência de fluxos com taxa mais alta que o TCP, desde que não causem maior impacto que um fluxo TCP causaria.

No caso do multicast, existe uma indefinição sobre o que seria mais correto considerar *TCP-friendly*, já que no multicast um mesmo pacote é aproveitado por vários receptores. Por um lado, existe a definição que considera o fluxo multicast como um fluxo normal, que deve se comportar como se fosse um fluxo *TCP-friendly* unicast. Por outro lado, existe o argumento que um fluxo multicast envia tantos dados quanto *n* fluxos unicast, sendo *n* o número de receptores compartilhando o mesmo enlace, portanto, o fluxo multicast deveria utilizar uma quantidade de banda maior do que um único fluxo TCP, e o valor dessa quantidade de banda seria proporcionalmente maior à medida que aumenta o número de receptores simultâneos. Shapiro [SHA 2002] propõe um método de controle de congestionamento para fluxos multicast promovendo incentivos para o uso de multicast, mas buscando não ser excessivamente agressivo a fim de evitar prejudicar demasiadamente os fluxos unicast concorrentes.

O algoritmo ALM, proposto nesta Tese, busca eqüidade com tráfego TCP e utiliza transmissão multicast, e apesar da segunda teoria vista no parágrafo anterior (fluxos multicast devem consumir mais banda que unicast) possuir um ponto de vista válido, a definição utilizada neste trabalho utiliza a primeira teoria, ou seja, que um fluxo multicast deve se comportar como um fluxo unicast tradicional.

Uma dificuldade para o uso disseminado de multicast atualmente é o desenvolvimento de algoritmos de controle de congestionamento adequados, com mecanismos de redução e aumento de banda análogos ao TCP, justamente buscando eqüidade de tráfego na Internet [BYE 2000].

Entretanto, existem certas dificuldades para obter essa eqüidade (tanto para unicast como para multicast), como o fato de que o roteador deve transferir pacotes e não bits individualmente. Um modelo ideal é conhecido como GPS (*Generalized Processor Sharing*) [HUS 2000]. Esse modelo implementa uma alocação de recursos "*max-min fairness*" usando uma quota de serviço infinitamente pequena. Esse servidor ideal não transmite pacotes, mas é baseado num modelo onde o servidor visita seqüencialmente todas as filas lógicas que estão acumuladas, enviando parcelas infinitamente pequenas de cada uma delas. Este é um algoritmo ideal, não implementável, pois desconsidera as distorções provocadas pelo empacotamento dos dados. Um protocolo que se aproxima deste modelo é o WF2Q (*Worst Case Fair Weighted Fair Queuing*), detalhado em [ROE 2003h].

A propriedade da eqüidade de banda não é satisfeita em redes de computadores que utilizam QoS. Nesse caso, alguns fluxos determinados possuem prioridade em relação aos outros, fazendo com que sejam beneficiados na transmissão.

---

[1] Gargalo é definido aqui como um enlace onde existe limitação na banda devido à congestionamento ou à própria topologia da rede. No inglês o termo equivalente é *bottleneck*.

De fato, existem algumas variantes do conceito de "*max-min fairness*" baseados em política de acesso. Por exemplo, o "*weighted max-min fairness*" [GEV 2001] permite um percentual de uso da banda (peso) para cada fluxo, e o máximo que um usuário pode utilizar é o seu percentual da quantidade de banda disponível no momento. Outra variante é conhecida como "*proportional fairness*", onde se argumenta, através de uma teoria de Nash, que os fluxos com menor RTT (*Round Trip Time*) devem ser favorecidos, pois possuem menos perdas e vão provocar um benefício global ao sistema [GEV 2001].

Para fins de validação do algoritmo proposto nesta Tese, os testes têm por objetivo obter uma eqüidade conforme definida no início deste item, ou seja, em relação ao protocolo TCP. Um detalhamento maior do funcionamento do TCP pode ser visto no item 2.6 (Funcionamento do TCP), onde são explicados com detalhes os algoritmos que regem o funcionamento deste protocolo. No relatório técnico visto em [ROE 2003e], o funcionamento do TCP também é analisado, bem como uma análise de desempenho desse protocolo em aplicações de transferência maciça de dados, com simulações de várias topologias para um melhor entendimento.

Na Internet, os fluxos podem ser divididos em três classes, conforme RFC 2309 [BRA 98]: a) fluxos compatíveis com TCP, que alteram sua taxa de transmissão de forma similar ao protocolo TCP, provendo eqüidade de tráfego com o TCP, como por exemplo o protocolo ALMTF, descrito nesta Tese, no capítulo 6; b) fluxos não responsivos, ou seja, que não alteram sua característica de transmissão na presença de congestionamento, como por exemplo um sinal de voz codificado a 64 kbit/s através do codificador G.711; c) fluxos responsivos, mas não compatíveis com TCP, como o protocolo RLM, descrito no item 3.1 [MCC 96]. As duas últimas classes são as mais prejudiciais para o desempenho da Internet, pois a eqüidade de tráfego só pode ser conseguida se todas as sessões cooperarem. Se uma sessão ignora os sinais de congestionamento e mantém sua taxa de transmissão, pode utilizar uma fatia desproporcional da banda existente, prejudicando as outras sessões concorrentes [BEN 96].

Uma dificuldade para se obter eqüidade de tráfego nas transmissões em camadas é em relação à granularidade da transmissão [LI 2003]. Se as diferentes camadas possuem uma diferença de taxa muito grande (granularidade grossa), fica difícil de situar o receptor de forma que ele receba o mesmo tráfego do TCP. Por exemplo, supondo um sistema com duas camadas de 200 kbit/s e banda eqüitativa de 300 kbit/s: se o receptor se cadastrar somente na primeira camada, vai receber menos do que a banda eqüitativa, se ele se cadastrar também na segunda camada, vai receber mais do que a banda eqüitativa. A utilização de camadas próximas também não é a solução, pois no caso do multicast vai gerar muito tráfego de controle na rede.

O algoritmo proposto nesta Tese utiliza UDP no nível de transporte, porém, existe controle de fluxo para manter a eqüidade com tráfego TCP. O controle de fluxo é feito no nível de aplicação através de adaptação no lado do receptor. Para isso, é necessário um mecanismo a fim de descobrir a banda eqüitativa que o receptor pode utilizar. Nos capítulos 5 (ALMP: ALM for Private Networks) e 6 (ALMTF: ALM TCP-Friendly) será descrito com detalhes o método utilizado para conseguir eqüidade de banda com fluxos compartilhados.

2.6 Funcionamento do TCP

Os dois principais protocolos de transporte utilizados na Internet são o TCP (*Transmission Control Protocol*), definido inicialmente na RFC 793 [POS 81c] e o UDP (*User Datagram Protocol*), definido na RFC 768 [POS 80]. Enquanto o UDP é usado para aplicações de tempo real ou de consulta/resposta simples, sem garantias de entrega do pacote ou seqüência da transmissão, o protocolo TCP permite uma série de garantias à aplicação, como controle de fluxo, garantia de entrega, seqüência dos pacotes e eqüidade de tráfego.

Mecanismos de controle de congestionamento foram introduzidos no TCP em 1988 [JAC 88], levando à implementação do TCP Tahoe, que tinha duas fases: *slow-start* e *congestion avoidance*. Posteriormente, em 1990, foram introduzidas as fases de *fast retransmission* e *fast recovery*, levando à versão do TCP Reno, utilizada largamente até hoje com pequenas modificações. Detalhes sobre o funcionamento básico do TCP podem ser encontrados em [ROE 2003e], na RFC 2581 [ALL 99] ou em [HUS 2000b]. Este item analisa os mecanismos essenciais do TCP Reno necessários ao bom entendimento dos algoritmos descritos nesta Tese.

Em linhas gerais, o TCP é composto de duas fases principais: *slow-start* e *congestion avoidance*. A detecção de perdas de pacotes pode ser por *timeout* ou recebimento de ACKs duplicados, sendo que neste último caso são utilizados os algoritmos de *fast retransmit* e *fast recovery*. Além disso, possui duas variáveis principais, que são o valor da janela de congestionamento (*cwnd*) e o valor que marca o início de provável congestionamento (*ssthresh*). O funcionamento dessas fases e variáveis está descrito a seguir:

1. Quando uma nova conexão é estabelecida, o valor inicial da janela de congestionamento *cwnd* (*Congestion Window*) é, no máximo, igual a 2*SMSS (*Sender Maximum Segment Size*) (RFC 2581 [ALL 99]). O valor de SMSS é o menor entre o RMSS (*Receiver Maximum Segment Size*) do receptor (obtido no *handshake*) e o MTU (*Maximum Transmission Unit*) da rede. Caso não se tenha usado a opção para descobrir o MTU (RFC 1191 [MOG 90]), usa-se o MTU da interface de saída ou, na ausência de outra informação, 536 bytes [HUS 2000b]. A variável *cwnd* é responsável pelo controle de fluxo do lado do transmissor, enquanto o RMSS é o controle de fluxo imposto pelo receptor. A primeira está relacionada com o congestionamento detectado na rede, enquanto a última está relacionada com a quantidade de espaço em memória disponível pelo receptor (RFC 2001 [STE 97]). A partir deste momento será assumido que *cwnd* foi configurado inicialmente para um SMSS.
2. O transmissor entra então no modo de controle de fluxo chamado *slow-start*, enviando somente um pacote, pois é o que permite a janela de congestionamento *cwnd*. Após o envio, o transmissor espera o ACK desse pacote antes de continuar.
3. Ao receber o ACK, o transmissor incrementa *cwnd* por, no máximo, um SMSS, fazendo *cwnd=cwnd+SMSS*, aumentando para dois o número de pacotes que pode transmitir. Quando transmite esses dois pacotes, a janela de congestionamento está cheia novamente, e o transmissor deve esperar algum ACK para poder continuar. A cada ACK recebido, o transmissor aumenta de, no máximo, um SMSS o tamanho da janela de congestionamento. Isso confere um caráter exponencial à taxa de subida da

velocidade de transmissão, como mostra a FIGURA 2.8, praticamente dobrando a taxa de transmissão a cada RTT. Pode haver diferença na taxa de subida caso o receptor utilize a técnica de *delayed* ACK (RFC 813 [CLA 82] e RFC 1122 [BRA 89]), onde envia um ACK a cada dois segmentos recebidos (ou determinado tempo). Segundo definição na RFC 2581 [ALL 99], o atraso de um ACK nunca pode ser maior do que 500ms.

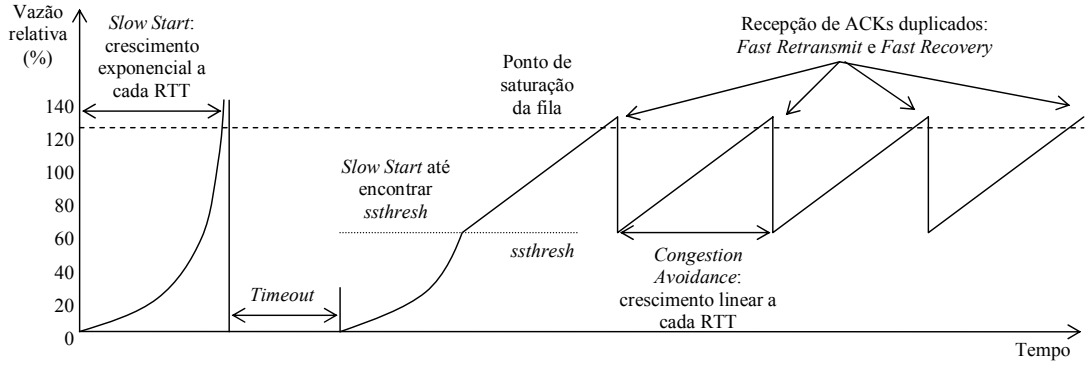


FIGURA 2.8 – *Slow-start* e *congestion avoidance* no TCP

4. Esse processo termina quando: a) o valor de *cwnd* atingir o limite superior da janela de transmissão; b) o limite da rede for atingido, que é detectado por perda de pacotes ou *timeout*; c) a janela *cwnd* exceder a variável *ssthresh* (*slow-start Threshold*), cujo objetivo é marcar a posição onde ocorreu o congestionamento. A fase de *slow-start* acontece quando *cwnd* < *ssthresh*, e a fase de *congestion avoidance* é utilizada quando *cwnd* > *ssthresh*.
5. Quando ocorre *timeout*, identificado pela falta de recebimento de ACK por um determinado tempo (causado pela perda de vários pacotes, ou pela perda de um pacote no fim da seqüência, onde não existem subseqüentes ACKS), o algoritmo do TCP assume que a rede está num estado de congestionamento pesado, portanto, reduz bastante sua taxa de transmissão, fazendo *cwnd*=1*SMSS. O algoritmo marca a suposta posição do congestionamento através da variável *ssthresh*, que é configurada para *ssthresh=max (flightsize/2*, *2*SMSS*) (RFC 2581 [ALL 99]), onde *flightsize* representa a quantidade de dados que estão circulando na rede sem terem recebido ACK. Como *cwnd* < *ssthresh*, o algoritmo entra novamente na fase de *slow-start* e retransmite os dados a partir do ponto que recebeu ACK válido do receptor. A diferença desse momento em relação ao início da sessão é que a variável *ssthresh* tem a memória do ponto de congestionamento, fazendo o transmissor mudar para a fase de *congestion avoidance* a partir desse ponto crítico, reduzindo sua taxa de transmissão, conforme mostra a figura 2.8.
6. A taxa de transmissão de dados é reduzida, e quando *cwnd* passar de *ssthresh*, o algoritmo de controle de fluxo do TCP muda de *slow-start* para *congestion avoidance*. Na fase de *congestion avoidance*, a janela de transmissão é incrementada da seguinte forma: *cwnd* = *cwnd* + SMSS*SMSS/*cwnd*, onde *cwnd* é mantido em bytes (RFC 2581 [ALL 99]). Percebe-se, através da fórmula, que a janela de transmissão aumenta em um SMSS cada vez que chegarem todos os ACKs dos pacotes enviados na janela anterior. Supondo que a transmissão de todos os pacotes é menor do que um RTT, pode-se dizer que a janela de congestionamento aumenta

em um SMSS a cada RTT. Isso provoca um crescimento linear na taxa de transmissão, conforme mostra a figura 2.8. No caso do receptor utilizar a técnica de *delayed* ACK, a janela de congestionamento aumenta em um SMSS cada dois RTTs [PAD 98].

7. Os ACKs são gerados pelo receptor e informam o último número de seqüência válido. Caso um pacote tenha sido perdido na rede, quando o pacote subseqüente chegar ao receptor, este vai enviar um ACK duplicado, pois o último número de seqüência válido é do pacote anterior ao que foi perdido. Esse ACK duplicado vai ser utilizado pelo transmissor a fim de detectar congestionamento leve na rede. Outros fatores que podem causar ACKs duplicados são quando pacotes chegam ao receptor em ordem inversa do que foram transmitidos ou quando ocorre uma replicação de ACK na rede (RFC 2581 [ALL 99]).
8. No caso de congestionamento leve na rede, definido como três ACKs duplicados, o TCP deve retransmitir somente os pacotes faltantes, e a taxa de transmissão deve ser ajustada para reduzir a probabilidade de novas perdas. A recepção de 3 ACKs duplicados (ou quatro ACKS com o mesmo número de seqüência) é assumido pelo TCP como um sinal de perda. Nesse caso, o transmissor envia somente o pacote faltante (algoritmo *fast retransmit*), e continua no modo *congestion avoidance*, entretanto, com uma redução na banda caracterizando o *fast recovery*, pois o algoritmo não precisa baixar a taxa de transmissão para o *slow-start* (RFC 2581 [ALL 99]).
9. A redução na variável *ssthresh* é a mesma que o caso do *timeout* visto acima, ou seja, *ssthresh=max (flightsize/2*, *2*SMSS*), entretanto, a variável *cwnd* é configurada para *cwnd = ssthresh + 3*SMSS*, pois como o receptor enviou 3 ACKs duplicados, ele já possui 3 segmentos na sua memória. Pela mesma razão, caso cheguem mais ACKs duplicados, incrementa *cwnd = cwnd + SMSS* para cada ACK duplicado recebido. Quando chegar o primeiro ACK não duplicado, ou seja, que indica o recebimento do pacote faltante, configura *cwnd = ssthresh* e continua a transmissão normalmente.

A taxa linear de subida dos fluxos TCP, aliado à sua descida brusca, confere um caráter conhecido como AIMD (*Additive Increase Multiplicative Decrease*), e esse é um dos principais aspectos que levam o TCP a funcionar eqüitativamente com outros fluxos, pois a sensibilidade do protocolo é maior quanto maior sua taxa, fazendo com que, quanto maior a taxa, maior a redução de banda. Dessa forma, a tendência é que as várias sessões atinjam o equilíbrio. Entretanto, da mesma forma que a utilização da filosofia AIMD permite a eqüidade de tráfego, ela gera um comportamento oscilatório ao tráfego sendo gerado, o que não é bom para aplicações inelásticas.

O algoritmo do TCP considera que perda é sinônimo de congestionamento na rede, porém, segundo Sanadidi [SAN 2002], em enlaces *wireless* nem sempre isso é verdade, pois seus níveis de perdas são bem mais elevados que enlaces terrestres. Isso está levando ao desenvolvimento de uma nova geração de protocolos TCP, como o TCP Peach [AKY 2001] e o TCP Westwood [GER 2001]. Para o objetivo desta Tese, será considerado o TCP tradicional conectado através de cabos, onde perda é sinônimo de congestionamento. Caso o algoritmo venha a ser utilizado em enlaces *wireless*, o modelo deverá ser validado nesse tipo de enlace. A expectativa é que seja necessário um novo método para informar congestionamento, como por exemplo o método descrito

por Shu-wen Teng [TEN 2003], que utiliza como base as informações de nível 2, assumindo que os erros nos pacotes que chegaram com CRC trocado foram causados pelo enlace *wireless*, e não devido a congestionamento.

Como o comportamento do TCP é conhecido, vários autores calcularam sua vazão analiticamente. Padhye [PAD 98] e [PAD 2000], calculou um modelo completo e uma aproximação levando em conta o tamanho da janela, a ocorrência de *timeouts* e ACKs duplicados. A equação aproximada é a seguinte:

$$B(p) \approx \min\left(\frac{W\max}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + To\min\left(1{,}3\sqrt{\frac{3bp}{8}}\right)p\left(1+32p^2\right)}\right) \qquad \text{(TCP\_1)}$$

Na equação, *B(p)* é a vazão equivalente a uma conexão TCP em função das perdas, *Wmax* é o valor máximo da janela de congestionamento, definida na negociação inicial de abertura de conexão (*three way handshake*). RTT (*Round Trip Time*) é o tempo de ida do pacote e volta do respectivo ACK, *b* corresponde ao número de pacotes que o receptor espera para enviar ACK (com *delayed* ACK, *b*=2). A variável *p* corresponde à taxa de perdas ocorridas na transmissão (total de pacotes perdidos dividido pelo total de pacotes transmitidos), enquanto *To min* equivale ao valor mínimo do *timeout* da sessão TCP (esse valor vai aumentando à medida que *timeouts* seqüenciais acontecem).

Outra equação, mais simplificada, foi definida em [MAH 97] e [FLO 99] para uma taxa de perdas menor que 16% (5% segundo [PAD 99]), não levando em consideração a diminuição de banda causada por *timeouts*.

$$Beq = C * \frac{MTU}{RTT * \sqrt{Loss}} \qquad \text{(TCP\_2)}$$

Na equação, *Beq* é a Banda equivalente a uma conexão TCP, MTU (*Maximum Transfer Unit*) é o tamanho máximo de pacote utilizado na conexão, RTT (*Round Trip Time*) é o tempo médio de ida e volta do pacote entre origem e destino, e *Loss* é a taxa média de perdas. A constante C é encontrada nos valores 1,22 e 1,31. Segundo [PAD 99], o valor de 1,31 é utilizado quando o receptor emprega a técnica de *delayed ACK*, e 1,22 é utilizado caso contrário.

2.7 Congestionamento em redes de computadores

Segundo Gevros [GEV 2001], congestionamento é um estado de sobrecarga prolongada na rede, onde a demanda por recursos da rede é próxima ou excede sua capacidade. Os principais recursos da rede são a largura de banda nos enlaces e o tamanho da fila dos roteadores.

Um roteador é um recurso compartilhado em uma rede, e cada pacote pode ser considerado uma solicitação de uso deste recurso. Algumas vezes os pacotes chegam nas interfaces de entrada de forma ordenada e numa taxa que permite ao roteador enviá-los imediatamente. Outras vezes, entretanto, ocorre contenção, pois múltiplas

solicitações podem chegar ao mesmo tempo através de interfaces diferentes, precisando ser redirecionados para a mesma interface de saída, ou o tempo despendido para atender algumas solicitações pode se sobrepor com novas solicitações. Nesse caso, a fila é utilizada temporariamente para os pacotes esperando sua vez de serem transmitidos.

Caso a taxa de entrada de pacotes seja superior à taxa de saída, haverá um aumento na utilização da fila, e se a situação persistir por determinado tempo, o roteador vai agir de acordo com sua política de controle de congestionamento, conforme será detalhado no item 2.8 (Esquemas de controle de congestionamento).

Para resolver os aspectos de contenção de recursos, a disciplina de filas do roteador possui dois componentes principais [HUS 2000]:

- Um componente de classificação dos pacotes na fila, controlando a ordem na qual os pacotes são processados. Esse componente será visto com detalhes no item 2.10 (Políticas de gerência de filas nos roteadores);
- Um componente de descarte, controlando o momento e a seleção do descarte de pacotes, conforme será descrito no item 2.11 (Políticas de descarte de pacotes nas filas dos roteadores).

Segundo Jain [JAI 90], existem certos mitos sobre a problemática do congestionamento nas redes de computadores, que são:

1. Congestionamento é causado por uma diminuição de espaço em memória, logo, aumentando a quantidade de memória (e tamanho das filas) vai resolver o problema do congestionamento;
2. Congestionamento é causado por enlaces lentos, logo, quando os enlaces forem de alta velocidade, o problema estará resolvido;
3. Congestionamento é causado por processadores lentos, e será resolvido quando a velocidade dos processadores for aumentada.

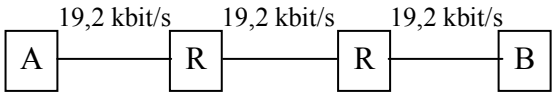
O problema do congestionamento não é resolvido com o aumento no tamanho da fila, pois isso acarretaria um aumento no atraso fim-a-fim do pacote, podendo até provocar *timeout* na sessão em casos extremos. Nesse caso, o pacote seria retransmitido pelos algoritmos de controle de fluxo dos protocolos de transporte. Na realidade, uma fila muito grande é mais prejudicial do que uma muito pequena, pois os pacotes seriam descartados somente após haverem consumido muitos recursos da rede, e seu atraso na entrega provocaria uma demora de detecção de congestionamento segundo os métodos atuais, baseados em perda de pacotes [GEV 2001].

A diminuição da capacidade da fila também não é solução. Com uma fila muito pequena, muitos pacotes seriam descartados, resultando numa alta taxa de pacotes retransmitidos. Segundo Croll [CRO 00], uma possibilidade para cálculo de tamanho máximo da fila, apropriada somente para tráfego TCP/IP elástico, seria multiplicar o número máximo de conexões TCP concorrentes pelo valor máximo de *window size* dessas conexões. O resultado seria o tamanho máximo da fila que o roteador precisaria.

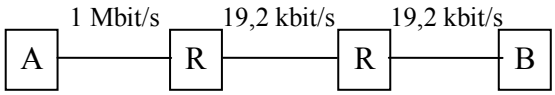
Esse cálculo, apesar de correto, não leva em conta as necessidades de baixo atraso das aplicações de tempo real, e só funciona para tráfego com controle de fluxo similar ao TCP, portanto, não é uma solução de cálculo de tamanho de fila.

O segundo mito é que o congestionamento é resolvido através de enlaces rápidos, porém, segundo Jain [JAI 90], o congestionamento surge da interconexão entre redes rápidas através de enlaces lentos ou congestionados. Para provar seu ponto de vista, Jain efetuou o experimento mostrado na figura 2.9, onde existem três enlaces entre a máquina A, de origem, e a máquina B, de destino. Quando todos os enlaces são de 19.2 kbit/s, mostrado na figura 2.9a, o tempo para transferir um determinado arquivo foi de 5 minutos. Quando o primeiro enlace teve sua velocidade aumentada para 1 Mbit/s, como mostrado na figura 2.9b, o tempo para transferir o mesmo arquivo aumentou para 7 horas. Isso aconteceu pois a taxa de chegada no primeiro roteador se tornou muito mais alta que a taxa de saída, levando a longas filas e perdas de pacotes, que aumentaram o tempo total de transmissão.

Em casos onde a velocidade é alta em todos os enlaces, evidentemente que a transmissão vai funcionar melhor.



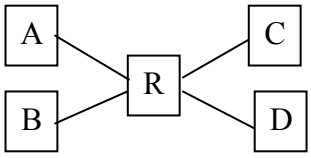
a) Tempo para transferir arquivo: 5 minutos



b) Tempo para transferir arquivo: 7 horas

FIGURA 2.9 – Introduzir um enlace de alta velocidade pode diminuir o desempenho

O terceiro mito descrito por Jain [JAI 90] é que a utilização de processadores de alto desempenho vai resolver o problema do congestionamento, entretanto, este persiste mesmo com todos enlaces de mesma velocidade e alto poder de processamento, como mostra a figura 2.10. Caso o nó A e B estejam transmitindo simultaneamente para o nó C, este teria uma taxa de entrada de 2 Gbit/s com uma taxa de saída de apenas 1 Gbit/s, causando congestionamento.



Todos enlaces de 1 Gbit/s

FIGURA 2.10 – Configuração balanceada e sujeita a congestionamento

É claro que Jain utilizou-se de algumas exceções para exemplificar seu ponto de vista, com o objetivo de mostrar que nenhuma das alternativas é 100% garantida. Vale lembrar que, de forma geral, a rede é beneficiada com enlaces e processadores mais rápidos, como se verifica na Internet atual.

Assim, controle de congestionamento é um problema dinâmico, e não pode ser resolvido somente com soluções estáticas, necessitando de protocolos que consigam reduzir sua taxa de transferência na ocorrência de congestionamento. O problema é complexo, e o objetivo almejado nas redes de computadores é que cada sessão

compartilhando recursos deva efetuar controle de congestionamento de tal forma que: a) a banda seja eqüitativa entre as várias sessões; b) os recursos da rede não sejam sub-utilizados, ou seja, a rede opere numa região próxima ao seu limite; c) os requisitos mínimos de qualidade de serviço, como atraso e taxa de transferência, sejam atendidos para as aplicações; d) o *overhead* do controle seja baixo, de preferência não incrementando o tráfego em períodos de congestionamento.

Para tanto, existem diversos esquemas de controle de congestionamento, tanto na camada de rede (política de filas, política de descarte de pacotes, política de roteamento, controle de tempo de vida do pacote), como na camada de transporte (estimativa de RTT, *timeout*, política de retransmissão, política de *acknowledge*, política de controle de fluxo), e na camada de enlace [JAI 90].

O principal método empregado na Internet é o controle de fluxo fim-a-fim baseado em janela, utilizado principalmente no TCP (RFC 2581 [ALL 99]). Pode-se afirmar, portanto, que o controle de fluxo baseado em janela é uma das formas de se obter controle de congestionamento na rede. A próxima seção analisa alguns desses mecanismos.

2.8 Esquemas de controle de congestionamento

Existem vários esquemas possíveis para que se obtenha controle de congestionamento e eqüidade de tráfego. Widmer [WID 2001] classifica esses esquemas nas seguintes categorias: a) orientado a janela e orientado a taxa; b) unicast e multicast; c) taxa única e multi-taxa; d) fim-a-fim e apoiado pelo roteador. Além desses, é possível classificar mais um esquema: e) orientado a transmissor ou a receptor.

A seguir cada um desses esquemas será analisado com maiores detalhes, e no capítulo 3 (Trabalhos Relacionados), alguns protocolos de cada categoria serão detalhados.

2.8.1 Baseado em janela e baseado em taxa

Algoritmos de controle de congestionamento baseados em janela possuem um limitador no número de bytes sendo transmitidos simultaneamente na rede, e esse limitador é conhecido como “janela”, pois representa uma fatia da transmissão total.

Cada pacote transmitido consome um número de bytes da janela. Enquanto o número de bytes em circulação for menor que o tamanho da janela, o transmissor pode enviar mais dados. Conforme o transmissor for recebendo ACKs, ele vai avançando a janela, abrindo espaço para a transmissão de novos bytes. Quando o número de bytes transmitidos for igual ao tamanho da janela, o transmissor deve parar de enviar bytes até que receba alguma confirmação por parte do receptor, permitindo a ele avançar a janela e abrir espaço para a transmissão de novos bytes.

Em períodos de congestionamento, o tamanho da janela é reduzido, provocando uma diminuição na taxa de transmissão. Em períodos sem congestionamento, o tamanho da janela é aumentado, aumentando a taxa de transmissão. Isso provoca uma adaptação às condições da rede.

Exemplos de algoritmos de controle de congestionamento baseados em janela, além do TCP, são o TEAR (*TCP Emulation At Receivers*) e o RAP (*Rate Adaptation Protocol*), que serão detalhados no capítulo 3.

Já os algoritmos de controle de congestionamento baseados em taxa buscam uma adaptação dinâmica de sua taxa de transmissão de acordo com o nível de congestionamento na rede. Segundo Widmer [WID 2001], eles podem ser subdivididos em esquemas AIMD simples ou AIMD baseados em modelo.

Os esquemas AIMD simples imitam o comportamento do TCP, criando um comportamento de aumento ou diminuição de taxa parecida com um fluxo TCP equivalente. Isso gera uma onda estilo "dente de serra", similar ao TCP, dificultando sua utilização para aplicações tipo *streaming* de vídeo [WID 2001].

Os esquemas AIMD baseados em modelo adaptam a taxa de transmissão de acordo com o desempenho médio do TCP em determinado período de tempo, produzindo um resultado bem mais estável que o obtido com AIMD simples.

Exemplos de algoritmos de controle de congestionamento baseados em taxa são o TFRCP (*TCP-Friendly Rate Control Protocol*), TFMCC (*TCP-Friendly Multicast Congestion Control*) e *Thin Streams*. Todos serão analisados no capítulo 3.

O modelo utilizado no algoritmo ALMTF, proposto nesta Tese, é baseado em taxa e janela, como pode ser visto no detalhamento do capítulo 6 (ALMTF: ALM TCP-Friendly). Esse modelo foi escolhido pois permite uma maior estabilidade ao sistema mantendo ao mesmo tempo uma eqüidade com o tráfego TCP, e como se trata de transmissão multimídia, esse é um fator importante a considerar.

2.8.2 Unicast e multicast

Segundo Widmer [WID 2001], os mecanismos de controle de congestionamento para multicast são bem mais complexos do que para unicast, e isso se deve aos seguintes fatores:

- Escala: o controle de congestionamento em multicast deve funcionar tanto para um receptor como para centenas de receptores simultâneos;
- Topologia e congestionamento: os receptores podem estar localizados em enlaces rápidos, lentos, congestionados ou livres, criando o problema da taxa de transmissão na qual os pacotes devem ser enviados. As opções são adaptar a taxa ao transmissor mais lento, ao mais rápido, utilizar o valor onde existem mais usuários ou utilizar transmissão em camadas;
- Perda de pacotes: receptores localizados em enlaces diferentes podem experienciar taxa de perda de pacotes diferentes, e o mecanismo de controle de congestionamento do transmissor deve saber como interpretar as perdas de forma correta;

Além da complexidade adicional citada acima, existem dois outros problemas relevantes no controle de congestionamento em multicast: o sincronismo entre receptores com transmissão em camadas e o problema do tempo de *leave*, ambos descritos a seguir.

Quando o esquema de controle de congestionamento utiliza transmissão em

camadas, é necessário haver sincronismo entre receptores, particularmente os situados na mesma sub-rede, pois se um único receptor estiver inscrito numa camada superior, vai gerar tráfego para todos os outros receptores que dividem o mesmo enlace, independente da camada que eles estão inscritos. Alguns podem, inclusive, interpretar as perdas devido ao congestionamento gerado por esse único receptor como um sinal de que estão num nível acima do que podem suportar, saindo da camada superior sem necessidade (através do comando *leave* do IGMP).

Um exemplo é ilustrado na figura 2.11, que mostra uma topologia onde o receptor R1 consegue se inscrever em três camadas (C1, C2 e C3), e os outros receptores conseguem se inscrever apenas nas camadas C1 e C2, entretanto, R4 é novo na sessão e ainda não tentou se inscrever na camada C2. Caso qualquer um deles tente se inscrever numa camada superior, isso vai gerar perdas em todas as camadas. Assim, se R2 tentar se inscrever na camada C3, a necessidade de banda do enlace 2 será de 448 kbit/s, gerando perdas. Normalmente, ao detectar as perdas, R2 sai da camada C3, regularizando a situação. Entretanto, se R3 fizer a mesma tentativa instantes depois, e outros receptores logo após, haverá perdas contínuas no enlace.

Isso mostra a necessidade de um sincronismo entre receptores localizados na mesma sub-rede, que poderiam "aprender" com a tentativa do outro receptor. A situação é mais complexa quando R1 tenta se inscrever na camada C4 ao mesmo tempo em que R4 tenta se inscrever na C2. Como a tentativa de R1 vai gerar perdas, pode dar a impressão a R4 que ele não pode se inscrever na camada C2, o que é falso. No capítulo 3 serão analisados trabalhos que lidam com esse tipo de problema, e nos capítulos 5 e 6 será feita a descrição da forma que o ALM lida com esse problema.

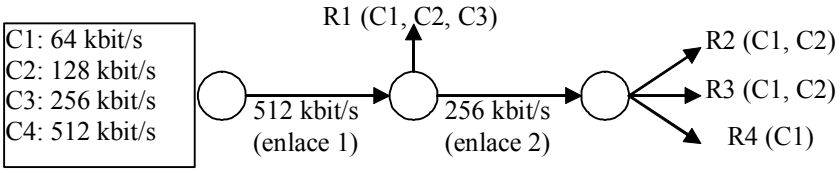


FIGURA 2.11 – Necessidade de sincronização entre receptores da mesma sessão

Contrariamente à necessidade de sincronização entre tentativas de *join* para receptores cadastrados na mesma sessão, existe a necessidade de evitar sincronização para tentativas de *join* de receptores cadastrados em sessões diferentes, conforme ilustrado na figura 2.12, onde existem duas sessões, S1 e S2, e o receptor R_{S1} cadastrado na sessão S1, enquanto R_{S2} está cadastrado na sessão S2. Nesse caso, o valor justo de banda seria 128 kbit/s para cada um dos receptores, entretanto, R_{S1} está recebendo menos banda, e deveria aumentar uma camada, enquanto R_{S2} deveria permanecer como está (quatro camadas de 32 kbit/s). Se os dois receptores fizerem tentativas de *join* sincronizadamente, a banda no enlace central vai ser superior à sua capacidade e vão acontecer perdas, mesmo tendo banda disponível para um dos receptores (de preferência R_{S1}) aumentar o número de camadas. Soluções para esse tipo de problema também serão vistas no capítulo 3 e nos capítulos 5 e 6.

Além disso, no multicast existe o problema do tempo de *leave*. Quando um receptor pretende sair de um grupo multicast, ele envia uma mensagem de IGMP *leave* ao seu roteador local avisando que não está mais interessado no grupo. Ao receber essa

mensagem, o roteador envia uma mensagem de IGMP *specific query* à sub-rede (IGMP versão 2.0), a fim de descobrir se existe algum outro receptor cadastrado nesse grupo. Se não existirem receptores ativos, o roteador envia uma mensagem de *prune* em direção ao transmissor, a fim de parar de receber os pacotes multicast desse grupo. Como a ausência de receptores ativos só pode ser determinada via *timeout*, a fase de *leave* pode demorar um tempo razoável (da ordem de segundos). Enquanto isso, os pacotes multicast continuam chegando e causando perdas no enlace.

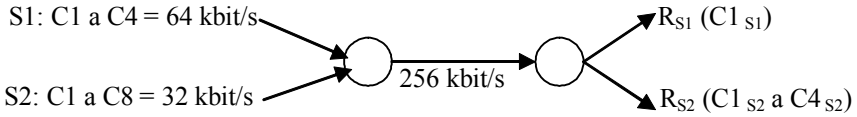


FIGURA 2.12 – Necessidade de evitar sincronização entre sessões diferentes

Linda Wu [WU 97] fez um estudo prático sobre a duração do tempo de *leave*. A conclusão foi que, utilizando a versão 1.0 do IGMP, o tempo de *leave* chega a dois minutos. Por outro lado, com a versão 2.0, o tempo de *leave* ficou em aproximadamente 250 ms.

Detalhes sobre as mensagens IGMP e funcionamento das fases de *join* e *leave* podem ser obtidas em [ROE 2003d] ou RFC 2236 [FEN 97]. Na implementação do algoritmo ALM testada em laboratório, cujos detalhes serão vistos no item 5.3 (Implementação do algoritmo ALMP), o roteador utilizou uma versão do *mrouted* com um tempo de *leave* mais rápido, conforme descrito por Rizzo [RIZ 98].

Neste trabalho, será abordado prioritariamente o controle de congestionamento em multicast, pois um dos objetivos do sistema é a transmissão de vídeo em tempo real, como canais de TV, shows e palestras. Entretanto, o sistema implementado suporta o controle em unicast, utilizado principalmente para permitir flexibilidade ao usuário sobre o momento de assistir à transmissão. Assim, uma determinada palestra pode ser transmitida em tempo real via multicast e ser gravada para ser transmitida posteriormente em unicast, através de um sistema de Vídeo sob Demanda com armazenamento em camadas.

2.8.3 Taxa única e multi-taxa

Para efeitos desta Tese, a transmissão em taxa única ou multi-taxa está relacionada ao transmissor, conforme a seguinte definição:

- **Taxa única**: a transmissão é efetuada para todos os receptores na mesma taxa de transmissão;
- **Multi-taxa**: a transmissão é enviada em mais de uma taxa simultaneamente, seja por divisão em camadas ou transmissão com taxas replicadas, conforme detalhamento no item 2.3.

As transmissões unicast sempre utilizam taxa única, pois se baseiam na comunicação de um transmissor com um receptor, não fazendo sentido efetuar transmissão em mais de uma taxa.

Em transmissões multicast, entretanto, é possível classificar os esquemas de controle de congestionamento em taxa única e multi-taxa, ou seja, se o transmissor

utiliza uma única taxa de transmissão para todos receptores ou se vai utilizar várias taxas simultaneamente, cada uma em um grupo multicast diferente. Nesse caso, pode ser utilizado a transmissão multicast em camadas, detalhada no item 2.3.

Um caso especial de transmissão multicast é a transmissão hierárquica, onde existem equipamentos transcodificadores intermediários adaptando a taxa do transmissor para a taxa dos receptores localizados abaixo dele, e transmitindo num grupo multicast separado, conforme será visto no item 3.4. Esse tipo de transmissão é considerado de multi-taxa, pois existem taxas diferentes na rede como um todo.

No esquema de taxa única, é possível que o transmissor adapte a taxa de transmissão de acordo com o retorno recebido dos receptores, entretanto, a nova taxa será enviada a todos receptores simultaneamente, dificultando a escalabilidade, pois todos receptores obrigatoriamente deverão possuir uma largura de banda suficiente para absorver essa taxa de dados. Um exemplo de transmissão multicast em taxa única será vista no item 3.9 (protocolo TFMCC).

2.8.4 Apoiado pelo roteador e fim-a-fim

Os esquemas de controle de congestionamento também podem ser classificados de acordo com o local onde o controle do congestionamento é efetuado: nos equipamentos centrais (roteadores) ou nas estações finais (controle fim-a-fim).

O controle de congestionamento apoiado pelos equipamentos centrais (roteadores) se fundamenta no fato que é através desses equipamentos que passa todo o tráfego da rede, e não há melhor lugar para garantir a eqüidade entre tráfegos de diferentes sessões. Devido ao grande poder computacional exigido nesses roteadores e à complexidade das implementações, essa classe ainda está em uso experimental em alguns pontos da Internet. Algumas propostas desse tipo de controle podem ser vistas em [FLO 99] e [LEG 2000b].

Uma das sugestões é a utilização de roteadores que façam escalonamento de pacotes por fluxo, regulando individualmente a largura de banda utilizada pelos fluxos. Outro método é dotar o roteador com mecanismos para incentivar o controle de congestionamento, restringindo a largura de banda de fluxos que utilizam uma quantidade de banda muito grande frente aos outros. A forma sugerida por [FLO 99] para conseguir isso é através do monitoramento dos descartes efetuados pelo algoritmo do RED. Nesse caso, o roteador descobre se um fluxo está utilizando mais banda que outros fluxos, pois ele teria descartado mais pacotes deste fluxo que dos outros.

Uma idéia que foi explorada pelo autor desta Tese e está descrita em [ROE 2003j] é cada roteador adicionar aos pacotes de ACK a banda eqüitativa para aquele fluxo. Para isso, os roteadores devem ter um contador do número de fluxos que estão passando pela interface no momento (*nf*). A banda eqüitativa para cada fluxo fica: $Beq = Benlace / nf$. *Benlace* é a banda física do enlace por onde está passando o fluxo em questão. O transmissor recebe, juntamente com o pacote de ACK, a informação da sua menor banda eqüitativa em todos os roteadores no caminho entre origem e destino. Isso pode gerar sobras de banda em determinados roteadores, que são detectadas e utilizadas de acordo, aumentando *Beq* para os outros fluxos.

Outros mecanismos de controle de congestionamento não precisam qualquer tipo

de apoio adicional por parte dos roteadores, podendo ser utilizados na Internet de hoje. Esses mecanismos são conhecidos por possuírem controle de congestionamento nas estações finais, ou fim-a-fim, e utilizam certo apoio da rede para inferirem o nível de congestionamento atual. O apoio da rede pode ser a reserva de recursos[1] (através do uso de controle de admissão e QoS) ou uma realimentação sobre o nível atual de congestionamento na rede[2], permitindo às estações reagirem a essa informação, aumentando ou diminuindo sua taxa de transmissão.

Existem diversos métodos através dos quais a rede fornece realimentação sobre seu nível de congestionamento, indo desde simples descarte de pacotes até mecanismos tipo ECN (*Explicit Congestion Notification*). Alguns desses métodos serão analisados com maiores detalhes na seção 2.9 (Mecanismos de realimentação do nível de congestionamento).

2.8.5 Orientados a transmissor e orientados a receptor

Nos mecanismos de controle de congestionamento orientados a transmissor, este utiliza as informações da rede e ajusta sua taxa ou janela de transmissão de acordo com o nível de congestionamento da rede. A única função dos receptores é enviar uma realimentação conforme os dados chegam. Um exemplo de protocolo que utiliza um mecanismo desse tipo é o próprio TCP.

Nos mecanismos orientados a receptor, é o receptor que descobre a sua taxa eqüitativa de tráfego, podendo informar ao transmissor para que o mesmo se adapte (como, por exemplo, no protocolo TEAR - *TCP Emulation At Receivers*), ou efetuar a adaptação por conta própria. Neste caso, normalmente é necessária a utilização de transmissão multi-taxa, e o receptor se inscreve em tantos grupos multicast quanto a capacidade da rede permitir (como, por exemplo, o protocolo RLM – *Receiver-driven Layered Multicast*).

O algoritmo ALM, proposto nesta Tese, utiliza um controle de congestionamento fim-a-fim e orientado a receptor, pois os receptores não enviam qualquer tipo de confirmação de chegada dos pacotes ao transmissor, e se baseiam somente nas perdas ocorridas para decidir se devem efetuar *join* ou *leave* em determinada camada.

2.9 Mecanismos de realimentação do nível de congestionamento

Existem diversos mecanismos utilizados para que transmissor ou receptor descubram o nível de congestionamento atual na rede, de forma que consigam adaptar adequadamente sua taxa de transmissão. Alguns mecanismos estão implícitos através da própria transmissão, como, por exemplo, o descarte de pacotes pelos roteadores ou o aumento no atraso da chegada dos mesmos (utilizado pelo TCP Vegas e pares de

[1] Esse caso caracteriza o controle de fluxo fim-a-fim de laço aberto, pois a banda está descrita e supostamente garantida pelos equipamentos da rede, não necessitando realimentação. Basta utilizar um conformador de tráfego para manter a transmissão dentro dos valores reservados.

[2] Esse caso caracteriza o controle de fluxo fim-a-fim de laço fechado, pois as estações devem adaptar-se conforme a realimentação informada pela rede.

pacotes). Outros mecanismos informam explicitamente o nível de congestionamento atual da rede, como o ECN e o ICMP *source quench*. O TCP Vegas é detalhado no item 3.12, e o mecanismo de pares de pacotes no Anexo A. Os outros mecanismos serão analisados a seguir.

2.9.1 Descarte de pacotes

Descarte de pacotes é a forma mais comum atualmente para realimentação de nível de congestionamento sendo utilizada pelos roteadores na Internet. Entretanto, nem sempre um pacote descartado indica congestionamento, principalmente quando se trata de transmissão *wireless*, onde maiores taxas de erros são comuns [GEV 2001].

O roteador descarta pacotes quando detecta uma condição de congestionamento, e o momento desse descarte pode variar, dependendo da política de descarte utilizada, como *droptail* e RED (*Random Early Detection*).

As simulações efetuadas para o algoritmo desenvolvido nesta Tese utilizaram política de descarte RED e também *droptail*, a fim de verificar a funcionalidade do sistema com os dois tipos de política. O item 2.11 (Políticas de descarte de pacotes nas filas dos roteadores) analisa com detalhes estas e outras políticas de descarte.

2.9.2 ECN: Explicit Congestion Notification

A proposta de realimentação de congestionamento através do uso dos bits de ECN (*Explicit Congestion Notification*) no cabeçalho IP está descrito por Ramakrishnan na RFC 2481 [RAM 99]. Ele propõe a utilização nos roteadores de uma política de filas do tipo RED com uma modificação: ao invés de descartar determinado pacote, o algoritmo marca o pacote através dos bits de ECN, indicando que esse pacote passou por uma região congestionada. O receptor copia essa informação ao pacote de ACK, avisando ao transmissor desse congestionamento. Quando o pacote de ACK chegar ao transmissor, o mesmo vai reagir ao congestionamento, diminuindo sua taxa de transmissão. A vantagem é que não é necessário haver descarte de pacotes para avisar congestionamento.

Na RFC 2481 são definidos dois bits de ECN: o primeiro foi denominado ECT (*ECN-Capable Transport*), e é ligado pelo transmissor indicando que as estações finais conhecem o protocolo. O segundo bit foi denominado CE (*Congestion Experienced*), e é ligado pelo roteador para indicar congestionamento.

Os bits 6 e 7 do campo TOS (*Type of Service*) do IPv4 (RFC 791 [POS 81a]) são sugeridos para uso no ECN, sendo que o bit 6 é ECT e o bit 7 é CE. No IPv6 (RFC 2460 [DEE 98]), o campo *Traffic Class* é equivalente ao TOS do IPv4, portanto, sugere-se a utilização dos mesmos bits. Os restantes 6 bits do campo são utilizados para Diffserv (RFC 2474 [NIC 98]), tanto no IPv4 como no IPv6.

Maiores detalhes sobre Diffserv e QoS em geral podem ser vistas em [CRO 2000], [BRA 97], [BLA 98] e [NIC 98]. Além disso, foi desenvolvido um relatório técnico sobre esse assunto, em [ROE 2003f].

2.9.3 ICMP Source Quench

O protocolo ICMP (*Internet Control Message Protocol*) é parte integrante do IP, e é utilizado em ocasiões onde os roteadores devem enviar alguma mensagem de erro ou de controle, como, por exemplo, avisando que o IP destino é inacessível, ou o TTL (*Time To Live*) expirou. Além disso, o ICMP define mensagens de *echo*, utilizadas para testar conectividade até um determinado destino (RFC 792 [POS 81b]), [HUI 2000].

Uma das mensagens do ICMP é o *source quench* (definida como tipo 4), indicando que o roteador não tem espaço suficiente na fila para inserir o pacote recebido. Quando o roteador descarta o pacote, ele pode enviar uma mensagem ICMP de *source quench* para o transmissor. Além disso, um receptor também pode enviar mensagens de *source quench* caso os pacotes estejam chegando muito rapidamente para serem processados. O objetivo dessa mensagem é solicitar ao transmissor uma redução na taxa de transmissão.

O transmissor pode diminuir sua taxa de transmissão até parar de receber mensagens de *source quench*, incrementando-a gradualmente até voltar a receber esse tipo de mensagem. Alternativamente, o roteador (ou o receptor) pode enviar essas mensagens quando a sua capacidade está se aproximando do limite ao invés de esperar o esgotamento total dos recursos, evitando descarte de pacotes (RFC 792 [POS 81b]).

Segundo Gevros [GEV 2001], a mensagem de *source quench* raramente é utilizada na Internet, e pode vir inclusive a ser obsoletada. Isso se deve porque seu consumo de banda acontece justamente em momentos de congestionamento, e geralmente é ineficiente. O autor desta Tese considera que essa forma de realimentação poderia ser bastante útil para diminuir o atraso na detecção de congestionamento, e a modificação necessária para isso é a utilização conjunta com a política de descartes do tipo RED.

2.10 Políticas de gerência de filas nos roteadores

O modelo ideal utilizado para estudar a gerência das filas dos roteadores possui um servidor e uma seqüência de solicitações de serviço (pacotes chegando através das interfaces de entrada) [HUS 2000].

O servidor não possui qualquer controle sobre a taxa de chegada de solicitações, da mesma forma que o tempo despendido para atender cada solicitação pode variar, dependendo, por exemplo, da capacidade de processamento e da velocidade das interfaces de saída. Assim, enquanto o servidor estiver atendendo uma solicitação, outras solicitações podem chegar, de modo que o servidor necessitará enfileirá-las, para atendê-las quando estiver liberado.

A política de gerência dessa fila pode variar dependendo do algoritmo utilizado no servidor, e sua função é determinar qual solicitação será atendida quando o servidor tiver terminado de atender a solicitação anterior.

Algumas políticas preservam a ordem de chegada dos pacotes, como por exemplo a FCFS (*First Come First Served*). Outras alocam prioridades a certos tipos de fluxo, como as do tipo CBQ (*Class-Based Queuing*). Dentro dessas políticas de gerência

existem diversas variantes, como FIFO, *Strict Priority*, CBQ, FQ, WFQ e WF^2Q. Todas elas estão descritas em Roesler [ROE 2003h].

2.11 Políticas de descarte de pacotes nas filas dos roteadores

Caso a taxa de chegada de pacotes exceder a taxa de processamento, a fila vai expandir. Em um determinado instante pode haver demasiado número de pacotes na fila, e o servidor deve tomar uma atitude, que pode ser marcar ou descartar algum pacote. Isso pode servir como realimentação ao transmissor para que ele diminua sua taxa de transmissão e mantenha o equilíbrio na rede, conforme visto no item 2.8. O instante de início de marcação / descarte de pacotes não é necessariamente quando a fila estiver cheia, como será visto nos itens a seguir.

As principais políticas de descarte são a *droptail, head drop, random drop*, RED e descarte por prioridades. Vantagens e desvantagens de cada uma, bem como detalhes de seu funcionamento podem ser obtidas em [ROE 2003h].

2.12 Tráfego CBR e VBR

É importante entender a diferença entre os tipos de tráfego CBR (*Constant Bit Rate*) e VBR (*Variable Bit Rate*) dentro do contexto desta Tese (ambiente IP), pois ambos serão utilizados no algoritmo desenvolvido.

Os pacotes que formam tráfego do tipo CBR são transmitidos a intervalos constantes, gerando uma única taxa de transmissão ao longo do tempo. Por exemplo, caso seja necessário uma taxa de 100 kbit/s com pacotes de 500 bytes (4000 bits), o transmissor deve enviar 25 pacotes por segundo, ou um pacote a cada 40ms. O conhecimento desse espaçamento é importante caso o receptor necessite inferir o nível de congestionamento da rede através da variação no atraso de chegada dos pacotes. Um exemplo de transmissão CBR é visto na figura 2.13, onde todos os quadros possuem o mesmo tamanho e devem ser transmitidos a uma taxa fixa, como por exemplo 30 quadros por segundo.

Quando o tráfego é do tipo VBR, existe uma taxa de transmissão média e também uma taxa de pico. A taxa média refere-se ao valor de longo prazo, enquanto a taxa de pico é permitida por determinados intervalos, onde normalmente acontecem rajadas. Esse tipo de tráfego é gerado por transmissores que necessitam enviar pacotes de diferentes tamanhos ao longo do tempo, como os CODECs para transmissão de vídeo MPEG. A figura 2.13 mostra um exemplo desse tipo [PEA 96], com 30 quadros por segundo utilizando o padrão GOP (*Group of Pictures*) IBBPBBPBB. Pode-se observar que a seqüência da transmissão possui um padrão de quadros grandes (I), médios (P) e pequenos (B).

Em [PEA 96], Murray Pearson demonstrou que o tráfego em redes do tipo Ethernet possui um comportamento VBR fractal auto-similar, provando que não há um tamanho natural de uma rajada de tráfego, ou seja, existem rajadas de tráfego em escalas de tempo variando de milisegundos a minutos e horas.

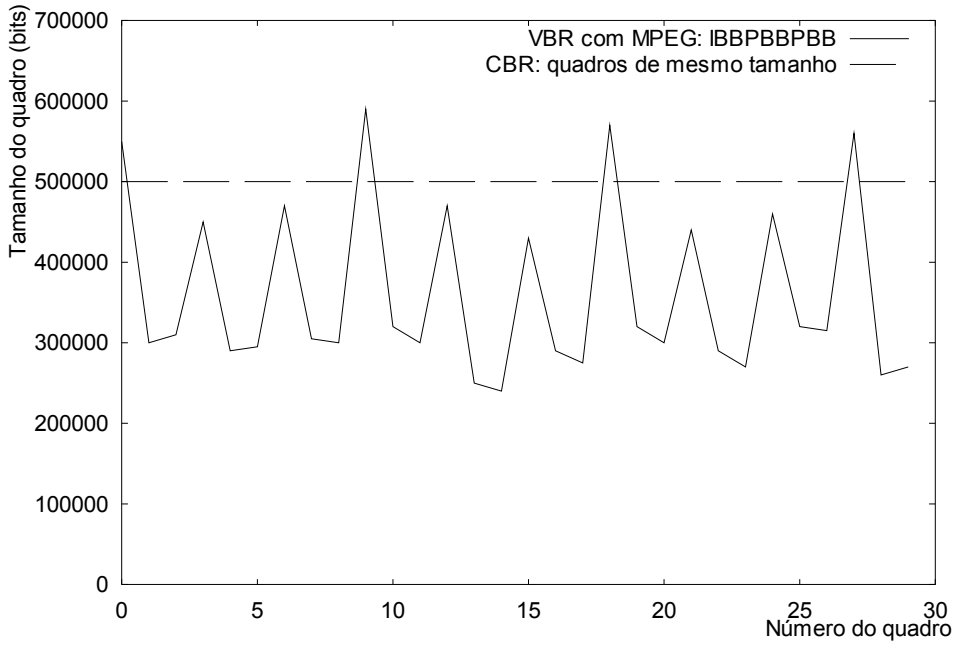


FIGURA 2.13 – Comparação entre CBR e VBR numa transmissão de um segundo

Para fins de teste do algoritmo desenvolvido nesta Tese, utilizou-se simulação com modelos de tráfego concorrente CBR e VBR. No modelo do codificador VEBIT, utilizado no sistema e descrito no capítulo 8 (Codificação de Vídeo Multi-Taxas), os quadros de vídeo são gerados numa taxa constante (24 quadros por segundo), mas cada quadro possui um número de bytes variável, causando variabilidade na taxa do sinal transmitido.

2.13 Protocolos de tempo real RTP e RTCP

Para transportar dados em tempo real, são necessários protocolos que levem consigo informações de sincronismo e de tempo, como o RTP (*Real-time Transport Protocol*). Para permitir o monitoramento da qualidade de recepção do sinal, existe o protocolo RTCP (*RTP Control Protocol*).

O protocolo RTP (*Real-time Transport Protocol*), descrito na RFC 1889 [SCH 89], especifica um formato para transmissão de dados em tempo real, facilitando o envio de sinais como áudio, vídeo ou dados de simulação. Alguns benefícios obtidos por esse protocolo são [PAU 98]:

- Detecção de perda de pacotes: observando o número de seqüência é possível saber se houve perda de pacotes ou não. Isso é útil para estimar a qualidade da recepção, adaptação da aplicação às características da rede, recuperação de dados, e assim por diante;
- Sincronização intramídia: o campo de *timestamp* do cabeçalho informa ao receptor o momento exato de passar os dados ao usuário. Essa informação é usada pelo receptor absorver o *jitter* da rede através de uma fila auxiliar;
- Sincronização intermídia: o campo de *timestamp* do cabeçalho de diferentes sessões RTP (como áudio e vídeo) pode ser usado em conjunto com o protocolo NTP

(*Network Time Protocol*) a fim de sincronizar as diferentes mídias. Um exemplo típico é o sincronismo voz-lábio. Outro é o sincronismo de uma seta na tela apontando objetos de acordo com um texto falado.

A garantia de entrega do pacote ou a qualidade de serviço da rede não são especificadas no RTP, e devem ser obtidas através de outro mecanismo de entrega, como o RSVP, Diffserv ou outro.

O protocolo RTCP (*RTP Control Protocol*) tem por objetivo fornecer *feedback* sobre a qualidade de serviço obtida na distribuição de dados RTP, e consegue isso através de transmissões periódicas de pacotes de controle a todos participantes da sessão RTP, utilizando o mesmo mecanismo de distribuição do RTP (unicast ou multicast), e possuindo uma porta específica de controle na sessão. Suas funções são [SCH 89]:

- Realimentar o transmissor sobre a qualidade obtida na transmissão RTP. Para isso ele gera relatórios periódicos, cuja periodicidade é variável, pois depende da quantidade de receptores participando da sessão. Exemplos de utilização são: controle de codificadores adaptativos (muda algoritmo de compactação dependendo da qualidade), diagnóstico de problemas na rede, e outros;
- Enviar o nome canônico (CNAME) do transmissor dos dados, utilizado para que todos saibam quem originou a transmissão;
- Controle da periodicidade de envio dos pacotes RTCP, a fim de permitir a escalabilidade do sistema;
- Função opcional para permitir o transporte de informações mínimas de controle, permitindo, por exemplo, que a identificação de cada participante seja apresentada na interface com o usuário.

No relatório técnico feito por Roesler [ROE 2003g], os protocolos RTP e RTCP são detalhados em maior profundidade.

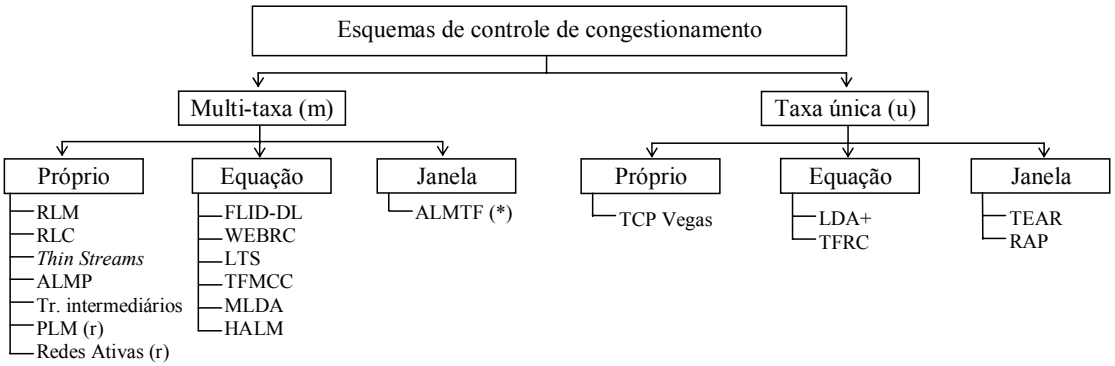
Para que o sistema descrito neste trabalho funcione, foi necessária a implementação de mecanismos similares ao RTP e RTCP, tanto na implementação da simulação como na implementação do sistema em si, e os detalhes podem ser vistos nos capítulos 5 (ALMP: ALM for Private Networks) e 6 (ALMTF: ALM TCP-Friendly).

3 Trabalhos Relacionados

O capítulo anterior introduziu, entre outros assuntos, os conceitos básicos sobre transmissão multimídia e a questão do congestionamento em redes de computadores, bem como a necessidade de que os novos algoritmos desenvolvidos mantenham uma taxa de transmissão que seja eqüitativa com o TCP, pois este é o protocolo dominante na Internet. Além disso, definiu-se o conceito de transmissão em camadas, que é uma alternativa para obter-se acesso universal às transmissões multimídia, tanto para receptores localizados em redes lentas, como para os localizados em redes rápidas.

O objetivo deste capítulo é introduzir alguns algoritmos relevantes e que tragam contribuições ao protocolo proposto nesta Tese. No item 2.8 (Esquemas de controle de congestionamento) foram vistos diversos esquemas existentes de controle de congestionamento, e os mesmos foram divididos nas seguintes categorias: a) orientado a janela e orientado a taxa; b) unicast e multicast; c) taxa única e multi-taxa (transmissão em camadas); d) fim-a-fim e apoiado pelo roteador; e) orientados a transmissor e orientados a receptor.

Para efeitos desta Tese, considerou-se importante a divisão dos algoritmos de controle de congestionamento em relação ao método que eles utilizam para conseguir eqüidade com o tráfego TCP, portanto, os algoritmos foram divididos da seguinte forma: a) multi-taxa e taxa única; b) mecanismo de adaptação de tráfego próprio, utilizando a equação do TCP ou através de adaptação por janela. Essas categorias foram privilegiadas, conforme ilustrado na figura 3.1. Também foi considerada importante a rápida visualização dos mecanismos que efetuam comunicação fim-a-fim ou exigem modificações nos roteadores intermediários, pois os últimos não podem ser utilizados na Internet atual, e foram marcados com a legenda (r). Todos os protocolos multi-taxa analisados são baseados em multicast e todos de taxa única são baseados em unicast.



(m)=multicast; (u)=unicast; (r)=apoiado pelo roteador.
(*) O ALMTF utiliza principalmente janela, mas também utiliza a equação do TCP

FIGURA 3.1 – Classificação de esquemas de controle de congestionamento

O protocolo "transcodificadores intermediários" foi classificado como fim-a-fim, entretanto, necessita agentes rodando dentro de domínios específicos para seu funcionamento. Ele foi considerado multi-taxa, pois utiliza vários grupos multicast, porém, cada receptor se cadastra em um único grupo multicast.

Todos os protocolos vistos na figura serão detalhados ao longo deste capítulo, com exceção do ALMP e do ALMTF, que são os algoritmos propostos nesta Tese e serão detalhados nos capítulos 5 (ALMP: ALM for Private Networks) e 6 (ALMTF: ALM TCP-Friendly).

Vale lembrar que existem muitos outros trabalhos nesta área, porém, os mesmos não foram detalhados por desviarem da proposta desta Tese, que objetiva a criação de um algoritmo para transmissão de sinais multimídia e que se adapte em ambientes heterogêneos sem a necessidade de modificação no algoritmo dos roteadores intermediários nem utilização de QoS. Entre tais protocolos não analisados encontram-se os de multicast confiável e muitos que exigem a utilização de QoS na rede.

Para o bom entendimento deste capítulo, assume-se que o leitor esteja familiarizado com os conceitos vistos no capítulo 2 (Conceitos Básicos). Os próximos itens descrevem detalhadamente cada um dos protocolos vistos na figura 3.1.

3.1 RLM (Receiver-driven Layered Multicast)

O primeiro método de controle de congestionamento utilizando transmissão multicast em camadas foi descrito por McCanne e denominado RLM (*Receiver-driven Layered Multicast*) [MCC 96].

O RLM utiliza controle de congestionamento por taxa baseado somente nas estações finais (controle "fim-a-fim"), sendo orientado a receptor. O receptor se adapta automaticamente escolhendo a taxa mais adequada para receber os dados, sempre dentro dos limites da taxa de cada camada enviada pelo transmissor.

Quando um novo receptor quer entrar na sessão sendo transmitida, ele se inscreve, efetuando *join* primeiramente no grupo multicast mais básico, passando a receber essa taxa de dados, como pode ser observado na figura 3.2, instante *t0*. Após um certo tempo, caso não aconteçam perdas de pacotes, ele faz *join* na próxima camada (instante *t1*), e assim sucessivamente até a ocorrência de determinado número de perdas (instante *t4*). Nesse momento, o receptor infere que a taxa de pacotes que está recebendo é superior às suas condições topológicas ou de congestionamento, portanto, faz *leave* na camada mais alta, buscando uma diminuição na taxa de recebimento de dados e conseqüente estabilidade no tráfego.

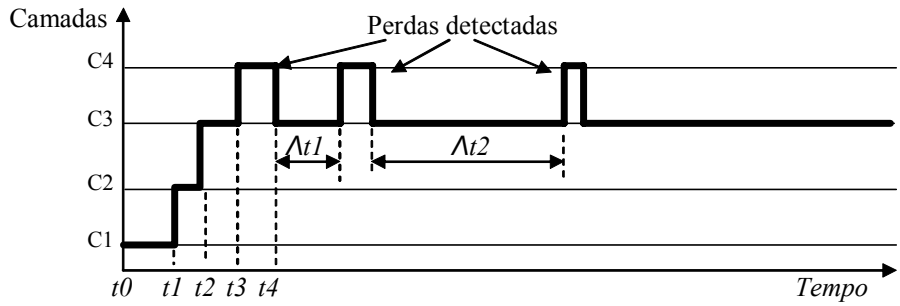


FIGURA 3.2 – Funcionamento do RLM

A cada intervalo de tempo Δt o receptor tenta subir uma camada (através de um *join* no grupo multicast correspondente), a fim de melhorar a qualidade da recepção,

pois as condições da rede podem ter mudado durante esse tempo. Caso detecte perda de pacotes, ele desce novamente para a camada anterior, conforme descrito anteriormente. As tentativas de *join* são cada vez mais esparsas, como mostra a figura 3.2 (intervalos $\Delta t1$ e $\Delta t2$), a fim de não ficar gerando insistentes congestionamentos numa rede de topologia lenta.

O modelo do RLM poderia não funcionar caso as tentativas de *join* fossem efetuadas de forma independente entre os receptores, pois sofreria o problema de sincronismo explicado no item 2.8.2. Para evitar isso, McCanne propôs uma solução de "aprendizado compartilhado", onde o receptor envia uma mensagem multicast para todo grupo avisando que vai fazer uma tentativa de *join* em determinada camada. Todos aprendem com isso, zerando seus contadores de novas tentativas. Da mesma forma, quando um receptor está fazendo uma tentativa de *join* numa camada superior, os outros não tentam, a fim de não interferirem entre si.

A figura 3.3 mostra as topologias utilizadas para simulações do protocolo RLM no simulador NS2 [MCC 96]. A topologia 1 tem por objetivo analisar a escalabilidade do protocolo em enlaces de atraso variável. O enlace central sofreu variações de atraso de 1 ms a 20 segundos. A topologia 2 busca explorar a escalabilidade do protocolo em relação ao número de receptores, que variaram de 1 a 100. A topologia 3 considerou dois conjuntos de receptores com enlaces de diferentes velocidades. O primeiro conjunto tinha uma largura de banda B, e o segundo uma largura de banda B/2, e o número de receptores variou de 1 a 35. Finalmente, a topologia 4 considerou várias sessões RLM concorrentes, onde cada receptor se conectou a uma sessão diferente que estava sendo executada no transmissor correspondente. O objetivo dessa topologia foi analisar a eqüidade de tráfego para várias sessões RLM concorrentes, onde o número de sessões variou de 1 a 35.

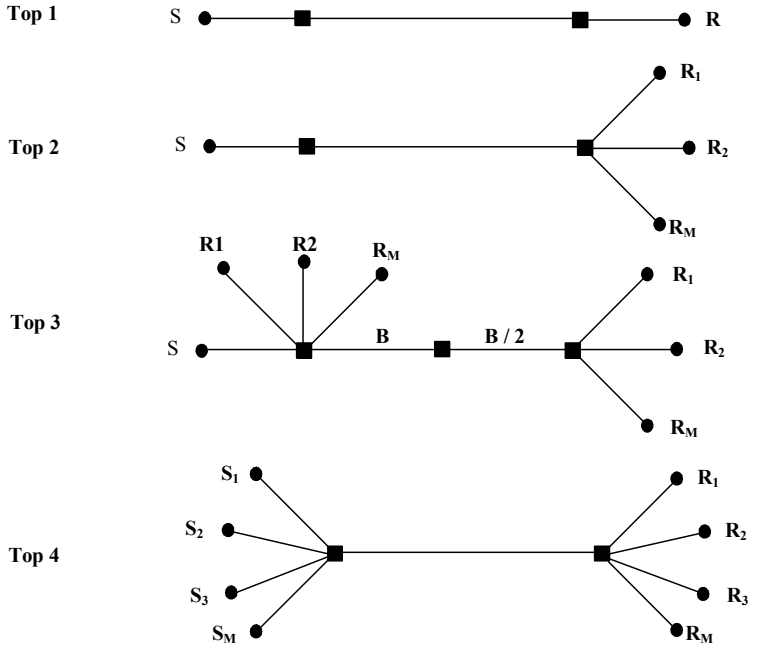


FIGURA 3.3 – Topologias utilizadas para simulação do RLM

3.1.1 Pontos negativos do RLM

No algoritmo do RLM, o receptor decide testar se a rede mudou sua condição de congestionamento a cada intervalo de tempo, entretanto, pode acontecer da topologia não permitir que o receptor se inscreva em uma nova camada. Por exemplo, na topologia da figura 2.11, R1 nunca terá condições de subir além da camada C3, e os outros receptores nunca poderão subir além da camada C2. Entretanto, o RLM utiliza somente o tempo como parâmetro, ignorando aspectos de topologia e redes com tráfego estável. A conseqüência disso é que, de tempos em tempos, haverá uma geração de congestionamento na rede.

Outro ponto negativo mostrado por Legout em [LEG 2000] é o demorado tempo de adaptação do RLM. Para analisar esse problema, foi desenvolvida uma simulação para o NS2 cujo código está disponível em http://www.inf.unisinos.br/~roesler/tese e o resultado é ilustrado na figura 3.4. A linha sólida no gráfico mostra uma simulação com 20 camadas lineares de 50 kbit/s, e o algoritmo demorou aproximadamente 250s para adaptar-se. A linha pontilhada mostra uma simulação com 8 camadas exponenciais, começando em 32 kbit/s. O algoritmo demorou aproximadamente 50s para adaptar-se. No capítulo 7 (Comparações com outros algoritmos) o RLM será comparado mais detalhadamente com outros algoritmos.

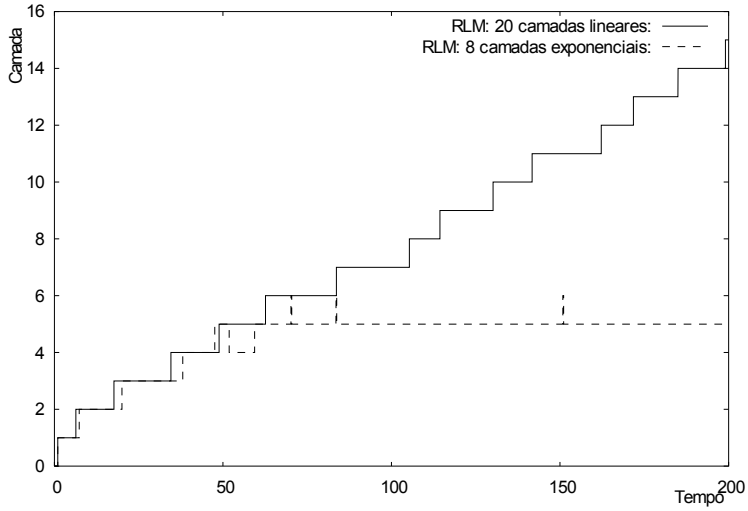


FIGURA 3.4 – Tempo para RLM adaptar-se

Além disso, foi observado que o RLM é altamente instável e obtém pouca utilização do enlace para tráfego VBR, além de gerar problemas de injustiça na rede com tráfego TCP concorrente. Os resultados das simulações podem ser vistos em [LEG 2000] e [BRU 2000].

3.2 RLC (Receiver-driven Layered Congestion Control)

O RLC (*Receiver-driven Layered Congestion Control*) foi proposto por Vicisano [VIC 98], e utiliza transmissão multicast em camadas, com controle de congestionamento por taxa, fim-a-fim e orientado a receptor, tal qual o RLM.

Da mesma forma que o RLM, o receptor se adapta automaticamente, escolhendo

a taxa mais adequada para receber os dados, sempre dentro dos limites da taxa de cada camada enviada pelo transmissor, porém, cada camada obrigatoriamente deve possuir uma taxa duas vezes a taxa da camada inferior. A principal diferença é no momento de decidir subir uma camada. Enquanto no RLM o receptor tenta subir de camada de tempos em tempos, no RLC o receptor só sobe camada em determinados pontos de sincronização, identificados através de pacotes com *flags* especiais.

O funcionamento do algoritmo é ilustrado na figura 3.5, onde são representadas 4 camadas exponenciais, com cada camada transmitindo o dobro de pacotes da camada inferior. O tempo é dividido em 8 intervalos de transmissão[1], sendo que no primeiro intervalo o transmissor envia uma taxa equivalente ao dobro do número de pacotes que seriam transmitidos normalmente nesse intervalo. No final do primeiro intervalo, o transmissor envia pacotes com *flags* especiais indicando o ponto de sincronização, que nesse caso será para todas as camadas, pois em todas foi enviado o dobro do número de pacotes (na figura 3.5 esse momento é ilustrado como "ponto de sincronização"). No segundo intervalo nenhum pacote é transmitido a fim de manter a média de transmissão constante. Nos intervalos "*Normal (1)"* a "*Normal (6)"*, o número de pacotes transmitido é coerente com a taxa de cada camada.

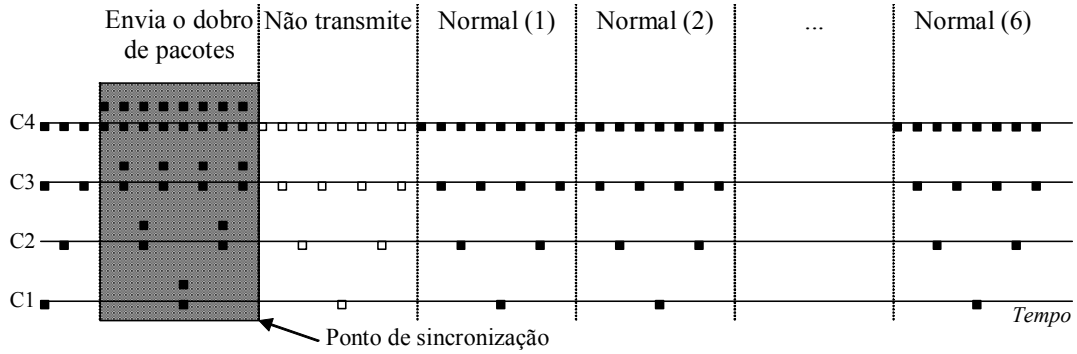


FIGURA 3.5 – Algoritmo do RLC

Através desse estilo de transmissão, o receptor sabe se pode ou não subir de camada, pois se o transmissor enviou o dobro do número de pacotes e não aconteceram perdas, significa que o receptor pode suportar a camada superior. O objetivo é eliminar a característica negativa do RLM de possuir freqüentes *joins* e *leaves*, fazendo com que o receptor tenha garantia de que, se tentar subir uma camada, vai conseguir. Segundo Vicisano, esse fator elimina o tempo de *leave* do grupo multicast, que é bastante lento, conforme descrito no item 2.8.2.

Os pontos de sincronização acontecem mais freqüentemente nas camadas inferiores, conforme mostra a figura 3.6, dando maior chance aos receptores com menos banda subirem de camada e aumentando a eqüidade de tráfego. A cada intervalo de tempo existe um ponto de sincronização para a camada C1; a cada 2 intervalos de tempo existem pontos de sincronização para as camadas C1 e C2; a cada 4 intervalos de tempo existem pontos de sincronização para as camadas C1, C2 e C3, e assim por diante.

Pode-se observar que quando o ponto de sincronização acontece em uma camada, ele acontece em todas as camadas abaixo dela simultaneamente. O objetivo

[1] O número de intervalos de transmissão pode ser variável, dependendo de parâmetros do algoritmo.

disso é evitar que a rajada de tráfego de uma camada (que pode gerar perda de pacotes em todas as camadas) resulte em interpretação de congestionamento por outro receptor, fazendo com que ele faça *leave* em uma camada erroneamente. Assim, quando um receptor está num ponto de sincronização fazendo tentativa de *join*, todos os receptores abaixo dele também estão fazendo tentativas de *join*.

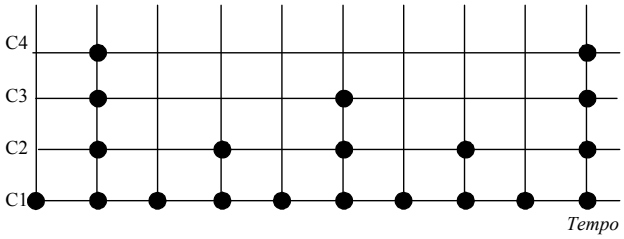


FIGURA 3.6 – Pontos de sincronização no RLC

Em resumo, o receptor pode subir, descer ou permanecer na mesma camada, de acordo com as seguintes definições:

- Subir de camada: se num ponto de sincronização, que é o ponto onde os pacotes chegam em dobro, não acontecerem perdas. Para o algoritmo, isso significa que a rede tem capacidade suficiente para suportar a próxima camada, que possui uma taxa de transferência igual ao dobro da atual.
- Descer de camada: sempre que houver perda de pacotes durante uma transmissão normal. O autor do RLC faz uma exceção logo após a descida de uma camada, aguardando um tempo chamado TP (tempo de *prune*), pois quando o receptor efetua o *leave*, só deixa de receber pacotes após o *prune* na árvore multicast;
- Não modifica estado: em todos os outros momentos. Se acontecerem perdas durante o intervalo de sincronização, não desce camada, pois a rajada de pacotes foi feita exatamente para testar isso.

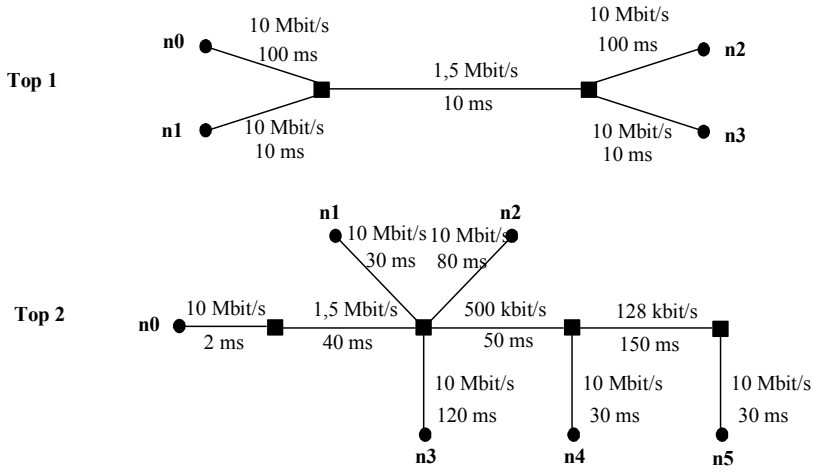


FIGURA 3.7 – Topologias utilizadas para simulação do RLC

A figura 3.7 mostra as topologias utilizadas para simulações do protocolo RLC no simulador NS2 [VIC 98]. A topologia 1 foi utilizada para verificar a eqüidade do tráfego entre várias sessões RLC e em relação a sessões TCP concorrentes. A topologia 2 foi utilizada para verificar o nível de adaptação do protocolo em redes contendo

enlaces de diferentes capacidades (redes heterogêneas).

3.2.1 Pontos negativos do RLC

Segundo simulações efetuadas por Legout [LEG 2000], o RLC não funciona, pois as rajadas de tráfego são absorvidas pelas filas dos roteadores intermediários, e o receptor assume que pode subir uma camada quando, na verdade, isso vai provocar congestionamento, tornando o RLC bastante instável.

Outra característica negativa é que cada camada deve ter o dobro da taxa da camada anterior, permitindo ao receptor somente dobrar ou dividir pela metade sua banda, porém, muitas aplicações não se adaptam a essa distribuição [WID 2001].

3.3 Thin Streams

A arquitetura de *Thin Streams* foi proposta por Linda Wu [WU 97], e utiliza transmissão multicast em camadas, com controle de congestionamento por taxa, fim-a-fim, orientado a receptor. Suas principais diferenças em relação a outros trabalhos nesta área específica são as seguintes:

- Utilização de camadas "finas", ou seja, com uma taxa de transmissão baixa. O motivo disso é evitar grandes oscilações na rede a partir das tentativas de *join* dos receptores. A justificativa é que uma tentativa de *join* gera um grande número de perdas que poderiam ser evitadas. Por exemplo, se um receptor efetua *join* num grupo multicast de 256 kbit/s (ou 32 kbytes/s) e o experimento demora 2 segundos, isso vai exigir uma capacidade de enfileiramento da rede de 64 kbytes a fim de evitar perdas;
- Utilização de um mecanismo similar ao TCP Vegas (item 3.12) para detectar congestionamento. Tal mecanismo é baseado na variação do atraso de chegada dos pacotes. Caso o atraso aumente, é uma indicação de que as filas estão aumentando, portanto, o congestionamento também está aumentando. Isso evita a ocorrência de perdas para detecção de congestionamento;
- Utilização de um sinal de *clock* para sincronizar as tentativas de *join* em receptores da mesma sessão. Esse sinal de *clock* deve ser diferente para diferentes transmissores, e isso é conseguido através de uma seqüência pseudo aleatória cuja semente é o endereço IP do transmissor concatenado com um número randômico.

A fim de absorver o impacto dos experimentos de *join*, o algoritmo limita a quantidade de banda a ser utilizada em cada camada. Para cálculo dessa banda máxima de cada camada sem gerar perdas foi considerado que cada tentativa de *join* gera um aumento de enfileiramento na rede (F), como visto no primeiro item acima. Pode-se dizer que, para evitar perdas, a capacidade de enfileiramento na rede deve ser: F <= B.T, onde B é a banda da nova camada, e T é uma estimativa de duração do tempo de *join*. Usando um valor máximo permitido de enfileiramento na rede (F) de 4 kbytes e T igual a 2 segundos, calcula-se que B = 2 kbytes, ou 16 kbit/s.

Para controle de congestionamento, o receptor monitora continuamente a banda recebida (A) e a banda estimada (E), armazenando a diferença entre E e A. Com essa

informação, o receptor infere se a rede está ficando congestionada, conforme detalhamento na descrição do TCP Vegas (item 3.12), efetuando *join* ou *leave* apropriadamente, de acordo com as variáveis *join_threshold* e *leave_threshold*, que representam os limites de tempo nos quais o receptor deve efetuar *join* ou *leave*. A vantagem desse mecanismo é que não é necessária a geração de perdas para que se detecte congestionamento.

A fim de obter convergência entre receptores cadastrados em um número diferente de camadas, o algoritmo aumentou a sensibilidade das camadas mais altas de duas formas: a) quanto mais grupos o receptor está cadastrado, menor o *threshold* para efetuar *leave*, assim, quanto mais banda o receptor estiver recebendo, mais fácil é para diminuir essa banda; b) quanto mais grupos o receptor está cadastrado, maior o *threshold* para efetuar novo *join*, assim, receptores com menos banda efetuam mais tentativas de *join* do que receptores com mais banda, acelerando a convergência.

Como foi visto no item 2.8.2, existe a necessidade de sincronismo entre receptores cadastrados na mesma sessão, e deve-se evitar o sincronismo para receptores cadastrados em sessões diferentes. Para obter isso, o protocolo Thin Streams faz com que cada transmissor envie um sinal de *clock* na camada base, que consiste de um bit que está sempre em zero, e quando passa para um indica que o receptor pode efetuar tentativa de *join*. O sinal de *clock* de cada transmissor é diferente, pois é baseado numa seqüência pseudo aleatória conforme visto anteriormente. Isso garante que receptores na mesma sessão vão efetuar tentativas de *join* sincronizadamente, ao passo que receptores em sessões diferentes não vão fazer isso.

As duas principais topologias simuladas com o algoritmo *Thin Streams* estão mostradas na figura 3.8. A primeira topologia avalia o protocolo em relação à adaptação em enlaces de diferentes velocidades e grande número de receptores, e foi utilizada uma largura de banda B = 512 kbit/s. Cada bloco de receptores (R1 a R4) representa um conjunto de 32 receptores. A segunda topologia avalia o protocolo concorrendo com tráfego de outras sessões *Thin Streams*.

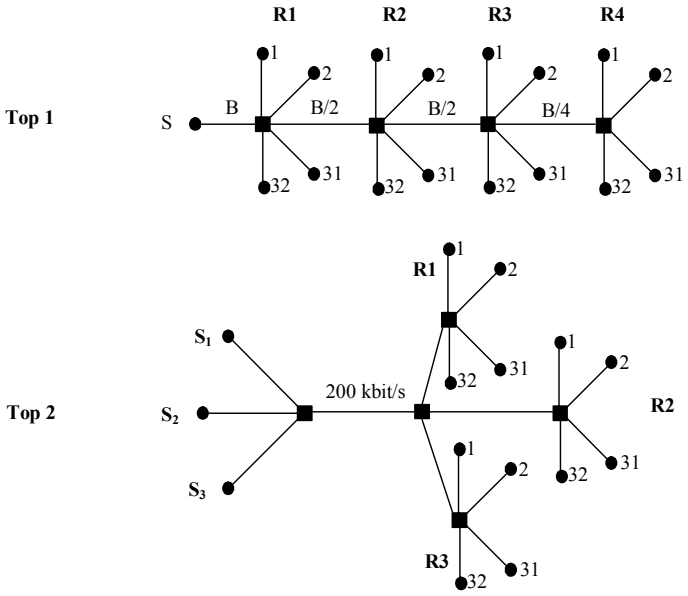


FIGURA 3.8 – Topologias usadas para simular o protocolo *Thin Streams*

Os resultados das simulações mostraram uma instabilidade alta, talvez devido à pequena banda de cada camada. Havia uma troca de camada aproximadamente a cada 10 ms, o que foi julgado bastante alto.

Um problema no controle de congestionamento similar ao TCP Vegas é que, como será visto no seu detalhamento, esse mecanismo não gera tráfego de forma eqüitativa quando concorre com o TCP Reno. Observou-se que o algoritmo Thin Streams não foi testado com tráfego TCP Reno concorrente. Outra desvantagem do algoritmo é a necessidade de um grande número de grupos multicast, gerando mais tráfego de controle na rede.

3.4 Transcodificadores Intermediários

O modelo de transcodificadores intermediários foi proposto por Kouvelas [KOU 98], e utiliza transmissão multicast com máquinas intermediárias (transcodificadores) que adaptam a taxa de transmissão de acordo com as possibilidades do grupo de receptores localizados abaixo dele hierarquicamente. O controle de congestionamento necessita de máquinas intermediárias, que funcionam como novos transmissores localizados em pontos estratégicos.

A figura 3.9 apresenta uma visão geral do sistema, que consiste de um transmissor T enviando um fluxo de alta qualidade, adequado para os receptores R1 e R2. Já os receptores R3, R4 e "coordenador" não possuem banda suficiente para receber a transmissão, por isso existe o transcodificador TR, que recebe o fluxo multicast de T e o converte para uma qualidade inferior, enviando-o num endereço multicast diferente. Os receptores R3, R4 e "coordenador" devem, simultaneamente, mudar para o novo grupo multicast, a fim de evitar mais congestionamentos (isso é coordenado pela entidade "coordenador").

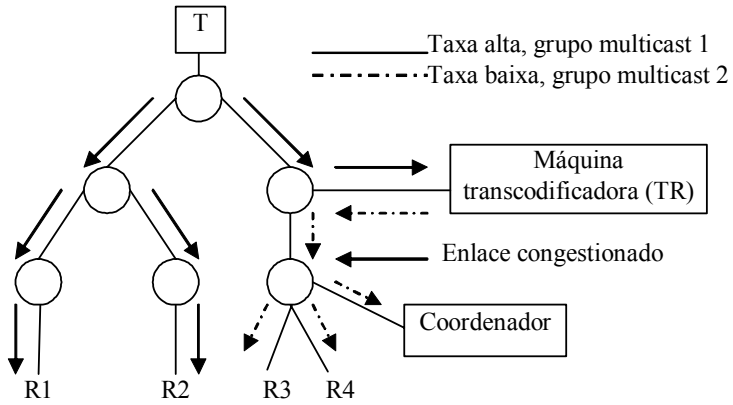


FIGURA 3.9 – Transcodificador adaptando para sub-rede lenta

Um ponto positivo desse sistema é que o receptor se cadastra em apenas um grupo multicast, reduzindo as mensagens de controle multicast.

Como ponto negativo pode-se dizer que cada sub-rede onde existam problemas de congestionamento vai depender da existência de uma máquina transcodificadora, que não pode ser desligada e deve estar sempre presente, senão o sistema não funciona. Isso aumenta bastante a complexidade do sistema e dificulta a escalabilidade.

Além disso, o sincronismo entre os receptores é conseguido graças a um algoritmo complexo, onde existe um coordenador central responsável por procurar um transcodificador ativo e efetuar a troca do endereço multicast. Esse modelo deve tratar ainda casos de falha no coordenador e no transcodificador intermediário, aumentando ainda mais a complexidade.

3.5 PLM

O protocolo PLM (*packet Pair receiver-driven cumulative Layered Multicast*) [LEG 2000b] foi criado por Legout e tem como objetivo a transmissão de aplicações multimídia (áudio e vídeo) através de multicast em camadas. O controle de congestionamento é por taxa, e a adaptação ao número correto de camadas acontece no receptor, que utiliza a técnica de pares de pacotes para inferir a quantidade disponível de banda na rede. Detalhes da técnica de pares de pacotes podem ser obtidas no Anexo A.

Para inferir a banda disponível ente o transmissor e o receptor, o algoritmo exige que todos os roteadores intermediários utilizem fila do tipo WF^2Q (detalhes em [ROE 2003h]). Com esse tipo de fila, o espaçamento entre o par de pacotes permite calcular a banda eqüitativa com uma boa precisão.

Uma característica do mecanismo de controle de congestionamento do PLM é que a inferência de banda é feita através do espaçamento dos pares de pacotes, não necessitando a ocorrência de perdas para a detecção de congestionamento e diminuição da taxa solicitada no receptor.

O transmissor PLM envia os dados em camadas e em pares de pacotes, assim, todos os pacotes em todas as camadas são enviados em pares, com o objetivo de maximizar o número de estimativas. Quando o receptor entra na sessão, ele descobre a composição das camadas através de um fluxo multicast enviado periodicamente pelo transmissor contendo o nome da sessão, sumário, distribuição das camadas, e assim por diante. Com essas informações, o receptor aguarda o primeiro par de pacotes, a fim de configurar o tempo atual do sistema (*t*). O algoritmo vai analisar a possibilidade de subir ou descer camadas a cada instante *Tc*, que se repete uma vez por segundo.

O código a seguir mostra o algoritmo que roda no receptor [LEG 2000b]. As variáveis significam o seguinte: *PPt* = total de banda medida pelo receptor via pares de pacotes; *Bn* = total de banda utilizada pelo receptor neste momento (inscrito na camada *n*); *t* = tempo atual; *Be* = banda estimada pelo receptor durante o tempo C. Alguns pontos-chave do código foram numerados para facilitar o detalhamento posterior.

```
Se PPt < Bn // banda medida menor que banda atual
  - Tc = (t + C) // C = 1 segundo
  – Enquanto Bn < PPt                                          (1)
    * drop layer (n)
    * n = (n – 1)
Senão, Se Tc < t // recebeu pacotes pelo tempo C
  – Be = min (PPt) // utiliza valor mínimo do intervalo
  – Tc = t + C // C = 1 segundo
  – Se Be >= Bn
      Enquanto (Bn + 1) < Be                                   (2)
        * add layer (n + 1)
        * n = (n + 1)
```

Assim, o algoritmo se desinscreve de camadas cada vez que uma determinada estimativa de banda (*PPt*) resultar num valor menor do que a banda utilizada (*Bn*), como pode ser visto pelo número "1" do código. Entretanto, para subir camadas é mais complexo, sendo feito somente em determinados intervalos de tempo (dado pela variável *C*) e considerando o valor mínimo de estimativas de banda, como pode ser visto no número "2" do código acima. Isso torna o algoritmo bastante conservador em relação ao congestionamento, tendo mais facilidade de descer camadas do que subir.

A figura 3.10 mostra as topologias utilizadas para simulações do protocolo PLM no simulador NS2 [LEG 2000b]. A topologia 1 foi utilizada para analisar a velocidade e capacidade de adaptação do protocolo em ambientes heterogêneos, e foram utilizadas camadas de 10 kbit/s. O objetivo da segunda topologia foi analisar a escalabilidade do algoritmo em relação ao número de receptores, com 50 kbit/s por camada e até 100 receptores iniciando no instante 5s. A topologia 3 tem múltiplos fluxos PLM, com um receptor ligado a cada fluxo, e seu objetivo foi analisar a justiça com o próprio tráfego[1]. O objetivo da topologia 4 foi testar o PLM com ambientes dinâmicos, e se utilizou uma mistura de fluxos PLM e CBR. No caso, foi utilizado M = 3 e U = 3. As seguintes siglas foram utilizadas: S = *sender*; R = *receiver*; M = número de transmissores ou receptores PLM; U = número de transmissores ou receptores CBR.

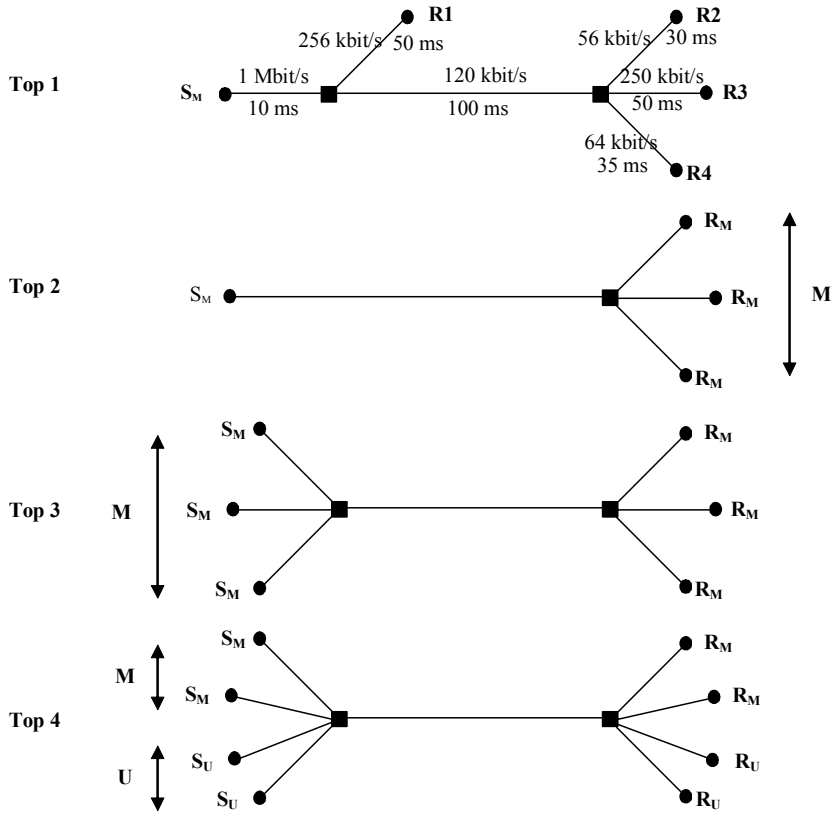


FIGURA 3.10 – Topologias de simulação no PLM

[1] *fairness*

3.5.1 Pontos negativos do PLM

Os autores do PLM argumentam que é possível modificar todos os roteadores da Internet para que utilizem filas WF^2Q, entretanto, o autor desta Tese e Widmer [WID 2001] acreditam que vai demorar muito para que os roteadores da Internet em geral sejam baseados em filas desse tipo, invalidando a utilização do PLM de forma prática.

3.6 Redes Ativas

O conceito de Redes Ativas consiste numa nova abordagem de arquitetura de redes onde programas customizados são executados nos elementos centrais da rede, ou seja, nos roteadores [WET 99].

A proposta de Gonçalves [GON 2000] baseia-se na utilização de camadas juntamente com redes ativas para efetuar o roteamento multicast, e não mais em um modelo multicast tradicional. Nesse modelo, envia-se o código para cada nó ativo na rede, e o mesmo efetua a adaptabilidade aos diferentes usuários do sistema, de acordo com o nível de qualidade solicitado pelo usuário. O protocolo é baseado em *soft-state*, necessitando mensagens periódicas para a manutenção de determinado usuário no sistema.

No protocolo proposto, existem cinco tipos de pacotes (denominados "cápsulas"), que são as seguintes: *inscrição / manutenção*, *disponibilidade*, *dados de vídeo*, *fim de sessão* e *desligamento*.

A cápsula de *inscrição / manutenção* é um pacote de dados que serve para formar e manter a árvore multicast. Cada receptor que quiser participar da transmissão deve enviar periodicamente uma dessas para o transmissor. Para evitar um excesso no número de cápsulas enviadas, existe um mecanismo de compartilhamento ao longo da árvore.

As cápsulas de *disponibilidade* são pacotes enviados pelos receptores e pelos nós ativos na direção da fonte, permitindo a construção de uma tabela em cada nó, com informações sobre a qualidade desejada pelo usuário e disponibilidade da rede. A figura 3.11 mostra um exemplo do sistema após a entrada dos receptores A, B e C. A taxa permitida em cada nó da rede é uma função da disponibilidade da rede (velocidade do enlace) mais a qualidade desejada pelo usuário (representado por Q).

Assim, os receptores A e B irão receber no máximo 2 Mbit/s, pois estão limitados pela rede, enquanto o receptor C vai receber os 8 Mbit/s solicitados.

A cápsula *dados de vídeo* é a transmissão do vídeo em camadas, e dependendo da taxa de saída do transmissor, mais de uma camada é enviada. Em cada nó intermediário, as tabelas de disponibilidade são utilizadas para saber se devem ser enviadas / replicadas todas as camadas ou somente as mais prioritárias.

O modelo apresentado é dinâmico, e conforme receptores entram, saem ou os usuários mudam as preferências de qualidade, as correspondentes tabelas de disponibilidade mudam, afetando o tráfego.

Um ponto negativo deste modelo é que não está disponível na Internet atual,

pois depende de nós intermediários com suporte a redes ativas. Além disso, o autor não tratou congestionamento na rede causado por tráfego concorrente, como TCP, nem analisou a possibilidade de proporcionar qualidade de serviço.

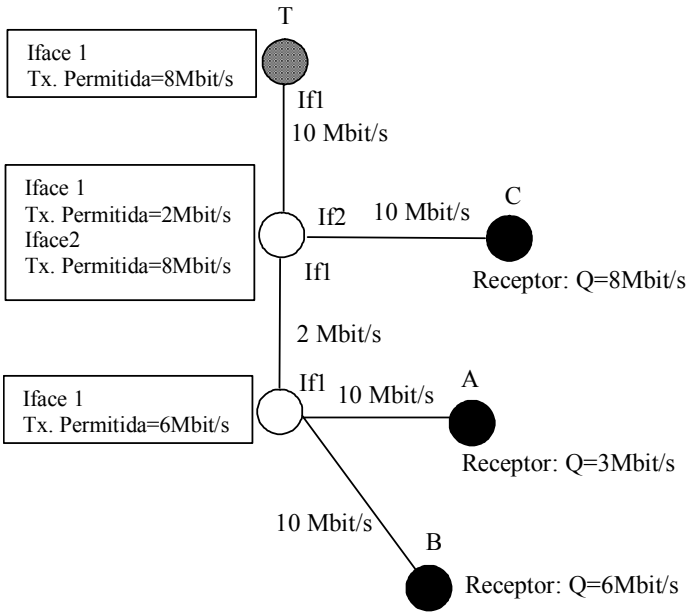


FIGURA 3.11 – Criação das tabelas de disponibilidade

3.7 FLID-DL e WEBRC

O algoritmo FLID-DL (*Fair Layered Increase / Decrease with Dynamic Layering*) foi proposto por Byers [BYE 2000], e é baseado em parte no RLC, buscando uma simplificação no seu algoritmo de controle de congestionamento, bem como uma redução no tempo de *join* e *leave* através da utilização do conceito de camadas dinâmicas, ou DL (*Dynamic Layering*). A transmissão é via multicast em camadas, com controle de congestionamento por taxa, fim-a-fim e orientado a receptor.

No método de camadas dinâmicas, a largura de banda consumida por uma camada (grupo multicast) diminui ao longo do tempo, dessa forma, um receptor deve se inscrever em novas camadas periodicamente a fim de manter sua taxa de recepção. Através do método proposto, a redução na taxa de recepção é obtida simplesmente com a ausência de *joins*, e para manutenção ou aumento de taxa o receptor deve fazer *joins* regularmente. A fim de evitar um excesso no uso de números IP multicast, o transmissor os reutiliza após um período de tempo sem uso.

Além disso, o transmissor utiliza o conceito de "fonte digital" [BYE 98], codificando os dados originais juntamente com dados redundantes de forma que, idealmente, quando um receptor tiver recebido quaisquer pacotes que somem a quantidade dos dados originais, ele esteja apto a reconstruir a informação. O nome "fonte digital" foi escolhido através da analogia com uma fonte de água, onde uma quantidade de água composta por quaisquer gotas da fonte está apta a saciar a sede de uma pessoa.

Para aproximar o conceito de "fonte digital", Byers utilizou códigos conhecidos

como “Tornado” [BYE 98]. Com o código “Tornado A”, o *overhead* médio na recepção de pacotes foi de 5,48%, e o máximo foi de 8,5%. Como o código “Tornado B”, o *overhead* médio foi de 3,06% e o máximo foi de 5,5%. Além disso, ambos códigos possuem tempos de codificação e decodificação bastante rápidos em relação ao código “*Reed-Solomon*”. Por exemplo, num arquivo de 1 Mbytes, o código “Tornado A” leva 0,26 s de codificação, enquanto o “*Reed Solomon Cauchy*” leva 93 s.

3.7.1 Funcionamento do FLID

O protocolo FLID tem por objetivo permitir aos receptores um aumento ou diminuição na sua taxa de transmissão de forma que a vazão média seja similar ao TCP nas mesmas condições. No FLID não existem ACKs, ou seja, os receptores devem adaptar-se de acordo com as características informadas pelo protocolo, permitindo escalabilidade ao sistema.

A transmissão é dividida em l camadas cumulativas, ou seja, cada camada adiciona qualidade à anterior, conforme definido no item 2.3 (Ambientes heterogêneos e codificação multi-taxas), e o receptor deve se inscrever em tantas camadas quanto a rede permitir. A taxa de cada camada é baseada em um determinado esquema, como, por exemplo, os seguintes:

- **Esquema igualitário**: as taxas de todas camadas são iguais, como, por exemplo, 24 kbit/s;
- **Esquema Exponencial**: a camada superior tem o dobro da taxa da anterior, como, por exemplo, 24 kbit/s, 48 kbit/s, 96 kbit/s, etc;
- **Esquema Multiplicativo**: dado uma constante c maior que 1, a taxa de cada camada i é proporcional a c^i. Considerando R_0 a taxa de transmissão inicial, a seqüência de taxas determinada pelo algoritmo foi $R_0(1, c\text{-}1, c^2\text{-}c, c^3\text{-}c^2, c^4\text{-}c^3, \text{etc})$.

No FLID, o transmissor divide o tempo em fatias de T segundos cada, e todo pacote é marcado com o número da fatia de tempo. Do ponto de vista do receptor, uma nova fatia de tempo começa quando o primeiro pacote da nova fatia de tempo chega.

Se durante uma fatia de tempo o receptor detectar perda de pacotes, ele deve efetuar *leave* na camada superior ao final da fatia de tempo. O receptor desconsidera todas perdas de pacotes subseqüentes, ou seja, considera que a fatia de tempo possui apenas dois estados: com ou sem perdas.

O receptor pode tentar subir uma camada sempre no início de uma nova fatia de tempo, dependendo de um *sinal de incremento* enviado pelo transmissor nos pacotes. Caso o receptor esteja inscrito na camada i e o sinal de incremento seja maior que i, então o receptor tenta subir uma camada. Caso o sinal de incremento seja -1, nenhum receptor deve tentar subir camadas.

O algoritmo utiliza um conjunto de probabilidades de incremento de camada, de forma que a probabilidade de incremento das camadas inferiores seja maior do que para as camadas superiores, buscando uma eqüidade de tráfego com o TCP e entre tráfego do próprio algoritmo.

O método utilizado pelo receptor para inferir a banda é baseado num cálculo equivalente para a fórmula de vazão do TCP, semelhante à equação descrita no item 2.6

(Funcionamento do TCP) e em [PAD 98], porém, no FLID-DL não existe informação de RTT, sendo fixo em 100ms. A fórmula utilizada é dada a seguir, e o significado das variáveis foi definido no item 2.6.

$$B(p) \approx \frac{1}{RTT\sqrt{p}\left(\sqrt{\frac{2}{3}} + 6\sqrt{\frac{3}{2}}p(1+32p^2)\right)}$$

Em experimentos simulados com tráfego TCP concorrente, foi utilizada a topologia mostrada na figura 3.12, e o simulador escolhido foi o NS2. Nesse caso, *n* fluxos TCP compartilhavam tráfego com *n* fluxos FLID-DL através de um gargalo de 0,5*n* Mbit/s. O número de fluxos variou de 1 a 15. Os roteadores utilizaram fila *droptail* com tamanho igual a 7*n* pacotes e também 25*n* pacotes. Como forma de comparação, foi computada a vazão para cada protocolo. O RTT do protocolo foi mantido fixo em 100ms e o tempo T de cada fatia de tempo foi fixo em 0,5 s.

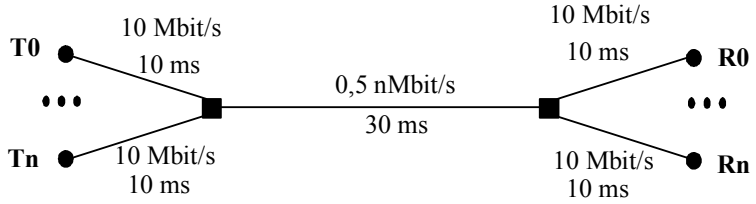


FIGURA 3.12 – Topologia utilizada para simulação do FLID com tráfego TCP

Com tamanho de fila 7*n*, a vazão foi equivalente para ambos protocolos. Com tamanho de fila 25*n*, o FLID-DL obteve vazão de 4 a 8 vezes maior do que o TCP, e o motivo disso foi que o RTT do FLID-DL é fixo e menor que o equivalente no TCP, fazendo com que o FLID utilize mais banda que o tráfego TCP concorrente. Byers comentou que o algoritmo não gera tráfego eqüitativo com o TCP em ambientes onde os atrasos na fila dos roteadores são variáveis e significativos em relação à latência fim-a-fim dos pacotes.

Uma evolução do FLID é o protocolo WEBRC (*Wave and Equation Based Rate Control*), proposto por Luby [LUB 2002], utilizando transmissão multicast em camadas, com controle de congestionamento baseado na equação do TCP, fim-a-fim e orientado a receptor. O WEBRC utiliza um conceito de camadas dinâmicas similar ao FLID, onde a taxa de cada camada diminui gradualmente com o tempo, forçando o receptor a se inscrever em novas camadas para manter a mesma taxa de recepção.

O cálculo do RTT é efetuado a partir da diferença de tempo entre o sinal de *join* e a respectiva chegada dos pacotes, com uma compensação de acordo com as perdas experienciadas pelo receptor. O RTT obtido é chamado MRTT (*Multicast* RTT)

A partir do RTT e das perdas, a equação do TCP é utilizada para determinar a taxa que o receptor pode utilizar, e o mesmo se inscreve em tantas camadas quanto possível. O WEBRC foi criado para ser "razoavelmente justo com o TCP", onde "razoavelmente justo" significa um fluxo com uma taxa de recepção geralmente dentro de um fator de 2 em relação a um fluxo TCP concorrente.

Como os receptores WEBRC não necessitam comunicação com o transmissor, o protocolo torna-se massivamente escalável, permitindo sessões com milhares de

receptores simultâneos.

3.8 LTS

O LTS (*Layered Transmission Scheme*) foi proposto por Turletti [TUR 97], e utiliza transmissão multicast em camadas, com controle de congestionamento por taxa, fim-a-fim e orientado a receptor, tal qual o RLM.

A principal diferença em relação ao RLM é o controle de congestionamento utilizado no LTS, que substitui os experimentos de *join* por uma estimativa de largura de banda equivalente à que seria utilizada por uma conexão TCP nas mesmas condições. Para efetuar essa estimativa, é utilizada a fórmula dada em [MAH 97] e vista no item 2.6.

$$Beq = 1{,}22 * \frac{MTU}{RTT * \sqrt{Loss}}$$

Para calcular a banda equivalente da conexão, cada receptor executa os seguintes passos:

- **Passo 0**: efetua *join* na camada base;
- **Passo 1**: mede MTU, que pode ser configurado para o mínimo valor aceitável do TCP, que é de 576 bytes, ou ser determinado utilizando um algoritmo de descoberta de MTU (RFC 1191 [MOG 90]);
- **Passo 2**: mede *Loss*, através do campo de seqüência do RTP. Quando a seqüência não estiver correta, incrementa contador de perdas;
- **Passo 3**: estima RTT: visto a seguir;
- **Passo 4**: Computa *Beq* e efetua *join* em tantas camadas quanto a banda permitir.

A estimativa do RTT é a parte mais complexa deste protocolo, visto que não existe retorno dos receptores para o transmissor (para permitir escalabilidade ao protocolo). Turletti sugere três formas para estimar o RTT, descritas a seguir.

A primeira forma de estimar o RTT é utilizar duas vezes o atraso da origem ao destino, e supor que o enlace seja simétrico e que os relógios do transmissor e receptor estejam sincronizados (o que nem sempre é verdade). Para saber o atraso da origem ao destino utiliza-se o campo *timestamp* do RTP, conforme descrição em [ROE 2003g].

A segunda forma é aumentar o protocolo RTCP (descrição em [ROE 2003g]), incluindo as mensagens de *rtt-request* e *rtt-reply*. De tempos em tempos o receptor envia uma mensagem de *rtt-request*, que é imediatamente respondida pelo transmissor com o *rtt-reply*. A freqüência que o receptor envia essas mensagens deve ser proporcional ao número de participantes da sessão (tal qual o RTCP) e não deve haver sincronismo entre as mensagens dos vários receptores, a fim de evitar implosão de mensagens e permitir escalabilidade. O transmissor inclui no cabeçalho da resposta o tempo que demorou em enviar o *rtt-reply*, e dessa forma o receptor calcula o RTT, independente do relógio do transmissor. Os problemas reportados por esse método são de que ele não escala para muitos receptores, ou seja, com 200 receptores, o tempo de estimativa de RTT é de 13 segundos.

A terceira forma é eliminando completamente o parâmetro RTT da equação de banda equivalente, utilizando ao invés dele o atraso entre origem e destino e o *jitter* entre pacotes medido no receptor. Dessa forma, o RTT seria medido esporadicamente através do método anterior, e as variações de RTT seriam obtidas através da variação do *jitter*, vistas através do campo *timestamp* dos pacotes RTP.

Após efetuar um *join* ou um *leave*, o receptor espera um período de tempo para verificar o impacto desta ação na rede. Esse período é normalmente de um RTT.

Para testar o mecanismo proposto, foi implementado um CODEC de áudio com 3 camadas, onde a primeira tem duas vezes a velocidade das outras duas (incluindo redundância para deixar o sistema mais robusto contra perda de pacotes). Os experimentos foram realizados no MBONE, com o transmissor no MIT (*Massachussetts Institute of Technology*) e receptores no MIT, INRIA, UCL (*University College London*) e UTS (*University of Technology in Australia*).

Os resultados mostraram uma certa instabilidade nos receptores, com perdas na ordem de 5%. Maiores detalhes e resultados gráficos podem ser obtidos em [TUR 97].

3.9 TFMCC e TFRC

O protocolo TFMCC (*TCP-Friendly Multicast Congestion Control*) foi proposto por Widmer [WID 2001b], utilizando transmissão multicast com controle de congestionamento por taxa, fim-a-fim, e adaptação no lado do transmissor, que recebe continuamente retorno por parte dos receptores indicando a taxa eqüitativa que cada um deles calculou. Com o valor da banda eqüitativa de todos os receptores, o transmissor ajusta sua taxa de transmissão para o receptor mais lento de todos.

A equação utilizada pelos receptores é semelhante à descrita em [PAD 98] e vista no item 2.6. A fatia média de banda utilizada por um fluxo TCP é dada por:

$$B(p) \approx \frac{MTU}{RTT\sqrt{\frac{2p}{3}} + \left(12\sqrt{\frac{3p}{8}}\right)p\left(1+32p^2\right)}$$

Um desafio do protocolo foi evitar que os pacotes de retorno dos receptores causassem implosão de *feedback* no transmissor. Para tanto, os receptores não enviam retorno caso sua taxa calculada seja maior do que a taxa sendo transmitida. Isso causa um problema, que é: como o transmissor aumenta sua taxa?

A solução utilizada por Widmer foi a criação do conceito de receptor CLR (*Current Limiting Receiver*), que é o receptor atualmente com a menor taxa entre todos. O único que pode enviar mensagens de retorno com taxa superior à atual é o CLR, permitindo ao transmissor um incremento na banda. O CLR é móvel, e muda caso um receptor mais lento entre no grupo multicast ou o CLR atual saia do grupo multicast.

Os receptores calculam sua taxa de perdas (necessária para utilização na equação) através da seguinte fórmula:

$$p = \frac{1}{\max(li(k), li(k-1))}$$

Na fórmula, *p* é a taxa de perdas a ser utilizada na equação do TCP. A variável *li* representa uma média de perdas nos últimos *k* RTTs, sendo que as medidas mais recentes têm um peso maior no cálculo da média. Um valor sugerido foi a análise dos últimos 8 RTTs, e o peso ficaria da seguinte forma: {5, 5, 5, 5, 4, 3, 2, 1}.

Os receptores também devem calcular o RTT com o transmissor, pois essa variável é utilizada na fórmula. Para tanto, inicialmente o protocolo assume receptores com o sinal de *clock* sincronizado com o transmissor, o que é conseguido através de receptores GPS (*Global Positioning System*) ou, com menor precisão, através do protocolo NTP (que possui aproximadamente 20 ms de erro). Esta é uma estimativa inicial, e caso o receptor não possua mecanismos de sincronização, ele parte de um valor de 500ms.

Para cálculo do RTT em regime permanente, o receptor utiliza dois métodos cumulativos:

1. De tempos em tempos (não muito freqüente para evitar implosão de *feedback*), o receptor envia ao transmissor um pacote de relatório de banda eqüitativa. Nesse pacote consta o *timestamp* do receptor (*tr*). O transmissor responde a esse relatório incluindo, no cabeçalho dos próximos pacotes de dados, seu *timestamp* (*tt*) e a identificação do receptor que efetuou a solicitação. Com essa informação, o receptor está apto a calcular o RTT de forma precisa no momento que chega o pacote (*tnow*). O cálculo é: *RTT = tnow - tr*.
2. Além da informação de RTT, o receptor pode calcular o atraso entre o transmissor e o receptor (juntamente com a diferença de relógio). Essa informação é calculada como: *Tempot_r = tnow - tt*. Essa informação é utilizada para calcular a variação do atraso entre transmissor e receptor, e é possível de ser calculada a cada pacote recebido, pois todos os pacotes incluem o *timestamp* do transmissor.

Para evitar excesso de tráfego de *feedback* por parte dos receptores, convencionou-se que somente o CLR pode enviar solicitações com maior freqüência, enquanto os outros receptores calculam com precisão seu RTT a intervalos maiores. Os receptores utilizam um filtro levando em consideração os últimos cálculos de RTT, suavizando a influência da última medição. Como o CLR calcula o RTT com maior freqüência, sua medição atual contribui com 95% do novo RTT. Para os outros receptores, a medição atual vale 50% do novo RTT.

Para calcular o intervalo de tempo de envio de mensagens ao transmissor, os receptores utilizam um temporizador que expira após *t* segundos e é dado por:

$$t = \max(T(1 + \log_N x), 0)$$

A variável *T* é um limite superior de tempo antes de enviar o pacote ao transmissor, *N* é o número total estimado de receptores, e *x* é uma variável randômica uniformemente distribuída entre 0 e 1. Maiores detalhes em [WID 2001].

O protocolo possui inicialmente uma fase de *slow-start*, onde o transmissor busca atingir com maior rapidez sua fatia eqüitativa de banda. O mecanismo utilizado limita a taxa de incremento de banda para duas vezes o mínimo reportado pelos receptores a cada nova rodada de *feedbacks*, ajustando essa taxa gradualmente durante o próximo RTT. Esse mecanismo aumenta a taxa de transmissão de forma mais

conservativa do que o equivalente TCP, o que é uma característica desejável em protocolos multicast. Quando qualquer um dos receptores percebe uma perda de pacotes, informa ao transmissor imediatamente, e este encerra a fase de *slow-start*.

Um protocolo semelhante ao TFMCC, porém no domínio unicast, é o TFRC (*TCP-Friendly Rate Control*), descrito por Sally Floyd em [FLO 2000], e por Handley, Floyd, Padhye e Widmer (RFC 3448 [HAN 2003]). O TFRC possui um esquema de controle de congestionamento por taxa onde o transmissor utiliza a equação do TCP para adaptar sua taxa de transmissão. Na verdade o TFMCC é uma extensão ao TFRC para o domínio multicast.

No TFRC, o mecanismo de controle de congestionamento AIMD não é seguido a fim de aumentar a estabilidade. Simulações mostram que são necessários quatro a oito RTTs para diminuir a taxa de transmissão pela metade, entretanto, como a taxa de incremento de banda também é menor (aproximadamente 0,14 pacotes por RTT), ocorre um aumento na estabilidade mantendo a eqüidade com os outros tráfegos.

Os protocolos TFMCC e TFRC foram validados através do simulador NS2.

3.10 MLDA e LDA+

O MLDA (*Multicast Enhanced Loss-delay based Adaptation Algorithm*) foi proposto por Sisalem [SIS 2000], e utiliza transmissão multicast em camadas, com controle de congestionamento por taxa, fim-a-fim, orientado a receptor e transmissor, ou seja, com adaptação tanto do lado dos receptores como do lado do transmissor.

O transmissor é responsável por ajustar seu comportamento de transmissão de acordo com as estimativas relatadas pelos clientes. Periodicamente, o transmissor utiliza essas informações dos clientes e modifica dinamicamente a taxa de codificação de suas camadas, enviando um pacote SR (*Sender Report*) com as novas informações. Cada pacote SR contém o número de camadas que está transmitindo, a taxa e o endereço multicast de cada camada. Como as camadas são dinâmicas e mudam de tempos em tempos, elas não possuem uma taxa fixa, mas sim flutuam dentro de certos limites.

Os receptores coletam informações sobre perdas, atrasos e gargalos na rede, utilizando essa informação para calcular uma largura de banda eqüitativa com tráfegos TCP concorrentes. Para cálculo da banda eqüitativa com tráfegos TCP, Sisalem utiliza o modelo analítico descrito em [PAD 98] e visto no item 2.6, onde a fatia média de banda utilizada por um fluxo TCP é dada pela seguinte equação:

$$B(p) \approx \frac{MTU}{RTT\sqrt{\frac{2bp}{3}} + To\min\left(1{,}3\sqrt{\frac{3bp}{8}}\right)p\left(1+32p^2\right)}$$

Ao receber um pacote SR, o receptor mede suas perdas por um período equivalente a *T*L*, onde T representa o tempo mínimo para que o receptor consiga uma medição válida de perdas, e *L* é o número da camada na qual o receptor se encontra atualmente. Vale observar que o tempo mais longo para as camadas superiores permite que os receptores localizados nas camadas mais baixas tentem subir camada antes. A

figura 3.13 mostra um exemplo: supondo que os receptores R2, R3 e R4 estejam compartilhando o mesmo enlace, cada um deles inscrito na camada 2, 3 e 4, respectivamente. Se o tempo *T* for igual a 5s, eles vão medir perdas por 10s, 15s e 20s. Se o receptor R2 fizer uma tentativa de subir para a camada 3, não vai interferir nas medições dos outros receptores, pois os mesmos já estão recebendo esta camada.

Depois de medir perdas pelo tempo *T*L*, o receptor calcula a banda eqüitativa, e dependendo do resultado, faz *join* na próxima camada, continua como está ou efetua *leave* das camadas mais altas. O *leave* é semelhante ao *join* visto na figura 3.13, ou seja, cada receptor efetua num momento diferente, e somente após o último ter efetuado é que a transmissão do grupo multicast pára efetivamente. Para evitar que o receptor se inscreva em uma taxa superior à existente na rede, Sisalem sugere a utilização de pares de pacotes.

Após o cálculo e adaptação ao número correspondente de camadas, o receptor se comunica com o transmissor, enviando sua banda eqüitativa para que o mesmo possa redimensionar as camadas de forma a melhor atender seus clientes.

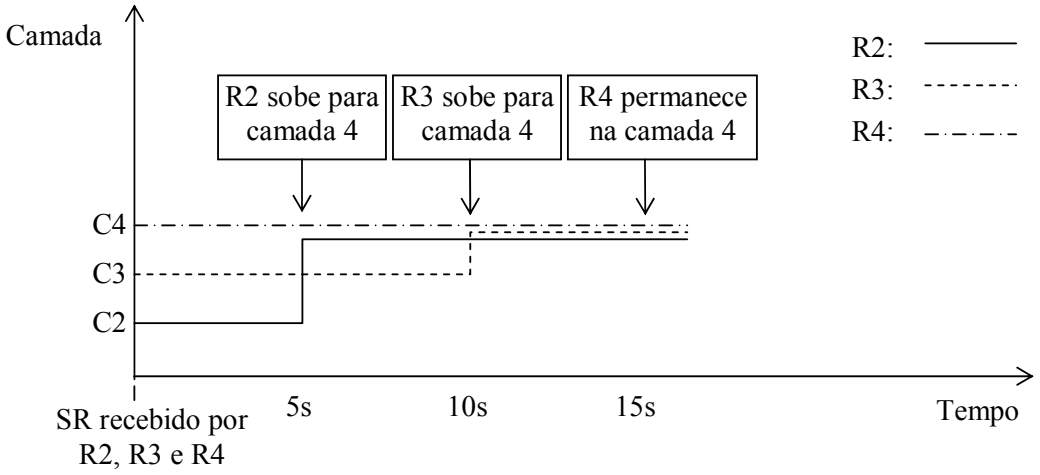


FIGURA 3.13 – Momentos diferentes para receptores adaptarem-se

A fim de permitir escalabilidade ao número de receptores, foi necessário criar um mecanismo denominado "supressão parcial", onde os receptores esperam um tempo denominado T_{wait} antes de enviar um retorno ao transmissor. Caso algum outro receptor envie um retorno durante este tempo, e o retorno possua valores de banda próximos ao que seria enviado, o receptor suprime a sua transmissão.

Os principais parâmetros da equação de cálculo de banda eqüitativa são a estimativa de perdas e o RTT.

Para estimar perdas, o algoritmo utiliza o número de seqüência dos pacotes que chegam no receptor, e as perdas são indicadas por lacunas nestes números. O valor total de perdas a ser utilizado na equação é dado por:

$$p = \frac{p_{C1} * taxa_{C1} + p_{C2} * taxa_{C2} + ... + p_{Cx} * taxa_{Cx}}{\sum_{k=1}^{x} taxa_{Ck}}$$

Onde P_{C1}, P_{C2} são as perdas de cada camada, e $taxa_{C1}$, $taxa_{C2}$, são as taxas de cada camada, conforme enviados nos pacotes de SR do transmissor.

Para estimativa de RTT, os relógios do transmissor e receptor não precisam estar sincronizados. O transmissor envia periodicamente mensagens contendo *timestamps_t* indicando o momento em que a mensagem foi enviada. Cada receptor também envia mensagens com *timestamp_r* ao transmissor, porém, em intervalos muito maiores que os utilizados pelo transmissor, a fim de evitar sobrecarregar a rede e permitir escalabilidade.

Quando o receptor envia a mensagem, o transmissor responde incluindo no pacote de retorno o *timestamp_r* do receptor e o tempo que demorou em enviar a resposta (T_{demora}). Com essa informação, o receptor calcula o RTT fim-a-fim do pacote, que é igual ao tempo de chegada do pacote no receptor menos o tempo de saída do pacote do receptor menos o tempo que demorou a ser retransmitido. $RTT_{ff} = T_{receptor} - timestamp\_r - T_{demora}$.

O receptor só envia a mensagem a intervalos grandes, como mencionado anteriormente. Assim, a fim de manter o sincronismo atualizado, o receptor deve se basear nas mensagens do transmissor, que são enviadas com mais freqüência. Quando o transmissor envia a mensagem, o receptor calcula a assimetria da rede: $A = T_{receptor} - timestamp\_t - RTT_{ff} / 2$.

Supondo que um pacote leve 500ms do transmissor ao receptor, e 100ms do receptor ao transmissor, e o transmissor responde imediatamente ao pacote. Calculando as duas equações acima, chega-se a: $RTT_{ff} = 600ms$, e $A = 200ms$.

Fixando o valor de A até sua nova medida (mensagem do receptor), o RTT_{ff} é estimado novamente a cada mensagem enviada pelo transmissor, e o valor de A é recalculado a cada mensagem do receptor.

A fim de criar uma atenuação sobre a medida obtida levando em conta as medidas anteriores, a equação fica: $RTT_{ff} = RTT_{ff} + 0{,}125 * (RTT_{ant} - RTT_{ff})$.

Um dos pontos negativos do MLDA é a complexidade tanto do protocolo como da aplicação, que deve distribuir os dados nas camadas dinâmicas [WID 2001]. Além disso, com o uso do mecanismo de "supressão parcial" utilizado no MLDA, caso um receptor transmita sua estimativa de banda e outros 10 suprimam a transmissão, pois possuem uma taxa parecida, o transmissor só recebe uma informação, e pode tomar uma decisão equivocada no novo dimensionamento de camadas (baseado em uma maioria de outros retornos de receptores), pois não leva em conta estatisticamente o impacto de cada camada em todos os receptores.

Outra dificuldade para utilizar o MLDA na prática seria obter um codificador em camadas flexível a ponto de se poder escolher uma taxa de transmissão específica para cada camada.

Outro trabalho desenvolvido por Sisalem foi o LDA+ (*Enhanced Loss-Delay based Adaptation Algorithm*) [SIS 2000b], porém, com adaptação em unicast. Nesse caso, o mecanismo de realimentação com o transmissor foi baseado totalmente em mensagens RTCP. O cálculo de banda eqüitativa utilizado no algoritmo também é

baseado na equação do TCP, tal qual o MLDA, e o RTT é medido através de um *timestamp* inserido nas mensagens RTCP. Nesse *timestamp* o receptor inclui o tempo de transmissão do último *sender report* (*Tsr*, que foi enviado pelo transmissor e copiado pelo receptor no pacote RTCP), e o intervalo de tempo entre a recepção deste *sender report* e o envio do pacote RTCP (*Tinterv*). Com essa informação mais o tempo de chegada do pacote RTCP (*T*), o transmissor calcula o RTT fazendo: *RTT = T – Tinterv – Tsr*. Dessa forma, não é necessário sincronismo entre os relógios do transmissor e receptor.

Para cálculo da banda máxima entre transmissor e receptor, o algoritmo LDA+ utiliza o mecanismo de trens de pacotes, e o ajuste de taxa é baseado na filosofia AIMD, com formulações definidas em [SIS 2000b].

Segundo [WID 2001], o intervalo de tempo entre mensagens RTCP é muito grande, na ordem de segundos, fazendo o algoritmo LDA+ lento para reagir a mudanças nas condições da rede. Além disso, a menor taxa de perdas que o RTCP pode reportar é limitada, assim, em ambientes com baixas taxas de perdas, o RTCP reporta zero e o algoritmo LDA+ utiliza mais do que sua fatia eqüitativa de banda.

3.11 HALM: Hybrid Adaptation Layered Multicast

O HALM (*Hybrid Adaptation Layered Multicast*) [LIU 2002] foi proposto por Jiangchuan Liu, utilizando transmissão multicast em camadas, com controle de congestionamento por taxa, fim-a-fim, orientado a receptor e transmissor, da mesma forma que o MLDA.

Ao contrário do MLDA, onde o transmissor ajusta seu comportamento dentro de um valor máximo e mínimo para cada camada, o HALM introduz um algoritmo de distribuição de banda de acordo com a possibilidade dos receptores. Esse algoritmo determina de forma ótima a melhor distribuição de camadas possível para o conjunto atual de receptores. Com essa informação e um codificador em camadas flexível, o transmissor se adapta e passa a transmitir dentro dessas novas taxas. De tempos em tempos, o transmissor envia um novo pacote *sender report*, que contém, entre outras informações, o novo vetor de distribuição de banda.

Liu sugeriu a utilização de um codificador baseado em *bitplanes*, como o visto em [LI 2001], e explicado no capítulo 8 (Codificação de Vídeo Multi-Taxas), por sua flexibilidade em relação à obtenção de taxas específicas.

Liu também definiu o conceito de *fairness index* (FI) como a relação entre a taxa que o receptor está utilizando (TU) frente à taxa eqüitativa para este receptor (TE), ou seja, FI = TU / TE. A taxa utilizada TU é a soma das taxas de todas as camadas em que o receptor está inscrito, e a taxa eqüitativa é obtida via equação do TCP. Esse índice foi utilizado também para medir o índice de satisfação do receptor, e se considerou que, quanto mais próximo do valor “um” estiver FI, mais satisfeito estará o receptor.

Um problema mencionado no artigo é o *overhead* de processamento para cálculo do novo vetor de camadas. Com 500 receptores e 5 camadas, o tempo de processamento num Pentium III 450 MHz é de 32 ms. Já para 1000, 2000 e 3000 receptores, o tempo de processamento passa para 126 ms, 514 ms e 1.154 ms, respectivamente. Isso sugere

que o algoritmo não seja escalável para um número muito grande de receptores.

Para cálculo da taxa, Liu utilizou a equação do TCP, descrito em [PAD 98] e visto no item 2.6. O cálculo do RTT, necessário na equação, é o mesmo utilizado pelo MLDA, visto no item 3.10. O RTT é uma composição de uma medição precisa porém bastante espaçada no tempo (laço fechado) e ajustes em todos os pacotes (laço aberto).

Outra característica utilizada no HALM é a coordenação local à sub-rede, visando acelerar a convergência para os receptores e redução de *overhead* do protocolo. Assim, quando um receptor obtém uma estimativa de RTT precisa (laço fechado), ele repassa essa informação a todos receptores localizados na mesma sub-rede (TTL = 1). Além disso, quando um novo receptor entra na sessão, o receptor que atualizou seu RTT mais recentemente deve informar a ele o RTT atual, taxa de perdas e camada que ele pode se inscrever.

O protocolo HALM foi validado através do simulador NS2, utilizando-se a topologia ilustrada na figura 3.14, que mostra também as velocidades utilizadas nos enlaces. O protocolo de roteamento foi o DVMRP, todas as filas foram FIFO com política de descarte *droptail*, o tamanho de todos pacotes foi de 500 bytes e a duração das simulações foi de 1000 segundos.

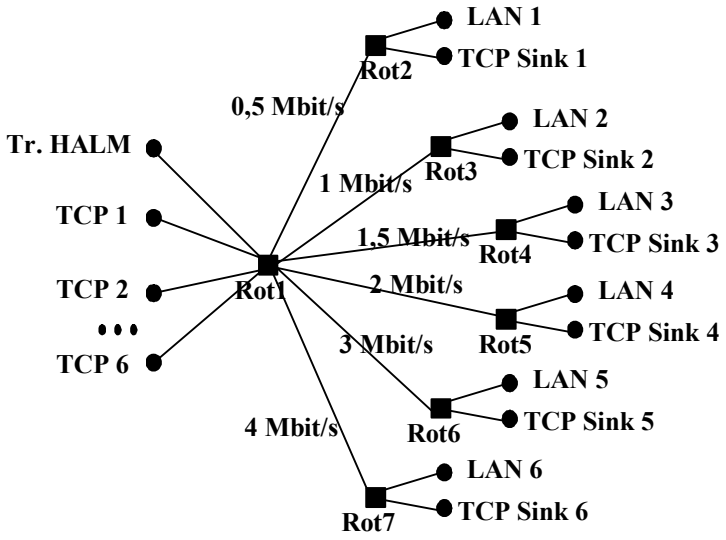


FIGURA 3.14 – Topologia utilizada na simulação do protocolo HALM

3.12 TCP Vegas

O TCP Vegas não é um protocolo para transmissões *multicast* em camadas, como a maioria dos trabalhos relacionados vistos neste capítulo, entretanto, está relacionado devido ao seu mecanismo de controle de congestionamento, que é baseado na variação do atraso na chegada de pacotes.

Segundo Brakmo [BRA 94], o TCP Vegas atinge uma vazão entre 40% e 70% a mais do que o TCP Reno, com 20% a 50% do número de perdas. Para tanto, foram criadas três novas técnicas no protocolo: a primeira consiste de um novo mecanismo de retransmissão de pacotes, a segunda é um novo mecanismo para evitar

congestionamento, onde o mesmo é previsto antecipadamente e a taxa ajustada de acordo. A terceira técnica modifica o mecanismo *Slow-Start* a fim de evitar perda de pacotes enquanto busca atingir a banda eqüitativa para o fluxo. A seguir essas técnicas são explicadas com detalhes.

O novo mecanismo de retransmissão proposto no TCP Vegas procura diminuir o tempo de espera na retransmissão de pacotes supostamente perdidos, que no TCP Reno é após 3 ACKs duplicados ou *timeout*. No TCP Vegas, o *timestamp* do momento que cada pacote é transmitido é armazenado, e o mecanismo de retransmissão funciona da seguinte forma:

- Quando um ACK duplicado é recebido, caso a diferença de tempo entre o pacote que foi enviado (cujo ACK deveria ter chegado) e o tempo atual seja maior que o *timeout*, então o pacote é retransmitido, conforme mostra a figura 3.15. Isso evita ter que esperar 3 ACKs duplicados para retransmissão, como acontece no TCP Reno.
- Quando um ACK não duplicado é recebido, caso seja o primeiro ou o segundo após uma retransmissão, Vegas confere o tempo para ver se já deveria ter chegado o ACK do próximo pacote. Caso negativo, o pacote é retransmitido, conforme figura 3.15. O objetivo é evitar que pacotes que foram perdidos intercaladamente tenham que esperar por 3 ACKs novamente.

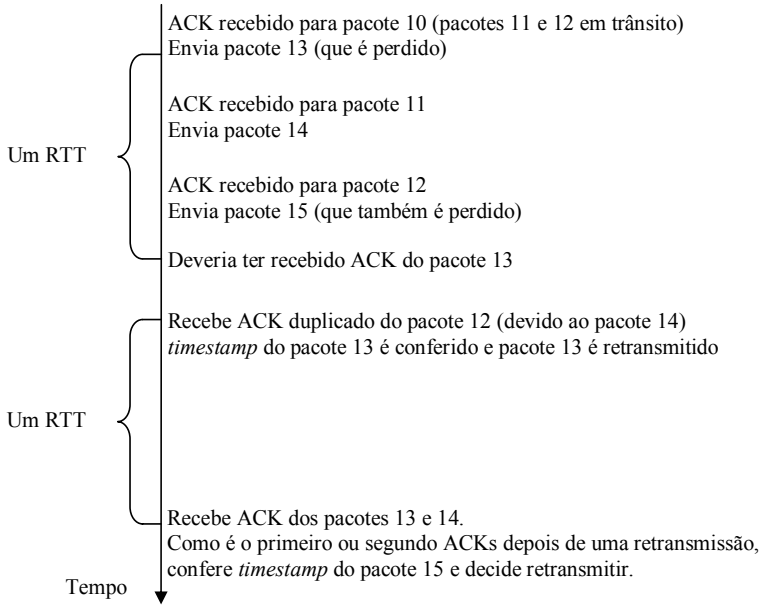


FIGURA 3.15 – Mecanismo de retransmissão do TCP Vegas

Em outras palavras, Vegas se baseia no *timeout* ao invés de ACKs duplicados para decidir se deve retransmitir determinado pacote, agilizando o mecanismo de retransmissão. Apesar disso, ainda mantém o mecanismo do TCP Reno, e caso três ACKs duplicados cheguem antes de ter passado o tempo do *timeout* Vegas, acontece retransmissão da mesma forma que no TCP Reno.

O mecanismo para evitar congestionamento, conhecido no TCP Reno como "*congestion avoidance*" também foi modificado no Vegas. O TCP Reno é reativo, necessitando a geração de perdas para descobrir a banda disponível na rede. No TCP

Vegas, o mecanismo é pró-ativo, baseado no aumento no RTT comparado com o aumento na taxa de transmissão.

A idéia explorada pelo TCP Vegas é que o número de bytes em trânsito é diretamente proporcional à vazão esperada, portanto, conforme o tamanho da janela aumenta (causando um aumento no número de bytes em trânsito), a vazão da sessão também deve aumentar proporcionalmente.

Primeiramente, Vegas configura a variável *BaseRTT* ao valor mínimo de todos RTTs medidos até o momento (normalmente é o RTT do primeiro pacote enviado pela conexão). Caso não exista congestionamento, a vazão esperada é dada por: V_E = *Windowsize / BaseRTT*. *Windowsize* é o tamanho atual da janela de congestionamento, que é assumido como o número de bytes em trânsito.

Em segundo lugar, Vegas calcula a taxa de transmissão real (V_R) por RTT, dividindo o número de bytes transmitidos pelo RTT obtido (caso o RTT seja menor que o mínimo, a variável *BaseRTT* é reconfigurada).

Finalmente, Vegas compara a vazão esperada com a vazão real, configurando a variável *diff* da seguinte forma: $diff = V_E - V_R$. Para ajuste de tamanho da janela, são definidos dois *thresholds*: $\alpha < \beta$, conforme ilustrado na figura 3.16: se $diff < \alpha$, Vegas aumenta linearmente a janela de congestionamento durante o próximo RTT, pois significa que a sessão está utilizando poucos recursos da rede; Se $diff > \beta$, Vegas diminui linearmente a janela de congestionamento durante o próximo RTT, significando que a sessão está utilizando muitos recursos da rede; se $\alpha < diff < \beta$, a janela de congestionamento não é modificada.

Na prática, os valores de α e β são configurados como posições ocupadas nas filas dos roteadores intermediários, e não como taxa. Por exemplo, caso o RTT seja 100ms e o tamanho do pacote seja 1 kbyte, se $\alpha = 30$ kbytes/s e $\beta = 60$ kbytes/s, pode-se dizer que a sessão não deve ocupar menos do que 3 posições nem mais do que 6 posições nas filas dos roteadores [BRA 94].

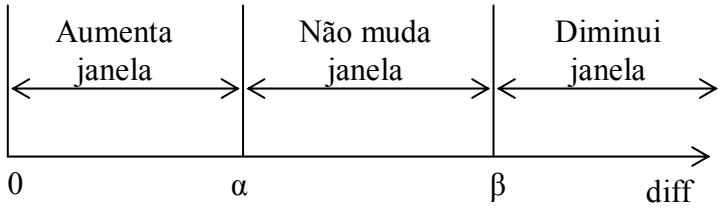


FIGURA 3.16 – Ajuste de janela no TCP Vegas

A terceira técnica modificada pelo TCP Vegas foi o *slow-start*, considerado um gerador de perdas muito grande, pois o transmissor duplica sua taxa de transmissão cada RTT, provocando um crescimento exponencial. No TCP Vegas, o crescimento exponencial continua, mas somente cada dois ACKs recebidos. Além disso, o controle de congestionamento está embutido no protocolo, e não é necessário gerar perdas para descobrir sua janela de congestionamento. Quando a diferença entre a vazão esperada e a vazão obtida ficar acima de um determinado *threshold* γ, Vegas muda seu controle de congestionamento para *congestion avoidance*.

Em [BRA 94], diversas simulações foram efetuadas e comparadas com o TCP Reno para duas versões do Vegas: Vegas-1,3 (com $\alpha = 1$ e $\beta = 3$) e Vegas-2,4 (com $\alpha =$

2 e $\beta = 4$). As duas versões mostraram resultados bastante semelhantes. Vegas-1,3 obteve vazão 53% melhor que um equivalente TCP Reno com somente 49% das perdas. Várias simulações mostraram resultados semelhantes.

Entretanto, o TCP Reno (ou variantes) domina atualmente mais de 90% do tráfego da Internet, e é muito mais agressivo que o TCP Vegas. Quando ambos concorrem pelo mesmo enlace, o TCP Reno obtém uma largura de banda superior, ou seja, o TCP-Vegas não provê eqüidade de tráfego com o TCP Reno, apesar de mostrar um desempenho superior quando roda sem tráfego concorrente [SAN 2002].

Outro problema observado por [SAN 2002] e [HEN 2000] é que não existe eqüidade com os próprios tráfegos. A conexão que começa mais tarde vai observar um RTT mais alto, provocando uma janela de congestionamento mais baixa e obtendo menos tráfego que a que começou mais cedo.

Hengartner [HEN 2000] isolou cada modificação no algoritmo TCP Vegas em relação ao Reno, buscando identificar os fatores que mais influenciam na sua melhoria de desempenho. Para tanto, a implementação foi modificada de forma que cada porção do algoritmo poderia ser ligada ou desligada independente das outras. Foi detectado que algumas porções do algoritmo não tinham sido relatadas anteriormente, como por exemplo a redução na janela de congestionamento por um fator de 25% ao invés dos 50% do TCP Reno, ou o tamanho inicial da janela de congestionamento configurado para dois segmentos ao invés de um, como é comumente utilizado no TCP Reno.

As principais conclusões obtidas em [HEN 2000] foram que o principal fator na melhora de desempenho do TCP Vegas é a modificação no *slow-start*, pois não gera múltiplas perdas nas filas e, conseqüentemente, não gera *timeouts* na rede (que são um dos maiores responsáveis pela queda de desempenho no TCP Reno). Outro grande fator de melhoria de desempenho é a diminuição na janela de congestionamento por um fator de 25% ao invés dos 50% do TCP Reno.

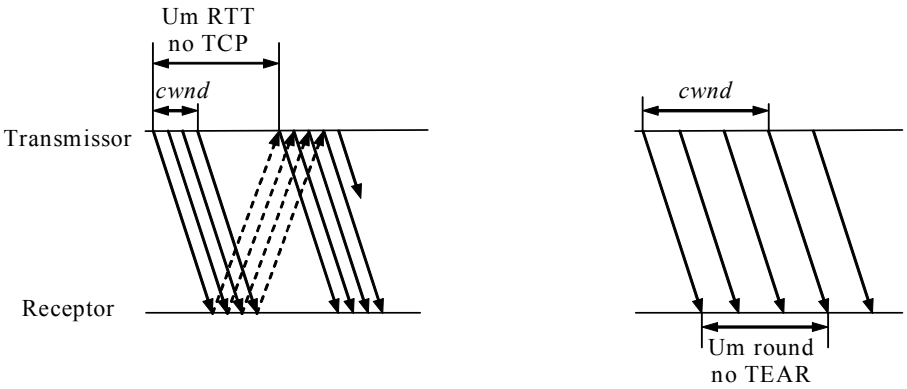
3.13 TEAR

O protocolo TEAR (*TCP Emulation At Receivers*) foi proposto por Injong Rhee [RHE 2000], e utiliza transmissão unicast com controle de congestionamento por janela, fim-a-fim, e mecanismos de controle de fluxo localizados nos receptores. O controle de congestionamento por janela no receptor é utilizado para determinação da melhor taxa a ser utilizada pelo transmissor, assim, a comunicação entre receptor e transmissor é bem reduzida, sendo utilizada basicamente para informar a taxa que o transmissor deve enviar o sinal.

Os objetivos principais do protocolo são de manter a eqüidade de tráfego com tráfego TCP concorrente (*TCP-friendliness*), obtendo uma maior estabilidade e permitindo escalabilidade (neste caso, a partir de uma versão multicast do TEAR a ser desenvolvida).

O TEAR utiliza um mecanismo de janelas imitando o do TCP a fim de determinar sua taxa eqüitativa, porém, com adaptações onde necessário. Por exemplo, o TCP mantém uma variável chamada *cwnd* que indica o número de pacotes em trânsito do transmissor ao receptor, e essa variável é atualizada conforme retornam os pacotes de

ACK, como mostra a figura 3.17a. No TEAR, como não existem ACKs, o conceito foi modificado para a utilização de "*rounds*". Neste caso, o transmissor estima um RTT e divide o valor de *cwnd* pela estimativa do RTT, intercalando os pacotes transmitidos ao longo deste tempo, como mostra o a figura 3.17b.



a) RTT no TCP b) *round* no TEAR

FIGURA 3.17 – Definição de "*round*" no TEAR

O TEAR possui estados equivalentes para as diversas fases do TCP, e o receptor busca mecanismos para descobrir a taxa equivalente em uma conexão TCP nas fases de *slow-start*, *congestion avoidance*, *fast recovery* e *timeout*. Durante a fase de *slow-start*, o receptor incrementa *cwnd* em um a cada *round*. Durante a fase de *congestion avoidance*, a janela é incrementada pelo valor *1/cwnd*. Quando um pacote é perdido (detectado através do número de seqüência), o receptor entra num estado chamado *GAP*, onde procura descobrir se deve efetuar *fast recovery* ou *timeout*.

Caso entre no estado de *fast recovery*, o receptor aguarda um RTT, reduzindo a janela de congestionamento uma única vez mesmo que tenha acontecido mais do que um pacote perdido. A janela é reduzida pela metade, da mesma forma que é feito no TCP. Para cálculo do RTT, o receptor envia um pacote informando perda ao transmissor, e espera o ACK do transmissor.

Se acontecer *timeout*, o receptor entra no modo *slow-start* novamente, configurando a variável *ssthresh* para *cwnd / 2* (ou 1, caso *cwnd = 1*).

Os procedimentos acima geram uma taxa de transmissão similar ao TCP, ou seja, com grandes oscilações. Para obter uma maior estabilidade, TEAR utiliza a taxa média durante determinado período. Este período é denominado "época", sendo equivalente ao tempo de uma onda "dente de serra" de um tráfego TCP equivalente. Assim, uma época começa quando o receptor entra no modo *slow-start* ou *congestion avoidance*, e termina quando acontecem perdas.

Para evitar mudanças bruscas de taxa, TEAR utiliza um filtro que considera as épocas anteriores. Atualmente, são consideradas as últimas 8 épocas, e elas tem peso variável, diminuindo ao longo do tempo. Os pesos utilizados no TEAR para as últimas 8 épocas foram, respectivamente: 1/6, 1/6, 1/6, 1/6, 2/15, 1/10, 1/15 e 1/30.

As simulações foram feitas através do simulador NS2, e procuraram verificar a eqüidade do tráfego com sessões TCP-Sack concorrentes e a estabilidade do algoritmo. A topologia utilizada nas simulações é mostrada na figura 3.18. As principais variáveis

foram as filas nos roteadores (*droptail* e RED), a largura de banda na conexão central (10Mbit/s, 5Mbit/s, 2,5Mbit/s e 128 kbit/s), e o número de fluxos TCP concorrentes (1, 2, 4, 8 e 16).

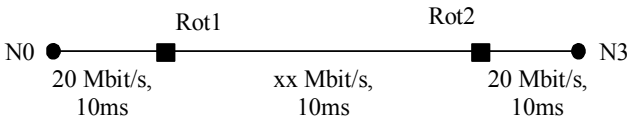


FIGURA 3.18 – Topologia utilizada na simulação do protocolo TEAR

O resultado das simulações mostrou que o algoritmo é bem mais estável que o TCP, principalmente pela utilização das épocas, descrito acima.

3.14 RAP

O protocolo RAP (*Rate Adaptation Protocol*) [REJ 99] é um mecanismo unicast fim a fim com controle de congestionamento baseado em taxa e janela. Seu objetivo é permitir a transmissão e recepção de sinais em tempo real, como vídeo, de forma a gerar tráfego eqüitativo com o TCP. O mecanismo utilizado para ajustar a qualidade no receptor é através de transmissão do vídeo em camadas.

O controle de congestionamento utilizado no RAP é similar ao TCP: a perda de pacotes é interpretada como sinal de congestionamento e são utilizados mecanismos similares ao TCP para aumento ou diminuição de banda (filosofia AIMD).

No protocolo RAP, cada pacote possui um número de seqüência que é retransmitido pelo receptor nos ACKs. Isso permite a detecção de perdas e o cálculo do RTT. O ajuste de taxa no transmissor é feito a cada RTT, buscando imitar o comportamento de um fluxo TCP nas mesmas condições. Os pacotes são espaçados igualmente dentro do período de tempo que têm para serem enviados, provocando uma taxa de transmissão mais suave em relação ao TCP.

O mecanismo de adaptação na qualidade do vídeo é baseado em camadas, e está resumido a seguir e detalhado em [REJ 99b]. O objetivo é evitar variações na qualidade da imagem, buscando estabilidade ao sistema, pois isso foi considerado menos prejudicial ao usuário.

A figura 3.19 mostra o mecanismo em ação. Inicialmente o transmissor começa a enviar pacotes da camada 1 (representado pela linha pontilhada do gráfico de cima), que são armazenados num *buffer* no lado do receptor (cujos momentos de chegada e utilização estão representados no gráfico de baixo). Quando a taxa de transmissão se iguala à taxa de consumo, o receptor começa a apresentar a camada 1 para o usuário, e continua armazenando pacotes num *buffer*, pois a taxa de transmissão é maior que a taxa de consumo. Em determinado momento, o transmissor começa a transmitir pacotes da camada 2 ao receptor, que vai armazenando esses pacotes num *buffer*. Quando a taxa de transmissão equivale à taxa de consumo da camada 2, o receptor começa a utilizar os pacotes armazenados, mostrando-os ao usuário e aumentando sua qualidade. Quando ocorrem perdas, a taxa de transmissão diminui de acordo com o TCP, porém, o receptor se mantém estável enquanto possuir pacotes armazenados no *buffer*.

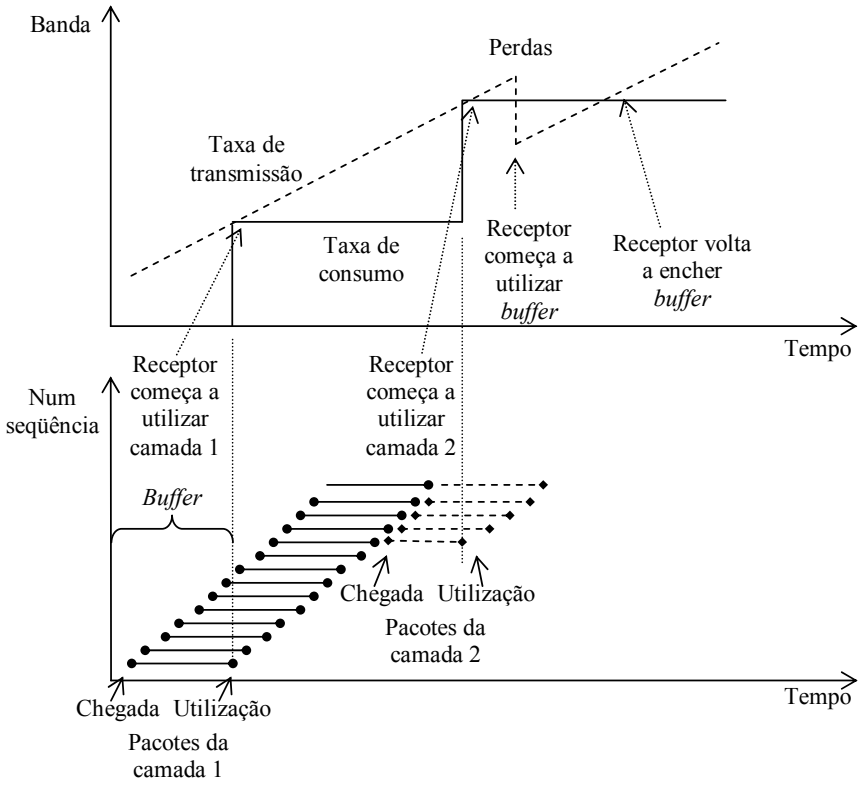


FIGURA 3.19 – Transmissão unicast em camadas com *bufferização* no receptor

Claramente pode-se perceber duas fases: a fase de enchimento no *buffer*, enquanto a taxa de transmissão é maior que a taxa de consumo, e a fase de utilização do *buffer*, quando a taxa de consumo é maior do que a taxa de transmissão.

O protocolo RAP foi validado através do simulador NS2, utilizando-se a topologia mostrada na figura 3.20. O enlace entre os roteadores é sempre o ponto mais congestionado. Existem *m* conexões RAP (*R1* a *Rm* conectados com *P1* a *Pm*) e *n* conexões TCP (*T1* a *Tn* conectados com *S1* a *Sn*). Nos resultados das simulações, percebe-se que o protocolo se mostrou um pouco mais agressivo que o TCP, atingindo taxas 1,5 a 2,5 vezes maiores que os fluxos TCP.

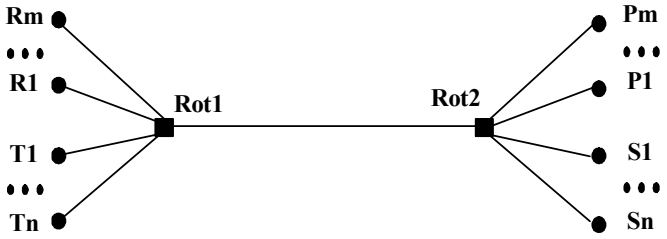


FIGURA 3.20 – Topologia utilizada na simulação do protocolo RAP

4 O SAM: Sistema Adaptativo para Multimídia

Segundo Li [LI 2003], a distribuição de vídeo em tempo real é uma das principais aplicações para transmissões multicast, devido ao rápido desenvolvimento de aplicações multimídia e à natureza dos programas de vídeo. Além disso, é um componente essencial de muitas aplicações emergentes na Internet, como videoconferência e ensino a distância.

De acordo com essa necessidade, e ciente da natureza heterogênea da Internet (conforme definição no item 2.3), está se propondo uma nova ferramenta de transmissão de vídeo através da Internet atual. Esta ferramenta foi denominada SAM (Sistema Adaptativo para Multimídia).

O objetivo deste capítulo é detalhar o modelo utilizado no SAM, bem como sua aplicabilidade e funcionalidades desejáveis. Além disso, será definida a metodologia para as simulações dos algoritmos utilizados no sistema.

As principais funcionalidades esperadas para o SAM são as seguintes:

- Transmitir em multicast, de forma escalável a centenas de receptores;
- Transmitir sinais gravados previamente em arquivo (vídeo sob demanda) ou ao vivo, à medida que acontecem, como por exemplo a cobertura de um evento esportivo;
- Transmitir sinais CBR ou VBR;
- Gerar tráfego de forma eqüitativa entre as próprias sessões quando elas concorrem entre si (*fairness*);
- Gerar tráfego eqüitativo com sessões TCP concorrentes (*friendliness*);
- Ser adaptável para a Internet atual, ou seja, com roteadores *best-effort* que encaminham os pacotes através de política de filas RED ou *droptail*;
- Ser adaptável automaticamente a ambientes heterogêneos, transmitindo tráfego na forma mais eficiente para o receptor em questão;
- Ser adaptável ao tipo de usuário, de forma que o mesmo possa escolher determinadas camadas opcionais;
- Ser adaptável às condições da máquina, ou seja, regular automaticamente a qualidade do vídeo de forma a não sobrecarregar a máquina;
- Possuir estabilidade na recepção do vídeo.

A figura 4.1 mostra o modelo proposto a fim de atingir as funcionalidades descritas acima. O transmissor gera a codificação do sinal multimídia em camadas e transmite cada camada como um grupo multicast diferente (módulos "codificador em camadas" e "Transmissão"). No lado do receptor, o módulo de controle de congestionamento controla a adaptação, que é baseada em detecção de perdas, estimativa de eqüidade de banda e estimativa de banda máxima (através da técnica de pares de pacotes).

Inicialmente, o receptor estabelece uma conexão unicast com o módulo "descrição das taxas" do transmissor, buscando os parâmetros de sessão, que são o nome e a descrição da sessão, bem como características de cada taxa ou camada

(número IP multicast, tráfego médio, taxa de pico, e assim por diante). Com essa informação, o módulo "controle de congestionamento e camadas (ALM)" inscreve o receptor no número supostamente correto de camadas, de acordo com a capacidade da rede, seu nível de congestionamento e perfil de usuário, na forma mais eqüitativa possível com os outros tráfegos. Finalmente, as camadas recebidas são enviadas ao módulo "decodificador em camadas", que reconstrói e ressincroniza o sinal multimídia, mostrando o resultado na tela do usuário.

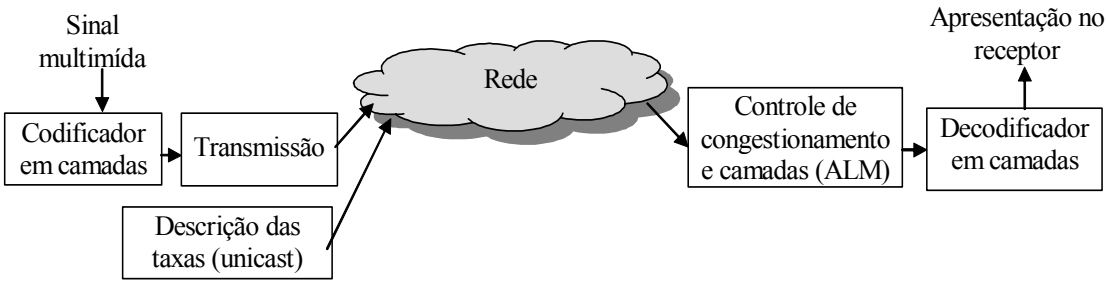


FIGURA 4.1 – Modelo do SAM

Em relação à aplicabilidade, o SAM é um sistema adequado para utilização em transmissões multimídia codificadas em camadas fixas, com a possibilidade da transmissão de camadas cumulativas (um mesmo vídeo dividido em várias camadas) e opcionais (o receptor escolhe a camada que mais lhe convém).

Como exemplos de aplicações para o SAM, pode-se citar as seguintes:

- Um canal de TV para centenas ou milhares de usuários, como por exemplo o descrito por Roesler [ROE 2000a]. A transmissão pode ser via TV digital e de forma interativa com o usuário, conforme visto em [ROE 2000c];
- Palestras de uma grande conferência, como as promovidas pelo IEEE;
- Eventos esportivos, como a copa do mundo ou as olimpíadas;
- Shows em geral, como teatro e musicais;
- Aulas a distância, como por exemplo a realizada pelo autor desta Tese e detalhada em [ROE 2003c] e [ROE 2003i].

Das funcionalidades desejadas para o sistema, vistas acima, algumas são simples implementações na interface com o usuário, e outras formam o núcleo desta Tese. Os próximos itens procuram fazer essa diferenciação, detalhando os módulos do SAM. Do item 4.1 (Fonte do sinal multimídia para entrada no SAM) até o item 4.6 (Controle de limitações da máquina) são detalhados os vários blocos ilustrados pela figura 4.1. Posteriormente, o item 4.7 (Metodologia para as simulações dos algoritmos) e o item 4.8 (Definição das simulações) descrevem a metodologia para análise dos algoritmos propostos.

4.1 Fonte do sinal multimídia para entrada no SAM

O sinal multimídia pode entrar no SAM através de uma placa de captura de vídeo ou de um arquivo. Em qualquer uma das possibilidades, o codificador e o decodificador devem possuir uma rapidez de execução superior à taxa de transmissão

dos quadros.

Assim, por exemplo, se o sinal é recebido a partir de uma filmadora ou videocassete conectado à placa de captura de vídeo, ele deve ser codificado e transmitido à medida que os quadros são capturados. Da mesma forma, quando a entrada é via arquivo previamente codificado, como num sistema de vídeo sob demanda, o sinal multimídia será lido a partir do dispositivo de armazenamento da máquina transmissora. Esse dispositivo deverá permitir a leitura de forma rápida suficiente para a codificação do sinal. Para o decodificador é indiferente de onde vem o sinal, pois sua função será a mesma nos dois casos.

Do ponto de vista do transmissor, a interface do SAM deve permitir a escolha do local de entrada do sinal, e o codificador deve estar apto a buscar o sinal de qualquer um dos locais, a partir de vários formatos.

4.2 Módulo "descrição das taxas"

O SAM utiliza transmissão multi-taxa, e é essa particularidade que permite a adaptação a ambientes heterogêneos, conforme definição no item 2.3 (Ambientes heterogêneos e codificação multi-taxas). A característica das camadas é dada pela flexibilidade do codificador em atender às definições do usuário. Na descrição deste item não foram consideradas as limitações do codificador utilizado no SAM, preferindo-se explicar o modelo de forma genérica, a fim de não limitar a ferramenta. O item 4.3 se refere especificamente ao codificador VEBIT, que será integrado ao sistema.

Existem dois tipos de camadas modeladas pelo SAM: camadas cumulativas (onde a próxima complementa a qualidade da anterior) e camadas opcionais (onde as camadas são independentes entre si e cada receptor escolhe as que lhe interessar).

As camadas cumulativas existem para transmissão de vídeo e áudio codificados através da codificação escalável, definida no capítulo 8 (Codificação de Vídeo Multi-Taxas), e as camadas opcionais servem para transmissão de fluxos independentes, como a dublagem em outro idioma ou taxas alternativas de um mesmo vídeo.

Em relação ao usuário, no lado do receptor, é importante que este saiba a definição exata de cada camada opcional. Isso vai permitir a ele efetuar uma escolha de prioridade e interesse em receber tais camadas, como por exemplo o sistema com as camadas descritas na tabela 4.1, onde "*cum*" representa uma camada cumulativa, e "*opc*" representa uma camada opcional.

A tabela 4.1 mostra uma transmissão com 5 camadas cumulativas e 5 opcionais, nas prioridades "2, 3, 4, 5, 6, 1". Isso significa que, se o receptor tiver uma banda máxima de 50 kbit/s, ele vai escolher a camada 6 (áudio em português, prioridade 1), e camada 1 (vídeo, camada base, prioridade 2), totalizando 35 kbit/s. As camadas sem prioridade definida não estão habilitadas para este usuário, ou seja, o sistema não está apto a se cadastrar nelas.

O usuário pode modificar as prioridades das camadas opcionais, como por exemplo escolhendo áudio em inglês ao invés de português para a prioridade 1, ou eliminando a recepção de alguma camada, como a de prioridade 6.

TABELA 4.1 – Exemplo hipotético de camadas transmitidas

Camada	Descrição	Prioridade
1 (cum)	Vídeo: camada base; IP 239.1.1.1; taxa: 25 kbit/s; ...	2
2 (cum)	Vídeo (agrega qualidade); IP 239.1.1.2; taxa: 25 kbit/s; ...	3
3 (cum)	Vídeo (agrega qualidade); IP 239.1.1.3; taxa: 50 kbit/s; ...	4
4 (cum)	Vídeo (agrega qualidade); IP 239.1.1.4; taxa: 100 kbit/s; ...	5
5 (cum)	Vídeo (agrega qualidade); IP 239.1.1.5; taxa: 180 kbit/s; ...	6
6 (opc)	Áudio em português; IP 239.1.1.6; taxa: 10 kbit/s; ...	1
7 (opc)	Áudio em inglês; IP 239.1.1.7; taxa: 10 kbit/s; ...	-
8 (opc)	Áudio em espanhol; IP 239.1.1.8; taxa: 10 kbit/s; ...	-
9 (opc)	Vídeo monocromático p/ PDAs; IP 239.1.1.9; taxa: 10 kbit/s; ...	-
10 (opc)	Vídeo com cores para PDAs; IP 239.1.1.10; taxa: 20 kbit/s; ...	-

Caso o usuário esteja conectando através de um terminal monocromático, pode modificar as prioridades e receber somente a camada 6 (áudio em português, prioridade 1) e a camada 9 (vídeo monocromático, prioridade 2), demandando um máximo de 20 kbit/s da rede.

Quando o algoritmo do ALM efetua controle de congestionamento na rede (que é o foco principal da Tese), cada camada é considerada simplesmente como um determinado fluxo de bits concorrendo pela banda com os demais fluxos. Com essa definição em mente, o uso de camadas opcionais ou cumulativas não faz diferença para o algoritmo, pois este vai permitir ao usuário se inscrever no máximo de banda que a rede permitir.

4.3 Módulos “codificador e decodificador em camadas”

O codificador e decodificador em camadas definido para a utilização com o SAM foi denominado VEBIT (Vídeo Escalável por Bitplanes), sendo desenvolvido em paralelo com esta Tese, através de uma dissertação de mestrado [BRU 2003].

O codificador VEBIT atende com sucesso a tarefa de codificar vídeo em diferentes taxas, e atualmente trabalha com 5 camadas. Entretanto, ainda não possui a integração e sincronização com áudio. Sua descrição detalhada encontra-se no capítulo 8 (Codificação de Vídeo Multi-Taxas).

4.4 Módulo “Transmissão”

Dependendo do codificador utilizado, o sinal a ser transmitido pode ser do tipo CBR e VBR, portanto, as simulações do algoritmo devem levar em consideração esses dois tipos de sinais.

O multicast deve ser utilizado sempre que o número de receptores for grande o suficiente para que a rede se beneficie com seu uso, caso contrário, poderia acontecer das mensagens de controle do multicast causarem um tráfego maior do que sem a sua utilização. Para efeitos desta Tese, considerou-se como prioridade a elaboração de um

algoritmo que efetuasse controle de congestionamento em multicast, mas este pode ser adaptado para a utilização em unicast.

No SAM, existe a possibilidade do transmissor enviar pacotes multimídia em pares, permitindo ao receptor uma estimativa de banda máxima na rede, conforme detalhamento no Anexo A e em [ROE 2003b]. Os dois principais algoritmos definidos nesta Tese utilizam a técnica de pares de pacotes, que não necessita de ACKs, portanto, não é gerado qualquer tráfego acima do que seria gerado sem o uso da técnica.

Em todos os casos considerados, os receptores podem estar localizados em enlaces com diferentes larguras de banda, adaptando-se de forma automática.

4.5 Módulo "controle de congestionamento e camadas (ALM)"

O controle de congestionamento é efetuado pelo algoritmo ALM (*Adaptive Layered Multicast*), e é o módulo principal do SAM. Este módulo é responsável por:

- Determinar o número de camadas que o receptor pode se inscrever, permitindo a adaptabilidade do sistema em ambientes heterogêneos;
- Possibilitar a escalabilidade para centenas de receptores, pois é o algoritmo que determina o número de mensagens trocadas entre transmissor e receptores;
- Permitir a geração de tráfego de forma eqüitativa entre as próprias sessões quando elas concorrem entre si (*fairness*), conforme definição no item 2.5 (Eqüidade de tráfego);
- Permitir a geração de tráfego eqüitativo com sessões TCP concorrentes (*friendliness*), conforme definição no item 2.5 (Eqüidade de tráfego);
- Permitir estabilidade na recepção do vídeo: a importância deste conceito foi definida no item 2.4 (Estabilidade de tráfego).

A principal contribuição desta Tese é o algoritmo de controle de congestionamento ALM, pois foi onde se efetuou o maior esforço, e disso resultou a elaboração de duas principais variantes (ALMP e ALMTF) e uma sugestão de protocolo de nível 4 (TCP-STABLE), conforme descrição a seguir:

- **ALMP (ALM para redes privativas)**: um algoritmo adequado ao uso em redes privativas, onde os parâmetros foram adaptados visando a obtenção de estabilidade. O ALMP será detalhado no capítulo 5;
- **ALMTF (ALM *TCP Friendly*)**: um algoritmo cujo principal objetivo é efetuar transmissão de tráfego de forma eqüitativa com o TCP, procurando manter ao máximo a estabilidade. O ALMTF será detalhado no capítulo 6;
- **TCP-STABLE**: uma sugestão de protocolo de nível 4 que utiliza apoio do roteador e tem como objetivo a transmissão confiável e não-confiável, de forma eqüitativa entre todas as sessões envolvidas. O detalhamento do TCP-STABLE é feito em [ROE 2003j].

Quando se referencia a palavra "ALM" nesta Tese, está ser referindo de forma geral a qualquer um dos dois algoritmos desenvolvidos (ALMP ou ALMTF).

Além das duas variantes mencionadas acima, no início deste trabalho foi criado

e simulado outro algoritmo, batizado como ALMJ (*Adaptive Layered Multicast Jitter Based*), que infere congestionamento baseando-se no *jitter* de chegada dos pacotes, ou seja, na variação do tempo de chegada dos pacotes devido ao atraso nas filas dos roteadores. Sua primeira versão foi publicada em outubro de 2000 [ROE 2000b], e posteriormente aperfeiçoada e publicada em 2001 [ROE 2001]. O algoritmo ALMJ trabalha com adaptação no receptor, não necessitando de comunicação com o transmissor. Isso torna o algoritmo escalável para um grande número de receptores. Provou-se que esta forma de adaptabilidade não se adapta ao tráfego TCP, pois o TCP é bem mais agressivo, tomando conta da capacidade disponível na rede, portanto, não será detalhado neste trabalho, entretanto, o código fonte implementado no simulador NS2, bem como o manual de instalação e resultado de todas as simulações efetuadas pode ser obtido em http://www.inf.unisinos.br/~roesler/tese.

Entre os diversos tráfegos concorrentes com o ALM, pode-se distinguir três tipos principais: TCP, UDP e o próprio ALM. Esses tráfegos podem acontecer juntos ou separadamente. Quando se fala em transmissão de forma eqüitativa, devem ser levados em consideração esses três tipos de concorrência.

O algoritmo ALM foi desenvolvido visando seu funcionamento numa rede *best-effort*, pois o autor julga que, mesmo com a entrada de mecanismos de QoS como Intserv e Diffserv (detalhados num relatório técnico em [ROE 2003f]), o tráfego em redes *best-effort* continuará significativo por muitos anos. Uma visão semelhante é compartilhada pelos autores da RFC 2309 [BRA 98]. Além disso, na mesma classe Diffserv continuará existindo a disputa por recursos.

Como já foi dito no item 2.8.5, o controle de congestionamento fim-a-fim pode ser encontrado no receptor e no transmissor. Quando é inserido no receptor, existe uma menor interação com o transmissor, gerando menos mensagens e, conseqüentemente, menos tráfego com o aumento no número de receptores, melhorando a escalabilidade. Quando o controle de congestionamento é inserido no transmissor, os receptores devem enviar periodicamente informações de qualidade de recepção ao transmissor, a fim de que o mesmo ajuste sua taxa de transmissão. Além de ser mais complexo, esse mecanismo reduz a escalabilidade, tendo como conseqüência um aumento no tráfego da rede em função do algoritmo. O ALMP segue a primeira abordagem, ou seja, adaptação no receptor.

4.6 Controle de limitações da máquina

Adicionalmente ao controle de congestionamento na rede, que foi explicado no item anterior, existe também a possibilidade de efetuar a adaptação automática ao número de camadas de acordo com as características da máquina. Existem duas formas de adaptação no receptor: a primeira é em relação à sua capacidade de processamento, e a segunda trata das limitações de outros aspectos, como resolução, número de cores, e assim por diante. Os próximos itens tratam, respectivamente, dessas duas formas de adaptação. Vale lembrar que esses aspectos não serão aprofundados posteriormente, já que a Tese está voltada ao controle de congestionamento na rede.

4.6.1 Limitação na capacidade de processamento do receptor

Quando a capacidade de processamento do receptor atinge seu máximo, o sinal de vídeo que está sendo decodificado perde qualidade de forma significativa, muito mais do que perderia caso um menor número de camadas estivesse sendo decodificado, portanto, é desejável que o mecanismo de adaptação do SAM trate esse tipo de situação, liberando a rede da transmissão desse tráfego adicional e melhorando a qualidade percebida pelo usuário.

Na implementação do algoritmo ALMP descrita no capítulo 5, percebeu-se uma forma de adaptação automática: o processador fica sobrecarregado e o sistema não consegue ler todos os pacotes capturados pela placa de rede, fazendo com que o algoritmo assuma que os pacotes foram perdidos. Como o algoritmo do SAM considera que todos pacotes perdidos são devidos à congestionamento, ele efetua *leave* na camada superior, liberando o decodificador de tratar essa camada. Isso diminui a necessidade de uso do processador e estabiliza o sistema.

Para evitar atingir o limite de sobrecarga de CPU, desenvolveu-se um programa de monitoramento do seu uso, que envia um sinal de controle quando o processador atinge um valor limite, como, por exemplo, 90%. Ao receber esse sinal, o decodificador pára de decodificar a última camada, liberando a CPU. Caso a situação persista por determinado tempo, o módulo de controle de congestionamento (ALM) recebe um aviso para efetuar *leave* na camada menos prioritária. Quando o processador deixa de exigir tanta CPU, o decodificador é avisado a voltar a decodificar a última camada. Para evitar instabilidade, deve haver uma curva de histerese e um atraso de reação à carga da CPU, evitando rápidas respostas para breves sobrecargas de CPU. O programa desenvolvido está disponível em http://www.inf.unisinos.br/~roesler/tese.

Um trabalho relacionado é visto em [LYO 2000], onde os pacotes são armazenados em filas FIFO assim que chegam na placa de rede. A aplicação efetua o controle de sobrecarga de processador no receptor através da análise de taxa de utilização dessas filas. Caso alguma fila aumente de tamanho, é sinal que a CPU não está conseguindo decodificar os pacotes que estão chegando no sistema, portanto, o algoritmo provoca uma degradação suave na qualidade do vídeo recebido, liberando a carga da CPU.

4.6.2 Limitações em outros aspectos do receptor

Outros aspectos que podem ser considerados no SAM são relacionados à outras características do receptor, como resolução, número de cores, e assim por diante. Esses tópicos não serão aprofundados, pois dependem de um codificador apropriado, conforme descrito anteriormente, neste capítulo.

4.7 Metodologia para as simulações dos algoritmos

Segundo Jain [JAI 91], existem três métodos para validar o funcionamento de um sistema: analítico, experimental ou através de simulação. A escolha efetuada nesta Tese foi a validação dos algoritmos via simulação, visto que já existe um simulador de redes desenvolvido e aceito mundialmente pela comunidade científica, que é o *Network*

Simulator (NS2) [MCC 2003]. Dessa forma, bastava implementar e compilar o código dos algoritmos desenvolvidos no NS2, inserindo no simulador os novos protocolos. Após a fase de simulações, um dos dois algoritmos desenvolvidos, no caso o ALMP foi implementado, e sua descrição encontra-se no item 5.3 (Implementação do algoritmo ALMP).

A maior parte dos trabalhos relacionados, detalhados no capítulo 3, também foram validados através do simulador NS2. Exemplos são o RLM, o RLC, o PLM, o FLID-DL, o TFMCC, o TFRC, o HALM, o TEAR, o RAP, e outros.

Através de um ambiente simulado se consegue analisar rapidamente detalhes do protocolo praticamente impossíveis de serem analisados na prática, como por exemplo seu comportamento com a mudança na largura de banda dos roteadores, variação no atraso dos enlaces, aumento no número de receptores, variações dos tipos e quantidade de fluxos concorrentes, e assim por diante.

Este item tem por objetivo mostrar a metodologia utilizada para a simulação dos dois algoritmos desenvolvidos no âmbito desta Tese. As mesmas métricas foram escolhidas para ambos algoritmos a fim de facilitar a comparação entre eles, além de serem importantes para a aplicação que está sendo desenvolvida.

Algumas práticas metodológicas, métricas e parâmetros (como o tamanho das filas, tipos de filas, velocidade nos enlaces centrais e necessidade de randomização), foram escolhidos com base no encontro feito na *Digital Fountain Incorporation*, em agosto de 2000, proposto por Mark Handley [HAN 2000], onde são propostos alguns itens considerados importantes para simulação de controle de congestionamento em novos protocolos.

Todos os testes realizados foram repetidos, no mínimo, três vezes, pois alguns parâmetros das simulações são randômicos, e o teste é repetido a fim de verificar se aconteceram discrepâncias. Neste trabalho são mostrados apenas alguns gráficos, entretanto, existem muitos outros disponíveis no CD anexo a este trabalho e em http://www.inf.unisinos.br/~roesler/tese. Os gráficos escolhidos mostram o comportamento da simulação de uma forma representativa em relação à todas as amostras efetuadas.

Quando se utilizar o termo “ALM”, ele se refere a qualquer um dos dois algoritmos implementados, ou seja, ALMP ou o ALMTF. Os próximos itens detalham a metodologia a ser utilizada nas simulações. O item 4.7.1 especifica as métricas utilizadas para a análise dos algoritmos. O item 4.7.2 especifica os parâmetros de rede que foram variados. No item 4.7.3 é definida a topologia utilizada para simular os algoritmos, enquanto o item 4.7.4 apresenta a metodologia utilizada para variação dos parâmetros no simulador. No item 4.7.5 são definidas as simulações efetuadas nos algoritmos.

4.7.1 Métricas estabelecidas para as simulações

As simulações efetuadas visam analisar o ALM nos seguintes critérios:

- **Adaptabilidade**: verifica a capacidade do algoritmo para adaptar-se ao correto número de camadas, mesmo em ambientes heterogêneos;

- **Escalabilidade**: verifica a escalabilidade do algoritmo para um grande número de usuários;
- **Estabilidade**: examina a variabilidade da inscrição em camadas ao longo do tempo, ou seja, se o algoritmo se mantém no número correto de camadas ou fica variando. Esse critério será analisado sem tráfego concorrente e com tráfego concorrente de outras sessões ALM, bem como tráfego TCP e UDP;
- **Tempo de adaptação**: verifica a rapidez do algoritmo para encontrar a sua parcela eqüitativa de banda;
- **Eqüidade de tráfego**: analisa se a quantidade de dados transferidos pelo algoritmo é semelhante à quantidade de dados transferida pelo tráfego concorrente. Devem ser analisadas simulações de uma sessão ALM concorrendo com tráfego de outras sessões ALM (*fairness*), e com tráfego TCP e UDP (*friendliness*). O comportamento será verificado para o ALM começando antes dos outros tráfegos e depois dos outros tráfegos, pois alguns algoritmos analisados no capítulo 3 (Trabalhos Relacionados) mostraram problemas nesse critério;
- **Taxa de perdas**: verifica a taxa de pacotes descartados pelos roteadores em função do algoritmo.

4.7.2 Principais parâmetros relacionados à topologia e ao tráfego na rede

A dinâmica do tráfego na rede depende de diversos parâmetros, como o número de usuários, tamanho do pacote e número de camadas. Os fluxos reais podem possuir camadas bastante variadas, dependendo do CODEC utilizado, portanto, as simulações foram repetidas para três diferentes configurações de camadas.

Os seguintes parâmetros foram utilizados visando analisar o comportamento do algoritmo em relação à dinâmica do tráfego na rede. Espera-se que, a partir das simulações efetuadas com esses parâmetros, seja possível inferir o comportamento do algoritmo para quaisquer outros casos.

- **Número de fluxos concorrentes**: nos testes de eqüidade de tráfego, convencionou-se utilizar o mesmo número de fluxos ALM e TCP concorrentes. Foram utilizados 1, 2, 5 e 10 fluxos simultâneos de cada algoritmo. Com fluxos CBR concorrentes, foi utilizado apenas um fluxo, pois o tráfego é CBR e não se adapta;
- **Número de usuários ou receptores**: como se trata de uma aplicação destinada a multicast e que tem o objetivo de atingir escalabilidade, o comportamento do algoritmo foi testado com 1, 5 e 25 usuários ligados a cada roteador nas simulações de adaptabilidade e escalabilidade. Como existem quatro blocos de receptores, o total de receptores será, respectivamente, 4, 20 e 100 usuários;
- **Largura de banda nos roteadores principais**: foi necessário variar a banda dos roteadores principais para que os fluxos não compartilhem uma banda muito pequena, dificultando a visualização, como aconteceria caso a banda eqüitativa do receptor fosse menor do que a banda da primeira camada. Assim, convencionou-se que cada fluxo, em média, deve ter uma banda próxima a 300 kbit/s ou 500kbit/s, dependendo da configuração das camadas. Entretanto, o comportamento do algoritmo também foi analisado para bandas de 100 kbit/s, 400 kbit/s, 1 Mbit/s e 2 Mbit/s;
- **Atraso físico dos enlaces**: um dos principais fatores que mudam em qualquer

topologia real é o atraso entre dois roteadores, pois a distância física entre eles pode variar, bem como o tipo de meio físico, que pode ser via fibra ótica ou satélite, por exemplo. Esse fator é um dos determinantes do RTT, que está relacionado diretamente com o desempenho do protocolo TCP, como foi visto no item 2.6. O atraso fim-a-fim unidirecional devido ao meio físico foi estabelecido variando entre 5ms e 50ms;

- **Camadas com taxas exponenciais padrão iniciando em 30 kbit/s**: utilizadas para analisar o comportamento do algoritmo com camadas cuja taxa da camada superior é o dobro da camada atual. O valor inicial de 30 kbit/s foi escolhido pois permite adaptação para receptores via modem. As taxas utilizadas foram 30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s e 960 kbit/s;
- **Camadas com taxas exponenciais imitando o codificador VEBIT**: criou-se uma taxa exponencial semelhante à obtida com o codificador VEBIT, que é o codificador a ser utilizado na Tese, sendo descrito no capítulo 8. A taxa utilizada foi aproximadamente a média dos quatro vídeos analisados, sendo: 15 kbit/s, 45 kbit/s, 80 kbit/s, 150 kbit/s, 385 kbit/s;
- **Camadas com taxas iguais a 50 kbit/s**: utilizadas para analisar o comportamento do algoritmo em camadas iguais e próximas entre si. Assim, por exemplo, as taxas de cada camada foram 50 kbit/s, 50 kbit/s, 50 kbit/s, e assim por diante;
- **Camadas com taxas iguais a 100 kbit/s**: igual ao item anterior, porém, cada camada acrescenta 100 kbit/s na camada anterior em vez de 50 kbit/s;
- **Tamanho do pacote**: numa aplicação multimídia real, o tamanho do pacote pode ser variado dependendo da codificação utilizada. Nas simulações efetuadas, foram utilizados 3 tamanhos de pacote: 250 bytes, 500 bytes e 1000 bytes. O objetivo foi analisar o impacto que pacotes de tamanho variável provocam nas métricas estabelecidas, tanto entre diferentes sessões ALM como entre sessões ALM com TCP e UDP concorrentes;
- **Tipo de fila nos roteadores intermediários**: as filas mais comuns atualmente são filas RED e *droptail*, conforme visto no capítulo 2, e essas duas filas foram simuladas. No *droptail*, foram utilizados dois tamanhos de fila: com 20 pacotes e com 60 pacotes. No caso do RED, os valores de *threshold* mínimo / máximo foram de 10 / 20 e 20 / 60;
- **Randomização**: os algoritmos foram implementados no simulador de forma que possuam uma randomização inicial, ou seja, o tempo inicial programado para o início do transmissor e do receptor pode variar em 1s, fazendo com que dois receptores programados para iniciar em 2s iniciem a recepção em instantes diferentes: um no instante 2,3s e outro no instante 2,9s, por exemplo. Além disso, como as várias taxas das camadas do ALM são CBR, foi implementada uma randomização no envio dos pacotes, como mostra a figura 4.2. Essa randomização pode ser de dois tipos: a) randomização de ±0,5s; b) CBR puro, ou randomização de ±0,000001s;
- **Tamanho da janela do TCP e ALMTF**: o comportamento do TCP varia bastante quando se utiliza um tamanho de janela calculado para que sua taxa de transmissão não ultrapasse o valor dado pelo produto “atraso X largura de banda”. O tamanho da janela também influencia o comportamento do algoritmo ALMTF. Assim, foram utilizados nas simulações dois tamanhos de janela: 30 e 60.

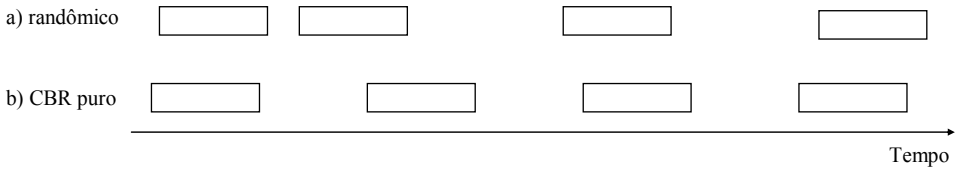


FIGURA 4.2 – Randomização do tráfego transmitido

4.7.3 Topologia utilizada para simulações

Como foi visto nos itens anteriores, em redes reais a topologia pode variar bastante, portanto, utilizou-se uma topologia simulada que refletisse essa possibilidade a fim de aumentar a probabilidade de semelhança entre a simulação e a realidade, quando da implementação do algoritmo.

Com base nas métricas estabelecidas no item 4.7.1 e nos parâmetros vistos no item 4.7.2, definiu-se duas topologias a fim de analisar o comportamento do algoritmo proposto. As topologias definidas abrangem todas as topologias vistas nos trabalhos relacionados (capítulo 3). As topologias estão ilustradas na figura 4.3, e descritas a seguir.

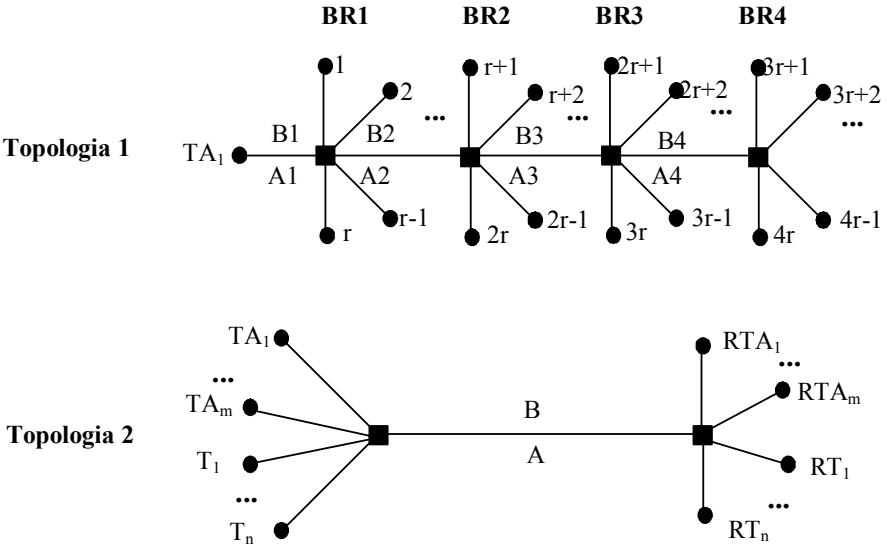


FIGURA 4.3 – Topologias utilizadas para as simulações

De forma geral, os transmissores ALM estão representados como TA_i, onde *i* varia de 1 a *n*, como ilustrado na figura 4.3. Os transmissores TCP e UDP são denominados T_i, onde *i* também varia de 1 a *n*. Os receptores foram separados em blocos denominados BR (Blocos de Receptores), e cada bloco contém até um máximo de *r* receptores, conforme ilustrado na topologia 1 da figura. A banda é determinada em cada enlace principal, sendo denominada B. Todas os outros enlaces possuem largura de banda de 10 Mbit/s, a fim de não interferirem nos resultados. Os atrasos também são controlados nos enlaces principais, sendo denominados A. Todos os outros enlaces possuem atraso fixo em 1ms.

O objetivo da topologia 1 foi analisar o algoritmo ALM nos critérios de adaptabilidade, escalabilidade, tempo de adaptação e estabilidade sem tráfego

concorrente. Para atingir tal objetivo, a topologia 1 possui um único transmissor ALM, quatro blocos de *r* receptores (BR1 a BR4), banda e atraso variáveis entre os enlaces (B1 a B4, A1 a A4). A escalabilidade é atingida através de um número alto de receptores, e a adaptabilidade se consegue através da limitação de banda nos enlaces.

O objetivo da topologia 2 foi analisar o algoritmo nos critérios de tempo de adaptação, estabilidade e eqüidade, todos com a presença de tráfego concorrente. Para tanto, existem até *m* transmissores ALM e até *n* transmissores TCP / UDP rodando simultaneamente. Cada transmissor se conecta a um único receptor. Por exemplo, TA_1 se conecta a RTA_1, TA_2 se conecta a RTA_2, e assim por diante. Nessa topologia também existe a possibilidade de variar a banda B e o atraso A no enlace principal.

Está se assumindo que os fluxos TCP são de longa duração, entretanto, existe muito fluxo de curta duração. Segundo Thompson [THO 97], na média, o número de pacotes TCP transferidos por fluxo varia entre 17 e 22, dependendo da hora do dia, e cada fluxo demora aproximadamente 15 segundos na rede.

4.7.4 Metodologia para variação dos parâmetros no simulador

Os algoritmos ALM foram implementados no simulador NS2, e consistem de duas partes distintas:

- **O código dos algoritmos ALM**: desenvolvidos e compilados junto com o NS2, resultando numa versão do simulador que suporta tais algoritmos. Os mesmos foram desenvolvidos em linguagem ‘C’ e ‘TCL’;
- **Os arquivos de simulação do código**: desenvolvidos em linguagem ‘TCL’, permitem a variação dos parâmetros e topologia de simulação, a fim de verificar a funcionalidade do algoritmo sob diversas situações.

Os parâmetros a seguir, configurados nos arquivos “almpsim.tcl” ou “almtfsim.tcl”, mostram a forma de variação do ambiente de simulação.

```
set run_nam 0            ;# define se vai rodar NAM ou não
set runtime 1000         ;# tempo de simulação
set topologia 2          ;# topologia da Tese (1 ou 2)
set inireceptores 1      ;# tempo de inicio dos receptores
set inicio_mesmotempo 0 ;# início ao mesmo tempo ou cada 100s
set ALM_Banda "2,1M 1,05M 525K 105K"   ;# Banda entre roteadores
set ALM_r  1             ;# numero de receptores por bloco
set ALM_m 2              ;# numero de transmissores ALM - topo 2
set ALM_n 0              ;# numero de transmissores TCP – topo 2
set ALM_Atraso 1ms       ;# atraso nos enlaces principais
set tipofila 0           ;# Escolhe entre droptail ou RED
set ALM_random 1         ;# 1=+-0,5s; 2=CBR puro
set ALM_packetsize 500   ;# tamanho do pacote ALM
set ALM_TipoCamadas 0    ;# exp., VEBIT ou contínuas
set ALM_tipotraf 1       ;# CBR, Exponencial ou Pareto
set ALM_gerafluxo 1      ;# gráficos por fluxo ou camada
set criaalm 1            ;# habilita tráfego ALM
set criaudp 0            ;# habilita trafego UDP
set criatcp 0            ;# habilita tráfego TCP
```

- **run_nam**: em alguns momentos foi necessário analisar a simulação através do NAM (*Network Animator*), que mostra uma visão gráfica da execução do algoritmo.

Essa opção permite habilitar ou não a execução do NAM;

- **runtime**: define o tempo de simulação, em segundos;
- **topologia**: define a topologia simulada, se a topologia 1 ou a topologia 2, conforme figura 4.3;
- **inireceptores**: define o tempo de início dos receptores, em segundos. Os receptores iniciam num tempo randômico de até um segundo acima do valor configurado. Isso permite uma aleatoriedade quando se repetem várias vezes as simulações. Os transmissores também iniciam num tempo aleatório, porém sempre entre 0 e 1s;
- **inicio_mesmotempo**: define se todos receptores iniciam simultaneamente (com randomicidade de 1 segundo) ou se iniciam a cada 100s (também com randomicidade de 1 segundo);
- **ALM_Banda**: configuração de banda entre os roteadores principais. Na topologia 1 da figura 4.3, existem 4 possibilidades de configuração de banda, que são dados pelos 4 valores neste vetor (neste caso 2,1 Mbit/s, 1,05 Mbit/s, 525 kbit/s e 105 kbit/s). Na topologia 2, só é considerado o primeiro valor do vetor (no caso, 2,1 Mbit/s). Escolheu-se valores que não fossem iguais à soma das camadas (tomando como padrão as de 100 kbit/s), pois nesse caso a visualização ficaria levemente prejudicada, pois o receptor adaptar-seia na camada anterior à ótima teórica, devido ao não esvaziamento da fila nas tentativas de subir camadas. Considerou-se isso irreal, pois dificilmente acontece num roteador comercial, e procurou-se simular algo mais próximo da realidade, adicionando 5% de banda ao número inicial. Assim, em vez de 100 kbit/s, utilizou-se 105 kbit/s, e assim por diante;
- **ALM_r**: número de receptores por bloco na topologia 1;
- **ALM_m**: número de transmissores e receptores ALM na topologia 2;
- **ALM_n**: número de transmissores e receptores TCP ou UDP na topologia 2;
- **ALM_Atraso**: atraso físico (em ms) dos enlaces principais, tanto na topologia 1 como na topologia 2;
- **tipofila**: define o tipo de fila nos roteadores. As opções são: 0=*droptail* de tamanho 20; 1=*droptail* de tamanho 60; 2=RED 10/20 (*threshold*=10 e *maxthreshold*=20); 3=RED 20/60;
- **ALM_random**: define a randomicidade da transmissão dos pacotes CBR. Isso é necessário pois num ambiente real os pacotes sofrem uma certa variação devido à concorrência na rede local, carga da CPU, filas nos roteadores anteriores, e assim por diante. Escolheu-se 2 opções: 1=±0,5s (o instante de transmissão de cada pacote pode variar randomicamente para mais ou para menos 0,5 segundos, provocando uma possibilidade de rajada nas transmissões); 2=CBR puro, que na realidade é uma variação de ±0,000001s;
- **ALM_packetsize**: define o tamanho do pacote ALM;
- **ALM_tipocamadas**: define a taxa de cada camada e o número de camadas do codificador de vídeo, conforme definição no item 4.7.2 (0=exponencial padrão; 1=VEBIT; 2=50k; 3=100k);
- **ALM_tipotraf**: permite a escolha do tipo de tráfego desejado. As opções são: 0=CBR utilizando transmissão em pares de pacotes; 1=CBR sem utilização de pares de pacotes; 3=exponencial; 4=Pareto;
- **ALM_gerafluxo**: permite a escolha do tipo de *trace* desejado. As opções são: 0=habilita somente *trace* "*outmsg.tr*", que permite a criação de gráficos por camada;

1=habilita também *trace* "*out.tr*" somente para pacotes recebidos e descartados, permitindo também a criação de gráficos por fluxo; 3=habilita "*out.tr*", porém completo (*trace-all*), ou seja, contendo todos pacotes recebidos, transmitidos, descartados, entrando e saindo das filas dos roteadores. A opção "3" ocasiona um tempo muito maior de simulação;

- **criaalm**: habilita criação de fluxos ALM;
- **criaudp**: habilita criação de fluxos UDP (do tipo CBR);
- **criatcp**: habilita criação de fluxos TCP.

Algumas simulações específicas exigem a modificação de algum parâmetro que não está automatizado na lista acima, e isso é feito diretamente no arquivo de simulação. Por exemplo, caso se queira testar uma fila *droptail* de 30 pacotes, deve-se alterar diretamente o arquivo de simulação, na parte de configuração de filas, pois esse valor não está disponível nos parâmetros definidos acima.

4.7.5 Metodologia para análise de resultados no simulador

Para analisar os resultados das simulações são utilizados os seguintes métodos de forma complementar, dependendo da necessidade no momento: gráfico de camadas, gráfico de banda, taxa de perdas, atraso médio de transmissão, total de pacotes transmitidos e NAM (*Network Animator*). Cada um desses itens será descrito com maiores detalhes a seguir.

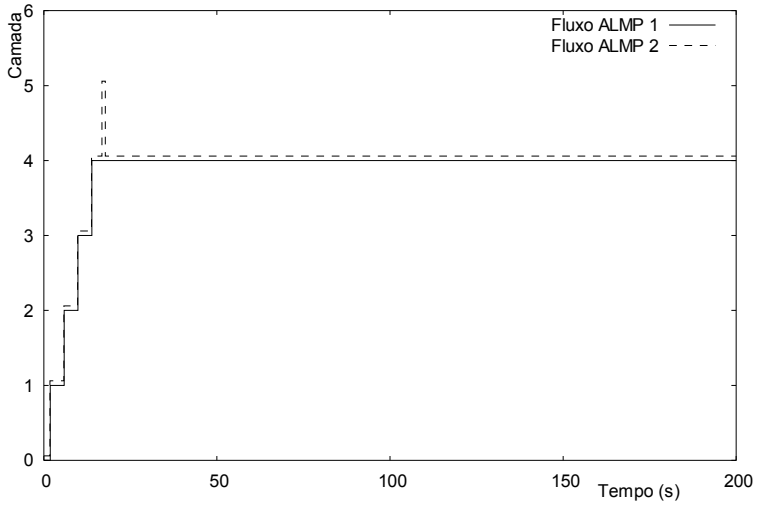


FIGURA 4.4 – Exemplo de gráfico de camadas

O *gráfico de camadas* mostra a variação das camadas para cada receptor ao longo do tempo, conforme pode ser visto na figura 4.4, onde se utilizou dois fluxos ALMP com camadas exponenciais concorrendo por 1 Mbit/s na topologia 2 da figura 4.3. Pode-se ver que cada receptor se cadastrou em quatro camadas, ou seja, o equivalente a 450 kbit/s (soma das camadas C1 a C4, ou seja, 30 + 60 + 120 + 240)

Para gerar o gráfico, criou-se um programa em linguagem *perl* que extrai as informações relevantes de um arquivo de mensagens gerado pelo simulador. O arquivo de mensagens gerado é o "*outmsg.tr*", e um extrato desse arquivo é visto no código a seguir.

```
v 1.3022 eval {… {1.302237 5 join-group -2147483642}}
v 1.3022 eval {…{ALM BWNova: 40000 BWatual: 32000.0 Perdas: 0}}
v 1.3022 eval {… {ALM node 5: Layer Mínimo}}
v 1.5509 eval {… {1.550934 4 join-group -2147483648}
```

A linha 1 indica que, no instante 1,30s, o receptor cadastrado no nodo 5 efetuou *join* no grupo multicast "-2147483642". A linha 4 indica que, no instante 1,55s, o receptor cadastrado no nodo 4 efetuou *join* no grupo "-2147483648". O programa "*gcamadanodo.pl*", escrito em *perl*, recebe como parâmetro o número do nodo a ser analisado, extraindo as informações de camada do arquivo de mensagens, gerando uma saída de "*tempo X camada*", compatível com os programas de geração de gráficos, como o *gnuplot*, por exemplo. Um exemplo de saída para o nodo 4 (equivalente ao receptor ALMP1) é mostrado a seguir.

```
0 0
1.550934 1
5.550934 2
9.550934 3
13.550934 4
200 4
```

Para aumentar a legibilidade e evitar sobreposição de gráficos quando vários receptores são analisados e estão na mesma camada, criou-se um programa em linguagem 'C', chamado "*og*" (*Offset Graphic*), que lê o arquivo de saída gerado pelo "*gcamadanodo.pl*" e aplica um deslocamento na camada. O tamanho do deslocamento é dado como parâmetro de entrada. O resultado de saída visto a seguir é para o nodo 5 (equivalente ao receptor ALMP2) após ser deslocado por 0,06. Nesse caso, quando o receptor estiver inscrito na camada 4, o gráfico vai mostrar a linha deste receptor como inscrito na camada 4,06 ao invés de 4,0. Um exemplo é visto na figura 4.4, onde os dois receptores estão inscritos na camada 4, porém é possível visualizar a linha do gráfico para os dois receptores, ou seja, não há sobreposição.

```
0 0.06
1.302237 1.06
5.302237 2.06
9.302237 3.06
13.302237 4.06
16.302237 5.06
17.302237 4.06
200 4.06
```

Para automatizar o processo, criou-se um *script* que executa o processo automaticamente após a simulação. O *script* é mostrado no código a seguir, e o resultado são os arquivos "*n4.dat*" e "*n5.dat*", prontos para serem abertos pela ferramenta de geração de gráficos.

```
perl gcamadanodo.pl 4 > n4.dat
perl gcamadanodo.pl 5 > n5.dt

og n5.dt n5.dat 0.06
del *.dt
```

Outro método de análise de resultados é o *gráfico de banda*, que mostra a variação da banda recebida por determinado receptor ao longo do tempo, conforme é

ilustrado na figura 4.5, onde se utilizou a mesma configuração do gráfico de camadas visto acima. O gráfico de banda é útil quando o objetivo é comparar protocolos diferentes, como TCP e ALM. Além disso, é importante quando se quer comparar tráfego variável e estabilidade nas camadas.

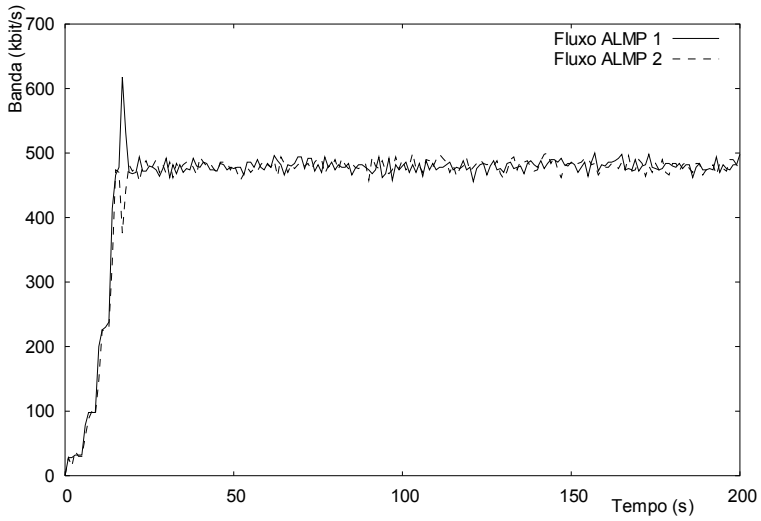


FIGURA 4.5 – Exemplo de gráfico de variação de banda

Para gerar o gráfico, criou-se um programa em linguagem *'C'* chamado "*almfluxo*", que extrai as informações relevantes de um arquivo de *trace* gerado pelo simulador. O arquivo de *trace* gerado é denominado "*out.tr*", e um extrato desse arquivo é visto no código a seguir. A primeira linha mostra um pacote sendo recebido (r) no instante 53,1 s, partindo do nodo 3 em direção ao nodo 4, do tipo CBR com 250 bytes. O número do fluxo é 100 (primeira camada do fluxo ALM 1). O primeiro dígito de cada linha indica o que está acontecendo com o pacote: "+" indica que entrou na fila de saída de um determinado nodo; "-" indica que saiu da fila e está trafegando em direção ao nodo destino; "r" indica que foi recebido pelo nodo destino; "d" significa que foi descartado.

```
r 53.1077 3 4 cbr 250 ------- 100 0.1 -2147483648.0 845 105608
r 53.1087 2 3 cbr 250 ------- 103 0.4 -2147483645.0 6665 105613
+ 53.1087 3 4 cbr 250 ------- 103 0.4 -2147483645.0 6665 105613
- 53.1087 3 4 cbr 250 ------- 103 0.4 -2147483645.0 6665 105613
+ 53.1094 1 2 cbr 250 ------- 202 1.3 -2147483640.0 3369 105626
- 53.1094 1 2 cbr 250 ------- 202 1.3 -2147483640.0 3369 105626
r 53.1097 3 4 cbr 250 ------- 103 0.4 -2147483645.0 6665 105613
r 53.1104 1 2 cbr 250 ------- 202 1.3 -2147483640.0 3369 105626
+ 53.1104 2 3 cbr 250 ------- 202 1.3 -2147483640.0 3369 105626
- 53.1104 2 3 cbr 250 ------- 202 1.3 -2147483640.0 3369 105626
r 53.1107 2 3 cbr 250 ------- 203 1.4 -2147483639.0 6701 105614
+ 53.1107 3 5 cbr 250 ------- 203 1.4 -2147483639.0 6701 105614
```

Como o arquivo "out.tr" no estilo original fica com um tamanho elevado (54 Mbytes na simulação de 200s acima), utilizou-se um método para que sejam gravados somente os pacotes recebidos ("r") e descartados ("d"). O código de simulação a seguir, em linguagem "*tcl*", mostra o processo, onde a função "*geratrace*" é chamada a cada linha de impressão entre o *nodeini* e o *nodefim*. A função "*geratrace*" escolhe quais linhas serão impressas no arquivo "*out.tr*".

```
set linktrace [$ns link $nodeini $nodefim]
$linktrace trace-callback $ns "$self geratrace"
```

Desta forma, utilizando esse tipo de filtragem, o tamanho do arquivo gerado é significativamente menor. Na simulação acima ficou em aproximadamente 12 Mbytes, ou seja, menos de 25% do tamanho original, que era de 54 Mbytes. Como as linhas geradas dessa forma possuem um espaço em branco inicial, criou-se um filtro para eliminar esse espaço, chamado "*convtrace*", que gera o arquivo chamado "out2.tr", sem o espaço indesejável.

Para automatizar o processo, criou-se um *script* que executa o processo automaticamente após a simulação (quando necessário). O *script* é mostrado no código a seguir, e o resultado, como pode-se ver pelo código, são os arquivos "*f4.dat*" e "*f5.dat*", prontos para serem abertos pela ferramenta de geração de gráficos.

```
convtrace
del out.tr
ren out2.tr out.tr
almfluxo 4 > f4.dat
almfluxo 5 > f5.dat
```

Quando se deseja obter o gráfico da banda em relação a um fluxo específico, como é o caso de fluxos TCP, utiliza-se outro programa em *perl* denominado "*gbandafluxo.pl*", que recebe dois parâmetros: o número do nodo e o número do fluxo. Para esse fluxo específico ele gera o gráfico de "*tempo x banda*", conforme descrito acima.

A eqüidade dos algoritmos concorrendo por determinado enlace será analisada através do conceito de *fairness index* (FI), que é a relação entre a taxa que o receptor está utilizando (TU) frente à taxa eqüitativa para este receptor (TE), ou seja, FI = TU / TE. A taxa utilizada TU é a soma das taxas de todas as camadas em que o receptor está inscrito, e a taxa eqüitativa é obtida através da análise de quanta banda o receptor deveria utilizar num determinado enlace (que é o total de banda no enlace dividido pelo número de fluxos compartilhando o enlace). Esse índice foi utilizado também para medir o índice de satisfação do receptor, e se considerou que, quanto mais próximo do valor "um" estiver FI, mais satisfeito estará o receptor. [LIU 2002]. Em alguns casos, o total de banda que o receptor deveria utilizar é a soma da taxa de cada camada que ele deveria estar inscrito. Esse conceito ficará mais claro na análise do resultado das simulações.

Outra informação utilizada para a análise dos resultados é a *taxa de perdas* de um determinado fluxo. Essa informação permite comparar a perda de pacotes entre diferentes sessões ALM ou entre diferentes protocolos. Na implementação do algoritmo já foi criada uma rotina que, quando detecta perda de pacotes, incrementa uma variável chamada "*nlost_*". Essa variável somente começa a ser incrementada após os receptores iniciarem as suas sessões, pois dessa forma elimina as perdas causadas pelo *flooding* inicial do multicast, que não estão relacionadas ao algoritmo em si. Ao final da simulação, basta acessar o valor dessa variável para cada receptor ALM, conforme mostra o código a seguir.

```
Numperdas = ALM_total_loss()
```

O *total de pacotes recebidos* é obtido da mesma forma que as perdas, ou seja, a partir de um código inserido no algoritmo do ALM que, após o início das sessões dos receptores, incrementa uma variável a cada pacote recebido pelo receptor. Essa variável é lida ao final da simulação, conforme código a seguir.

```
Totpackets = ALM_total_packets_delivered()
```

Para obter a taxa de perdas utilizou-se a seguinte fórmula, onde TP é o total de perdas e TR é o total de pacotes recebidos. O total de pacotes transmitidos é "TR + TP".

$$perdas(\%) = \frac{TP * 100}{TR + TP}$$

Um exemplo de saída de simulação sobre a adaptabilidade do ALMP em relação à camadas exponenciais é ilustrado no texto a seguir, para quatro receptores, R1 a R4:

```
R1: vazão: 1860 kbit/s; tot: 462680 pacotes; perdas: 0 (0%)
R2: vazão: 918 kbit/s; tot: 228526 pacotes; perdas: 387 (0,17%)
R3: vazão: 427 kbit/s; tot: 106289 pacotes; perdas: 438 (0,41%)
R4: vazão: 89 kbit/s; tot: 22281 pacotes; perdas: 711 (3,09%)
```

A partir da saída da simulação obtém-se diretamente a taxa de perdas para cada receptor, que foi: 0%, 0,17%, 0,41% e 3,09%.

O *atraso médio da transmissão* é obtido através do monitoramento da fila no enlace principal, conforme mostra o código a seguir, em linguagem "*tcl*", onde os nós "*rot1*" e "*rot2*" são por onde passa todo o tráfego, sendo o ponto de congestionamento do sistema.

```
set fmon_ [$ns makeflowmon Fid]
$ns attach-fmon [$ns link $node($rot1) $node($rot2)] $fmon_
```

O código no simulador em linguagem "*tcl*" que permite a visualização do atraso nesse ponto de congestionamento é visto a seguir, onde o vetor "*fids*", referenciado na terceira linha do código abaixo, contém todos os fluxos que se pretende analisar.

```
set fcl [$fmon_ classifier]
set total 0.0
foreach i $fids {
  set flow [$fcl lookup auto 0 0 $i]
  if { $flow != "" } {
    set dsamp [$flow get-delay-samples]
    set mean [$dsamp mean]
    set total [expr $total + $mean]
    puts "Atraso médio (ms) no fluxo $i = $mean*1000]"
  }
}
```

Visualiza-se o total de perdas, total de pacotes, vazão e atraso através de um único arquivo de saída, que é gerado em todas as simulações. O exemplo desse arquivo para a simulação em questão (figura 4.4 ou figura 4.5) é visto a seguir.

```
Transmissores - inicio em: 0.008473 e 0.080415
Tempo de inicio do receptor = 1.2341755462038217
Tempo de inicio do receptor = 1.5981431475831862
Atraso médio (ms) no fluxo 100 = 3.409440
Atraso médio (ms) no fluxo 101 = 3.401710
```

```
Atraso médio (ms) no fluxo 102 = 3.402710
Atraso médio (ms) no fluxo 103 = 3.394940
Atraso médio (ms) no fluxo 104 = 16.451000
Atraso médio (ms) no fluxo 105 = 13.977100
Atraso médio (ms) no fluxo 200 = 3.460350
Atraso médio (ms) no fluxo 201 = 3.408710
Atraso médio (ms) no fluxo 202 = 3.421510
Atraso médio (ms) no fluxo 203 = 3.421200
Atraso médio (ms) no fluxo 204 = 12.436100
Atraso médio (ms) no fluxo 205 = 12.806200

Atraso médio total (ms) = 6.915914

Receptor 0: Vazão:460 kbit/s; T:45822 pacotes; P:150(0.32%)
Receptor 1: Vazão:458 kbit/s; T:45603 pacotes; P:77 (0.17%)
```

Assim, após a simulação, pode-se verificar, a partir do arquivo de saída, os seguintes valores obtidos na simulação:

- **Tempo de início dos transmissores**: aleatório entre 0 e 1s – pode-se ver no arquivo de saída que, nessa simulação, os dois transmissores iniciaram em 0,008s e 0,08s;
- **Tempo de início dos receptores**: aleatório entre o parâmetro "*inireceptores*" e um segundo depois – instantes 1,23s e 1,59s;
- **Atraso médio na fila para cada um dos fluxos envolvidos**: entre 3ms e 16ms;
- **Atraso médio total na fila**: 6,9 ms;
- **Vazão dos receptores**: 460 kbit/s e 458 kbit/s. Pode-se inferir que está coerente com o esperado, pois para 1Mbit/s de banda no enlace principal, cada receptor deve ficar com 500 kbit/s (na verdade, 450 kbit/s, pois cada receptor se cadastra em quatro camadas);
- **Total de pacotes recebidos**: 45.822 e 45.603 pacotes, respectivamente;
- **Total de perdas**: 150 e 77 pacotes. Isso perfaz um total de 0,32% e 0,17% de perdas, respectivamente.

Outra informação necessária utilizada para a análise de estabilidade do algoritmo foi definida nesta Tese como a variação de camadas por minuto (VCM). Para verificar a VCM de determinado fluxo que utiliza adaptação em camadas, criou-se um programa em *perl* denominado *vcm.pl* que analisa o arquivo "*outmsg.tr*" e calcula o valor de VCM para aquele fluxo.

Assim, se um algoritmo não mudar de camada durante toda a simulação, o número de variações de camada por minuto (VCM) será zero. Se variar uma vez por segundo, o VCM será 60. Vale observar que cada tentativa fracassada de subir camada provoca duas variações: uma para subir e uma para voltar à camada anterior.

Para efeitos de comparação com fluxos TCP, criou-se um programa em linguagem C denominado "*vcmtcp.exe*" que calcula o valor VCM equivalente para esse fluxo caso o mesmo utilizasse transmissão em camadas. Assim, cada vez que a banda utilizada por determinado fluxo TCP ultrapassar a banda utilizada por uma camada ALM, será considerada uma variação. Por exemplo, supondo que o ALM esteja utilizando camadas de 100 kbit/s e o fluxo TCP esteja estável em 500 kbit/s. Quando o fluxo TCP ultrapassar 600 kbit/s (uma camada equivalente no ALM), será considerada uma variação. Quando o fluxo TCP voltar a ficar abaixo de 600 kbit/s, será considerada

outra variação.

Além das metodologias de análise de resultados vistas anteriormente, é possível utilizar o *Network Animator*, que permite a visualização do que está acontecendo na simulação em questão. Em situações em que existe um sintoma que não se consegue descobrir a causa, essa ferramenta é útil para permitir uma análise visual da simulação. A figura 4.6 mostra a topologia utilizada para as simulações anteriores, no instante 16s. Na figura, pode-se ver os pacotes passando e as filas nos roteadores. Além disso, existe uma janela para receber mensagens do algoritmo, como *join*, *leave*, variáveis específicas, e assim por diante. O *Network Animator* foi utilizado inúmeras vezes para resolução de problemas durante o processo de desenvolvimento dos algoritmos propostos nesta Tese.

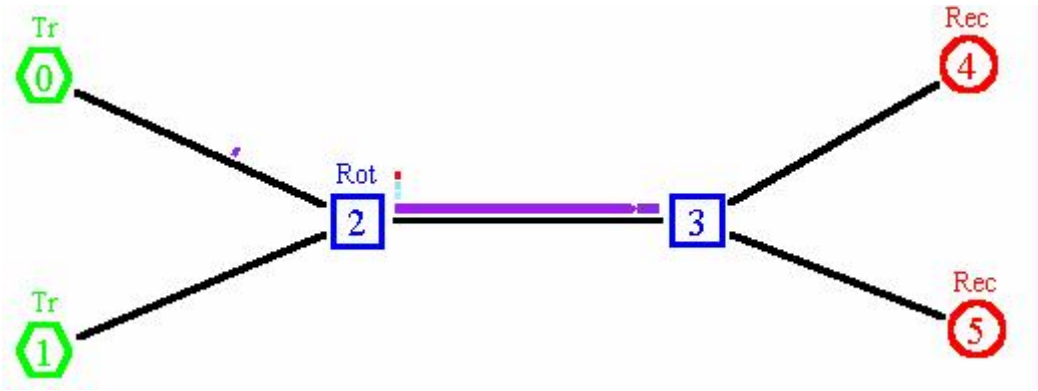


FIGURA 4.6 – Exemplo do *Network Animator*

4.8 Definição das simulações

Este item tem por objetivo descrever o conjunto de simulações efetuadas nos capítulos 5 e 6. Serão especificados parâmetros relevantes para analisar o algoritmo ALM. As métricas foram definidas no item anterior, e são: adaptabilidade, escalabilidade, tempo de adaptação, estabilidade, eqüidade de tráfego e taxa de perdas.

Por motivos práticos, definiu-se que não seriam feitas simulações envolvendo a variação de todas as combinações de parâmetros, pois o número de simulações seria muito grande, portanto, escolheu-se variar os parâmetros mais significativos em cada métrica a ser simulada.

A seguir serão detalhados os experimentos efetuados para cada métrica. Vale lembrar que alguns experimentos servem para mais de uma métrica. Por exemplo, no teste de adaptabilidade é possível verificar o tempo de adaptação. Todos experimentos obtiveram a taxa de perdas e a quantidade de bytes transferidos para cada receptor, bem como o atraso médio nas filas, conforme explicado no item anterior.

4.8.1 Adaptabilidade em ambientes heterogêneos

Para testar a adaptabilidade do algoritmo em ambientes heterogêneos, foi utilizada a topologia 1, com um único fluxo. Os seguintes experimentos foram realizados com a utilização de pares de pacotes e sem a utilização de pares de pacotes.

Os seguintes parâmetros são especificados para cada um deles:

- **Adaptabilidade 1**: a) número de fluxos ALM = 1; b) r = 1 (receptor por bloco), perfazendo um total de 4 receptores; c) banda B1 = 2,1 Mbit/s, B2 = 1,05 Mbit/s, B3 = 525 kbit/s, B4 = 105 kbit/s; d) atraso A1 a A4 = 10ms; e) camadas exponenciais padrão; f) tamanho do pacote = 500 bytes; g) tipo de fila = *droptail* com 20 pacotes; h) randomização alta (±0,5s);
- **Adaptabilidade 2**: igual ao "1", porém com atraso A1 a A4 = 1ms;
- **Adaptabilidade 3**: igual ao "1", porém com camadas exponenciais iguais ao do VEBIT;
- **Adaptabilidade 4**: igual ao "1", porém com camadas de 50 kbit/s;
- **Adaptabilidade 5**: igual ao "1", porém com camadas de 100 kbit/s;
- **Adaptabilidade 6**: igual ao "1", porém com tamanho do pacote de 250 bytes;
- **Adaptabilidade 7**: igual ao "1", porém com tamanho do pacote de 1000 bytes;
- **Adaptabilidade 8**: igual ao "1", porém com tipo de fila *droptail* de 60 pacotes;
- **Adaptabilidade 9**: igual ao "1", porém com tipo de fila RED 10/20;
- **Adaptabilidade 10**: igual ao "1", porém com tipo de fila RED 10/20 e camadas de 100 kbit/s;
- **Adaptabilidade 11**: igual ao "1", porém com tipo de fila RED 20/60;
- **Adaptabilidade 12**: igual ao "1", porém sem randomização na transmissão dos pacotes (CBR puro).

Quando algum dos experimentos gerou uma diferença significativa em relação ao resultado esperado, o parâmetro que causou essa diferença foi estudado em detalhes, e o experimento foi repetido com a variação de outros parâmetros. Além disso, efetuaram-se novas simulações a fim de descobrir uma explicação sobre o motivo das divergências nos resultados.

4.8.2 Escalabilidade

Para análise de escalabilidade, também foi utilizada a topologia 1, porém, com muitos receptores. Nesse caso, a proposta foi utilizar os parâmetros do experimento mais significativo obtido no item anterior e variar o número de receptores. Os seguintes experimentos foram realizados:

- **Escalabilidade 1**: igual ao "adaptabilidade 1", porém com r = 5 (receptores por bloco), perfazendo um total de 20 receptores;
- **Escalabilidade 2**: igual ao "Escalabilidade 1", porém com r = 25 (receptores por bloco), perfazendo um total de 100 receptores;
- **Escalabilidade 3**: igual ao "2", porém com camadas de 100 kbit/s;
- **Escalabilidade 4**: igual ao "2", porém com fila RED 10/20.

4.8.3 Eqüidade de tráfego

Conforme definição no item 2.5, existem dois tipos de eqüidade de tráfego: a eqüidade entre sessões do próprio tráfego concorrendo entre si (*fairness*), e eqüidade com tráfego concorrente de outro tipo (*friendliness*). A análise de eqüidade foi feita com a topologia 2, que possui um enlace central onde o tráfego congestiona, permitindo

verificar a adaptação de cada fluxo existente do algoritmo. Os seguintes experimentos foram realizados:

- **Eqüidade 1**: a) total de 10 fluxos (5 ALM e 5 TCP); b) banda B igual a aproximadamente 500 kbit/s por receptor; c) atraso A = 10ms; d) camadas exponenciais; e) tamanho do pacote = 500 bytes; f) tipo de fila = *droptail* com 20 pacotes; g) randomização alta (±0,5s); h) tamanho da janela TCP e ALMTF = 10 pacotes;
- **Eqüidade 2**: igual ao "1", porém com total de 2 fluxos (1 ALM e 1 TCP);
- **Eqüidade 3**: igual ao "1", porém com total de 20 fluxos (10 ALM e 10 TCP);
- **Eqüidade 4**: igual ao "1", porém com taxas exponenciais iguais ao VEBIT e banda B de 350 kbit/s por receptor;
- **Eqüidade 5**: igual ao "1", porém com taxas de 100 kbit/s;
- **Eqüidade 6**: igual ao "1", porém com tamanho de pacote = 250 bytes;
- **Eqüidade 7**: igual ao "1", porém com tamanho de pacote = 1000 bytes;
- **Eqüidade 8**: igual ao "1", porém com fila RED 10/20;
- **Eqüidade 9**: igual ao "1", porém com atraso A = 1ms;
- **Eqüidade 10**: igual ao "2" (total de dois fluxos), porém com um fluxo iniciando 100s antes de outro. Se forem de tipos diferentes (ALM e TCP), devem ser feitas duas simulações, a primeira com o ALM iniciando antes, e a segunda com o TCP iniciando antes;
- **Eqüidade 11**: igual ao "1", porém com um total de cinco fluxos, cada fluxo iniciando 100s depois do outro;
- **Eqüidade 12**: igual ao "1", porém com um total de cinco fluxos, cada fluxo iniciando 100s depois do outro e utilizando camadas de 100 kbit/s;
- **Eqüidade 13**: igual ao "1", porém com geração de tráfego VBR tipo exponencial em vez de CBR (somente para o ALMP);
- **Eqüidade 14**: igual ao "1", porém com geração de tráfego VBR tipo pareto em vez de CBR (somente para o ALMP);
- **Eqüidade 15**: igual ao "2" (total de dois fluxos), concorrendo com um fluxo UDP de 500 kbit/s por uma banda de 1 Mbit/s. Um fluxo UDP inicia em zero e termina no instante 200s, e outro fluxo UDP inicia no instante 600s e vai até o final da simulação;
- **Eqüidade 16**: variação do parâmetro *windowsize* para o dobro do seu valor. Testar com 2 fluxos ALMTF e 2 fluxos TCP (somente para o ALMTF).

4.8.4 Estabilidade

A estabilidade do algoritmo foi analisada através dos resultados obtidos nos experimentos anteriores. Para facilitar comparações, foi utilizado um valor relacionado ao número de variações de camada por minuto (VCM), pois considera-se que a variação de camada pode causar um distúrbio no sinal de vídeo. A definição de VCM foi detalhada no item 4.7.5.

4.8.5 Tempo de adaptação

O tempo de adaptação representa o número de segundos que o algoritmo demora

até atingir o regime permanente, ou seja, até estar estabilizado. Para transmissão de vídeo, a adaptação não necessita ser muito rápida, pois está se tratando de reações humanas assistindo a uma transmissão de vídeo.

Os experimentos realizados anteriormente são suficientes para descobrir o tempo de adaptação do algoritmo.

4.8.6 Taxa de perdas

A taxa de perdas foi analisada em todos os experimentos anteriores, seguindo a metodologia descrita no item 4.7.5.

5 ALMP: ALM for Private Networks

No âmbito do SAM, descrito no capítulo 4, foram definidos dois algoritmos para o módulo de controle de congestionamento: o ALMP e o ALMTF. O objetivo deste capítulo é o detalhamento do algoritmo ALMP.

Uma prática que está se tornando comum atualmente é a criação de uma rede privativa virtual (VPN) com reserva de banda para a transmissão do sinal multimídia. Essa reserva pode ser temporária (como no caso da transmissão de uma conferência) ou permanente (transmissão de canais de TV). Assim, no caso de uso do SAM para a transmissão desses sinais, é necessário apenas que ele transmita de forma eqüitativa entre as próprias sessões, pois não haveria tráfego TCP concorrente.

O algoritmo ALMP, feito no âmbito desta Tese, tem seu foco na transmissão através de uma rede privativa, ou seja, o algoritmo é liberado da concorrência do TCP, podendo ter seus parâmetros ajustados para que se obtenha uma maior estabilidade de transmissão. Este algoritmo foi denominado ALMP (ALM através de redes privativas), e teve uma série de publicações descrevendo a ferramenta e sua implementação [ROE 2002a], [ROE 2002b], [ROE 2002c] e [ROE 2003a]. O código fonte para o simulador, bem como o manual de instalação e resultado de todas simulações efetuadas pode ser obtido no CD anexo a esta Tese ou em http://www.inf.unisinos.br/~roesler/tese.

Uma rede privativa poderia ser criada dentro da Internet tradicional através de um mecanismo de QoS, como o Diffserv. Caso fosse padronizado um número DS (*Differentiated Services*) específico para sessões ALMP, e todos roteadores estivessem configurados para utilizar uma fila diferente nesse tipo de pacote, todo tráfego ALMP seria separado do tráfego tradicional. Essa abordagem seria fundamental para criar um mecanismo de transmissão de tráfego multimídia sem a interferência da agressividade do TCP, ou seja, a Internet estaria dividida em duas redes separadas, uma para tráfego confiável de dados (TCP) e outra para tráfego multimídia (ALMP).

Na verdade, é possível modificar os parâmetros do algoritmo para concorrer com tráfego TCP, porém, isso diminuiria sua estabilidade. Além disso, o TCP não gera tráfego eqüitativo para receptores com diferentes RTT (*Round Trip Time*), e o ALMP não segue essa limitação, pois não depende do RTT.

A variável que controla a agressividade do algoritmo foi denominada *ALM_EI_delay* (onde, EI significa *Execution Interval*), ou seja, o algoritmo é executado a cada intervalo dado por essa variável.

A idéia básica do algoritmo é manter uma variável, denominada *bwshare*, que deve refletir a largura de banda disponível na rede para aquele receptor (de forma eqüitativa entre todos receptores), e deve fazer isso sem a necessidade de comunicação com os outros receptores. Para conseguir isso, o lema do algoritmo é "aumentar mais a banda para os tráfegos que têm menos" e "diminuir mais a banda para os tráfegos que têm mais", resultando num equilíbrio entre os receptores. Assim, o receptor cuja variável *bwshare* é menor vai ter um incremento maior (e um decremento menor) em relação ao receptor concorrente.

A adaptação no ALMP acontece no receptor, e este não necessita qualquer comunicação com o transmissor para adaptar-se, assim, o algoritmo é massivamente escalável para um grande número de receptores.

O presente capítulo está dividido da seguinte forma: o item 5.1 apresenta o detalhamento do algoritmo de controle de congestionamento ALMP. No item 5.2, as simulações serão mostradas, enquanto o item 5.3 descreve a implementação realizada e seus testes iniciais para análise de adaptação num ambiente real controlado. Finalmente, no item 5.4, será feita uma conclusão sobre os pontos positivos e negativos do algoritmo.

5.1 Detalhamento do algoritmo ALMP

O algoritmo é executado no receptor a intervalos regulares de tempo, e o mesmo pode estar inscrito em tantas camadas quanto a variável *bwshare* permitir, porém, o algoritmo limita a um *join* por vez a fim de evitar aumento excessivo de banda. Entretanto, é possível efetuar mais de um *leave* na mesma execução, caso necessário. Os principais parâmetros do algoritmo são:

- **Intervalo de Execução**: tempo entre duas execuções consecutivas do algoritmo. Em cada execução do algoritmo, a variável *bwshare* é incrementada ou decrementada, e o receptor decide fazer *join* ou *leave* baseado nesta variável;
- **Taxa de incremento**: quantidade de banda adicionada à variável *bwshare* em cada intervalo de execução quando não ocorrem perdas;
- **Taxa de decremento**: quantidade de banda subtraída da variável *bwshare* em cada intervalo de execução quando ocorrem perdas.

Um resumo do código do algoritmo ALM será visto a seguir e explicado em seguida com o objetivo de mostrar os pontos principais da implementação. Os pontos principais que serão explicados foram numerados para facilitar a referência durante a explicação.

```
Init ()  ;# rotina de inicialização das variáveis          (1)
  add-layer (camada 1) ;# começa na camada 1
  cngmode = start-state
  AlmEI() ;# executa agora e cada Intervalo de Execução

AlmEI () ;# código executado cada Intervalo de Execução
  # repete essa função cada ALM_EI_delay
  $ns_ at [T_atual + ALM_EI_delay] "AlmEI()"                (2)

  switch [cngmode]
    "start-state"                                           (3)
      Se (numloss==0)
        bwshare = bwshare + (bwshare / 4)
      Senão ;# perdas detectadas
        # divide banda e vai para fase steady-state
        bwshare = bwshare*0,7
        cngmode = steady-state

    "steady-state"  ;# regime permanente                   (4)
      qincr = (17-ln(bwshare))*500 ;# incremento
```

```
      qdecr = bwshare * 5% ;# quantidade de decremento
      Se (numloss==0)
          bwshare = bwshare + qincr ;# sobe "qincr" sem perdas
      Senão
          bwshare = bwshare - qdecr ;# desce "qdecr" com perdas

# Quando desce camada espera uma execução para estabilização (5)
  Se (desceucamada == 1)
    desceucamada = 0
    Se (numloss > 0)
      numloss = 0
      bwshare = (bwused + bwlevelup) / 2

# decide se receptor pode fazer join                              (6)
  Se (bwshare > bwlevelup)
    Se (TerminouEsperaRandomica())
      # faz join na próxima camada (add-layer)
      add-layer (bwshare)

  # decide se receptor deve fazer leave                           (7)
  Se (bwshare < bwused)
    # leave em tantas camadas quanto necessário (leave-layer)
    leave-layer (bwshare)
    # configura banda na média entre camadas adjacentes           (8)
    bwshare = (bwused + bwlevelup)/2
```

Inicialmente, a sub-rotina *init* é chamada (número "1" no código). Ela inicializa algumas variáveis do ALMP, faz *join* na camada 1 e chama a sub-rotina *ALM_EI*, que é o algoritmo a ser executado em cada Intervalo de Execução (conforme explicado anteriormente). A sub-rotina *ALM_EI* é repetida consecutivamente de acordo com a variável global *ALM_EI_delay*, que fornece o intervalo entre duas execuções consecutivas do algoritmo (número "2" no código).

A mudança da variável *ALM_EI_delay* modifica o tempo de adaptação do receptor e a estabilidade do sistema. Com um tempo relativamente pequeno (por exemplo, 100ms), o sistema alcança sua fatia eqüitativa no compartilhamento mais rapidamente, mas não é tão estável como seria com a execução em Intervalo de Execução maiores, como, por exemplo, 1s. Isso acontece pois quanto mais vezes o algoritmo é executado por minuto, mais vezes vai incrementar ou decrementar a variável *bwshare*.

A forma empregada pelo receptor para atingir rapidamente sua largura de banda eqüitativa com os outros receptores (representado pela variável *bwshare*) e permanecer estável depois disso é através da divisão do algoritmo em duas fases: fase *start-state* e fase *steady-state*, conforme explicado a seguir.

Durante a fase *start-state* (número "3" no código), o algoritmo tenta alcançar rapidamente sua banda eqüitativa com os fluxos concorrentes, e faz isso incrementando rapidamente *bwshare* até a detecção de perdas (ou informação de congestionamento, se usando ECN). Quando perdas são detectadas, o algoritmo considera que sua banda passou o valor eqüitativo (por causa das perdas), assim, diminui *bwshare* por um fator (no caso, 30%) e vai para a fase *steady-state*. Existe uma proteção que impede o algoritmo de subir mais de uma camada por vez, buscando evitar um número muito alto de perdas caso o receptor se inscreva em várias camadas, gerando um tráfego muito

acima do que a rede permite.

A detecção das perdas é feita com base no número de seqüência existente em cada pacote transmitido (semelhante ao protocolo RTP (*Real Time Transport Protocol*)). Caso o receptor detecte em qualquer camada uma falha na seqüência recebida, ele incrementa uma variável de perda.

Durante a fase *steady-state* (número "4" no código), o lema do algoritmo é "dar mais largura de banda para quem tem menos", e "tirar mais largura de banda de quem tem mais". Isso significa que, se há dois fluxos compartilhando o mesmo gargalo, aquele com menos banda vai incrementar mais rapidamente em relação ao outro, e ambos vão tender a um ponto de equilíbrio. No caso de acontecerem perdas, o fluxo com maior largura de banda vai decrementar mais rapidamente em relação ao outro, também gerando uma tendência de equilíbrio. A quantidade de incremento (*qincr*) e de decremento (*qdecr*) serão detalhados adiante.

Quando o receptor efetua *leave*, após a adaptação devido às perdas, ele espera um tempo equivalente a uma execução do algoritmo para fins de estabilização, pois a diminuição na camada pode não ter efeito imediato. Isso é mostrado no número "5" do código e detalhado no item 5.1.1.

Tanto a fase *start-state* como a fase *steady-state* mudam a variável *bwshare*, incrementando ou decrementando de acordo com o nível de congestionamento medido (visto através das perdas). Depois disso, a variável *bwshare* é usada para descobrir se o receptor pode fazer *join* na próxima camada (número "6" no código) ou se ele deve fazer *leave* em uma ou mais camadas (número "7" no código). A variável *bwused* representa a quantidade de banda atualmente utilizada pelo receptor, que é a soma da banda de cada camada na qual o receptor está inscrito. A variável *bwlevelup* representa a quantidade de banda que seria necessária pelo receptor caso o algoritmo fizesse *join* na próxima camada.

Antes do receptor efetuar *join* em uma camada, ele espera um número randômico de execuções do algoritmo (número "6" no código). O objetivo é evitar que vários receptores que convergiram para a mesma banda cheguem à conclusão que podem efetuar *join* no próximo grupo multicast e façam isso ao mesmo tempo, causando instabilidade a todos. Isso é detalhado nos itens 5.1.2 e 5.1.3.

Quando um receptor faz *leave* numa camada, ele configura a variável *bwshare* para a média aritmética entre as duas camadas adjacentes (superior e inferior, conforme apresentado no número "8" no código). Isso evita que a variável *bwshare* fique próxima da banda necessária para subir camada novamente, que poderia permitir ao receptor fazer outra tentativa de *join* em poucos Intervalos de Execução, tornando o sistema menos estável.

É importante perceber que para adaptar-se eqüitativamente um receptor não precisa conhecer a variável *bwshare* dos outros receptores a fim de que a mesma seja semelhante. Isso acontece pois o cálculo realizado leva a uma sincronização e equilíbrio dessa variável entre os receptores, conforme explicado a seguir.

Para que diferentes receptores tendam a um ponto de equilíbrio no uso da banda, a quantidade de incremento (*qincr*) e de decremento (*qdecr*) são proporcionais à largura de banda permitida ao receptor (variável *bwshare*). A quantidade de decremento foi

especificada em 5% da largura de banda atual, assim, por exemplo, se um receptor tem *bwshare* = 500 kbit/s e outro tem *bwshare* = 100 kbit/s, e perdas aconteçam a ambos, ambos vão decrementar 5%. O decréscimo do primeiro receptor será de 25 kbit/s, e o do segundo será 5 kbit/s. Isso vai reduzir a diferença entre a variável *bwshare* de ambos (de 400 kbit/s para 380 kbit/s). O mesmo acontece ao incrementar a variável, mas em escala menor. Isso explica o motivo dos receptores não necessitarem de comunicação para possuírem uma banda eqüitativa entre eles.

A fórmula da quantidade de incremento é dada pela equação a seguir, que permite um incremento maior com baixa banda, e um incremento menor com alta banda, gerando uma tendência para um ponto de equilíbrio. Por exemplo, se *bwshare* é 50 kbit/s, *qincr* será de aproximadamente 3 kbits, levando *bwshare* para 50,3 kbit/s. Se *bwshare* é 900 kbit/s, *qincr* será de aproximadamente 1,6 kbits, levando *bwshare* para 901,6 kbit/s. Isso provoca uma aproximação entre as duas possibilidades de banda.

$$qincr = (17 - \ln bwshare) * 500$$

A fórmula foi escolhida para transmissões multimídia com banda total de até 10 Mbit/s. A partir deste ponto existe uma proteção para evitar incrementos negativos. A equação se mostrou adequada aos cenários simulados, com uma largura de banda destinada ao tráfego multimídia atual, até 2 Mbit/s, entretanto, deve ser ajustada para cenários com altas taxas de codificação, pois torna muito lento o processo de adaptação para *bwshare* muito grande. A mesma observação vale para a taxa de decremento, que foi fixa em 5%. Nos novos cenários, deve-se observar sempre a necessidade da convergência entre os diferentes fluxos, fazendo com que, no caso de perdas, os receptores que recebem mais banda diminuam mais em relação aos que recebem menos banda. As taxas atuais foram escolhidas visando uma adaptação num tempo menor que poucos minutos, sem alterações bruscas a fim de manter a estabilidade no sistema.

Conclui-se, então, que a banda permitida a cada receptor vai tender para um ponto de equilíbrio, tanto se os receptores estiverem localizados em diferentes sessões ou localizados na mesma sub-rede.

A figura 5.1 mostra graficamente a evolução da variável *bwshare*. Na figura, há 5 camadas exponenciais (30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s). O eixo Y representa o total de banda usada pelo receptor, que é a soma das larguras de banda de cada camada na qual o receptor está inscrito.

As linhas mais grossas na figura indicam a quantidade de banda que o receptor 1 (representado pela linha contínua) e o receptor 2 (representado pela linha pontilhada) estão utilizando. Essa quantidade é relacionada diretamente com a quantidade de camadas que cada receptor se inscreveu. Pode-se ver que os receptores estão estáveis quando inscritos nas primeiras quatro camadas, e as tentativas de se inscrever na camada de número 5 geraram perdas.

A largura de banda permitida aos receptores é incrementada ou decrementada a cada Intervalo de Execução. Isso é representado na figura através das linhas finas, sendo que a linha contínua representa o receptor 1 e a linha pontilhada o receptor 2. Essas linhas representam os resultados obtidos para a variável *bwshare* em cada Intervalo de Execução. Observa-se que, quanto maior a banda (caso do receptor 1 até o instante 100s), menor a taxa de subida, e as duas linhas tendem ao equilíbrio. Os instantes de

tempo na figura foram escolhidos apenas de forma representativa, pois na prática esse tempo será dado pelo funcionamento do algoritmo, conforme será detalhado nos gráficos referentes ao resultado das simulações.

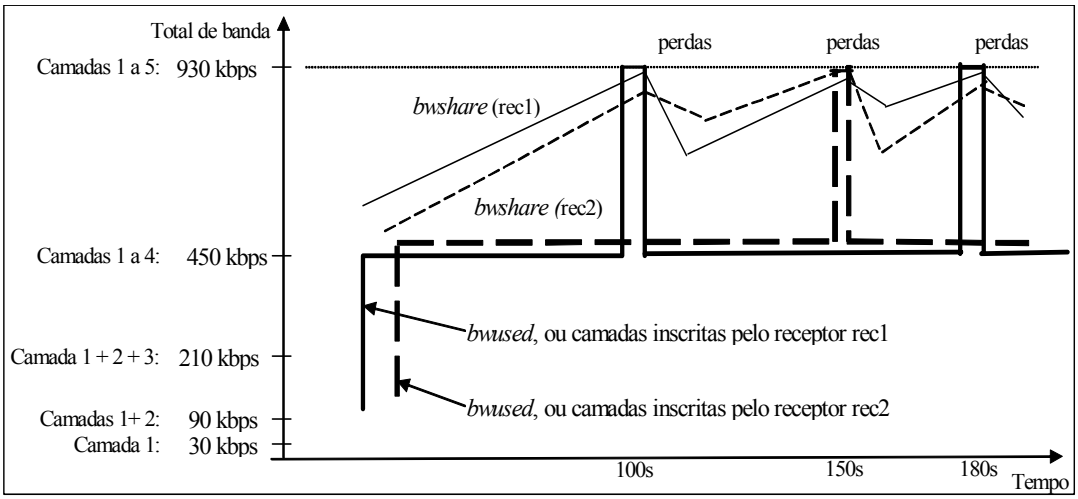


FIGURA 5.1 – Visão geral do algoritmo ALMP em termos da variável *bwshare*

Caso a largura de banda permita (linhas finas na figura atingindo o valor de *bwlevelup*, ou seja, uma largura de banda suficiente para o receptor se inscrever em uma nova camada), o receptor vai se inscrever em outra camada. Caso essa tentativa gere perdas, o receptor vai deixar uma camada (ver representação na figura 5.1). Normalmente as perdas acontecem para ambos os receptores, e eles diminuem o valor de *bwshare*, entretanto, o receptor com mais banda vai decrementar mais o valor de *bwshare*, conforme descrito anteriormente e representado na figura.

Como pode ser inferido, se a variável *ALM_EI_delay* fosse maior, o algoritmo seria executado menos vezes por segundo, e a variável *bwshare* seria incrementada um menor número de vezes. Isso deixaria o sistema mais lento, levando mais tempo para que o receptor tentasse subir camada. Por outro lado, a estabilidade iria aumentar.

5.1.1 Tempo de estabilização

Depois de desincrever-se de uma camada, o receptor deve esperar por um tempo antes de voltar a modificar *bwshare*. Isso serve para esperar a rede se estabilizar, evitando uma reação exagerada à perdas. O tempo de estabilização foi estipulado empiricamente em um segundo, que é o próprio Intervalo de Execução. Utilizando valores muito menores que 1s não daria tempo suficiente após um *leave* para os roteadores efetuarem completamente o *prune* na árvore multicast, e durante esse tempo as perdas continuariam acontecendo. Utilizando valores muito acima de 1s deixaria a reação do sistema lenta nos casos de congestionamento prolongado.

5.1.2 Receptores inscritos na mesma sessão

A abordagem do ALMP para divisão eqüitativa de banda em receptores localizados na mesma sub-rede que estão inscritos na mesma sessão é a seguinte: todos receptores localizados na mesma rede tendem a ter aproximadamente a mesma variável *bwshare*. Assim, quando *bwshare* permitir a um receptor efetuar *join* em uma nova

camada, o mesmo deve estar acontecendo nos outros (com uma pequena diferença de tempo). Se não existisse qualquer mecanismo de prevenção, poderia acontecer de todos receptores efetuarem *join* quase ao mesmo tempo na camada superior, levando mais tempo para que todos detectem congestionamento e efetuem *leave*.

Para minimizar esse conflito devido ao sincronismo, antes de se cadastrar numa camada o algoritmo gera um número randômico entre 0 e 9 (Intervalos de Execução), entrando num estado de espera e efetuando o *join* somente após um número de execuções igual ao número randômico gerado. O receptor que gerou o menor número randômico vai efetuar o *join*, enquanto os outros estarão aguardando. Se acontecerem perdas na rede durante esse tempo, a variável *bwshare* dos receptores que estão no estado de espera é reduzida para a média entre a camada superior e atual (fórmula igual à vista no número "8" do código do algoritmo) e o receptor nem tenta se inscrever na nova camada.

Assim, quando um dos receptores efetuar *join* na camada superior (e causar perdas), os outros receptores que estão em estado de espera irão sentir essas perdas (pois elas acontecem em todas as camadas), e irão aprender com isso, decrementando a variável *bwshare* e deixando a zona que permite fazer *join* na camada superior. Esta estratégia não garante que dois receptores não tentem efetuar *join* ao mesmo tempo. O número randômico de 0 a 9 permite que, na média, somente 10% dos receptores efetuem tentativas simultâneas de *join*.

A alternativa sugerida funciona para receptores na mesma sub-rede e também para aqueles receptores que dividem o mesmo gargalo, conforme ilustra a figura 5.2 (mesmo em diferentes sub-redes). Isso é possível porque os receptores tendem a possuir uma variável *bwshare* similar, que é garantido através do próprio funcionamento do algoritmo. Assim, se R1 efetua uma tentativa de *join* (pois gerou um número randômico menor), pode fazer com que R2 aprenda com sua tentativa.

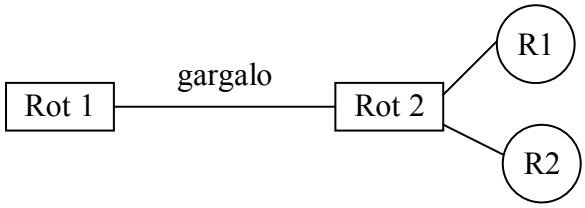


FIGURA 5.2 – Receptores em sub-redes diferentes compartilhando um gargalo

5.1.3 Receptores inscritos em sessões diferentes

No caso de sessões diferentes, cada uma delas transmitindo diferentes fluxos multimídia codificados em camadas (cada qual no seu próprio grupo multicast), o funcionamento do ALMP é o seguinte: os diferentes receptores vão tender para um equilíbrio na variável *bwshare*, conforme explicado anteriormente. Isso acontece mesmo para receptores localizados em sessões diferentes. Neste caso, a divisão das camadas nas diferentes sessões pode ser coincidente ou não. Se a divisão das camadas não é coincidente, os receptores inscritos nas diferentes sessões não estarão sincronizados para fazer *join* na camada superior, e não haverá problemas. Caso a divisão das camadas for coincidente, a situação será similar à explicada no item anterior (receptores na mesma sessão), mas o conceito é estendido para sessões diferentes, e o

algoritmo funciona da mesma forma que explicado no item anterior.

5.2 Simulações do ALMP

A metodologia das simulações foi definida no item 4.7, e as simulações a seguir visam analisar o algoritmo nos experimentos propostos no item 4.8. Nem todas as simulações serão mostradas neste capítulo por questões de espaço, porém, todas as simulações podem ser encontradas em http://www.inf.unisinos.br/~roesler/tese e no CD anexo a esta Tese.

As simulações foram divididas conforme as métricas estabelecidas no item 4.7, que são detalhadas em cada um dos itens a seguir, buscando analisar o algoritmo em relação a: adaptabilidade, escalabilidade, tempo de adaptação, estabilidade, eqüidade de tráfego e taxa de perdas.

5.2.1 Análise da adaptabilidade do ALMP em ambientes heterogêneos

Foram efetuadas todas as simulações de adaptabilidade definidas no item 4.8, e as mais significativas estão descritas a seguir. Para as simulações, utilizou-se a topologia 1 da figura 4.3, que é repetida na figura 5.3 já com os parâmetros específicos utilizados para testar a adaptabilidade do algoritmo em ambientes heterogêneos. Os parâmetros configurados para a primeira simulação estão descritos a seguir, e serão utilizados como base para as outras simulações, que vão modificar alguns desses parâmetros:

- Atraso A1 a A4 = 10ms;
- Camadas exponenciais padrão (30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s e 960 kbit/s);
- Tamanho do pacote = 500 bytes;
- Tipo de fila = *droptail* com 20 pacotes;
- Randomização alta (±0,5s);
- Sem uso de pares de pacotes.

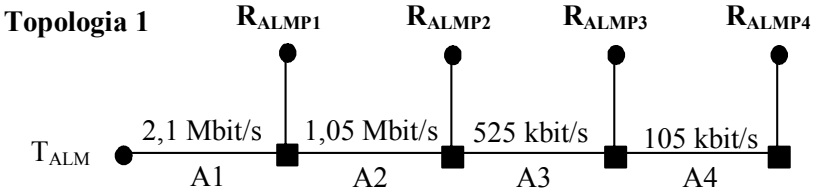


FIGURA 5.3 – Topologia para simulações de adaptabilidade

A figura 5.4 mostra o resultado da simulação. Pode-se constatar que todos os receptores adaptaram-se de acordo com a largura de banda disponível. O receptor ALMP1 se inscreveu em 6 camadas (total de 1890 kbit/s), o que é consistente com o enlace de 2,1 Mbit/s. O receptor ALMP2 se inscreveu em 5 camadas (930 kbit/s), coerente com o enlace de 1,05 Mbit/s. O receptor ALMP3 se inscreveu em 4 camadas (450 kbit/s), o que é adequado para o enlace de 525 kbit/s. Já o receptor ALMP4 se inscreveu em 2 camadas (90 kbit/s), pois a banda do enlace é somente 105 kbit/s.

O número maior de tentativas de subir camada do receptor ALMP4 é explicado pelo fato de que, quanto menor a banda, maior a taxa de subida do algoritmo, conforme explicado anteriormente. Isso faz com que a variável *bwshare* atinja o ponto de subida para a próxima camada de forma mais rápida. Além disso, como as camadas são exponenciais, o intervalo de banda até a próxima camada é menor do que nos receptores inscritos em mais camadas. Foi simulada uma modificação no algoritmo onde o receptor "aprendia" com as tentativas anteriores, efetuando menos tentativas ao longo do tempo, porém, com vários transmissores competindo pela banda e iniciando em instantes diferentes de tempo, essa alternativa mostrou não transmitir de forma eqüitativa.

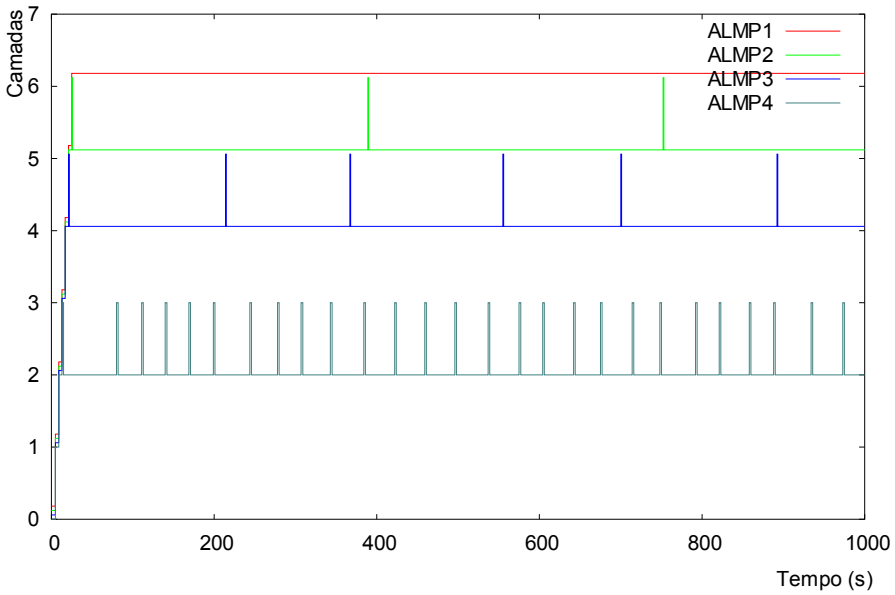


FIGURA 5.4 – ALMP com camadas exponenciais padrão sem utilizar pares de pacotes

A vazão para cada receptor está descrita a seguir, e foi obtida através da saída da simulação, conforme descrição no item 4.7.5 (Metodologia para análise de resultados no simulador). Como pode ser visto pelos resultados, o valor obtido é bastante próximo do ideal. O resultado é considerado ideal se a vazão é igual à soma das camadas possíveis do receptor se inscrever. Devido à granularidade das camadas, não é possível de obter um resultado igual à largura de banda do enlace, conforme explicado no item 2.5 (Eqüidade de tráfego). O resultado mostra uma vazão de aproximadamente 98% da vazão ideal que poderia ser conseguida para um algoritmo baseado em camadas, o que foi considerado excelente, pois nenhum algoritmo existente atualmente consegue atingir a vazão ideal da rede. A figura 5.5 mostra o resultado graficamente.

- Receptor ALMP1: vazão: 1861 kbit/s. Vazão ideal: 1890 kbit/s (camadas 1 a 6);
- Receptor ALMP2: vazão: 919 kbit/s. Vazão ideal: 930 kbit/s (camadas 1 a 5);
- Receptor ALMP3: vazão: 446 kbit/s. Vazão ideal: 450 kbit/s (camadas 1 a 4);
- Receptor ALMP4: vazão: 90 kbit/s. Vazão ideal: 90 kbit/s (camada 1 + camada 2).

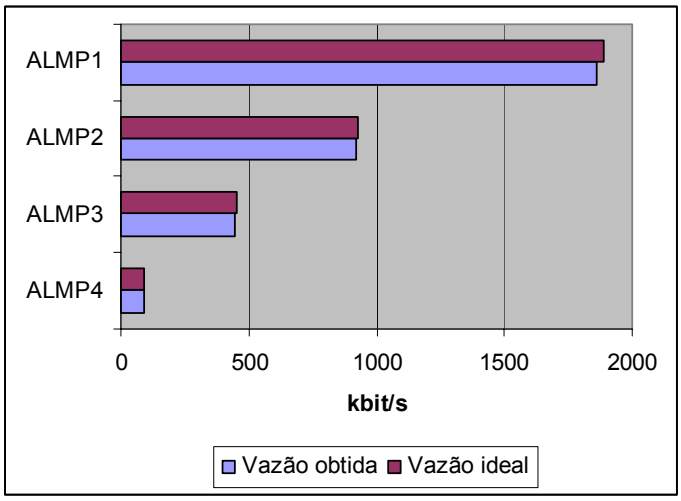


FIGURA 5.5 – Comparação entre a vazão obtida e ideal

A figura 5.6 mostra a mesma simulação, porém com a utilização de pares de pacotes. Nesse caso, como os pares de pacotes permitem a cada receptor o cálculo da banda máxima disponível na rede, e os receptores são os únicos usuários do enlace, praticamente não existem tentativas de subir camada, o número de perdas é zero e a adaptabilidade é consistente com a banda disponível.

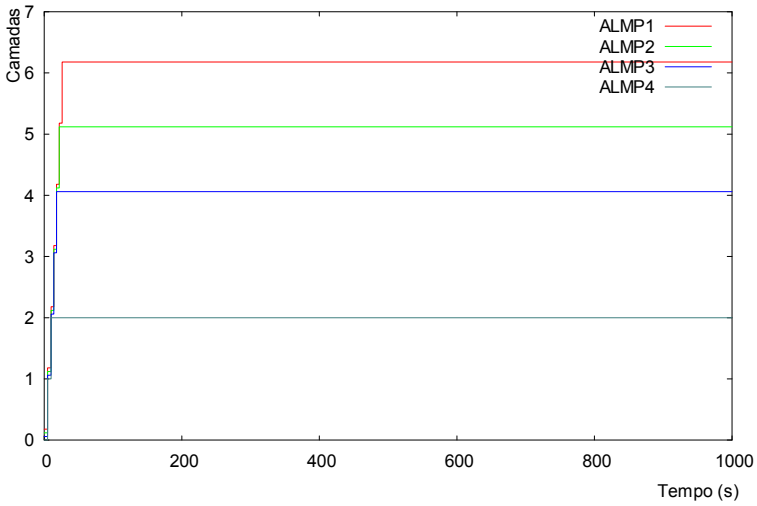


FIGURA 5.6 – ALMP com camadas exponenciais padrão utilizando pares de pacotes

A vazão para cada receptor está descrita a seguir e ilustrada graficamente na figura 5.7. A pequena diferença de vazão em relação à simulação anterior é devido a dois fatores: a) o arredondamento do tempo de início dos receptores (que começam de forma aleatória a partir de um instante pré-definido, podendo gerar uma diferença de até um segundo); b) cada tentativa de subir camada faz com que o receptor receba um excedente de tráfego enquanto a fila não esgotar, pois o valor nominal do enlace é maior do que o tráfego exigido pelo receptor. Por exemplo, o enlace do receptor ALMP4 é de 100 kbit/s, e se cadastrando em duas camadas o receptor consome 90 kbit/s, ou seja, menos que o valor nominal do enlace.

- Receptor ALMP1: vazão: 1858 kbit/s. Vazão ideal: 1890 kbit/s (camadas 1 a 6);

- Receptor ALMP2: vazão: 918 kbit/s. Vazão ideal: 930 kbit/s (camadas 1 a 5);
- Receptor ALMP3: vazão: 445 kbit/s. Vazão ideal: 450 kbit/s (camadas 1 a 4);
- Receptor ALMP4: vazão: 89 kbit/s. Vazão ideal: 90 kbit/s (camada 1 + camada 2).

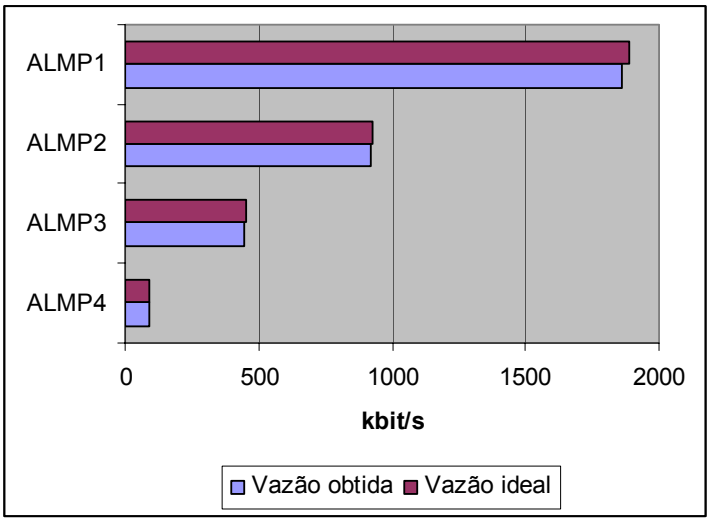


FIGURA 5.7 – Comparação entre a vazão obtida e ideal

Passando para tipo de camadas iguais ao do VEBIT (15 kbit/s, 45 kbit/s, 80 kbit/s, 150 kbit/s e 385 kbit/s), pode-se verificar que o sistema também se adapta corretamente, como ilustra a figura 5.8. Os receptores ALMP1 e ALMP2 se cadastraram em todas as camadas, pois as mesmas consomem 675 kbit/s de banda, e ambos tem mais do que isso. Já o receptor ALMP3 se cadastrou somente nas 4 primeiras camadas (consumindo 290 kbit/s de um enlace com 525 kbit/s), e o receptor ALMP4 se cadastrou somente nas primeiras duas camadas (total de 60 kbit/s de um enlace com 105 kbit/s).

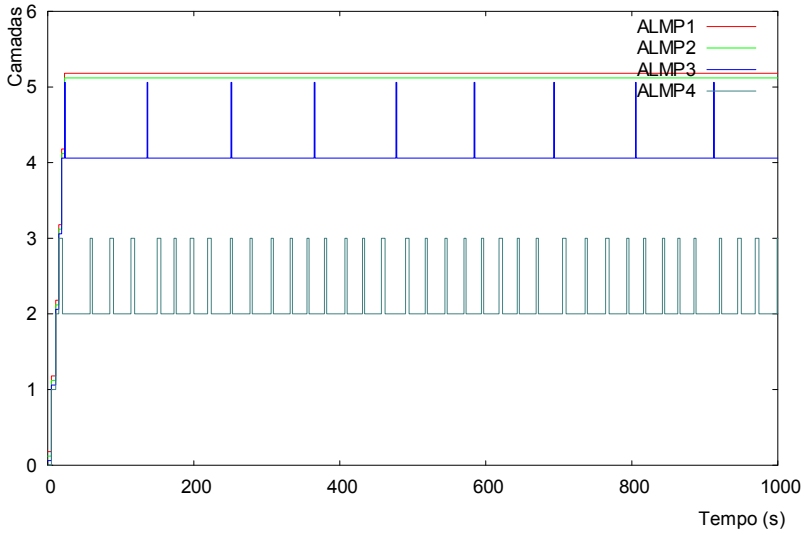


FIGURA 5.8 – Camadas iguais ao VEBIT sem pares de pacotes

A vazão obtida e ideal para cada receptor é descrita nos itens a seguir e ilustrada graficamente na figura 5.9. Os receptores ALMP1 a ALMP3 atingiram praticamente 100% do valor ideal. Já o receptor ALMP4 atingiu mais de 100%, e isso se deve às

tentativas de *join* na camada superior. Como o enlace tem 105 kbit/s, cada tentativa de *join* na camada 3 cria uma demanda de 140 kbit/s, demorando certo tempo até que o roteador comece a descartar pacotes. Durante esse tempo, o receptor acumula pacotes da camada 3, fazendo com que o total de pacotes recebidos seja superior ao esperado.

- Receptor ALMP1: vazão: 665 kbit/s. Vazão ideal: 675 kbit/s (5 camadas);
- Receptor ALMP2: vazão: 665 kbit/s. Vazão ideal: 675 kbit/s (5 camadas);
- Receptor ALMP3: vazão: 289 kbit/s. Vazão ideal: 290 kbit/s (4 camadas);
- Receptor ALMP4: vazão: 67 kbit/s. Vazão ideal: 60 kbit/s (2 camadas).

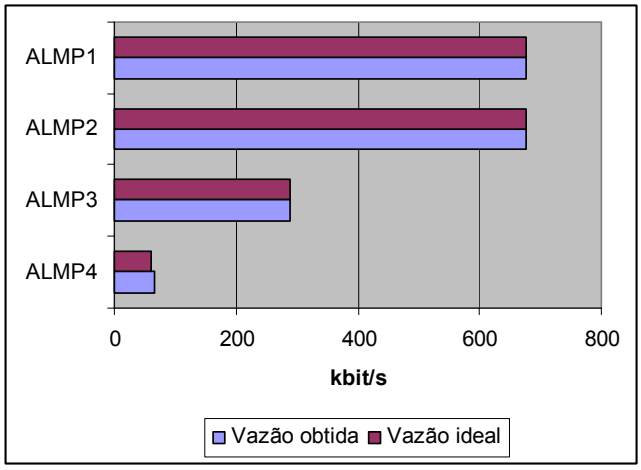


FIGURA 5.9 – Comparação entre a vazão obtida e ideal

Repetindo a simulação com o uso de pares de pacotes, conforme ilustrado na figura 5.10, o resultado é semelhante ao obtido com camadas exponenciais, ou seja, sem tentativas de subir camada além da banda máxima.

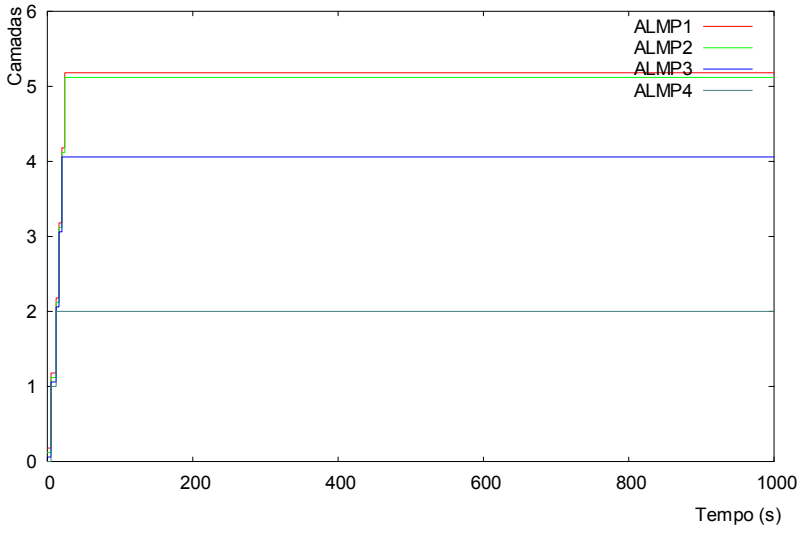


FIGURA 5.10 – Camadas iguais ao VEBIT com pares de pacotes

Quando se utilizam camadas de 50 kbit/s sem pares de pacotes, o sistema também se adapta de acordo com a banda disponível aos receptores, conforme ilustra a figura 5.11. O receptor ALMP1 adaptou-se em 40 camadas (2 Mbit/s), o ALMP2 em 20

camadas (1 Mbit/s), o ALMP3 em 10 camadas (500 kbit/s) e o ALMP4 em 2 camadas (100 kbit/s).

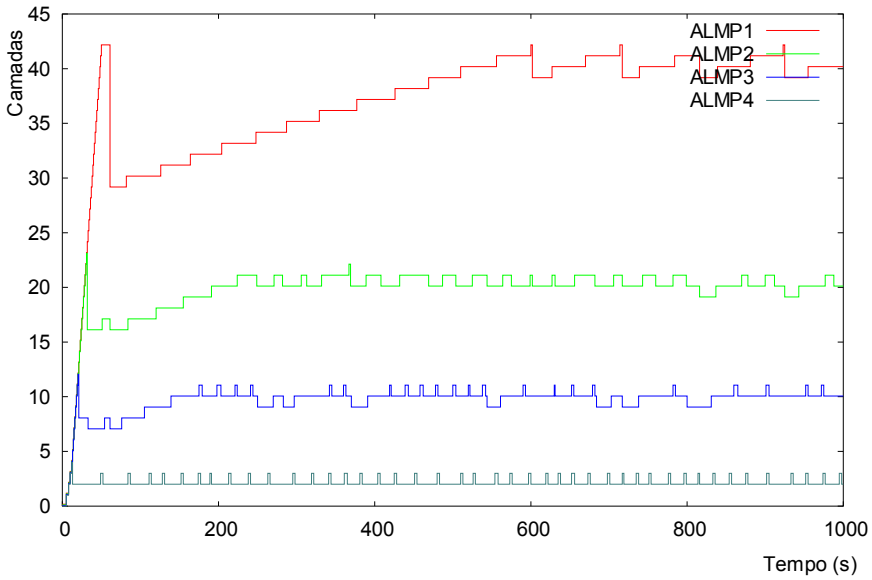
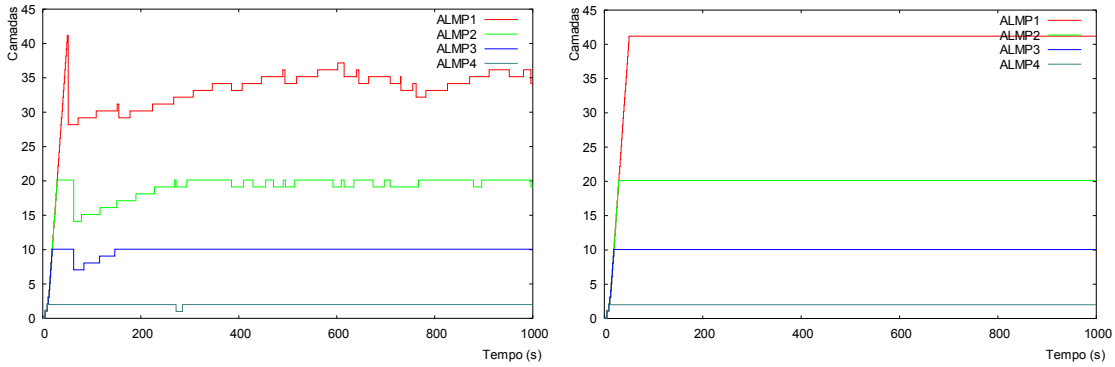


FIGURA 5.11 – ALMP com camadas de 50 kbit/s sem pares de pacotes

Com pares de pacotes a situação muda um pouco devido ao elevado número de fluxos (42 para uma banda de 2,1 Mbit/s). Isso causa um problema no ambiente simulado, que é a fila ser insuficiente para conter todos os fluxos simultaneamente, ou seja, quando o transmissor envia os pacotes randomicamente, muitos são enviados ao mesmo tempo, causando um grande número de pacotes na fila do roteador. A conseqüência disso é a perda de pacotes, que pode ser observada no receptor ALMP1 da figura 5.12a. Com uma fila de 20 pacotes, alguns deles são perdidos, independente do sistema ter ou não banda suficiente, fazendo com que o algoritmo se adapte diminuindo suas camadas, como pode ser visto no fluxo ALMP1, onde o receptor nem chega no limite de banda, que é de 2,1 Mbit/s, correspondendo à camada 42.



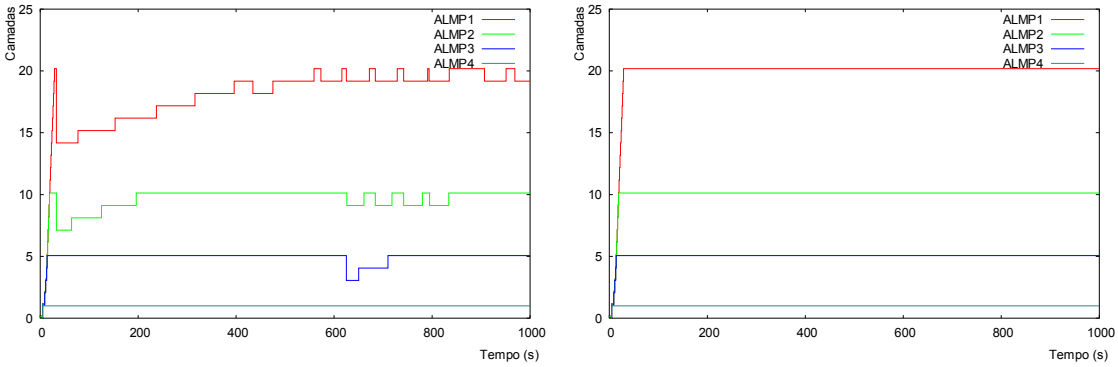
a) Fila de 20 pacotes b) Fila de 60 pacotes

FIGURA 5.12 – Camadas de 50 kbit/s sem pares de pacotes

Os outros receptores adaptaram-se de acordo com a banda disponível: o receptor ALMP2 na camada de número 20 (correto para banda de 1,05 Mbit/s), o receptor ALMP3 na camada 10 (enlace de 525 kbit/s) e o receptor ALMP4 na camada 2 (correto para 105 kbit/s).

A figura 5.12b mostra a mesma simulação, porém com uma fila de 60 pacotes. Nesse caso, como a fila consegue absorver mais pacotes nas rajadas do transmissor, o número de perdas devido a esse problema diminui consideravelmente, e o fluxo ALMP1 consegue atingir sua totalidade de banda (camada de número 41). Considera-se que o fato ilustrado não constitui um problema do algoritmo ALMP, pois o mesmo se adapta através da perda de pacotes e não tem como saber se a fila é pequena ou não para o número de fluxos estipulado, entretanto, mostra que o uso de pares de pacotes exige um pouco mais das filas dos roteadores.

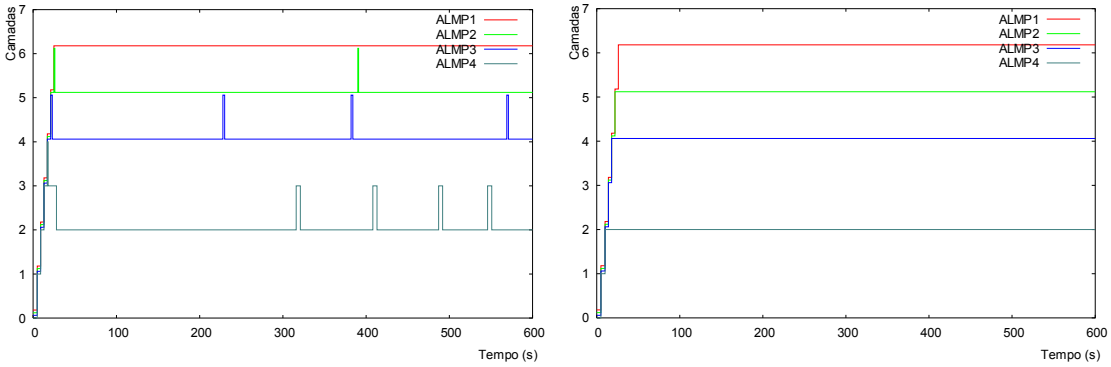
Modificando o tipo das camadas para 100 kbit/s, ocorre um sintoma semelhante ao visto nas camadas de 50 kbit/s. Com pares de pacotes, o tráfego gerado extrapola uma fila de tamanho 20 pacotes, fazendo com que o receptor não consiga adaptar-se ao correto número de camadas, como mostra a figura 5.13a, onde o receptor ALMP1 adaptou-se na camada de número 19 ao invés da camada 20. A figura 5.13b mostra a mesma simulação, porém com fila de 60 pacotes. Pode-se ver que no segundo gráfico o receptor ALMP1 consegue atingir as 20 camadas. Os receptores ALMP2 a ALMP4 adaptaram-se ao correto número de camadas (respectivamente 10, 5 e 1 camada) independente do tamanho da fila, pois o número de fluxos é menor, não acarretando o problema visto no receptor ALMP1.



a) Fila de 20 pacotes b) Fila de 60 pacotes

FIGURA 5.13 – Camadas de 100 kbit/s com pares de pacotes

A simulação ilustrada na figura 5.14 modifica o tipo de fila para RED 10/20, voltando o tipo das camadas para exponenciais padrão. O gráfico da figura 5.14a representa a simulação sem pares de pacotes, enquanto o da figura 5.14b com pares de pacotes. No primeiro gráfico, observa-se que o receptor ALMP4 efetuou um número bem menor de tentativas para subir camadas em relação à mesma simulação com fila *droptail* de 20 pacotes (vista na figura 5.4). Isso pode ser explicado através do funcionamento do RED, que descarta pacotes com determinada probabilidade quando a fila atinge 10 posições (*threshold* mínimo), chegando a descartar 100% dos pacotes quando a fila atinge a posição 20 (*threshold* máximo). O resultado é um maior número de perdas, e o receptor nem tenta subir camada. O segundo gráfico mostra que, com pares de pacotes, o receptor não tenta subir camadas além da sua banda disponível, gerando uma maior estabilidade ao sistema.



a) Sem pares de pacotes b) Com pares de pacotes

FIGURA 5.14 – ALMP com fila RED 10/20

Foram feitas outras simulações, conforme descrição no item 4.8 (Definição das simulações), entretanto, elas não geraram alterações significativas nos resultados, e somente serão comentadas brevemente a seguir, entretanto, estão disponíveis em http://www.inf.unisinos.br/~roesler/tese e no CD anexo a esta Tese.

- **Atraso com 1ms em cada enlace ao invés de 10ms**: nesse caso, o gráfico gerado não apresenta diferença significativa em relação ao com atraso de 10ms;
- **Tamanho de pacote 250 bytes ou 1000 bytes ao invés de 500 bytes**: foram feitos simulações com camadas exponenciais e de 100 kbit/s. Os gráficos ficaram muito parecidos com os obtidos para tamanho de pacote de 500 bytes. Basicamente o que muda é a quantidade de pacotes enviados por segundo. Com pacotes de 250 bytes, o transmissor envia um número maior de pacotes para manter a mesma taxa de transmissão da camada. A conseqüência é um maior enchimento da fila, provocando um acréscimo nas perdas. No caso de pacotes de 1000 bytes, são menos pacotes por segundo, onerando menos a fila;
- **Fila RED 10/20 e camadas de 100 kbit/s**: nesse caso, observa-se o mesmo sintoma das camadas exponenciais com fila RED 10/20, ou seja, o maior número de perdas do RED provoca um menor número de tentativas para subir camada;
- **Fila RED 20/60 com camadas exponenciais**: essa simulação gerou um resultado muito parecido com o resultado obtido com filas RED 10/20, mostrado na figura 5.14;
- **Sem randomização (CBR puro)**: nesse caso, o gráfico gerado é praticamente o mesmo do que com randomização na transmissão dos pacotes (±0,5s), tanto para camadas exponenciais como para camadas de 100 kbit/s. Entretanto, observa-se que, como o transmissor gera os pacotes numa cadência fixa, a fila dos roteadores intermediários permanece mais estável, pois não existe a incerteza da randomização. A conseqüência é visível quando existem muitas camadas, como por exemplo na utilização de camadas de 100 kbit/s com fila *droptail* de 20, mostrado na figura 5.13a. Sem randomicidade, não aconteceu o sintoma descrito, e o gráfico gerado com fila de 20 pacotes (gráfico "a") ficou muito parecido com o que se utilizou fila 60 pacotes (gráfico "b").

5.2.2 Análise de escalabilidade do ALMP

Para analisar a escalabilidade do algoritmo, a metodologia prevê aumentar o

número de receptores buscando verificar mudanças no comportamento dos fluxos e o impacto para um grande número de usuários. Nas simulações de adaptabilidade, foram feitos os testes para quatro receptores, cada um deles localizado num enlace com largura de banda diferente. Nos testes de escalabilidade, esse número será aumentado, e serão analisados os resultados para um total de 20 e 100 receptores. Com essa informação, pode-se analisar o algoritmo em três escalas: 1 receptor por bloco (visto no item 5.2.1), 5 receptores por bloco e 25 receptores por bloco, ou seja, um total de 4 receptores (visto no item 5.2.1), 20 receptores e 100 receptores.

As simulações de escalabilidade também visam testar a capacidade do algoritmo em adaptar-se a ambientes heterogêneos, no caso da existência de vários receptores localizados atrás de cada roteador.

As simulações tomaram como base os mesmos parâmetros utilizados nos testes de adaptabilidade, e a topologia utilizada também é a topologia 1 da figura 4.3, que é repetida na figura 5.15 já com os parâmetros específicos utilizados para testar a escalabilidade do algoritmo em ambientes heterogêneos. Os parâmetros configurados para a primeira simulação estão descritos a seguir, e serão utilizados como base para as outras simulações, que vão modificar alguns desses parâmetros:

- Número de receptores por bloco: r = 5, totalizando 20 receptores, e r = 25, totalizando 100 receptores;
- Camadas exponenciais padrão (30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s e 960 kbit/s);
- Tamanho do pacote = 500 bytes;
- Tipo de fila = *droptail* com 20 pacotes;
- Randomização alta (±0,5s);
- Sem uso de pares de pacotes.

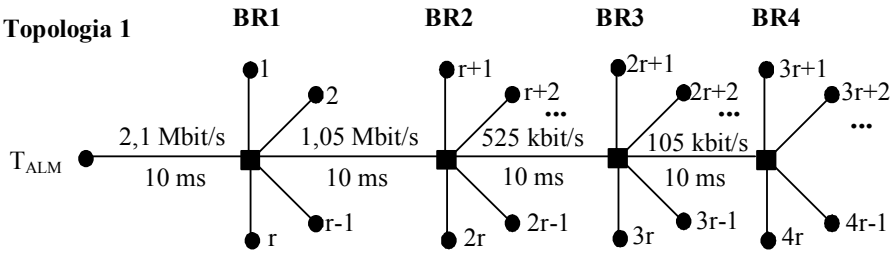


FIGURA 5.15 – Topologia para simulações de escalabilidade

Um fator importante é que todos receptores em uma determinada sub-rede devem chegar às mesmas conclusões sobre o número de camadas a se inscrever, caso contrário, o algoritmo não vai funcionar adequadamente numa situação de congestionamento, visto que um único receptor cadastrado em determinado grupo multicast mantém o tráfego dos pacotes para toda a sub-rede.

O gráfico da figura 5.16 mostra o resultado das simulações para 20 receptores sem a utilização de pares de pacotes, e a legenda foi eliminada para melhorar a legibilidade. Pode-se ver que todos receptores adaptaram-se de acordo com a largura de banda disponível. Os receptores ALMP1 a ALMP5 se inscreveram em 6 camadas (1890 kbit/s cada), que é consistente com o limite de 2,1 Mbit/s. Os receptores ALMP6 a

ALMP10 se inscreveram em 5 camadas (930 kbit/s), que é consistente com o gargalo de 1,05 Mbit/s. Os receptores ALMP11 a ALMP15 se inscreveram em 4 camadas (450 kbit/s), coerente com o gargalo de 525kbit/s, e os receptores ALMP16 a ALMP20 se inscreveram em 2 camadas (90 kbit/s), que é adequado ao gargalo de 105 kbit/s.

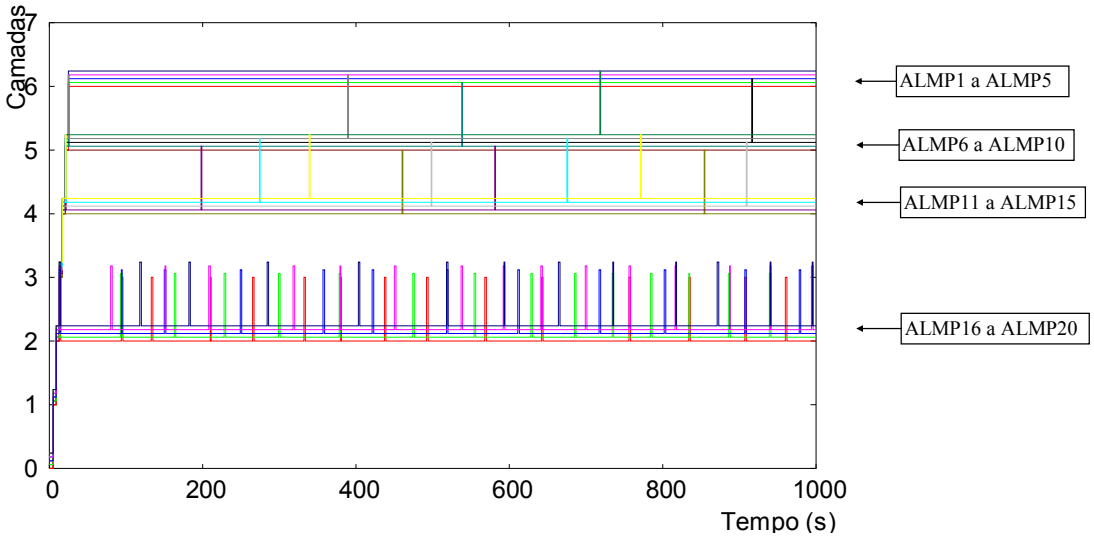


FIGURA 5.16 – Escalabilidade do ALMP - 20 receptores sem pares de pacotes

A vazão obtida e ideal para cada bloco de receptores é descrita nos itens a seguir e representada graficamente na figura 5.17. Todos os receptores atingiram uma vazão próxima a 98%.

- Receptores ALMP1 a ALMP5: vazão: 1859 a 1861 kbit/s. Vazão ideal: 1890 kbit/s (camadas 1 a 6);
- Receptores ALMP6 a ALMP10: vazão: 916 a 918 kbit/s. Vazão ideal: 930 kbit/s (camadas 1 a 5);
- Receptores ALMP11 a ALMP15: vazão: 443 a 444 kbit/s. Vazão ideal: 450 kbit/s (camadas 1 a 4);
- Receptores ALMP16 a ALMP20: vazão: 86 a 87 kbit/s. Vazão ideal: 90 kbit/s (camada 1 + camada 2);

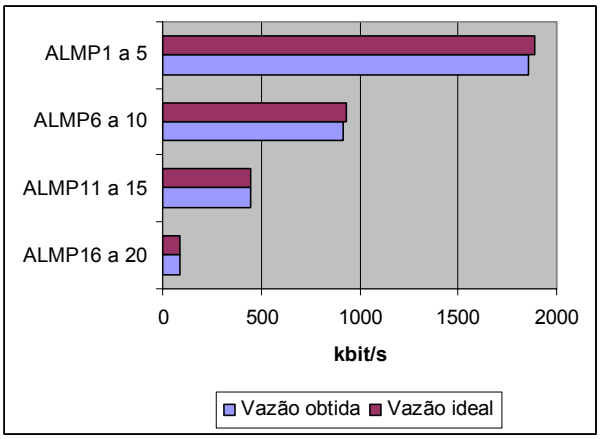


FIGURA 5.17 – Comparação entre vazão obtida e vazão ideal para 20 receptores

A figura 5.18 mostra a escalabilidade do ALMP para 100 receptores, sem a

utilização de pares de pacotes. Pode-se constatar que todos adaptaram-se de forma coerente com a largura de banda disponível, da mesma forma que a simulação com 20 receptores. Algumas tentativas de subir camada também podem ser observadas, principalmente nas camadas de baixo, onde a taxa de incremento de banda é maior, conforme detalhado anteriormente.

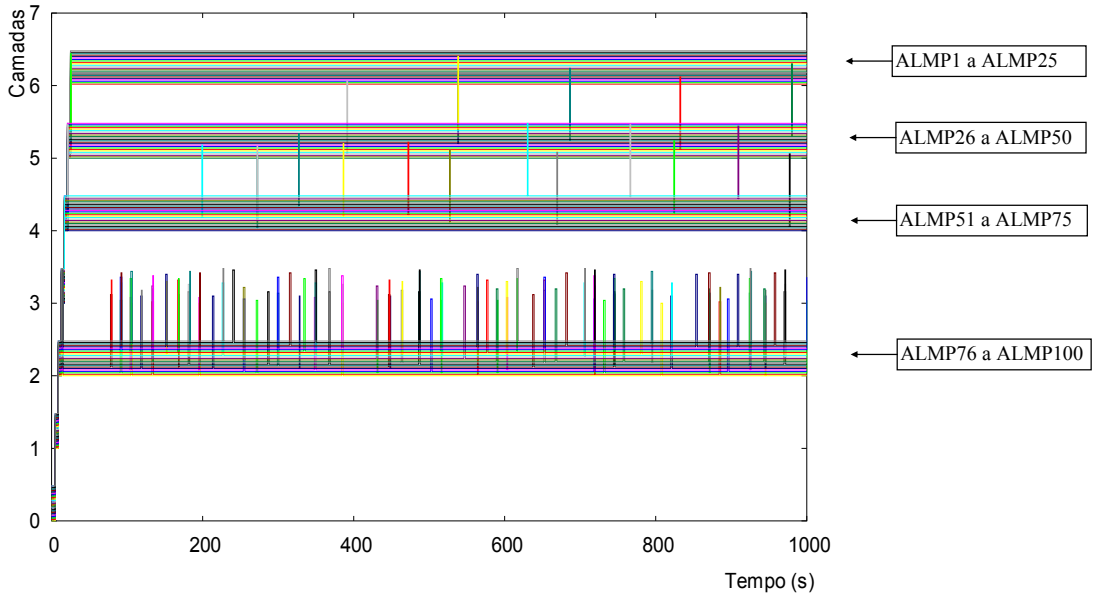


FIGURA 5.18 – Escalabilidade do ALMP – 100 receptores sem pares de pacotes

A vazão obtida e ideal para cada bloco de receptores é descrita nos itens a seguir e representada graficamente na figura 5.19. Todos os receptores atingiram uma vazão próxima a 98%.

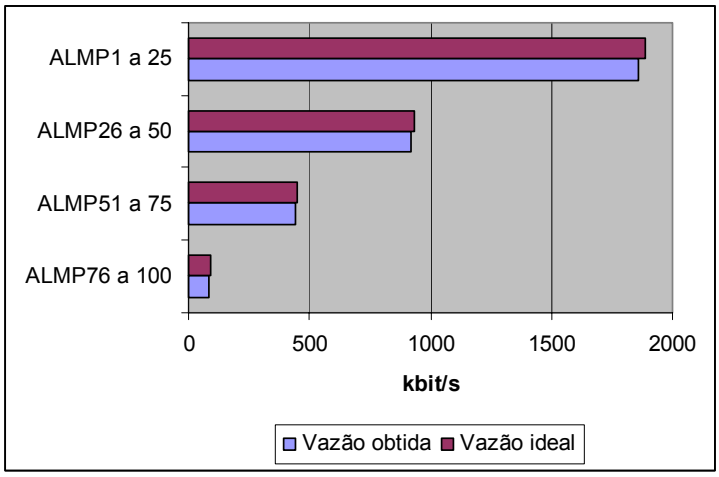


FIGURA 5.19 – Comparação entre vazão obtida e vazão ideal para 100 receptores

- Receptores ALMP1 a ALMP25: vazão: 1859 a 1861 kbit/s. Vazão ideal: 1890 kbit/s (camadas 1 a 6);
- Receptores ALMP26 a ALMP50: vazão: 915 a 917 kbit/s. Vazão ideal: 930 kbit/s (camadas 1 a 5);
- Receptores ALMP51 a ALMP75: vazão: 442 a 443 kbit/s. Vazão ideal: 450 kbit/s

(camadas 1 a 4);

- Receptores ALMP76 a ALMP100: vazão: todos receptores com 85 kbit/s. Vazão ideal: 90 kbit/s (camada 1 + camada 2).

A figura 5.20 mostra a mesma simulação de 100 receptores com o uso de pares de pacotes. Pode-se perceber que o sistema repetiu o comportamento mostrado nas simulações anteriores com pares de pacotes, ou seja, adaptou-se de acordo com a banda disponível sem efetuar tentativas de subir camadas acima da banda nominal do enlace.

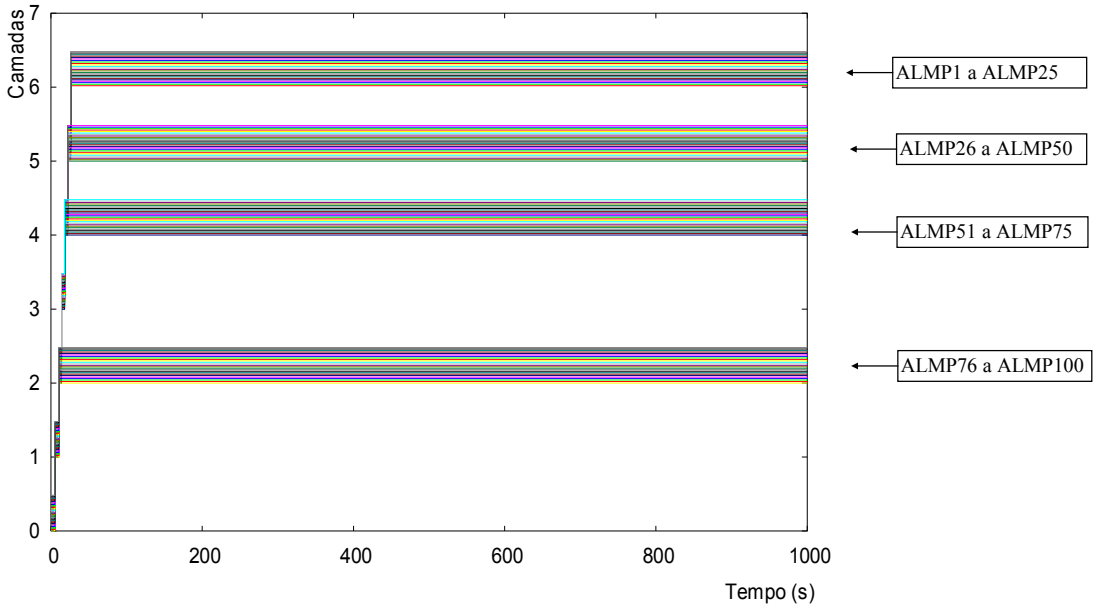


FIGURA 5.20 – Escalabilidade do ALMP – 100 receptores com pares de pacotes

A tabela 5.1 mostra uma comparação entre estabilidade, perdas e vazão do algoritmo ALMP para um único transmissor e vários receptores, onde r = 1, 5 e 25, ou seja, um total de 4, 20 e 100 receptores. A taxa de cada camada é exponencial, e as simulações são sem pares de pacotes. Os valores para 20 e 100 receptores são os valores médios obtidos para cada bloco de receptores. O valor e significado de VCM (Variações de Camada por Minuto) foi explicado no item 4.7.5, e será detalhado para o ALMP no item 5.2.4 (Análise da estabilidade do ALMP). O método de obtenção da taxa de perdas foi explicado no item 4.7.5, e será detalhado para o ALMP no item 5.2.6 (Análise da taxa de perdas do ALMP).

Através da comparação da estabilidade (coluna VCM na tabela 5.1) linha a linha entre as três escalas do algoritmo, percebe-se que a estabilidade aumenta quando existem mais receptores. Por exemplo, no pior caso, o fluxo ALMP4 teve um VCM de 3,12, enquanto que a média dos fluxos ALMP76 a 100 foi de 0,48. Isso se explica pois quando um receptor efetua uma tentativa de subir camada, faz com que os outros aprendam com isso, conforme detalhamento no início do capítulo, e outros receptores nem tentam subir camada, deixando cada fluxo individualmente mais estável. O sistema como um todo, entretanto, sofre uma diminuição de estabilidade, como pode ser percebido comparando os gráficos das figuras 5.4, 5.16 e 5.18.

Analisando a taxa de perdas entre as três escalas do algoritmo (coluna “taxa

média de perdas" na tabela 5.1), percebe-se que existe um incremento na taxa de perdas, porém esse aumento não é proporcional ao aumento no número de receptores. Por exemplo, no pior caso, o fluxo ALMP4 teve uma média de perdas de 3,09%, enquanto a média de perdas para os receptores ALMP76 a 100 foi de 6,04%, mostrando que houve um aumento de 25 vezes o número de receptores, mas somente duas vezes a taxa de perdas. Vale lembrar que a transmissão é multicast, ou seja, um pacote perdido para um fluxo é perdido para todos os fluxos após esse roteador. O algoritmo VEBit, que deve ser integrado ao SAM, não é muito sensível a perdas, como pode ser visto no capítulo 8, porém, a verificação se a taxa de perdas está muito elevada somente será vista após a integração de ambos e implementação final do sistema.

Analisando a vazão (coluna "vazão média" na tabela 5.1), observa-se que o algoritmo diminui um pouco a vazão com o aumento no número de receptores. Por exemplo, no pior caso, o receptor ALMP4 obteve uma vazão de 90 kbit/s. Cada um dos 25 receptores mostrados na última linha da tabela (ALMP76 a 100) obtiveram uma vazão de 85 kbit/s, ou seja, 5% menor. A diminuição foi menor para os outros receptores, como pode ser visto na tabela.

A coluna de vazão também serve para mostrar a adaptabilidade do algoritmo com o aumento no número de receptores. De acordo com os resultados observados, percebe-se que o algoritmo é adaptável até 100 receptores, e existe uma tendência que consiga se adaptar com mais receptores.

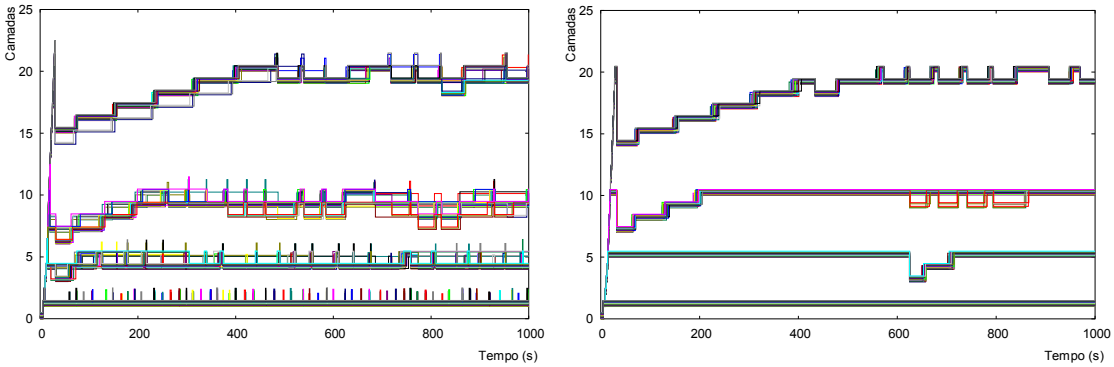
TABELA 5.1 – Análise de estabilidade, perdas e vazão para o algoritmo ALMP

Total de receptores	VCM médio	Taxa média de perdas	Vazão média (kbit/s)
4 receptores	ALMP1: 0,00 ALMP2: 0,24 ALMP3: 0,60 ALMP4: 3,12	ALMP1: 0,00% ALMP2: 0,17% ALMP3: 0,41% ALMP4: 3,09%	ALMP1: 1861 ALMP2: 919 ALMP3: 446 ALMP4: 90
20 receptores	ALMP1 a 5: 0,00 ALMP6 a 10: 0,05 ALMP11 a 15: 0,19 ALMP16 a 20: 1,58	ALMP1 a 5: 0,00% ALMP6 a 10: 0,28% ALMP11 a 15: 0,73% ALMP16 a 20: 4,8 %	ALMP1 a 5: 1860 ALMP6 a 10: 917 ALMP11 a 15: 443 ALMP16 a 20: 87
100 receptores	ALMP1 a 25: 0,00 ALMP26 a 50: 0,02 ALMP51 a 75: 0,11 ALMP76 a 100: 0,48	ALMP1 a 25: 0,00% ALMP26 a 50: 0,25% ALMP51 a 75: 1,02% ALMP76 a 100: 6,04%	ALMP1 a 25: 1860 ALMP26 a 50: 916 ALMP51 a 75: 442 ALMP76 a 100: 85

Com camadas de 100 kbit/s o sistema também adaptou-se corretamente, conforme ilustrado na figura 5.21, onde o gráfico "a" representa a simulação sem pares de pacotes, e o gráfico "b" com pares de pacotes. Como existem mais receptores do que visto nas simulações anteriores, existe um maior número de tentativas de subir camada, mesmo com o tempo randômico implementado. Isso provoca um acréscimo no nível de perdas, fazendo com que alguns receptores não consigam atingir plenamente sua banda. na figura 5.21a essa situação pode ser percebida para o segundo e terceiro bloco de receptores, onde vários deles não conseguem atingir a camada 10 e 5, respectivamente.

Com a utilização de pares de pacotes, vista na figura 5.21b, diminui o número de tentativas para subir camada, conseqüentemente, diminui o número de perdas nas

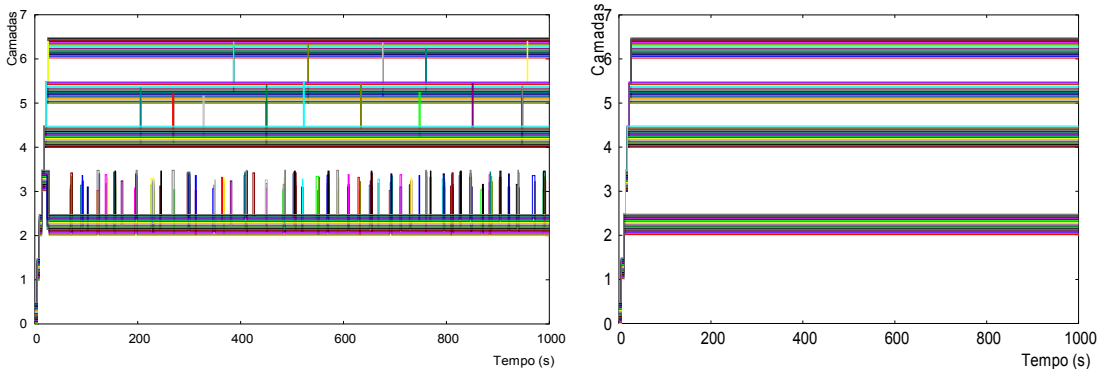
camadas baixas, deixando o sistema mais estável.



a) Sem pares de pacotes b) Com pares de pacotes

FIGURA 5.21 – 100 receptores ALMP: camadas de 100 kbit/s

Outra simulação efetuada foi com fila RED 10/20 e 100 receptores, conforme ilustra a figura 5.22, onde o gráfico "a" representa a simulação sem o uso de pares de pacotes, e o gráfico "b" com pares de pacotes.



a) Sem pares de pacotes b) Com pares de pacotes

FIGURA 5.22 – 100 receptores ALMP: camadas exponenciais, RED 10/20

Sem pares de pacotes (figura 5.22a) a estabilidade é muito parecida com a obtida utilizando filas *droptail*. Com o uso de pares de pacotes (figura 5.22b) o sistema adaptou-se perfeitamente.

A partir das simulações de escalabilidade obtidas, pode-se concluir que o sistema aceita um grande número de usuários sem perder suas características. Uma das razões para isso é o fato do algoritmo não exigir que os receptores se comuniquem com o transmissor. Entretanto, o número de tentativas de subir camada provoca um aumento no número de perdas, provocando certa instabilidade em alguns casos. Uma diminuição nesse número de tentativas é obtida através do mecanismo de geração de número aleatório antes de subir camada, que permite um aprendizado de alguns receptores a partir das tentativas efetuadas por outros receptores (que geraram números aleatórios menores).

5.2.3 Análise de eqüidade de tráfego do ALMP

As simulações de análise de eqüidade de tráfego têm por objetivo analisar o

comportamento do algoritmo com tráfego concorrente, verificando se o mesmo tem capacidade para compartilhar tráfego igualmente com outros fluxos. No caso do ALMP, o único tráfego concorrente é de outras sessões do próprio algoritmo, visto que ele foi projetado para uso em redes privativas, sem a presença de TCP.

A topologia utilizada para as simulações foi a topologia 2 da figura 4.3, que é reproduzida na figura 5.23 já com os parâmetros específicos utilizados para testar a eqüidade do algoritmo. Na topologia, existe um número *m* de fluxos ALMP concorrendo através de um enlace único. Os parâmetros configurados para a primeira simulação estão descritos a seguir, e serão utilizados como base para as outras simulações, que vão modificar alguns desses parâmetros:

- Banda "B" no enlace central especificada como *m*500kbit/s*, onde *m* é o número de transmissores ALM no momento. Isso foi feito para acomodar 4 camadas exponenciais para cada sessão (que demandam um total de 450 kbit/s: 30+60+120+240);
- Atraso A = 10ms;
- Camadas exponenciais padrão (30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s e 960 kbit/s);
- Tráfego de cada camada constante (tipo CBR);
- Tamanho do pacote = 500 bytes;
- Tipo de fila = *droptail* com 20 pacotes;
- Randomização alta (±0,5s);
- Com uso de pares de pacotes.

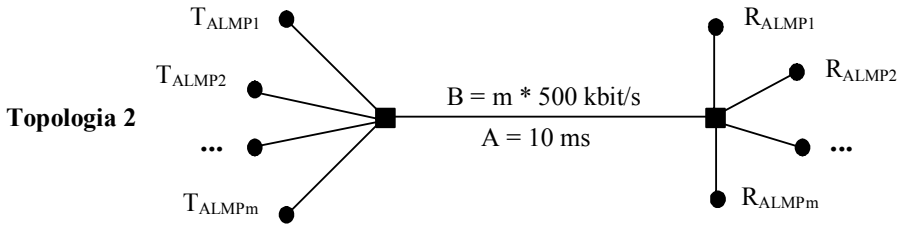


FIGURA 5.23 – Topologia para simulações de eqüidade

A seguir serão analisados os resultados obtidos para as simulações especificadas no item 4.8 (Definição das simulações) para eqüidade de tráfego.

Em alguns resultados, utilizou-se o conceito de *fairness index* (FI), definido no item 4.7.5, que é a relação entre a taxa que o receptor está utilizando (TU) frente à taxa eqüitativa para este receptor (TE), ou seja, FI = TU / TE, e quanto mais próximo do valor "um", mais eqüitativo é o fluxo.

A simulação da figura 5.24 mostra dois fluxos ALMP concorrendo por 1 Mbit/s de banda no enlace central. Pode-se constatar que os dois receptores adaptaram-se na camada correta, dividindo igualmente a banda. A vazão do receptor ALMP1 ficou em 440 kbit/s, enquanto a vazão do receptor ALMP2 foi de 437 kbit/s, ou seja, diferença de menos de 1% entre eles. O *fairness index* de cada fluxo foi 0,978 e 0,971.

Outro resultado obtido é que o gráfico gerado é idêntico com e sem pares de pacotes. O motivo disso é que os pares de pacotes impedem o receptor de utilizar banda

além da banda máxima do enlace, porém, a banda máxima do enlace é de 1 Mbit/s, e cada receptor está utilizando apenas 450 kbit/s, tendo possibilidade de se inscrever em novas camadas sem atingir o limite de banda obtido pelo uso dos pares de pacotes. A única diferença é em relação à utilização das filas, que é mais exigida com pares de pacotes, conforme visto nas simulações de adaptabilidade. A partir desse momento, todas as simulações vão utilizar o mecanismo de pares de pacotes, pois este mecanismo é parte integrante do algoritmo ALMP.

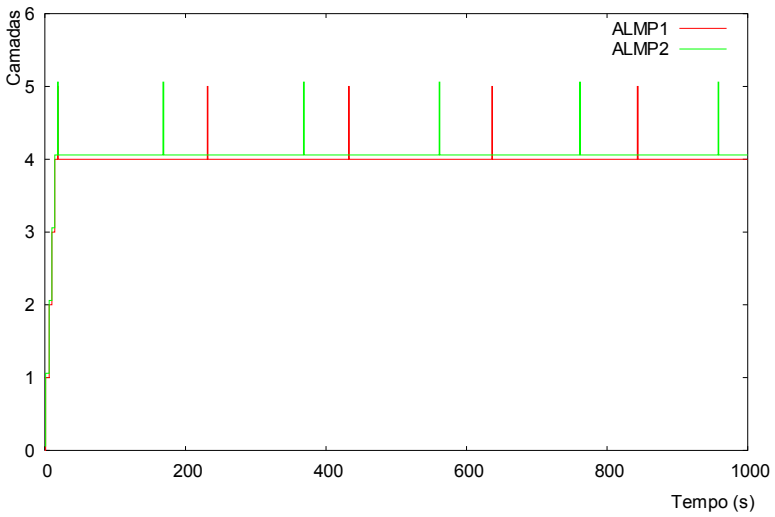


FIGURA 5.24 – Eqüidade do ALMP para 2 fluxos concorrentes

A figura 5.25 mostra a adaptação do algoritmo para 10 fluxos concorrentes. A banda no enlace central foi de 5 Mbit/s, perfazendo um total de 500 kbit/s por receptor. Observa-se que todos os receptores adaptaram-se de acordo com a banda disponível, de forma eqüitativa. A vazão variou de 445 kbit/s a 447 kbit/s, ou seja, todos valores muito próximos, mostrando a alta eqüidade de tráfego obtida através do algoritmo do ALMP. O *fairness index* entre os fluxos ficou em FI = 0,99.

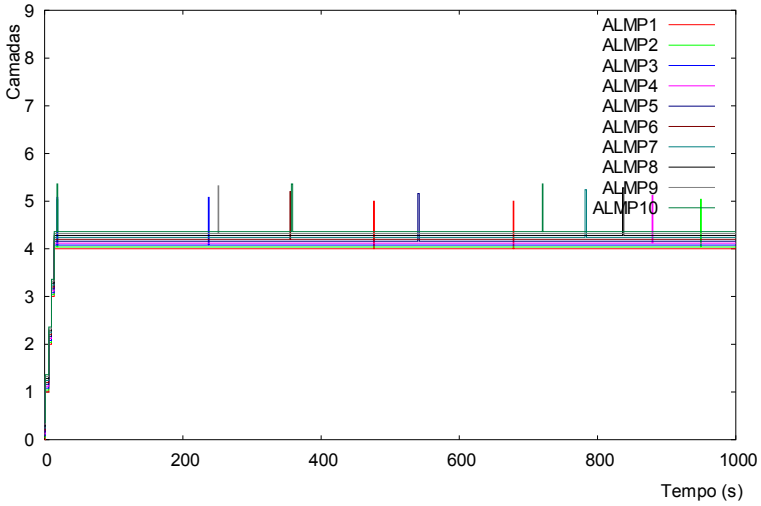


FIGURA 5.25 – Eqüidade para 10 fluxos concorrentes com camadas exponenciais

Com 20 fluxos concorrentes, acontece o mesmo problema visto nas simulações de adaptabilidade com camadas de 100 kbit/s, ou seja, a fila é bastante exigida devido à

randomicidade da transmissão, e alguns pacotes são perdidos, forçando alguns fluxos a se estabilizarem numa camada inferior. Essa situação é ilustrada na figura 5.26, onde 3 do total de 20 fluxos ficam na camada 3 ao invés de subirem para a camada 4.

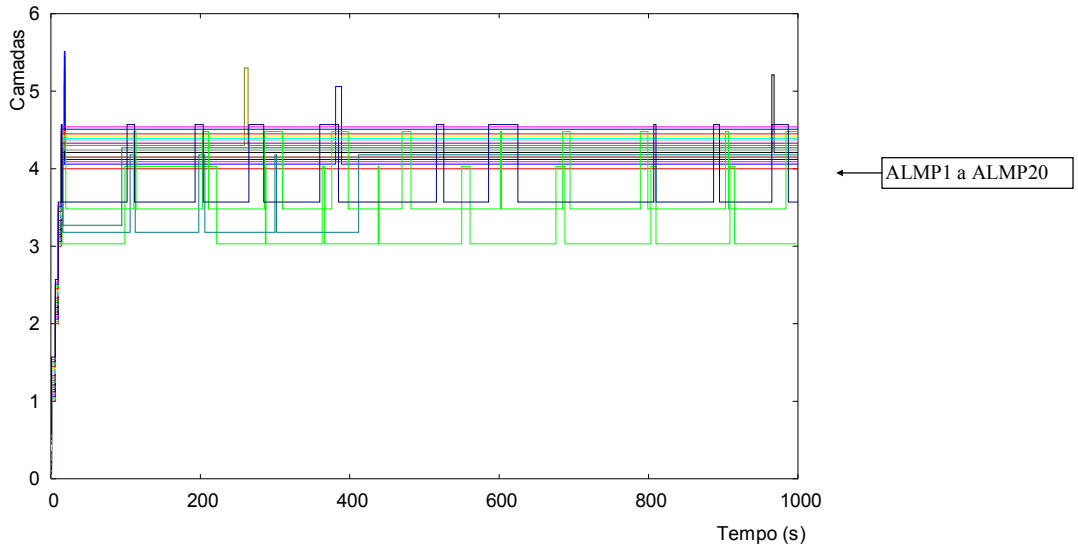


FIGURA 5.26 –20 fluxos concorrentes com fila *droptail* de 20

A solução para essa situação é aumentar a fila, e isso é ilustrado pela figura 5.27, onde se repetiu a simulação para uma fila *droptail* de 60 pacotes. Como pode ser visto no gráfico, a instabilidade deixou de existir. Esse fato não é considerado como problema do algoritmo, visto que acontece em todos os mecanismos de controle de congestionamento por camadas, e é devido ao excesso de fluxo na fila dos roteadores, devendo ser resolvido através do aumento do tamanho da fila nos roteadores intermediários. Pode-se inferir que, num ambiente real, é melhor a utilização de poucas camadas, caso seja possível, pois diminui o número de fluxos (número de grupos multicast), reduzindo o tráfego de controle multicast necessário.

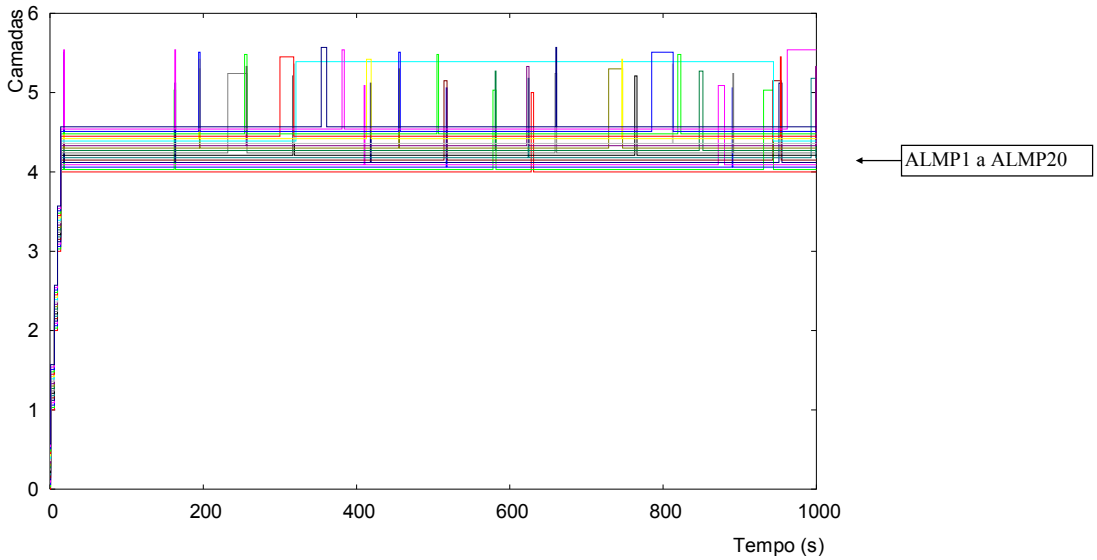


FIGURA 5.27 –20 fluxos concorrentes com fila *droptail* de 60

O gráfico da figura 5.28 mostra a eqüidade de tráfego do ALMP para 10 receptores e camadas exponenciais iguais às geradas pelo codificador VEBIT. As camadas do VEBIT foram definidas como 15 kbit/s, 45 kbit/s, 80 kbit/s, 150 kbit/s e 385 kbit/s. Como o enlace central possui 5 Mbit/s, ou 500 kbit/s por receptor, não existe um valor ótimo para que todos se adaptem, pois se os 10 receptores adaptarem-se na camada de número 4, vão ocupar 2,9 Mbit/s do enlace central (290 kbit/s por receptor), causando uma sobra de 2,1 Mbit/s, o que não é possível pelo algoritmo, pois ele foi projetado para sempre tentar ocupar toda a banda disponível. O ideal seria que 5 receptores se cadastrassem na camada 5 (total de 3,375 Mbit/s) e os outros 5 na camada 4 (total de 1,45M), perfazendo uma utilização de banda de 4,825 Mbit/s. Entretanto, os receptores não têm como saber o que está acontecendo com os outros, e efetuam tentativas para subir de camada. O resultado é que acontece uma troca de uso das camadas superiores por parte dos receptores, como pode ser visto no gráfico.

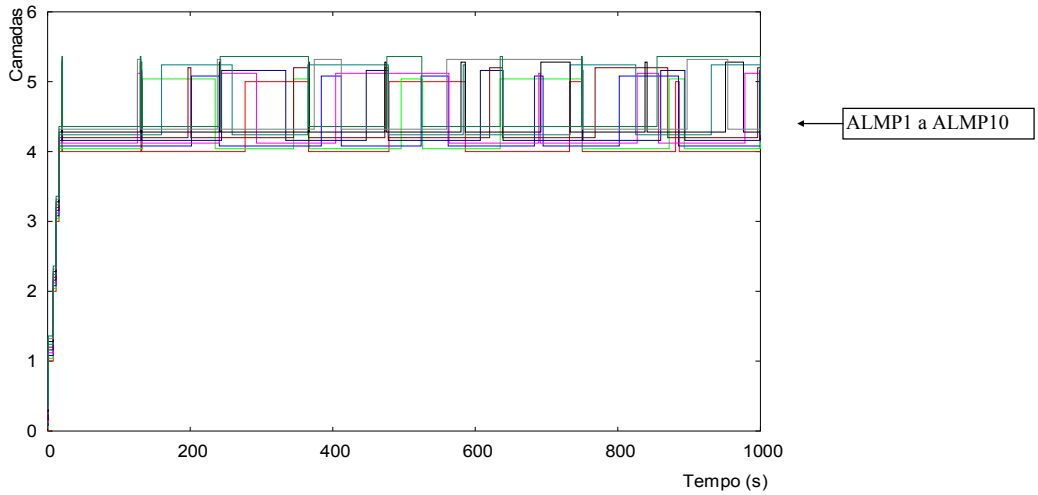


FIGURA 5.28 – Camadas iguais ao VEBIT – enlace de 5 Mbit/s

Refazendo a simulação para uma banda de 3,5 Mbit/s no enlace central, ou seja, 350 kbit/s por receptor, a adaptação se torna mais eficiente, pois todos receptores podem adaptar-se na mesma camada. Isso pode ser visto no gráfico da figura 5.29.

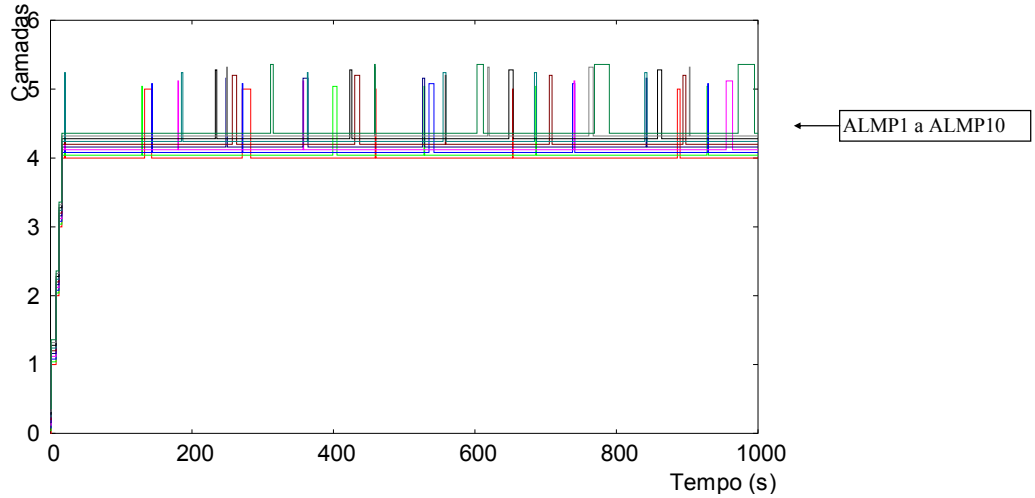


FIGURA 5.29 – Camadas iguais ao VEBIT – enlace de 3,5Mbit/s

Quando se utilizam camadas de 100 kbit/s, o espaçamento entre a camada atual e a próxima é menor do que com camadas exponenciais, tornando o sistema menos estável, como pode ser observado na figura 5.30. Pode-se observar que todos receptores ficaram a maior parte do tempo nas camadas 4 e 5, que correspondem à banda eqüitativa para o enlace central com 5 Mbit/s, perfazendo aproximadamente 500 kbit/s por receptor.

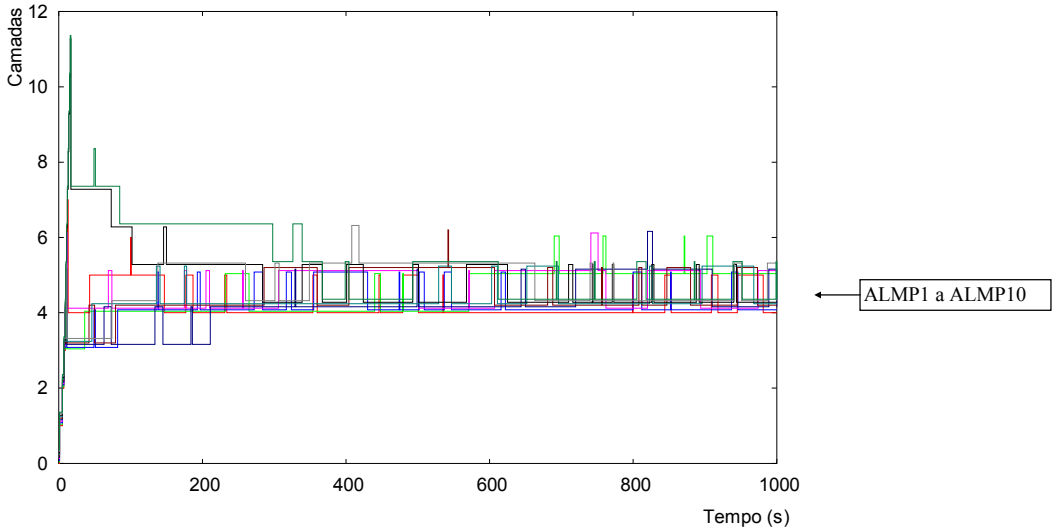


FIGURA 5.30 – 10 fluxos e camadas de 100 kbit/s

Para facilitar a visualização, o mesmo gráfico foi gerado mostrando a utilização de banda para cada receptor, e pode ser visto na figura 5.31. Da mesma forma que na simulação do VEBIT, pode-se perceber a troca de camada de alguns receptores, que está correto, pois o algoritmo foi projetado para ser eqüitativo com tráfego concorrente, e algumas vezes um receptor sobe de camada, forçando outro a descer.

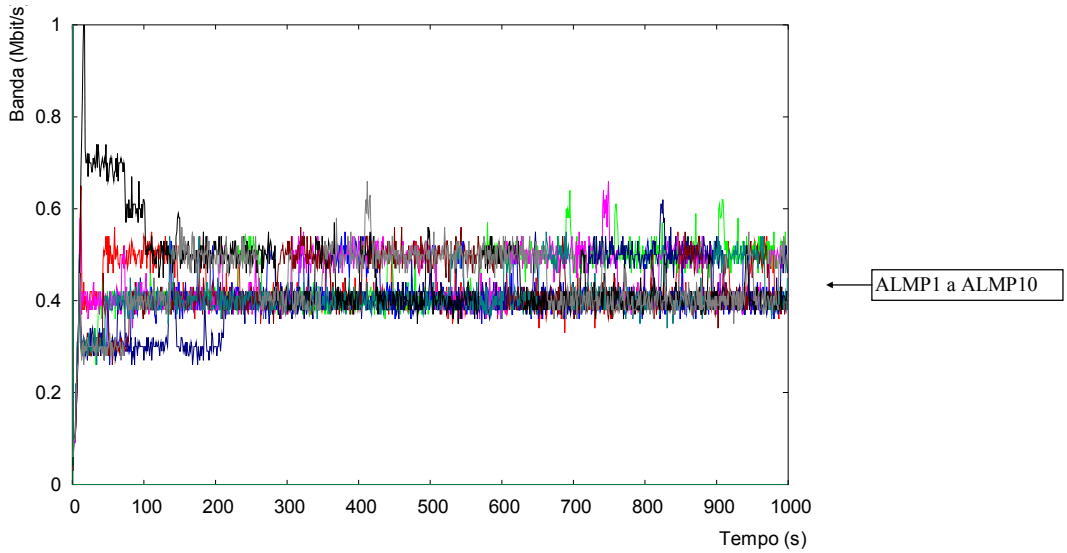


FIGURA 5.31 – 10 fluxos e camadas de 100 kbit/s (utilização da banda)

Em termos de vazão, a diferença entre o menor e o maior receptor foi de 21%. A vazão para cada receptor foi a seguinte: 419 kbit/s, 445 kbit/s, 405 kbit/s, 446 kbit/s,

401 kbit/s, 427 kbit/s, 410 kbit/s, 456 kbit/s, 439 kbit/s e 488 kbit/s.

A simulação ilustrada na figura 5.32 mostra a capacidade do algoritmo reduzir sua banda para dividir eqüitativamente o tráfego após estar em regime permanente. O fluxo ALMP1 já estava estabilizado quando entrou o fluxo ALMP2. Com a entrada do segundo fluxo, ocorreram perdas e os dois fluxos reduziram a banda, porém, como o fluxo ALMP1 possui uma banda maior, a redução é maior, gerando uma tendência de equilíbrio. Após um tempo de aproximadamente 300s, os dois fluxos estabilizaram em 4 camadas (equivalente a aproximadamente 450 kbit/s), coerente com a banda disponível de 1 Mbit/s.

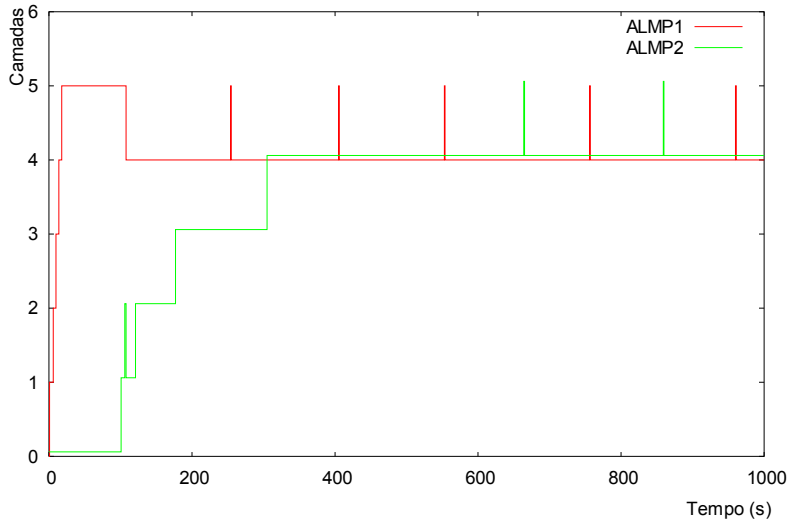


FIGURA 5.32 – Eqüidade do ALMP para dois fluxos, um iniciando 100s após o outro

Com cinco fluxos exponenciais, o algoritmo também atinge a eqüidade de tráfego, como mostra a figura 5.33, onde cada fluxo inicia 100s após o outro. Pode-se perceber a convergência do algoritmo para uma divisão eqüitativa de banda, que é alcançada na camada 4 (total de 450 kbit/s), pois o total de banda no enlace central é de 2,5 Mbit/s.

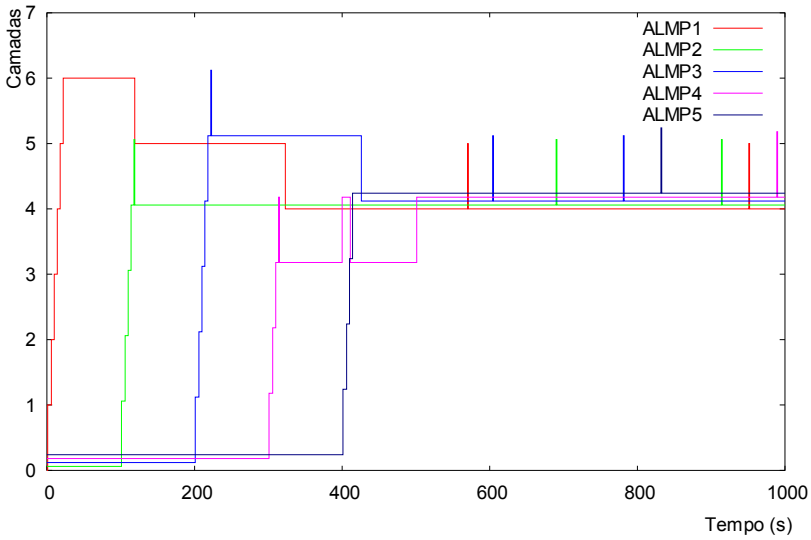


FIGURA 5.33 – Cinco fluxos exponenciais, cada um iniciando 100s após o outro

A figura 5.34 mostra a mesma simulação, porém com camadas de 100 kbit/s, cada fluxo iniciando 100s após o anterior. Devido ao maior número de camadas, o algoritmo leva um tempo maior para adaptar-se, mas após aproximadamente 600s ele atinge o regime permanente, com todos os fluxos distantes no máximo duas camadas.

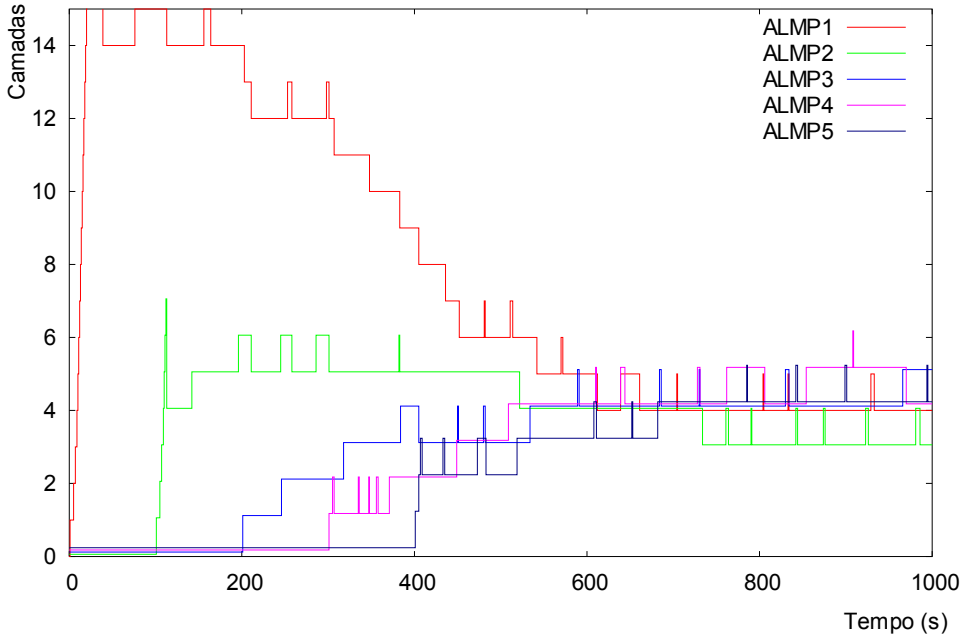


FIGURA 5.34 – Camadas de 100 kbit/s, cada fluxo iniciando 100s após o outro

Os resultados vistos nas figuras 5.32, 5.33 e 5.34 mostram que o ALMP consegue adaptar-se e compartilhar banda quando entram novos fluxos na rede.

Até o momento considerou-se que cada camada gera tráfego praticamente constante (tipo CBR), e introduziu-se uma certa randomicidade de ±0,5s no momento de transmissão do pacote. O codificador em camadas VEBIT, detalhado no capítulo 8, gera tráfego variável, entretanto, pode-se constatar através das figuras 8.12 e 8.13 que a taxa se mantém numa variabilidade baixa dentro da sua camada.

Para verificar como seria o funcionamento do algoritmo para uma alta variabilidade de tráfego, foram efetuadas simulações onde a característica de transmissão de cada camada é VBR do tipo exponencial ou do tipo pareto. Os resultados são apresentados a seguir.

Com tráfego VBR do tipo exponencial, as variáveis *burst-time* e *idle-time* foram configuradas para 500ms. O gráfico da figura 5.35 mostra o número de kbit/s transferido a cada segundo, que é bastante variável comparado ao CBR. O gráfico mostra a quantidade de bits por segundo que chega em cada um dos 10 receptores ALMP que estão compartilhando um único enlace de 5 Mbit/s na topologia 2.

A figura 5.36 mostra a mesma simulação vista na figura 5.35, porém, do ponto de vista da camada na qual cada receptor está inscrito. Pode-se ver que os receptores, apesar de estarem recebendo uma característica de tráfego bastante variável (figura 5.35), conseguem se manter de forma estável na camada correta (camada 4, total de 450 kbit/s), e compartilhando banda eqüitativamente com os outros fluxos ALMP.

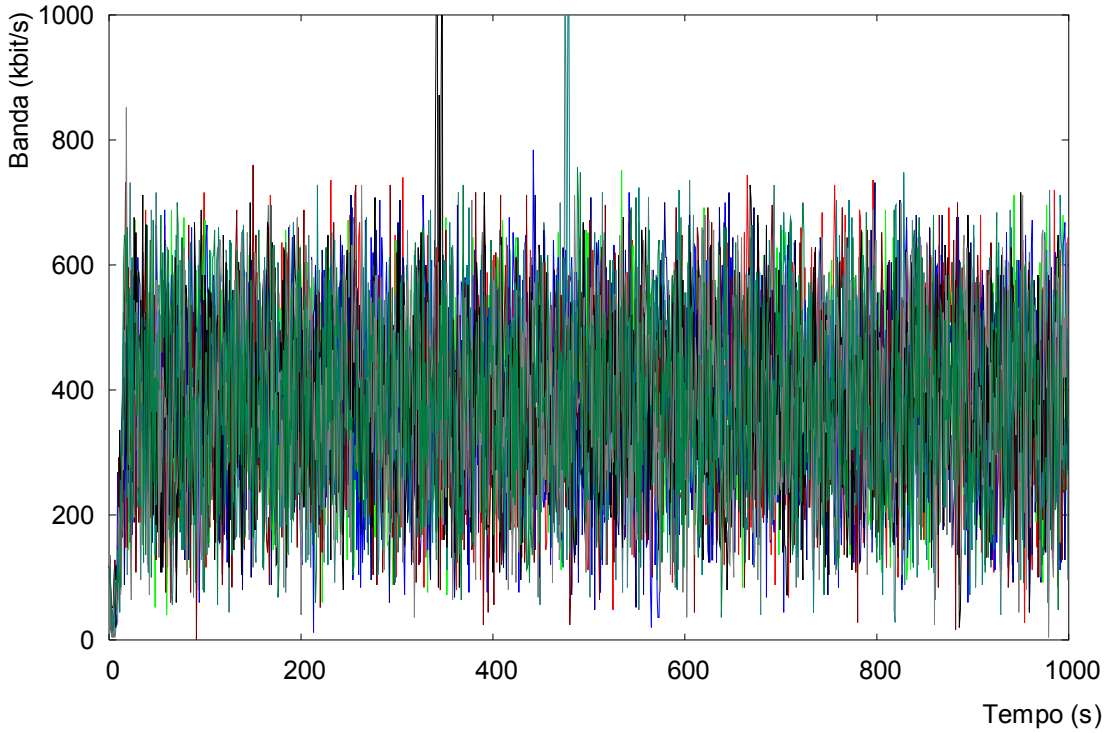


FIGURA 5.35 – Tráfego VBR Exponencial. Banda nos receptores

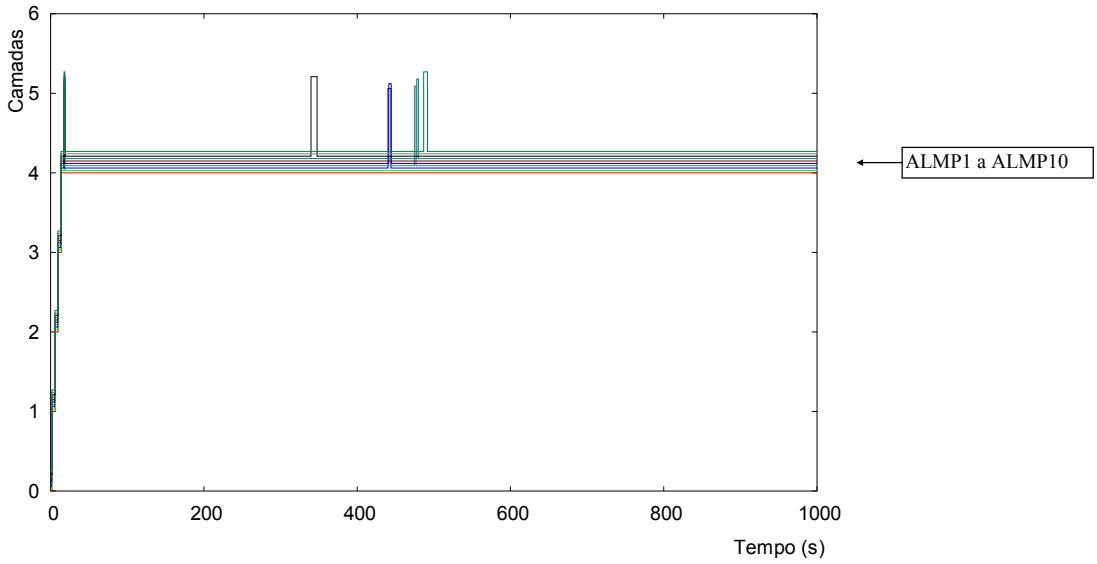


FIGURA 5.36 – Tráfego VBR Exponencial. Camadas nos receptores

A figura 5.37 mostra a visão da banda transferida para outra simulação de 10 fluxos ALMP compartilhando um enlace de 5 Mbit/s, porém com tráfego VBR do tipo pareto. As variáveis *burst-time* e *idle-time* foram configuradas para 500ms, e o parâmetro *shape* foi configurado para 1,5.

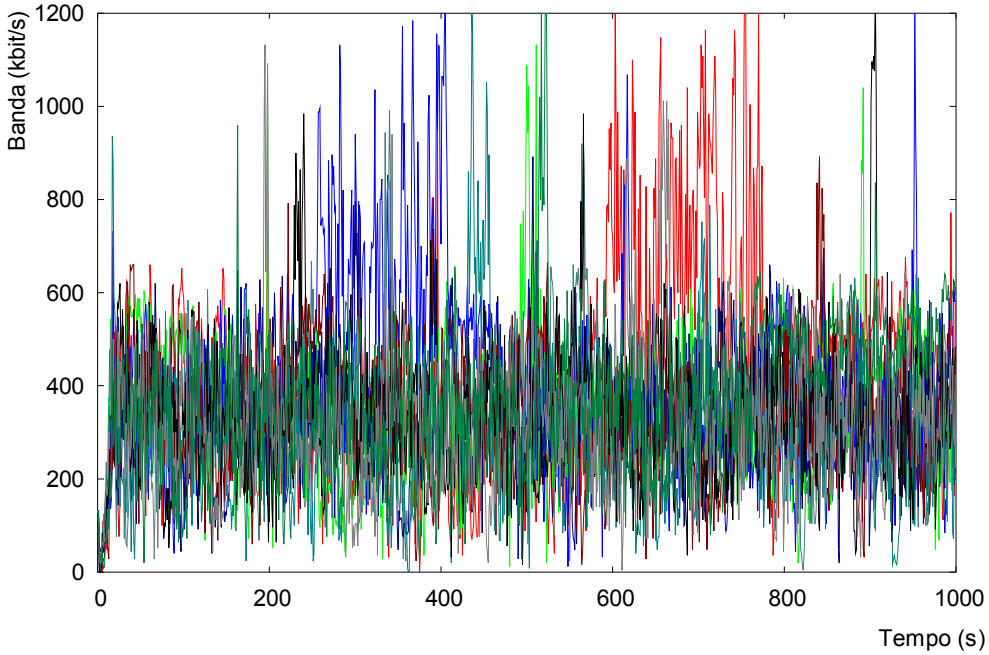


FIGURA 5.37 – Tráfego VBR Pareto. Banda nos receptores

A figura 5.38 mostra a mesma simulação da figura 5.37, porém, do ponto de vista da camada na qual cada receptor está inscrito. Pode-se ver que os receptores, apesar de estarem recebendo uma característica de tráfego bastante variável (figura 5.37), conseguem se manter de forma estável na camada correta (camada 4, que dá um total de 450 kbit/s), e compartilhando banda eqüitativamente com os outros fluxos ALMP. O número de tentativas para subir camada foi levemente superior ao do tráfego exponencial, entretanto, de forma geral os receptores adaptaram-se adequadamente.

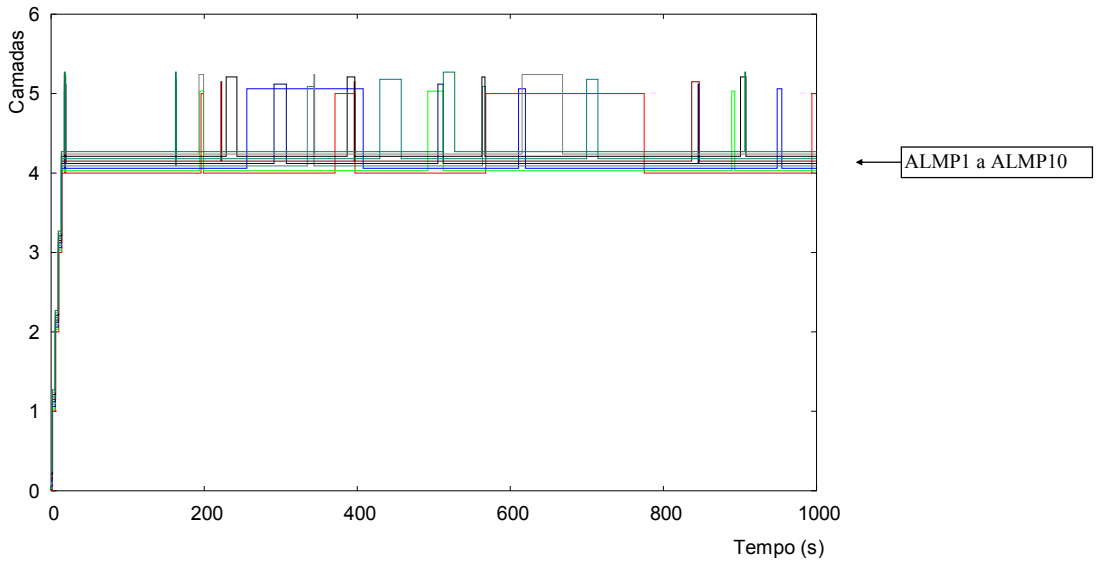


FIGURA 5.38 – Tráfego VBR Pareto. Camadas nos receptores

Os gráficos das figuras 5.35 a 5.38 mostram que o ALMP se adapta corretamente, mantendo equidade de tráfego, mesmo para transmissão VBR. Pode-se concluir que o sistema vai funcionar com a utilização de um codificador que gere tanto

tráfego CBR como tráfego VBR.

Em relação ao compartilhamento de banda com tráfego CBR, o algoritmo adaptou-se corretamente, efetuando uma divisão eqüitativa da banda em todas as situações analisadas, como mostra a figura 5.39, onde um fluxo CBR de 500 kbit/s foi transmitido nos primeiros 200s de simulação, juntamente com dois fluxos ALMP, todos através de um enlace de 1 Mbit/s. O fluxo CBR não se adapta, forçando os dois fluxos ALMP a dividirem o restante da banda, ou seja, 500 kbit/s (equivalente a 3 camadas, ou 210 kbit/s cada um). Quando o fluxo CBR termina, no instante 200s, os fluxos ALMP se adaptam e utilizam a banda total do enlace principal, ou seja, 1 Mbit/s (equivalente a 4 camadas, ou 450 kbit/s cada fluxo). Os dois fluxos ALMP demoram entre 30s e 50s para adaptar-se, e dividem igualmente a banda, como pode ser visto na figura.

No instante 600s, inicia o segundo fluxo CBR, provocando perdas para os fluxos ALMP. Eles se adaptam, e voltam a compartilhar o restante da banda, ou seja, 500 kbit/s (no máximo três camadas, ou 210 kbit/s por fluxo).

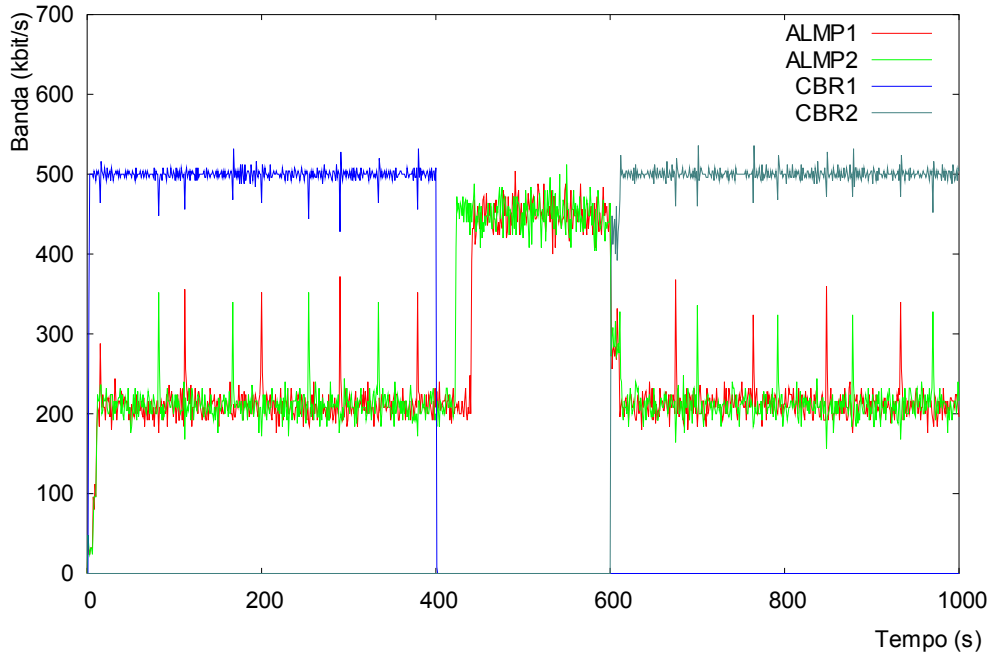


FIGURA 5.39 – Eqüidade de tráfego do ALMP com tráfego CBR

Foram feitas outras simulações, conforme descrição no item 4.8 (Definição das simulações), entretanto, elas não geraram alterações significativas nos resultados, e somente serão comentadas brevemente a seguir, entretanto, estão disponíveis em http://www.inf.unisinos.br/~roesler/tese.

- **Fila RED 10/20 e camadas exponenciais**: O resultado com 10 fluxos concorrentes foi muito semelhante ao visto na figura 5.25;
- **Atraso de 1ms no enlace principal**: com um menor atraso, o resultado ficou semelhante ao visto na figura 5.25;
- **Pacote com tamanho 250 bytes ou 1000 bytes ao invés de 500 bytes**: o resultado ficou bastante semelhante ao visto na figura 5.25. Da mesma forma que foi visto nos testes de adaptabilidade, o que muda é a quantidade de pacotes enviados por

segundo para manter a taxa constante da camada.

5.2.4 Análise da estabilidade do ALMP

A estabilidade do algoritmo define a quantidade de variações de camada que aconteceram por minuto após a fase inicial (*start-state*), conforme definição de VCM no item 4.8.4. Assim, pode-se construir uma tabela com os principais resultados de estabilidade obtidos nas simulações anteriores do ALMP. Vale lembrar que, quando um algoritmo sobe de camada, gera perdas e desce de camada, conta como duas variações.

A tabela 5.2 mostra o número de variações de camada por minuto (VCM) de várias simulações anteriores. A primeira coluna fornece uma descrição sucinta da simulação, enquanto a segunda coluna associa a figura referente à simulação em questão. A terceira coluna mostra o total de variações de camada por minuto dos fluxos participantes. No caso onde existem vários receptores, é fornecido o pior resultado, o melhor resultado e a média. Para o cálculo do VCM, utilizou-se o programa em linguagem *perl vcm.pl*, descrito no item 4.7.5 (Metodologia para análise de resultados no simulador), que fornece o VCM do fluxo especificado como parâmetro de entrada. Em linhas gerais, o programa calcula todas as trocas de camada do fluxo e divide pelo total de minutos da simulação (no caso, 16,66, pois as simulações são de 1000 segundos). As seguintes conclusões podem ser obtidas a partir da tabela 5.2 e dos gráficos relacionados:

- Fluxos com menos banda possuem, geralmente, um VCM mais alto, e isso se deve ao maior número de tentativas de subir camada, pois a taxa de incremento é maior (isso pode ser constatado pela primeira linha da tabela, referente à figura 5.4, onde o fluxo ALMP1 ficou constante, sem variações, enquanto o fluxo ALMP4 sofreu 3,12 variações de camada por minuto);
- Camadas utilizando VEBIT possuem mais variação do que as camadas exponenciais padrão. Isso acontece pois, quanto mais baixa a banda em que o algoritmo se estabiliza, mais tentativas de subir camada ele faz, devido à própria concepção do ALMP. Assim, como as camadas do VEBIT possuem banda mais baixa que as camadas exponenciais padrão, o número de tentativas é maior (visto através da comparação entre as figuras 5.4 e 5.8);
- A utilização de pares de pacotes impede que o algoritmo tente subir camadas além da sua banda máxima, provocando uma maior estabilidade ao sistema (visto através de todas figuras que tem as duas simulações, sem e com pares de pacotes);
- Quanto mais próximas estiverem as camadas, menor a estabilidade, pois o algoritmo vai efetuar tentativas de subir camada de forma mais freqüente (visto através da comparação entre as figuras 5.18 e 5.21);
- Alguns receptores aprendem através das tentativas de outros receptores, conforme detalhado no algoritmo. Isso pode ser constatado nas figuras com 100 receptores, onde o grau individual de estabilidade é maior do que em relação aos fluxos individuais. Por exemplo, comparando o gráfico da figura 5.18 com o mostrado na figura 5.4, que é a mesma simulação, porém com apenas um receptor por bloco, pode-se perceber que o número de tentativas para subir camada é maior de uma forma geral, entretanto, o número de tentativas por receptor é menor. Por exemplo, enquanto o receptor da camada 2 na figura 5.4 tentou subir de camada 26 vezes,

cada receptor na figura 5.18 fez entre 3 e 5 tentativas para subir camada. Isso acontece pois o erro causado pela tentativa de um receptor gera perda para todos os outros, fazendo com que eles nem tentem subir de camada;

- Quando diversos fluxos compartilham uma mesma banda, eles aprendem com os outros fluxos, da mesma forma que visto no item anterior, com 100 receptores, onde eles aprendem com outros receptores da mesma sessão. Isso pode ser visto através das figuras de eqüidade de tráfego, onde os receptores possuem um índice VCM menor do que os fluxos individuais (Comparando o VCM do fluxo ALM3 da figura 5.4 com a média da figura 5.25, por exemplo);

TABELA 5.2 – Estabilidade – número de variações de camada por minuto (VCM)

Descrição (adaptabilidade)	**Figura**	**VCM** **ALM1, ALM2, ALM3, ALM4**
Camadas exponenciais sem PP[1]	figura 5.4	0,00 0,24 0,60 3,12
Camadas exponenciais com PP	Figura 5.6	0,00 0,00 0,00 0,00
Camadas VEBIT sem PP	Figura 5.8	0,00 0,00 0,96 4,20
Camadas VEBIT com PP	Figura 5.10	0,00 0,00 0,00 0,00
Camadas 100 kbit/s com PP (fila 20)	Figura 5.13	1,25 0,17 0,11 0,00
Camadas 100 kbit/s com PP (fila 60)	Figura 5.13	0,00 0,00 0,00 0,00
Camadas exponenciais fila RED sem PP	Figura 5.14	0,00 0,23 0,59 1,19
Camadas exponenciais fila RED com PP	Figura 5.14	0,00 0,00 0,00 0,00
Descrição (escalabilidade)	**Figura**	**VCM: Bloco1 a Bloco4.** **(Pior, melhor e média)**
Camadas exponenciais (100 receptores) sem PP	Figura 5.18	B1: 0,00 0,00 0,00 B2: 0,11 0,00 0,02 B3: 0,89 0,00 0,11 B4: 0,83 0,35 0,48
Camadas exponenciais (100 receptores) com PP	Figura 5.20	VCM = 0 em todos casos
Camadas de 100k (100 receptores) sem PP	Figura 5.21	B1: 1,43 0,83 1,20 B2: 1,85 0,95 1,44 B3: 1,91 0,97 1,30 B4: 1,07 0,35 0,48
Camadas de 100k (100 receptores) com PP	Figura 5.21	B1: 1,20 0,00 1,15 B2: 0,65 0,11 0,25 B3: 0,11 0,00 0,10 B4: 0,00 0,00 0,00
Descrição (eqüidade)	**Figura**	**VCM** **(Pior, melhor e média)**
Eqüidade p/ 10 fluxos – camadas exponenciais	Figura 5.25	0,35 0,00 0,05
Eqüidade p/ 20 fluxos – camadas exponenciais	Figura 5.27	0,41 0,00 0,19
Eqüidade com 10 fluxos – camadas de 100k	Figura 5.30	1,91 0,83 1,34
Eqüidade com 10 fluxos – tráfego VBR expon.	Figura 5.36	0,11 0,00 0,01

Algumas simulações evidenciaram uma característica da transmissão em camadas quando não se utiliza QoS. O primeiro gráfico da figura 5.13 mostra, próximo

[1] Pares de Pacotes

ao instante 600s, que quando o receptor ALMP1 começa a perder pacotes (no caso por causa do excesso de fluxo nas filas do roteador), as perdas acontecem em todas as camadas. A conseqüência disso é que o receptor ALMP2 e o ALMP3 sentem essas perdas e também diminuem de camada. A alternativa para isso é o uso de QoS com alta prioridade nas camadas baixas, decrescendo de prioridade para os fluxos mais altos, assim, na hora de descartar pacotes, o roteador iria descartar somente dos fluxos altos, mantendo a estabilidade para os fluxos mais baixos.

5.2.5 Análise do tempo de adaptação do ALMP

O algoritmo do ALMP é dividido em duas fases: *start-state* e *steady-state*. A fase *start-state* é utilizada para tentar fazer o algoritmo adaptar-se rapidamente à sua banda eqüitativa, conforme detalhado anteriormente, e pode ser observada claramente na figura 5.8, cujos primeiros 50s de simulação foram reproduzidos na figura 5.40 a fim de detalhar o processo de adaptação.

Observa-se, através da figura 5.40, que o receptor ALMP4 foi o primeiro a adaptar-se, em 17s. Em seguida os receptores ALMP1 e ALMP2, em 23s. O receptor ALMP3 diminuiu uma camada, se adaptando em 24s.

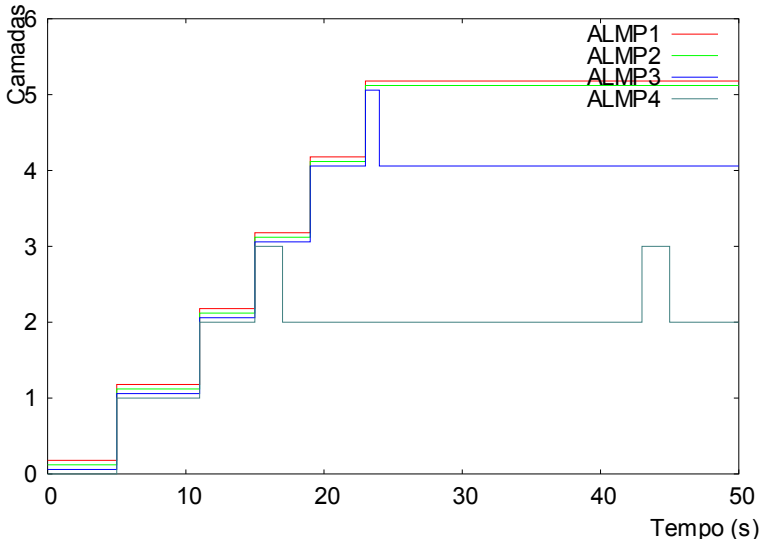


FIGURA 5.40 – Fase *start-state* do ALMP com camadas exponenciais iguais ao VEBIT

O algoritmo possui uma proteção para que o receptor suba no máximo uma camada por vez, a fim de evitar um grande número de perdas caso o receptor suba muitas camadas de uma vez só, portanto, normalmente quanto maior o número de camadas que o receptor deve se inscrever, maior o tempo de adaptação. Isso é ilustrado na figura 5.41 para camadas de 50 kbit/s. A fase *start-state* vai até a detecção de perdas, quando o algoritmo passa para a fase *steady-state*, decrementando 30% da banda para o receptor, como pode ser visto no instante 50s para o receptor ALMP1.

A fase *start-state* tem por objetivo levar o receptor a utilizar uma banda próxima à eqüitativa, e caso necessário, ele deve atingir o restante da adaptação através da fase *steady-state*. Isso pode ser constatado para todos os receptores na figura 5.41, que mostra um extrato de simulação com camadas de 50 kbit/s, transmissão *CBR puro* e fila *droptail* de 20 pacotes. Observa-se que o receptor ALMP4 atingiu a adaptação próxima

ao instante 13s, quando foi para a camada 2. O receptor ALMP3 saiu da fase *start-state* no instante 21s, indo para a camada 8, descendo para a camada 7 no instante 32s devido às perdas geradas pelo receptor ALMP2 (conforme foi detalhado no item sobre estabilidade). Próximo ao instante 60s os receptores ALMP2, ALMP3 e ALMP4 sofreram com as perdas geradas pelo receptor ALMP1.

Após a fase *start-state*, a adaptação é bem mais lenta, buscando aumentar a estabilidade do sistema como um todo. O receptor ALMP3 foi atingir a adaptação aproximadamente no instante 150s, quando foi para a camada 10, como pode ser constatado no gráfico da figura 5.11. Na mesma figura pode ser visto que o receptor ALMP2 atingiu a adaptação próximo ao instante 250s. Já o receptor ALMP1 foi para a camada 29 quando saiu da fase *start-state*, e como sua banda é alta, a taxa de incremento é baixa, fazendo com que o valor de *bwshare* demore a atingir banda suficiente para se cadastrar na próxima camada. Isso faz com que o receptor leve do instante 60s até o instante 600s para atingir a plena adaptação (conforme visto na figura 5.11).

A simulação ilustrada na figura 5.41 mostrou que, quando o receptor detecta perdas muito cedo, saindo da fase *start-state* sem atingir uma camada próxima da correta, leva mais tempo para que ele se adapte plenamente. O algoritmo foi projetado desta forma a fim de aumentar a estabilidade do sistema, partindo do pressuposto que a estabilidade é mais importante do que o tempo de adaptação em transmissões de vídeo.

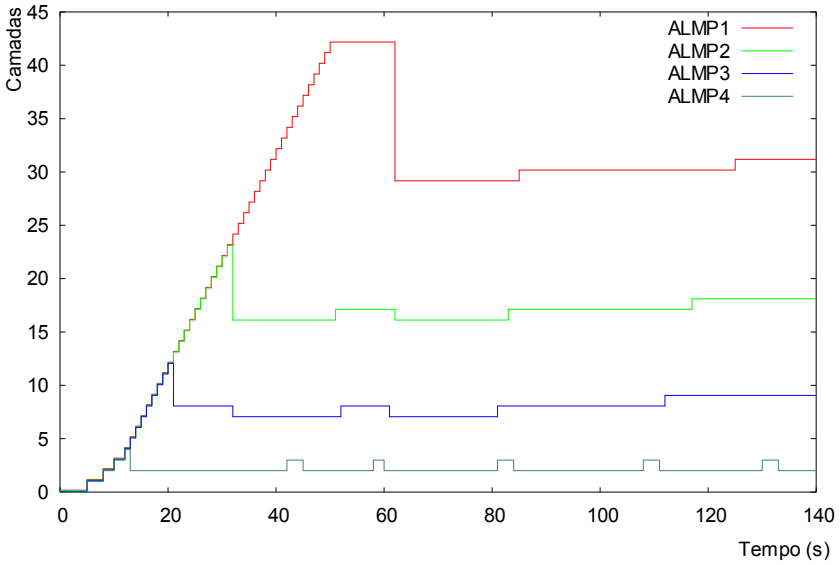


FIGURA 5.41 – Fase *start-state* do ALMP com camadas de 50 kbit/s

Quando existem vários fluxos concorrendo pela mesma banda, a adaptação pode ser rápida, conforme pode ser visto na figura 5.25, onde os 10 fluxos concorrentes atingiram rapidamente a banda eqüitativa já na fase *start-state* (aproximadamente 20s). Em outros casos, o receptor sofre perdas no início da fase de *start-state*, e deve atingir a banda eqüitativa através da fase *steady-state*, que é mais lenta. Essa situação pode ser constatada através da figura 5.32, onde o segundo receptor iniciou no instante 100s, sofrendo perdas e ficando na camada 1, necessitando atingir a eqüidade de banda através da fase *steady-state*. A adaptação demorou aproximadamente 5 minutos, pois o receptor estabilizou somente próximo ao instante 300s.

5.2.6 Análise da taxa de perdas do ALMP

A saída de todas as simulações mostra, entre outras informações, o total de pacotes recebidos e a quantidade de perdas, conforme foi descrito no item 4.7.5 (Metodologia para análise de resultados no simulador). Os parágrafos a seguir vão analisar alguns gráficos com base na taxa de perdas obtida.

A tabela 5.3 mostra um resumo da taxa de perdas para as principais simulações efetuadas. A primeira coluna fornece uma descrição sucinta da simulação, enquanto a segunda coluna associa a figura referente à simulação em questão. A terceira coluna mostra a taxa de perdas dos fluxos participantes. No caso onde existem vários receptores, é fornecido o pior resultado, a média e o melhor resultado.

TABELA 5.3 – Taxa de perdas para simulações do ALMP

Descrição (adaptabilidade)	**Figura**	**Taxa de perdas: ALMP1 a ALMP4**
Camadas exponenciais sem PP	Figura 5.4	0,00%; 0,17%; 0,41%; 3,09%
Camadas exponenciais com PP	Figura 5.6	0,00%; 0,00%; 0,00%; 0,00%
Camadas VEBIT sem PP	Figura 5.8	0,00%; 0,00%; 0,33%; 3,34%
Camadas VEBIT com PP	Figura 5.10	0,00%; 0,00%; 0,00%; 0,00%
Camadas 100 kbit/s com PP (fila 20)	Figura 5.13	0,01%; 0,11%; 0,40%; 4,70%
Camadas 100 kbit/s com PP (fila 60)	Figura 5.13	0,01%; 0,01%; 0,00%; 0,01%
Camadas exponenciais fila RED sem PP	Figura 5.14	0,00%; 0,25%; 1,05%; 5,48%
Camadas exponenciais fila RED com PP	Figura 5.14	0,00%; 0,00%; 0,00%; 0,00%
Descrição (escalabilidade)	**Figura**	**Perdas: Bloco1 a Bloco4. (Pior, melhor e média)**
Camadas exponenciais (100 receptores) sem PP	Figura 5.18	B1: 0,01% 0,00% 0,00% B2: 0,81% 0,21% 0,25% B3: 6,02% 0,74% 1,02% B4: 6,18% 5,95% 6,04%
Camadas exponenciais (100 receptores) com PP	Figura 5.20	B1: 0,00% 0,00% 0,00% B2: 0,00% 0,00% 0,00% B3: 0,00% 0,00% 0,00% B4: 0,00% 0,00% 0,00%
Camadas cont. 100k (100 receptores) sem PP	Figura 5.21	B1: 0,01% 0,01% 0,01% B2: 0,01% 0,07% 0,06% B3: 0,35% 0,33% 0,34% B4: 4,88% 4,72% 4,79%
Camadas cont. 100k (100 receptores) com PP	Figura 5.21	B1: 0,01% 0,01% 0,01% B2: 0,01% 0,01% 0,01% B3: 0,01% 0,00% 0,01% B4: 0,00% 0,00% 0,00%
Descrição (eqüidade)	**Figura**	**Perdas (Pior, melhor e média)**
Eqüidade com 10 fluxos – camadas exp.	Figura 5.25	0,07% 0,05% 0,05%
Eqüidade com 20 fluxos – camadas exp.	Figura 5.27	0,10% 0,04% 0,06%
Eqüidade com 10 fluxos – camadas cont. 100k	Figura 5.30	0,07% 0,04% 0,05%
Eqüidade com 10 fluxos – VBR exponencial	Figura 5.36	0,23% 0,13% 0,15%

As seguintes conclusões podem ser obtidas a partir dos valores obtidos na tabela 5.3:

- O número de perdas com pares de pacotes tende a ser menor, devido ao menor número de tentativas de subida de camada quando o limite da banda é atingido (visto através das figuras 5.4 e 5.6, por exemplo);
- Quanto menor a banda, maior o número de tentativas de subir camada, portanto, maior a taxa de perdas (visto através das figuras 5.4 e 5.8). Isso é consistente com o algoritmo, pois quanto menor a banda, maior a taxa de incremento utilizada pelo receptor, provocando mais tentativas de subir camada e, conseqüentemente, maior número de perdas, como pode ser verificado também no gráfico;
- Quanto maior o número de fluxos concorrentes, maior a taxa de perdas, pois vão existir mais tentativas para subir camada (visto através da figuras 5.4 e 5.18 (média)).

5.3 Implementação do algoritmo ALMP

A implementação do ALMP foi publicada em [ROE 2002b], e foram desenvolvidos dois aplicativos: um transmissor CBR de multicast em camadas e um receptor que executa o algoritmo do ALMP, se adaptando conforme descrito no item 5.1. O código-fonte e executável do ALMP podem ser obtidos em http://www.inf.unisinos.br/~roesler/tese e no CD anexo a esta Tese.

O transmissor lê um arquivo de configuração contendo uma linha de descrição para cada camada. Nesse arquivo consta: IP multicast da camada; porta destino a ser enviado o pacote; taxa em kbit/s e tamanho do pacote em bytes. Através dessa configuração, ele cria uma *thread* para transmitir cada camada. No momento que vai transmitir determinado pacote, a *thread* adiciona o número de seqüência e o *timestamp* ao pacote. Através da taxa de cada camada, a *thread* calcula um tempo de espera antes de enviar o próximo pacote, e fica assim indefinidamente.

Na implementação mostrada a seguir, o arquivo de transmissão não transmite dados multimídia, pois o codificador em camadas ainda não está integrado. O transmissor fica em laço enviando um arquivo qualquer, obedecendo as taxas especificadas para cada camada.

O receptor lê o mesmo arquivo de configuração utilizado pelo transmissor, através de uma conexão unicast, conforme descrito no capítulo 4. Dessa forma, descobre a taxa de cada camada e a banda necessária para se inscrever em uma nova camada (grupo multicast). O receptor também abre uma nova *thread* para cada camada, e abre uma *thread* de controle (sub-rotina *ALM_EI*, conforme descrito no item 5.1). Conforme a variável *bwshare* permite, o receptor faz *join* em novas camadas, passando a receber os dados do grupo multicast correspondente, ou efetua *leave* de acordo com a necessidade, parando de receber o tráfego associado.

A figura 5.42 mostra a interface com o usuário no lado do receptor. Para cada camada pode-se ver seu número (*Layer*), sua taxa (*Rate*), IP multicast (*IP address*), número de pacotes perdidos nos últimos 5 segundos (*Lost 5s*) e o número de pacotes perdidos desde o início da recepção (*Lost tot*).

A interface também permite a visualização de algumas variáveis do algoritmo,

como *bwshare*, que indica ao receptor a quantidade de banda que ele está autorizado a utilizar, *bwused*, que representa a quantidade de banda que o receptor está utilizando atualmente, e *total lost*, que indica o total de pacotes perdidos em todas as camadas desde o início da recepção. O gráfico facilita na visualização da estabilidade do sistema.

Pode-se observar na figura 5.42 a coerência entre *bwshare*, que autoriza o receptor a 890,61 kbit/s, e *bwused*, que mostra que o receptor está utilizando 480 kbit/s, pois para se cadastrar na camada 5 ele necessita de 992 kbit/s (soma das taxas 1 a 5).

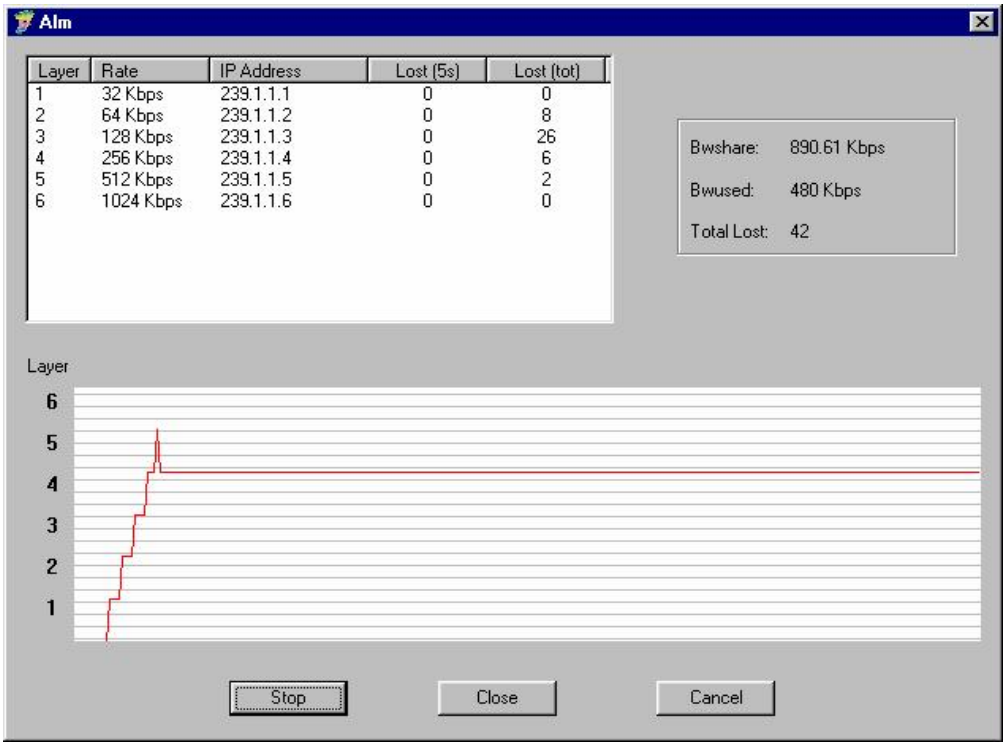


FIGURA 5.42 – Interface com o usuário – *receiver1* associado ao *sender1*

O ambiente de teste utilizado pode ser visto na figura 5.43. Cada transmissor envia 6 camadas distintas que crescem de forma exponencial conforme visto na figura 5.42. O transmissor *sender1* envia as camadas 239.1.1.1 a 239.1.1.6, e o transmissor *sender2* envia as camadas 239.1.1.11 a 239.1.1.16.

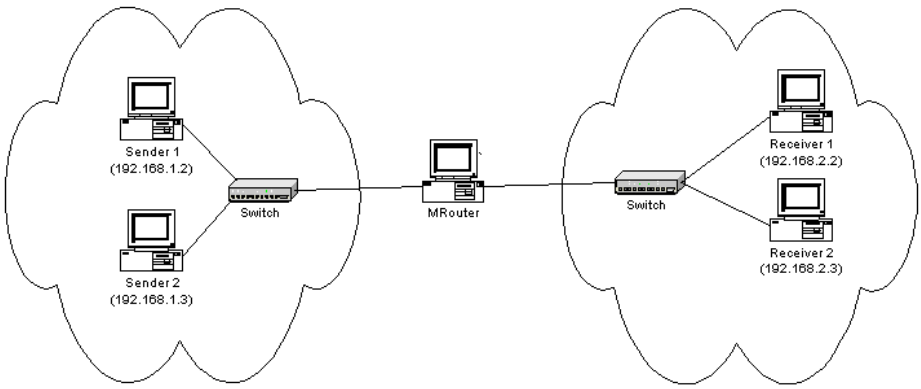


FIGURA 5.43 – Ambiente de teste

Os transmissores são ligados a um *switch* e este é ligado a um roteador. Na outra

sub-rede ficam os receptores (em outro *switch*). Nos testes, o roteador (máquina FreeBSD) estava com o multicast habilitado e limitava o tráfego em 1,2 Mbit/s. A versão do *mrouted* foi a 3.9 beta 3, alterado por Rizzo [RIZ 98] para menor tempo de *leave*.

Como o objetivo é ter uma ferramenta adaptativa ao usuário, foi especificado que ela funcionasse também em vários sistemas operacionais. Inicialmente, foi desenvolvido para Unix e Windows, pois isso deve atingir a maioria dos usuários.

A figura 5.44 mostra o resultado da saída do programa para o receptor *receiver1* e *receiver2*. Cada receptor se inscreveu nos grupos multicast enviados pelo transmissor correspondente. Como pode ser visto, o algoritmo funcionou de acordo com a teoria e as simulações, entretanto, algumas questões não foram tão simples, como o caso do tempo de *leave*, que demorava mais tempo na prática do que na simulação, gerando períodos de congestionamento levemente superiores.

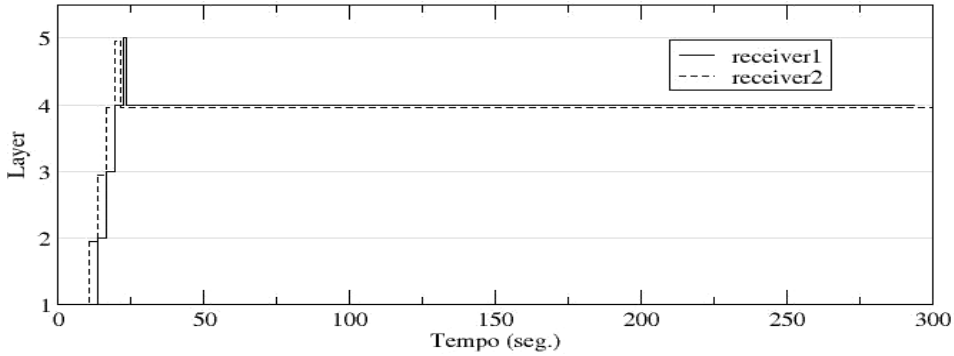


FIGURA 5.44 – Gráfico da adaptação do ALMP num ambiente real controlado

5.3.1 Modelo proposto

Para modelagem do sistema se utilizou um esquema que, além de servir como base para o teste do ALM, pode ser reutilizado para facilitar a implementação de novos protocolos de controle de congestionamento e CODECS em camadas. Esse ambiente de desenvolvimento também poderá ser portado facilmente para diferentes plataformas. A figura 5.45 apresenta o ambiente presente no transmissor (a) e no receptor (b), que se baseia em 3 camadas: a primeira, com informações dependentes do sistema operacional, possui as funções básicas para o tratamento da rede, interface com o usuário e recursos multimídia. Uma camada intermediária serve de conexão entre os recursos do sistema operacional e os módulos de transmissão / recepção e gerador / receptor dos dados. A última camada contém os módulos de transmissão, conformadores de tráfego e geração de dados.

Esse ambiente permite que o desenvolvedor de novas tecnologias de controle de congestionamento e codificação de dados em camadas possa criar uma aplicação facilmente, através de novos módulos. No caso do transporte, o *framework* possui funções básicas de controle de camadas, com diretivas como “aumentar uma camada” e monitorar a rede, de modo que os módulos de controle de congestionamento possam recolher informações como perda de pacotes. A comunicação de dados entre os módulos também é feita pelo *framework*. Um gerador de tráfego passa suas informações ao *framework* que, após conformar o tráfego via módulo conformador, executa a

transmissão. O receptor, ao receber os dados, repassa os mesmos aos seus módulos, que podem ser um decodificador de áudio e vídeo ou um gerador de estatísticas. Tanto na transmissão como na recepção o controlador de congestionamento informa ao *framework* como deve agir de modo a minimizar a perda de informações.

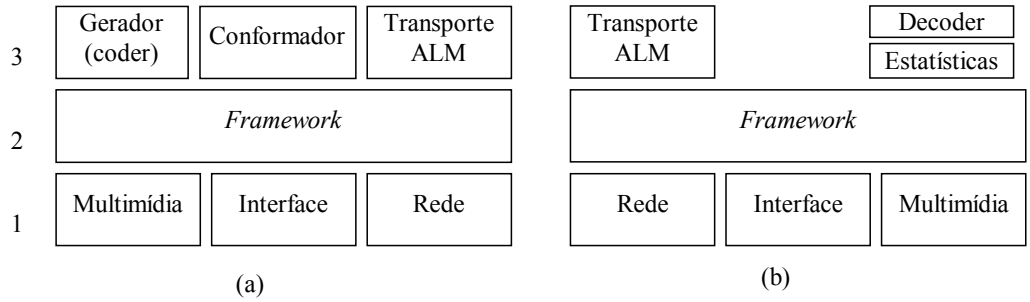


FIGURA 5.45 – Modelo da implementação

5.3.2 Problemas enfrentados

1. A versão do *mrouted* padrão do FreeBSD era IGMP 1, e não analisava o comando de *leave*. Isso fazia com que houvesse um tempo muito grande entre o receptor dar o *leave* num grupo multicast e os pacotes pararem de passar. Foi resolvido utilizando a versão 3.9 beta 3, conforme descrito anteriormente.
2. O comando *ipfw* utilizado no roteador era feito nas duas interfaces. Por exemplo, para limitar o enlace em 1 Mbit/s, o comando era: *ipfw pipe1 bw 1 Mbit/s queue droptail* e a mesma coisa para *pipe2*. Limitando o tráfego para 1Mbit/s causava o descarte aleatório de pacotes multicast para todos os grupos que estavam chegando na placa de rede do roteador (mesmo sem serem redirecionados para a outra sub-rede). Isso fazia com que o sistema não estabilizasse. A causa disso é que os testes iniciais foram de apenas duas *sub-redes*, como mostra a figura 5.43, e na primeira sub-rede existia permanentemente a presença de todos os grupos multicast. A solução foi utilizar o "*pipe1*" a 100Mbit/s, ou seja, descongestionar a interface por onde chegam os grupos multicast. O resultado é que o descarte de pacotes ficou coerente, ou seja, somente na interface onde estavam os receptores.
3. A Interface em modo gráfico em linux é diferente do Windows (baseado em MFC). Para diminuir a diferença entre sistemas, utilizaram-se arquivos de configuração em modo texto ao invés de configuração na própria interface (tipo *file options*).
4. Problema de processamento (ou demora na atualização da interface). Na atualização da janela da interface com o usuário, o programa levava muito tempo atualizando, não lendo todos pacotes que chegavam da rede (em torno de 992 kbit/s). Como não lia todos pacotes, encarava como perdas, e o sistema diminuía camadas.

5.4 Conclusões sobre o ALMP

O objetivo deste capítulo foi mostrar o funcionamento algoritmo de adaptação do ALMP, bem como sua estabilidade com tráfegos CBR e VBR. O cenário de simulação foi o de uma linha virtual privativa onde os fluxos concorriam por um determinado enlace de baixa velocidade. A metodologia das simulações foi definida no item 4.8 (Definição das simulações).

O ALMP destina-se à transmissão de fluxos multimídia em multicast através de uma rede virtual privativa, que poderia ser criada na Internet através de um mecanismo de QoS, que separaria a rede tradicional (TCP) da rede multimídia. A vantagem seria a possibilidade de criar um mecanismo de adaptação não dependente do TCP, como o ALMP, que é mais estável, transmite de forma eqüitativa com suas próprias sessões e é escalável para um grande número de receptores, já que não existe necessidade de comunicação do receptor para o transmissor.

O algoritmo é composto de três parâmetros principais: Intervalo de Execução, taxa de incremento e taxa de decremento. Juntos eles são responsáveis pela estabilidade, agressividade e tempo de adaptação do algoritmo.

A mudança do intervalo de execução (dado pela variável *ALM_EI_delay*) modifica o tempo de adaptação do receptor e a estabilidade do sistema. Com um tempo relativamente pequeno (por exemplo, 100ms), o sistema alcança sua fatia eqüitativa no compartilhamento mais rapidamente, mas não é tão estável como seria com a execução em Intervalo de Execução maiores, como, por exemplo, 1s. Isso acontece pois quanto mais vezes o algoritmo é executado por minuto, mais vezes vai incrementar ou decrementar a variável *bwshare*.

Como a taxa de incremento é maior para os receptores que estão utilizando menos banda, e a taxa de decremento é maior para os receptores que estão utilizando mais banda, conclui-se, que a banda permitida a cada receptor vai tender a um ponto de equilíbrio, tanto se os receptores estiverem localizados em diferentes sessões ou localizados na mesma sub-rede.

Em relação à adaptação em redes heterogêneas, foi visto que o algoritmo se adapta de forma coerente com a topologia existente, entretanto, os receptores localizados em redes lentas aumentam mais rapidamente a variável *bwshare*, possuindo uma maior taxa de tentativas de subir camada e conseqüentemente gerando mais instabilidade que os receptores localizados em redes mais rápidas.

A utilização de pares de pacotes mostrou aumentar a estabilidade do algoritmo, entretanto, seu uso é particularmente útil quando a inscrição na próxima camada gera uma banda superior à do enlace. Nos casos onde há concorrência entre vários fluxos ALMP diferentes, percebe-se pouca diferença em relação à não utilização dos pares de pacotes. Outra conclusão é que o uso de pares de pacotes gera um congestionamento levemente superior nas filas dos roteadores intermediários, devido à rajada de dois pacotes. Em algumas simulações a solução foi aumentar o tamanho da fila nos roteadores intermediários, como visto nas figuras 5.12, 5.26 e 5.27.

Pode-se inferir que, num ambiente real, é melhor a utilização de poucas camadas, caso seja possível, pois isso diminui o número de fluxos nas filas dos roteadores intermediários, diminuindo também o tráfego de controle e o número de grupos multicast necessários ao algoritmo.

Em algumas simulações observou-se que a tentativa de subir camada de um receptor podia gerar perdas num roteador congestionado, e essas perdas aconteciam em todas as camadas, dependendo do fluxo que estivesse entrando na fila do roteador naquele momento. A conseqüência disso é que outro receptor conectado em outro roteador na seqüência poderia sentir essas perdas, chegando mesmo a efetuar *leave* em

algumas situações (conforme pode ser visto na figura 5.41). Somente numa rede que utilize descarte por prioridades [GOP 2000] é possível evitar o problema das perdas acontecerem em todas as camadas. Nesse caso, a prioridade da camada mais alta é menor que das camadas mais baixas, fazendo com que elas sejam descartadas antes. No caso do ALMP, o objetivo é que o algoritmo rode sem filas especiais nos roteadores, portanto, as simulações não levaram em conta essa possibilidade. Vale lembrar que o algoritmo se beneficiaria desse tipo de fila por prioridade.

A partir das simulações de escalabilidade obtidas, pode-se concluir que o sistema aceita um grande número de usuários sem perder suas características. Uma das razões para isso é o fato do algoritmo não exigir que os receptores se comuniquem com o transmissor. Entretanto, o número de tentativas de subir camada provoca um leve aumento no número de perdas, provocando certa instabilidade em alguns casos. Uma diminuição nesse número de tentativas é obtida através do mecanismo de geração de número aleatório antes de subir camada, que permite um aprendizado de alguns receptores a partir das tentativas efetuadas por outros receptores (que geraram números aleatórios menores).

Em termos de eqüidade de tráfego, os resultados mostraram que o algoritmo é adequado para uso em redes privativas sem tráfego TCP concorrente. A adaptação mostrou ser eqüitativa com as sessões concorrentes para várias situações. Os resultados vistos nas figuras 5.32, 5.33 e 5.34 mostram que o ALMP consegue compartilhar banda adequadamente quando está em regime permanente e entram novos fluxos na rede.

Os gráficos das figuras 5.35 a 5.38 mostram que o ALMP se adapta corretamente, mantendo equidade de tráfego, mesmo para transmissões VBR. Pode-se concluir que o sistema vai funcionar com a utilização de um codificador que gere tanto tráfego CBR como tráfego VBR.

A simulação ilustrada na figura 5.41 mostrou que, quando o receptor detecta perdas muito cedo, saindo da fase *start-state* sem atingir uma camada próxima da correta, leva mais tempo para que ele se adapte plenamente. O algoritmo foi projetado desta forma a fim de aumentar a estabilidade do sistema, partindo do pressuposto que a estabilidade é mais importante do que o tempo de adaptação em transmissões de vídeo.

A implementação inicial deste algoritmo mostrou que os resultados num ambiente controlado de teste estão de acordo com os obtidos nas simulações.

Um problema do ALMP é que ele não é competitivo com tráfego TCP concorrente. Para competir com o TCP, é necessário efetuar uma modificação nos seus parâmetros, tornando-o mais agressivo. Mesmo assim, como o ALMP possui intervalo de execução fixo, e o intervalo de execução do algoritmo do TCP é dependente do RTT, o sistema nunca terá tráfego totalmente competitivo com o TCP. Para aumentar a agressividade do ALMP, é necessário modificar os parâmetros responsáveis pela taxa de subida, descida e intervalo de execução.

Apesar disso, o ALMP é muito mais estável que o TCP, se adaptando de forma suave a outros tráfegos de mesmo tipo, conforme visto nas simulações, tornando-o propício para utilização com tráfego multimídia. Outra vantagem do algoritmo é que ele não depende do RTT, gerando tráfego eqüitativo para receptores localizados em enlaces lentos e rápidos.

6 ALMTF: ALM TCP-Friendly

Após a experiência obtida com o algoritmo ALMP, detalhado no capítulo 5 (*ALMP: ALM for Private Networks*), buscou-se o desenvolvimento de um segundo algoritmo, desta vez com o objetivo de obter estabilidade juntamente com transmissão eqüitativa de tráfego mesmo na presença de sessões TCP concorrentes. Este algoritmo foi denominado ALMTF (*ALM TCP-Friendly*), e seu código fonte para o simulador, bem como o manual de instalação e resultado de todas simulações efetuadas pode ser obtido em http://www.inf.unisinos.br/~roesler/tese, bem como no CD anexo a esta Tese.

Quando não existe uma rede privativa virtual, as transmissões vão estar concorrendo com tráfego existente na Internet, que é composto 95% de TCP (conforme item 2.5), portanto, é importante que o ALMTF gere tráfego eqüitativo com TCP, mesmo que o TCP não seja eqüitativo com seu próprio tráfego, pois sua taxa de subida depende do RTT, conforme detalhado no item 2.6. Uma alternativa para obter tráfego eqüitativo com o TCP é utilizar um padrão equivalente de aumento e diminuição de banda.

Conforme detalhado no item 3.9 (TFMCC e TFRC), Floyd [FLO 2000] utiliza um mecanismo de subida e descida baseado em taxa buscando imitar o comportamento do TCP, entretanto, não segue o padrão AIMD (*Additive Increase Multiplicative Decrease*) a fim de obter maior estabilidade. O tempo necessário para diminuir pela metade a taxa de transmissão é entre quatro e oito RTTs, e o incremento na taxa é na ordem de, no máximo, 0,14 pacotes por RTT. No TCP Vegas, detalhado no item 3.12, Hengartner [HEN 2000] isolou o fator que provoca o maior desempenho do TCP Vegas em relação ao TCP utilizado atualmente (*new Reno*). Esse fator é a redução na janela de congestionamento por um fator de 25% ao invés de 50%.

Mesmo com os mecanismos alternativos citados, que são mais suaves, a agressividade necessária para obter tráfego eqüitativo com o TCP é bastante alta, pois o TCP é um protocolo que incrementa e decrementa rapidamente sua largura de banda, conforme visto no item 2.6.

Este capítulo tem por objetivo analisar o algoritmo ALMTF, detalhando a metodologia utilizada para que o mesmo tenha uma característica semelhante ao tráfego TCP, porém com maior estabilidade. Isso é conseguido com:

- Dois métodos complementares para controle de fluxo no receptor: a) modelo baseado em "janela de congestionamento" detalhado no item 6.1.1; b) modelo baseado na equação do TCP, detalhado no item 6.1.2. O receptor se comunica a intervalos regulares com o transmissor a fim de calcular o RTT, porém, o número de mensagens de retorno é limitado a fim de obter escalabilidade, conforme detalhes no item 6.1.4. A metodologia de cálculo do RTT é descrita no item 6.1.3, e o RTT é utilizado, juntamente com a detecção de perdas (descrita no item 6.1.5), para efetuar o controle de fluxo;
- Premissa de que, quanto mais agressivo for um algoritmo, menos estável fica o tráfego gerado, portanto, o ALMTF imita o comportamento do TCP de forma 10 vezes menos agressiva, tanto na taxa de subida como na taxa de descida. Isso torna a

resposta do algoritmo mais lenta, aumentando a estabilidade, porém, mantendo a eqüidade com tráfegos TCP. Detalhes no item 6.1.1;
- Sincronismo entre os receptores inscritos na mesma sessão, para que todos efetuem *join* no mesmo instante. Detalhamento no item 6.1.8;
- Dessincronismo entre os receptores localizados em sessões diferentes, a fim de que eles não efetuem o *join* no mesmo instante. Detalhamento no item 6.1.9;
- Utilização da técnica de pares de pacotes, descrita no Anexo A, para evitar exceder a capacidade da rede. Detalhes no item 6.1.7.

Este capítulo está dividido da seguinte forma: no item 6.1 será apresentado o detalhamento do algoritmo de controle de congestionamento ALMTF. No item 6.2 serão mostradas as simulações, e no item 6.3 será feita uma conclusão sobre os pontos positivos e negativos do algoritmo.

6.1 Detalhamento do algoritmo ALMTF

Para a implementação do algoritmo ALMTF, criou-se dois novos agentes no simulador NS2, em linguagem C++: o agente transmissor (*ALMTF Agent*) e o agente receptor (*ALMTFSink Agent*), além do código que implementa a lógica do algoritmo em si, feito na linguagem *tcl*.

O agente transmissor gera os pacotes em multicast nas taxas especificadas em cada camada. O agente receptor recebe os pacotes, calcula o RTT, o número de perdas e outras informações, atualizando algumas variáveis do código em *tcl*, que são utilizadas pelo algoritmo.

O cabeçalho dos pacotes ALMTF tem os campos mostrados na figura 6.1, com um total de 16 bytes, sendo utilizado para cálculo de erros, cálculo de RTT, cálculo de intervalo entre mensagens de *feedback* e sincronismo para subir camadas. O significado de cada campo é descrito a seguir.

32 bits

Numseq	Num_nodereceptor / Num_receptores	
Timestamp		
Timestamp_echo		
Timestamp_offset	sincjoin	Flags

FIGURA 6.1 – Cabeçalho dos pacotes ALMTF

- *Numseq*: número de seqüência do pacote. Utilizado para detecção de perda de pacotes no receptor;
- *Num_nodereceptor*: identificação do receptor para o qual está sendo enviada a resposta à "Solicitação de Eco". Utilizado para cálculo de RTT;
- *Num_receptores*: número de receptores atuais. Utilizado para evitar implosão de *feedback*;
- *Timestamp*: instante de tempo que foi transmitido o pacote, seja pacote de dados pelo transmissor ou pacote de *feedback* pelo receptor. Utilizado para cálculo de RTT;
- *Timestamp_echo*: cópia do instante de tempo no qual o transmissor recebeu um

pacote de “Solicitação de Eco”. Utilizado para cálculo de RTT;

- *Timestamp_offset*: diferença de tempo entre o transmissor receber uma “Solicitação de Eco” e enviar a resposta. Utilizado para cálculo de RTT;
- *Sincjoin*: aviso de sincronismo de *join*, enviado pelo transmissor em intervalos regulares de tempo. Essa variável determina até qual camada o receptor pode efetuar uma tentativa de *join*.
- *Flags*: as seguintes *flags* foram definidas:
  - FL_INIRAJADA (10000000): indicador de inicio de rajada. Utilizado para pares de pacotes ou trem de pacotes;
  - FL_MEDERTT (01000000): indica que este pacote é para medição de RTT, logo, o segundo campo do cabeçalho contém a identificação do receptor, ou seja, o campo significa “*Num_nodereceptor*”.

A lógica do algoritmo ALMTF foi implementada em linguagem *tcl*, interagindo com o agente transmissor e receptor quando necessário. As principais variáveis do algoritmo são:

- *bwshare*: banda máxima que o receptor pode utilizar inferida pelo mecanismo de janelas. O algoritmo do ALMTF é executado no receptor a cada RTT, tentando inferir a largura de banda eqüitativa disponível na rede, representada por *bwshare*. Essa variável deve refletir a banda de forma que seja eqüitativa entre todos receptores, e deve fazer isso sem necessidade de comunicação com os outros receptores. Seu uso será detalhado no item 6.1.1 (Detalhe do algoritmo e controle de congestionamento baseado em janela);
- *banda_eqn***:** banda máxima que o receptor pode utilizar inferida através da equação do TCP. Esse valor é utilizado somente como apoio à decisão, pois o *bwshare* é a variável principal na qual o ALMTF se baseia. Seu uso será detalhado no item 6.1.2 (Controle de congestionamento baseado na equação do TCP).
- *RTT (Round Trip Time)*: determina o tempo entre duas execuções consecutivas do algoritmo ALMTF, sendo utilizado também para cálculo da taxa equivalente através da equação do TCP. Em cada execução do algoritmo, a variável *bwshare* é incrementada ou decrementada, e o receptor decide fazer *join* ou *leave* baseado nesta variável.
- *bwmax*: atualizado pelo receptor quando se utiliza transmissão por pares de pacotes. O valor máximo de *bwshare* e *banda_eqn* é determinado por essa variável, que reflete a banda máxima da rede. Assim, sua utilização impede o receptor de utilizar uma banda maior do que a disponível na rede.

Para controle de fluxo, o ALMTF utiliza uma combinação de dois métodos para inferência de banda eqüitativa: por janela (J – atualizando variável *bwshare*), e por equação (E – atualizando variável *banda_eqn*), conforme mostra a figura 6.2. Na figura, a linha cheia representa a taxa descoberta pelo mecanismo de janela, e a linha pontilhada a taxa descoberta pelo mecanismo de equação. Existem duas camadas, C1 e C2, e o receptor se baseia nos dois mecanismos para tomar uma decisão. As seguintes considerações podem ser feitas:

- **Se J = E ou J > E**: utiliza a taxa descoberta pelo mecanismo de janela, que é o principal do ALMTF, pois segundo [RHE 2000], os mecanismos que utilizam

equação sofrem de problemas de inferência, e algumas vezes a taxa cai para zero. Entretanto, a fim de evitar diferenças muito grandes, limita *bwshare* em, no máximo, duas vezes *banda_eqn*, conforme mostra o bloco "a" na figura 6.2, fazendo com que a taxa de subida seja mais suave;

- **Se J < E**: Através de simulações, chegou-se à conclusão de que o mecanismo de janela diminui banda de forma muito rápida, podendo causar instabilidades. Assim, utiliza-se uma proteção que, se a variável *bwshare* descer abaixo do limite de determinada camada (C1 + C2 na figura 6.2), o sistema avalia se a banda obtida pela equação está acima ou abaixo deste limite. Se estiver acima, como no bloco "b" da figura, o algoritmo não desce de camada protegido pela equação.

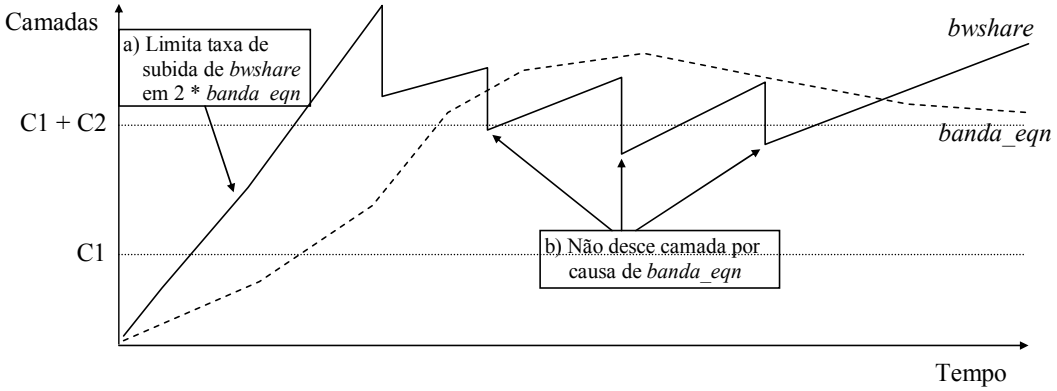


FIGURA 6.2 – Controle de fluxo do ALMTF (janela e equação)

Os próximos itens detalham os mecanismos utilizados no algoritmo do ALMTF. Os itens 6.1.1 e 6.1.2 explicam a implementação dos mecanismos de janela e taxa, respectivamente. O item 6.1.3 mostra como foi implementado o cálculo de RTT, visto que a transmissão é em multicast, não existindo *feedback* inerente ao protocolo. Para tanto, são utilizadas mensagens de retorno, cujo intervalo deve ser calculado dependendo do número de receptores, a fim de permitir escalabilidade ao sistema, conforme detalhado no item 6.1.4. Para o controle de fluxo, também é necessário o cálculo de perdas, visto no item 6.1.5, e o cálculo de *timeout*, visto no item 6.1.6. Os receptores não se inscrevem em taxas maiores do que a banda máxima da rede (variável *bwmax*), calculada através dos pares de pacotes, visto no item 6.1.7. Os itens 6.1.8 e 6.1.9 explicam como o ALMTF provê sincronização para tentativas de *join* entre receptores inscritos na mesma sessão, bem como dessincronização dos receptores inscritos em sessões diferentes.

6.1.1 Detalhe do algoritmo e controle de congestionamento baseado em janela

O mecanismo de controle de congestionamento por janela é o principal no ALMTF, portanto, neste item será feita também uma análise geral do algoritmo.

Para se criar um mecanismo de controle de congestionamento por janela semelhante ao TCP, utilizou-se uma variável denominada *cwnd*, que representa o tamanho da janela de congestionamento atual do ALMTF, em pacotes. Essa variável está relacionada diretamente à quantidade de banda permitida para o receptor se cadastrar (variável *bwshare*). A relação entre *cwnd* e *bwshare* é dada pela função

win2rate(), que efetua basicamente o seguinte cálculo:

$$bwshare = \frac{cwnd * packetsize * 8}{RTT}$$

Esse cálculo representa a banda utilizada em uma conexão TCP equivalente para um determinado tamanho de janela. Por exemplo, se o RTT é 100 ms e o tamanho de pacote é 500 bytes, tendo *cwnd* igual a 2 pacotes, a banda é igual a 80 kbit/s.

Em alguns pontos do algoritmo, é necessário efetuar o inverso, ou seja, passar de uma determinada banda para o equivalente em janela de congestionamento. Isso acontece, por exemplo, quando é necessário adaptar *bwshare* de acordo com o valor obtido via equação do TCP (variável *banda_eqn*). Para isso, altera-se *bwshare*, e a variável *cwnd* deve ficar equivalente à essa taxa. Isso é feito pela função *rate2win()*, que utiliza o seguinte cálculo:

$$cwnd = \frac{bwshare * RTT}{packetsize * 8}$$

Por exemplo, se a variável *bwshare* foi limitada em 400 kbit/s, com *packetsize* = 500 bytes e RTT = 100ms, o valor calculado equivalente para *cwnd* é 10.

A variável *banda_eqn* representa a banda calculada pela equação do TCP, que será vista no item 6.1.2. Um resumo do código é visto a seguir e explicado em seguida com o objetivo de mostrar os pontos principais da implementação. Os pontos principais que serão explicados foram numerados para facilitar a referência durante a explicação.

```
Init ()  ;# rotina de inicialização das variáveis                    (1)
  add-layer (camada 1) ;# começa na camada 1
  cngmode = start-state
  AlmIE() ;# executa agora e cada Intervalo de Execução

AlmIE() ;# código executado cada Intervalo de Execução
  # repete essa função cada RTT
  $ns_ at [T_atual + RTT] "AlmIE()"                                  (2)

  # Se em estabilização, ignora perdas e sincronismo de join
  Se (time_libera > T_atual)                                         (3)
    numloss = 0
    sincjoin = 0
    return

  # se deu timeout, reduz banda
  Se (deu_timeout ())                                                (4)
    cwnd = 1
    numloss = 0
    bwshare = win2rate()
    time_libera = T_atual + t_estab

  switch [cngmode] ;# Fases start-state e steady-state
    "start-state"                                                    (5)
      Se (numloss==0)
        cwnd = cwnd * 1,5 ;# aumenta 50%
        Se (cwnd > windowsize)
          cwnd = windowsize
        bwshare = win2rate()
        Se (bwshare > bwmax)
          bwshare = bwmax
```

```
      Senão
        cwnd = cwnd*0,7 ;# reduz 30% e passa para steady-state
        reduzbanda()
        cngmode = steady-state

    "steady-state"                                              (6)
      Se (numloss == 0)
        cwnd = cwnd + 0,1 ;# dez vezes menos agressivo que TCP
        if (cwnd > windowsize)
          cwnd = windowsize
        bwshare = win2rate() ;# converte janela para taxa
      Senão
        cwnd = cwnd * 0,95 ;# dez vezes menos agressivo que TCP.
        Reduzbanda ()

# Limita taxa da janela ao dobro da taxa da equação
  Se (bwshare > banda_eqn*2)                                    (7)
    bwshare = banda_eqn*2
    cwnd = rate2win ;# converte taxa para janela

# Trata subir camada. Só sobe com sincjoin ou se start-state (8)
  Se (bwshare > bwlevelup)
    Se (sincjoin > subscription || cngmode == 0)
      add-layer (bwshare)
  Senão
    # evita efeito colateral de join de outros receptores
    if (sincjoin > subscription + 1 && cngmode == 1)
      time_libera = T_atual + t_estab ;# espera estabilização

  set sincjoin 0 ;# faz um único join por vez

# Trata descer camada
  Se (bwshare < bwatual)                                        (9)
    # proteção. Se tiver banda pela equação, não desce camada
    if (banda_eqn >= bwatual)
      return
    # leave em tantas camadas quanto necessário
    leave-layer (bwshare)
    time_libera = T_atual + t_estab ;# espera estabilização
    # configura banda na média entre camadas adjacentes      (10)
    bwshare = (bwused + bwlevelup)/2
    cwnd = rate2win() ;# converte taxa para janela
```

Inicialmente no código, a sub-rotina *init()* é chamada (número "1" no código). Ela inicializa algumas variáveis do ALMTF, faz *join* na camada 1 e chama a sub-rotina *ALM_IE*, que é o algoritmo a ser executado a cada RTT (conforme explicado anteriormente). A sub-rotina *ALM_IE* é repetida consecutivamente de acordo com o RTT calculado pelo agente receptor do sistema (número "2" no código). Se o RTT for maior, o algoritmo é executado menos vezes, refletindo o que acontece com uma conexão TCP.

Em determinadas ocasiões o ALMTF ignora perdas, conforme mostrado no número "3" do código. Isso é feito, por exemplo, logo depois de uma condição de congestionamento, onde é necessário um tempo para que a rede consiga se estabilizar. A variável *time_libera* define o tempo que o algoritmo espera pela estabilização da rede. Nas simulações foi definido que esse tempo é de 1 segundo (definido pela variável

t_estab).

Numa condição de *timeout*, ou seja, quando o receptor fica um determinado tempo sem receber pacotes, a banda do receptor deve ser diminuída para o mínimo, que é quando *cwnd* é igual a um. Isso é ilustrado através do número "4" do código.

Da mesma forma que o algoritmo ALMP, o ALMTF foi dividido em duas fases: fase *start-state* e fase *steady-state*, conforme explicado a seguir.

Durante a fase *start-state* (número "5" no código), o algoritmo tenta alcançar rapidamente sua banda eqüitativa com os fluxos concorrentes, e faz isso incrementando rapidamente *bwshare* até a detecção de perdas (ou informação de congestionamento, se usando ECN). Quando perdas são detectadas, o algoritmo considera que sua banda passou o valor eqüitativo (por causa das perdas), assim, diminui *bwshare* por um fator (no caso, 30%) e vai para a fase *steady-state*. Segundo Widmer [WID 2001], não é aconselhável em multicast uma fase inicial que incremente de forma muito agressiva sua banda, pois as conseqüências são desastrosas em termos de perdas. No ALMTF, existe uma proteção que impede o algoritmo de subir mais de uma camada por vez, diminuindo a agressividade. Além disso, o incremento de banda é de 50% a cada RTT, em vez dos 100% utilizados pelo TCP.

Durante a fase *steady-state* (número "6" no código), o algoritmo incrementa a variável *cwnd* a cada RTT quando não houve perdas, entretanto, a fim de melhorar a estabilidade, assumiu-se que é possível manter a eqüidade com o TCP sendo dez vezes menos agressivo. Dessa forma, o ALMTF incrementa *cwnd* em "0,1" por RTT, ao invés de "1" por RTT (como utilizado no TCP). Da mesma forma, no caso de perdas, a redução de banda é dez vezes menos agressiva que o TCP. No ALMTF a redução de banda é de 5%, enquanto que no TCP a redução de banda é de 50%.

Escolheu-se uma diminuição na agressividade em dez vezes de forma empírica. Um valor muito maior do que dez tornaria o sistema muito lento, e um valor muito menor tornaria o sistema parecido com a agressividade do TCP, ou seja, muito instável.

O número "7" no código mostra uma proteção do algoritmo, que é quando a taxa obtida pelo controle por janela (*bwshare*) é maior que o dobro da taxa obtida pelo controle pela equação (*banda_eqn*). Nesse caso, o valor de *cwnd* é modificado para refletir a taxa de, no máximo, o dobro da variável *banda_eqn*.

O número "8" do código mostra o sincronismo entre receptores para subir camada. Quando *bwshare* permitir (*bwshare* > *bwlevelup*), o receptor pode tentar efetuar *join* na camada superior, entretanto, isso é feito somente em instantes específicos a fim de sincronizar todos receptores da mesma sessão. Os detalhes desse mecanismo serão vistos no item 6.1.8.

No número "9" do código, é verificado se o receptor deve efetuar *leave* em alguma camada, pois o mesmo não tem banda suficiente para sustentar a taxa atual (*bwshare* < *bwatual*). Nesse caso, o receptor somente vai efetuar *leave* se a banda obtida pela equação (*banda_eqn*) for menor que *bwatual*, caso contrário, o sistema mantém a camada atual. Isso já foi explicado anteriormente, na introdução ao capítulo.

Quando um receptor faz *leave* numa camada, ele configura a variável *bwshare* para a média aritmética entre as duas camadas adjacentes (superior e inferior – número

"10" no código). Isso evita que a variável *bwshare* fique próxima da banda necessária para subir camada novamente, que poderia permitir ao receptor fazer outra tentativa de *join* em poucos Intervalos de Execução, tornando o sistema menos estável.

6.1.2 Controle de congestionamento baseado na equação do TCP

O controle de congestionamento pela equação do TCP utilizado no ALMTF é semelhante ao utilizado no TFMCC [WID 2001b]. A cada pacote recebido, o ALMTF chama a função *estimabanda()*, que retorna o valor estimado através da equação, que é semelhante à descrita em [PAD 98] e vista no item 2.6. A fatia média de banda utilizada por um fluxo TCP é dada por:

$$B(p) \approx \frac{packetsize}{RTT\sqrt{\frac{2p}{3}} + \left(1{,}3\sqrt{\frac{3p}{8}}\right) * 4 * RTT * p\left(1 + 32p^2\right)}$$

Como pode-se verificar a partir da equação, é necessário saber o tamanho do pacote, o RTT e as perdas (variável *p*). O ALMTF utiliza um tamanho de pacote fixo, definido no arquivo de simulação. O cálculo de RTT será descrito no item 6.1.3, e o cálculo de perdas será descrito a seguir.

Para o cálculo de perdas (variável *p*), o ALMTF considera os oito últimos eventos de perdas, que são calculados conforme ilustra a figura 6.3. Quando acontece uma perda, o algoritmo verifica se já passou um RTT, se passou, inicia um novo evento de perda, caso contrário, mantém o mesmo evento. Na figura, pode-se ver que durante o "evento de perda 1" ocorreram duas perdas e chegaram 13 pacotes (visto através do número de círculos preenchidos, que representam os pacotes recebidos), durante o "evento de perda 2" ocorreram três perdas e chegaram 24 pacotes, e durante o "evento de perda 3" ocorreu apenas uma perda, e chegaram 27 pacotes. Através desse método, é gerado um vetor de oito entradas contendo o número de pacotes recebidos em cada evento de perdas, sendo que o valor inicial do vetor refere-se ao evento mais recente. No caso da figura, o vetor seria: *vet_perdas* = [27; 24; 13; 0; 0; 0; 0; 0].

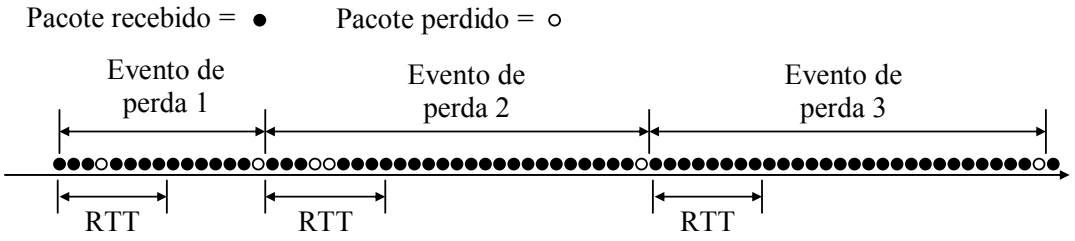


FIGURA 6.3 – Criação do vetor "evento de perda" para cálculo da variável *p*

No vetor de perdas, é dado um maior peso para os eventos mais recentes. O filtro utilizado é o mesmo definido em [RHE 2000], [WID 2001] e [FLO 99], que se baseiam nos 8 tempos anteriores, definindo um vetor de pesos da seguinte forma: [5; 5; 5; 5; 4; 3; 2; 1], ou, como é utilizado no ALMTF, *vet_pesos* = [1,0; 1,0; 1,0; 1,0; 0,8; 0,6; 0,4; 0,2] .

O valor de *p* será o inverso da média do valor dos eventos de perdas, conforme

mostra o código a seguir, supondo que "*soma*" é a soma do vetor de pesos, ou 6,0.

```
media = 0
for (i = 0, i < 8, i++)
    media = media + vet_perdas(i)*vet_pesos(i)/soma
p = 1/media
```

Após a obtenção da variável *p*, basta aplicar a fórmula da equação, como mostra o código a seguir.

```
tmp1 = RTT * sqrt(2*p/3)
tmp2 = 1.3*sqrt(3*p/8)
Se (tmp2 > 1.0)
  tmp2 = 1.0
tmp3 = 4*RTT*p*(1+32*p*p)
banda_eqn = (packetsize*8) / ($tmp1 + $tmp2*$tmp3) ;# em bits/s
```

A banda máxima que o receptor pode utilizar baseado na equação é dada pela variável *banda_eqn*, em bit/s. A utilização dessa variável já foi definida nos itens anteriores.

6.1.3 Cálculo do RTT

O RTT é um dos elementos que gera o maior impacto no protocolo TCP, pois é o RTT que determina o aumento na janela do TCP (variável *cwnd*), que está diretamente ligada ao aumento de taxa, conforme detalhado no item 2.6. Isso faz com que um algoritmo que queira gerar tráfego eqüitativo ao TCP deva se basear nessa variável.

O ALMTF é baseado em *multicast*, e tem o objetivo de atingir um grande número de receptores, não admitindo, portanto, comunicações freqüentes entre transmissor e receptores. O método utilizado pelo ALMTF para cálculo de RTT tem semelhanças com o utilizado pelo TFMCC (item 3.9) e MLDA (item 3.10).

Para estimativa de RTT, os relógios do transmissor e receptor não precisam estar sincronizados. Os campos do cabeçalho destinados ao cálculo do RTT são: *num_nodereceptor, timestamp*, *timestamp_offset* e *timestamp_echo*, conforme visto anteriormente no cabeçalho do pacote (figura 6.1).

No instante inicial e a cada intervalo entre *feedbacks* (conforme será detalhado no item 6.1.4), o receptor envia ao transmissor um pacote de "Solicitação de Eco", necessário para cálculo do RTT. Nesse pacote os seguintes campos são utilizados:

- *Timestamp*: instante que o receptor enviou o pacote (*tstamp_receptor*);
- *Num_nodereceptor*: identificação do receptor.

O transmissor responde a esse pacote preenchendo, no cabeçalho do próximo pacote de dados relativo à camada um, os seguintes campos:

- *Timestamp*: instante que o transmissor enviou o pacote (*tstamp_transmissor*);
- *Timestamp_offset*: diferença de tempo entre a chegada do pacote de "Solicitação de Eco" e a transmissão da resposta;
- *Timestamp_echo*: cópia de *tstamp_receptor*, obtido no pacote de "Solicitação de Eco";

- *Num_nodereceptor*: cópia de *num_nodereceptor* obtido no pacote de "Solicitação de Eco".

É necessário que seja na camada um pois todos a recebem, fazendo com que a informação atinja todos os receptores. Com essa informação, o receptor está apto a calcular o RTT de forma precisa no momento que chega o pacote, conforme mostra a figura 6.4. Na figura, o transmissor envia três pacotes de dados (Dados 1, Dados 2 e Dados 3). No cabeçalho de todos os pacotes o transmissor envia seu *timestamp*, representado por TT1, TT2 e TT3. Na chegada do primeiro pacote, o receptor envia um pacote de "Solicitação de Eco", contendo no cabeçalho seu *timestamp* (representado na figura por TR1) e sua identificação, representado por *numrec*. Quando o transmissor recebe a solicitação de eco, ele armazena seu tempo de chegada, e calcula a diferença de tempo entre a chegada da solicitação e a transmissão do novo pacote de dados (representado na figura por ΔT). No próximo pacote de dados, o transmissor envia, além do seu *timestamp*, o tempo do receptor (TR1), a identificação do receptor e ΔT, conforme visto no pacote "Dados 2" da figura.

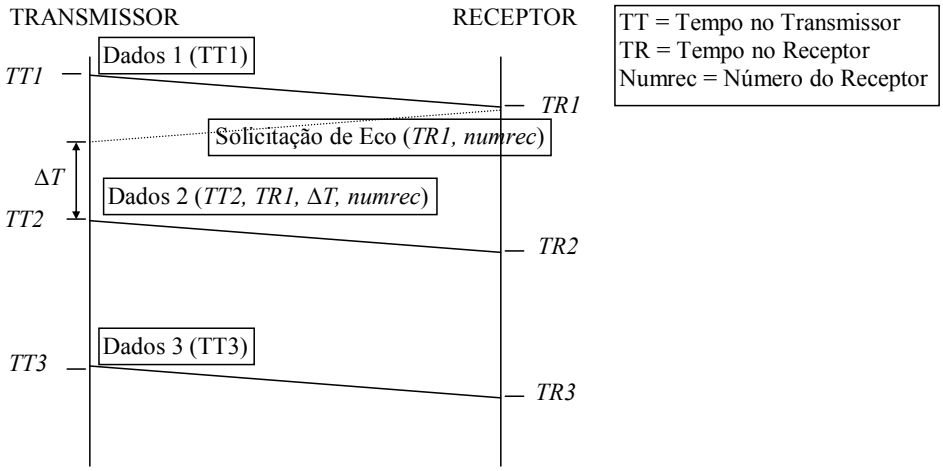


FIGURA 6.4 – Cálculo do RTT do ALMTF no receptor

O cálculo do RTT é composto de duas fases: a) quando o receptor recebe uma resposta à sua solicitação de eco (como no instante TR2 da figura 6.4), e calcula o RTT com precisão, bem como a diferença entre os relógios do transmissor e do receptor; b) quando o receptor recebe um pacote de dados somente com *timestamp* do transmissor (como nos instantes TR1 e TR3 da figura 6.4), e calcula a variação do RTT.

Na primeira fase, o RTT é calculado conforme a fórmula F1 a seguir, onde os tempos TR1 e ΔT são obtidos a partir do cabeçalho do pacote. Essa medida é bastante precisa, e é utilizada como base para os outros cálculos.

$$RTTcalc = TR2 - TR1 - \Delta T \qquad (F1)$$

Nesta primeira fase ainda é possível calcular a assimetria da rede, que pode ser ocasionada por diferença de tempo entre o relógio do transmissor e do receptor ou por diferença na topologia de rede da transmissão e recepção (como satélite e fibra ótica, por exemplo). Supõe-se inicialmente que o tempo de propagação do pacote (do receptor para o transmissor), seja metade do RTT, conforme fórmula F2 a seguir.

$$T_{ida} = RTT_{calc} / 2 \qquad (F2)$$

A assimetria (T_{assim}) é calculada através da diferença no tempo de propagação do pacote a partir do transmissor para retornar ao receptor em relação a uma rede simétrica, sendo calculada conforme fórmula F3, a seguir. Esse valor é ajustado a cada pacote recebido, pois todos os pacotes incluem o *timestamp* do transmissor.

$$T_{assim} = TR2 - TT2 - T_{ida} \qquad (F3)$$

Na segunda fase, ou seja, quando o receptor obtém somente um pacote de dados com o *timestamp* do transmissor (instantes TR1 e TR3 da figura 6.4), o receptor pode calcular a variação no atraso da rede, ajustando o RTT. Para o pacote *Dados 3*, o ajuste no cálculo do RTT seria dado pela fórmula F4, a seguir.

$$RTT_{calc} = T_{ida} + (TR3 - TT3 - T_{assim}) \qquad (F4)$$

Por exemplo, supondo uma topologia de rede simétrica e relógios entre transmissor e receptor sincronizados, como mostra a figura 6.5, onde o tempo de propagação no meio físico é de 200ms. Após os cálculos da primeira fase (fórmulas F1, F2 e F3), obtém-se RTT_{calc} = 400ms (1,7 – 1,2 – 0,1), T_{ida} = 200ms e T_{assim} = 0 (1,7 – 1,5 – 0,2). No terceiro pacote, por congestionamento na rede ou qualquer outro fator, o tempo de propagação aumentou para 220ms. O cálculo da segunda fase (fórmula F4) obtém RTT_{calc} = 420ms (0,2 + (2,22 – 2 – 0)), ou seja, refletindo imediatamente o aumento no tempo de propagação da rede.

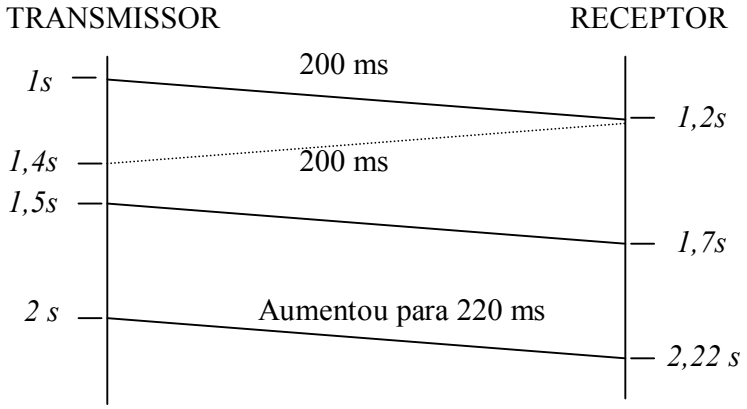


FIGURA 6.5 – Cálculo do RTT em redes simétricas com relógio sincronizado

O sistema também funciona se os relógios do transmissor e receptor estiverem dessincronizados, como mostra a figura 6.6, onde o relógio do transmissor está 1s adiantado em relação ao receptor. Após os cálculos da primeira fase (fórmulas F1, F2 e F3), obtém-se RTT_{calc} = 400ms (1,7 – 1,2 – 0,1), T_{ida} = 200ms e T_{assim} = –1s (1,7 – 2,5 – 0,2). No terceiro pacote, quando o tempo de propagação aumentou para 220ms, efetua-se o cálculo da segunda fase (fórmula F4), obtendo-se RTT_{calc} = 420ms (0,2 + (2,22 – 3 – (–1))), ou seja, refletindo imediatamente o aumento no tempo de propagação da rede e compensando a diferença de relógio.

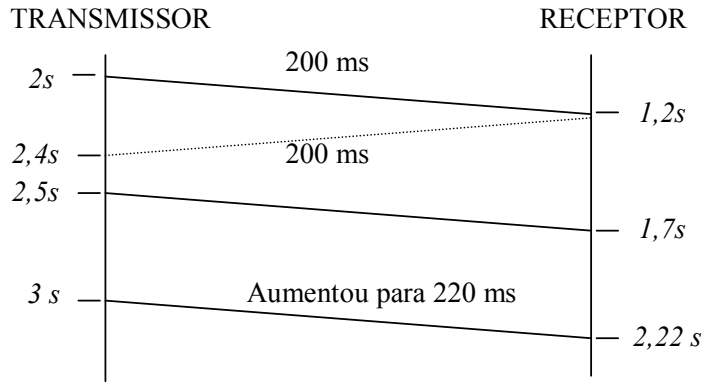


FIGURA 6.6 – Cálculo do RTT em redes simétricas com relógio dessincronizado

Um último caso é mostrado na figura 6.7, onde a rede é assimétrica (100ms para ir e 300ms para voltar) e os relógios estão dessincronizados (relógio do transmissor 1s adiantado). Após os cálculos da primeira fase (fórmulas F1, F2 e F3), obtém-se RTT_{calc} = 400ms (1,6 – 1,1 – 0,1), T_{ida} = 200ms e T_{assim} = –1,1s (1,6 – 2,5 – 0,2). No terceiro pacote, quando o tempo de propagação aumentou para 120ms, efetua-se o cálculo da segunda fase (fórmula F4), obtendo-se RTT_{calc} = 420ms (0,2 + (2,12 – 3 – (–1,1))), ou seja, refletindo imediatamente o aumento no tempo de propagação da rede e compensando a diferença de relógio e a diferença na assimetria da rede.

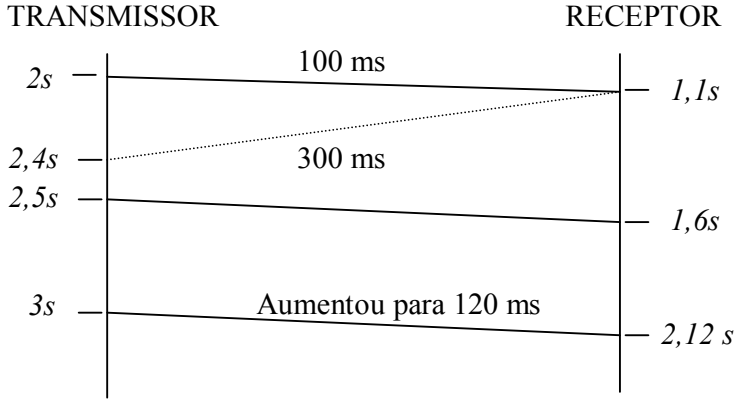


FIGURA 6.7 – Cálculo do RTT em redes assimétricas com relógio dessincronizado

O único caso que o método não detecta imediatamente é o aumento no tempo de propagação entre receptor e transmissor (T_{ida}). Caso aconteça uma variação nesse tempo, o sistema só irá calcular o valor correto após um pacote de "Solicitação de Eco".

No início dos tempos, o RTT é fixo em 0,5s. Na primeira medida do receptor, o RTT calculado é atribuído ao RTT, fazendo-se **RTT = RTT_{calc}**. Em todas as outras medidas, utilizou-se um filtro a fim de evitar variações bruscas e manter a estabilidade. Este filtro está descrito no código a seguir, e tem as seguintes características:

- Se RTT_{calc} menor que RTT (primeira e segunda linha no código): nesse caso, a banda disponível ao receptor (*bwshare*) tende a aumentar, conforme visto no item 6.1.1. Para evitar um aumento brusco, utiliza-se 90% do RTT anterior e 10% do RTT calculado, resultando: **RTT = 0,9 * RTT + 0,1 * RTT_{calc}**;
- Se RTT_{calc} maior que RTT (a partir da terceira linha no código): nesse caso, a banda disponível ao receptor (*bwshare*) tende a diminuir, conforme visto no item 6.1.1.

Para evitar que o receptor efetue *leave* desnecessariamente devido a pequenas variações de RTT, definiu-se que o RTT só vai mudar se for maior que 25% do RTT atual.

```
Se (RTT_calc < RTT)
  RTT = 0,9 * RTT + 0,1 * RTT_calc
Senão
  Se (RTT_calc > 1,25 * RTT)
    RTT = RTT_calc
```

6.1.4 Intervalo entre *feedbacks*

O receptor deve enviar *feedbacks* ao transmissor a intervalos regulares, a fim de calcular com precisão o tempo de ida (T_{ida}) do pacote, conforme visto no item 6.1.3, entretanto, o número de pacotes não deve ultrapassar um determinado limite, pois caso contrário provocaria uma implosão de *feedbacks* quando houvesse um grande número de receptores cadastrados na sessão. No ALMTF, definiu-se o seguinte:

Até 60 receptores o total de mensagens deve estar limitada a uma por segundo, assim, se existir um receptor, o mesmo vai enviar uma mensagem por segundo. Se existirem 60 receptores, cada um deles pode enviar uma mensagem a cada 60 segundos, obtendo uma atualização por minuto.

Acima de 60 receptores, mantém-se uma atualização por minuto, a fim de evitar uma espera muito grande para cálculo do RTT com precisão. Isso faz com que o sistema gere um tráfego adicional caso tenha muitos receptores. Com 6.000 receptores, tem-se uma mensagem a cada 10ms, ou 100 pacotes por segundo. Se cada pacote de *feedback* possuir 64 bytes (512 bits), tem-se um tráfego adicional de 51,2 kbit/s para 6.000 receptores.

O intervalo ótimo entre cada atualização (definida aqui em 60s) depende da característica do canal de comunicação entre os receptores e o transmissor. Se esse canal variar de forma significativa, o RTT vai ficar desatualizado em relação ao tempo de propagação receptor / transmissor por aproximadamente um minuto, gerando problemas de divisão eqüitativa de banda com tráfego TCP concorrente (que é baseado fortemente no RTT). Nesta Tese não se aprofundou esse tópico, e se está supondo que uma atualização precisa por minuto é suficiente para manter o tempo "T_{ida}", utilizado no cálculo do RTT, suficientemente próximo ao de uma sessão TCP concorrente.

Para diminuir a probabilidade de rajadas de *feedbacks* entre receptores, o cálculo do intervalo é feito utilizando um multiplicador aleatório, conforme visto no código a seguir. A variável *num_receptores* (primeira linha no código) representa o número de receptores, e é enviada pelo transmissor no cabeçalho dos pacotes da camada um (conforme visto na figura 6.1). T_{envio} (terceira linha no código) representa o instante de transmissão da mensagem de "Solicitação de Eco". T_{atual} representa o instante atual no receptor, e *random (0, 1)* é uma função que retorna um número aleatório entre zero e um.

```
Se (num_receptores > 60)
  num_receptores = 60;
T_envio = T_atual + random (0, 1) * num_receptores + 1
```

6.1.5 Cálculo de perdas

No ALMTF, a detecção das perdas é feita com base no número de seqüência existente em cada pacote transmitido (semelhante ao protocolo RTP). Caso o receptor detecte em qualquer camada uma falha na seqüência recebida, ele incrementa a variável *numloss*, que reflete a quantidade de perdas.

Em [RHE 2000], [WID 2001] e [FLO 99] foi assumido que várias perdas durante o mesmo RTT eram consideradas como uma única ocorrência de perdas. O ALMTF utiliza o mesmo conceito, ou seja, o tratamento do algoritmo é o mesmo independente do número de pacotes perdidos. No caso, o ALMTF diminui *bwshare* em 5%, conforme detalhado no item 6.1.1.

6.1.6 Identificação de *timeout*

Em [FLO 99] é descrito o cálculo de *timeout* para colocar na equação do TCP. Normalmente, o algoritmo do TCP utiliza *Ttout = SRTT + 4*RTT*, onde SRTT é a estimativa atual do RTT, entretanto, esse valor varia muito entre implementações do TCP. O que Floyd utilizou no TFRC foi: *Ttout = 4*RTT*, dizendo que esse valor empírico funciona muito bem na prática.

No caso do ALMTF, não é possível basear-se no RTT para cálculo de *timeout*, pois o algoritmo utiliza transmissão multicast em camadas, e pode acontecer do RTT ser muito maior do que a taxa de transmissão dos pacotes da camada zero. Por exemplo, supondo que a camada zero seja de 30 kbit/s e os pacotes sejam de 500 bytes, ou 4.000 bits, resulta em 7,5 pacotes por segundo, ou um pacote cada 133 ms, conforme mostra a figura 6.8.

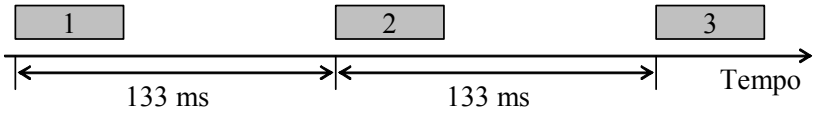


FIGURA 6.8 – Cálculo de *timeout* no ALMTF

Se um receptor estivesse localizado numa rede rápida, com um RTT de 30ms, um *timeout* de 4*RTT resultaria em 120ms. Nesse caso, haveria *timeout* em todos os pacotes, pois como o intervalo entre dois pacotes seria de 133ms, sempre resultaria num valor maior do que *4*RTT*.

Para contornar essa característica da transmissão multicast, o mecanismo utilizado no ALMTF calcula o *timeout* com base no tempo esperado para chegada de um determinado número de pacotes. Na implementação do algoritmo, utilizou-se o tempo equivalente a 10 pacotes. Assim, caso o receptor fique um tempo equivalente a 10 pacotes sem receber dados (equivalente a 1,33 segundos na figura 6.8), ele considera que houve *timeout*, reduzindo a variável *cwnd* para 1, conforme visto no item 6.1.1. O código para efetuar o cálculo é:

```
Se (T_ultimo_pacote + (10 * 8 * tampacote / banda) < T_atual)
  Timeout = 1
```

A variável T_{ultimo_pacote} representa o momento de chegada do último pacote da camada zero no receptor. *Tampacote* representa o tamanho médio dos pacotes em bytes,

e a variável *banda* é referente à camada zero, em bit/s.

6.1.7 Uso de pares de pacotes

Da mesma forma que o algoritmo ALMP, o ALMTF utiliza pares de pacotes para determinar a banda máxima na qual o receptor pode se inscrever. A descrição do método de pares de pacotes encontra-se no Anexo A (Análise do mecanismo de pares de pacotes). As simulações foram efetuadas com a implementação do mecanismo de pares de pacotes, entretanto, em algumas se retirou o mecanismo para demonstrar algum detalhe do funcionamento do algoritmo.

Como pode ser visto no Anexo A, o mecanismo de pares de pacotes obtém medições bastante variadas ao redor da banda máxima. A fim de evitar variações bruscas e errôneas na banda máxima da rede, foi utilizado um filtro simples para obter o valor com maior probabilidade de estar correto. A banda máxima calculada pelo algoritmo é atualizada na variável *bwmax*, já mencionada anteriormente. O código do filtro está descrito a seguir, na função *calcbwmax*, que recebe como parâmetro de entrada o valor calculado na medição atual dos pares de pacotes (*PP_value*). Os pontos principais que serão explicados foram numerados para facilitar a referência durante a explicação que será feita após o código.

```
Calcbwmax (PP_value)
  # cria buffer de 10 medições antes de atribuir bwmax
  Se (PP_namostras < 10)                                          (1)
    PP_buf(PP_namostras) = PP_value
    PP_somatotal = PP_somatotal + PP_value
    PP_namostras = PP_namostras + 1
    # Quando PP_namostras = 10, faz a média
    Se (PP_namostras==10)
      bwmax = PP_somatotal/10                                     (2)
      PP_indice = 0
    return

  Se (PP_value > bwmax*0.7 && PP_value < bwmax*1.3)               (3)
    PP_descartou = 0
    PP_somatotal = PP_somatotal - PP_buf(PP_indice)
    PP_somatotal = PP_somatotal + PP_value
    PP_buf(PP_indice) = PP_value ;# substitui valor pelo novo
    bwmax = PP_somatotal/10 ;# altera banda máxima
    PP_indice = PP_indice + 1
    if (PP_indice == 10)
      PP_indice = 0
  Senão                                                           (4)
    # recomeça se dez descartes seguidos
    PP_descartou = PP_descartou + 1
    Se (PP_descartou > 10)                                        (5)
      PP_namostras = 0
      PP_descartou = 0
```

O filtro é bastante simples, e está representado na figura 6.9. As primeiras dez medidas são simplesmente inseridas numa fila, conforme pode ser visto através do número “1” no código. O primeiro valor de banda máxima só é obtido após as dez primeiras medições, sendo a média entre os dez primeiros valores (item “a” na figura 6.9 e número “2” no código). Após esse instante, cada medida nova que chega deve

estar entre ±30% da média atual, caso contrário é descartada (número “4” no código). Estando dentro desta faixa, a nova medida substitui a mais antiga da fila (medida 11 substitui medida um, medida 12 substitui medida dois, e assim por diante). Com o novo valor, a média é recalculada, obtendo-se o novo valor de *bwmax* (item “b” na figura e número “3” no código).

Caso aconteçam dez descartes em seqüência (durante dez medições o valor obtido não estava entre ±30% da média atual), considera-se que a topologia da rede mudou por algum motivo e o processo é reiniciado (número “5” no código).

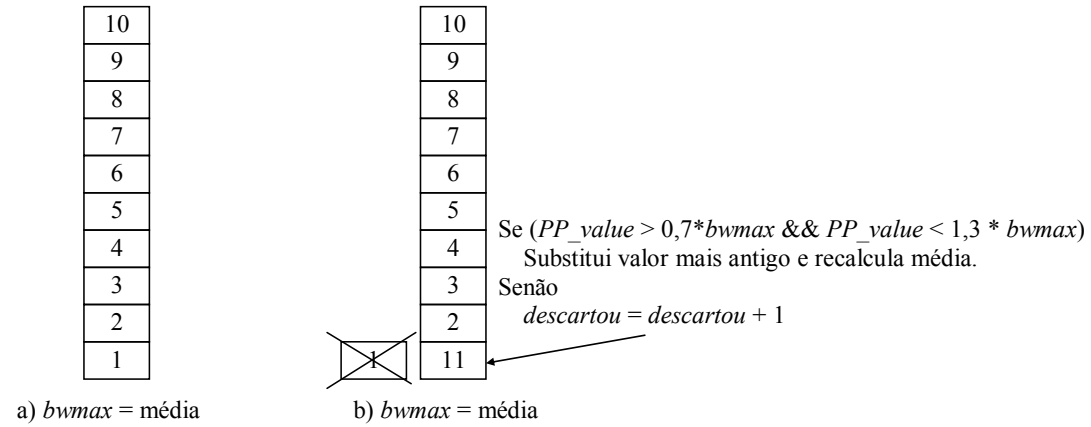


FIGURA 6.9 – Filtro de pares de pacotes para cálculo de banda máxima no ALMTF

6.1.8 Sincronização entre receptores da mesma sessão

No item 2.8.2 foi explicado o motivo da necessidade de sincronização entre receptores da mesma sessão. Caso não existisse essa sincronização, haveria uma diminuição na estabilidade do sistema.

O ALMTF utiliza pontos de sincronização contendo o limite até o qual algum receptor pode efetuar *join*, como mostra a figura 6.10, que representa o valor de *sincjoin* numa transmissão com 5 camadas (C1 a C5). Os círculos na figura representam o limite de camada no qual o receptor pode se inscrever, que também é representado pela variável *sincjoin* (transmitida pelo transmissor no cabeçalho do pacote, como pode ser visto na figura 6.1). No primeiro intervalo de tempo (*sincjoin = 2*) todos os receptores cadastrados na camada C1 podem efetuar tentativas de *join* para a camada C2, entretanto, um receptor cadastrado na camada C2 não poderá se inscrever na camada C3. No próximo intervalo de tempo (*sincjoin = 3*), os receptores podem se cadastrar até a camada C3, e assim por diante. Se existissem mais camadas, o número seria dividido por quatro, a fim de seguir a distribuição mostrada na figura. Por uma característica do algoritmo, todos receptores estão sempre cadastrados na camada C1, portanto, não faz sentido enviar *sincjoin = 1*.

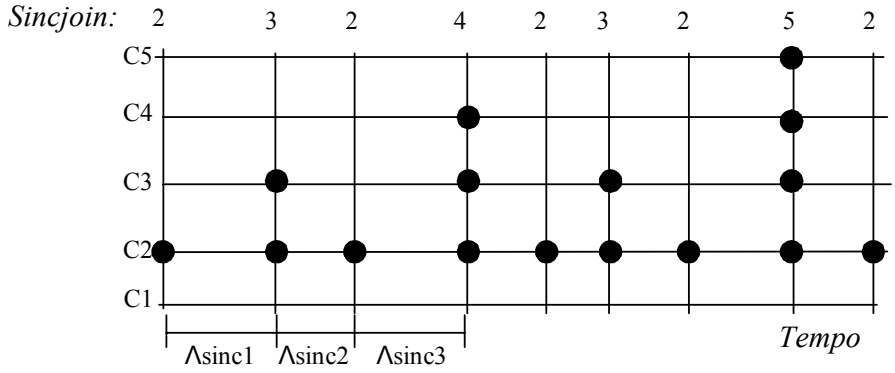


FIGURA 6.10 – Pontos de sincronização no ALMTF

O código que implementa o valor de *sincjoin* é mostrado a seguir e visualizado através da figura 6.10, onde a variável *vet_sinc* (primeira linha do código) é o vetor da distribuição, que foi escolhido como [2; 3; 2; 4; 2; 3; 2; 5]. A variável *div_sinc* (primeira linha do código) representa o número de camadas dividido por quatro, e *ind_sinc* (primeira linha do código) indica o índice para o valor do vetor que deve ser multiplicado para determinar *sincjoin*. Na segunda linha do código abaixo, o campo *sincjoin* do cabeçalho do pacote é preenchido.

```
sincjoin_ = (int)(vet_sinc[ind_sinc]*div_sinc);
almtf_header->sincjoin = sincjoin_;
   Se (++ind_sinc > 7)
     ind_sinc=0;
```

Como pode ser visto através da figura 6.10, os pontos de sincronização acontecem mais freqüentemente nas camadas inferiores. Dessa forma, em mais intervalos de sincronização os receptores podem tentar fazer *join* nas camadas inferiores, dando maior chance aos receptores com menos banda subirem de camada e aumentando a eqüidade de tráfego.

Quando o ponto de sincronização acontece em uma camada, ele acontece em todas as camadas abaixo dela simultaneamente. O objetivo disso é evitar que a rajada de tráfego de uma camada (que pode gerar perda de pacotes em todas as camadas) resulte em interpretação de congestionamento por outro receptor, fazendo com que ele faça *leave* em uma camada erroneamente. Assim, quando um receptor está num ponto de sincronização fazendo tentativa de *join*, todos os receptores inscritos numa camada igual ou menor que ele também podem estar fazendo tentativas de *join*. Caso um receptor não faça tentativa de *join*, pois sua variável *bwshare* não permite, ele ignora as perdas por um determinado tempo de estabilização, definido no algoritmo ALMTF como um segundo.

O intervalo entre os pontos de sincronização não é fixo, variando de forma aleatória a fim de sincronizar receptores na mesma sessão e dessincronizar receptores localizados em sessões diferentes (conforme descrição no item 6.1.9 e representação na figura 6.10).

O ALMTF implementa essa sincronização através da transmissão de sinais de sincronismo de *join* (*sincjoin*) com base em um número randômico cuja semente é diferente para cada transmissor. Inicialmente, quando o transmissor é criado, é gerada

uma semente aleatória para criação de número randômico, que é dependente da hora do sistema, diferente para cada transmissor.

Durante a transmissão dos pacotes, o transmissor se baseia na semente criada para gerar o novo ponto de sincronismo, conforme mostra o código a seguir. No código, a variável $T_{ultsinc}$ (primeira e segunda linha do código) representa o momento que foi enviado o último *sincjoin*. Δ*sinc* é a diferença de tempo para o envio do próximo *sincjoin*, e seu cálculo pode ser visto na terceira linha do código. O valor de Δ*sinc* é randômico, fazendo com que aumente a probabilidade do sincronismo para receptores na mesma sessão e o dessincronismo para receptores em sessões diferentes. Uma representação de Δ*sinc* pode ser vista na figura 6.10. A variável *media_sinc*, vista na terceira linha do código, refere-se ao tempo médio entre pontos de sincronismo. No ALMTF, o intervalo médio foi escolhido como um segundo, portanto, a variável Δ*sinc* assume valores de 0,5s a 1,5s.

Na quarta linha do código, a variável *sincjoin* é inserida no cabeçalho do pacote que está para ser transmitido, e é feito o controle para que essa variável não exceda o número máximo de camadas (variável *total_camadas*), conforme representado na figura 6.10 e visto nas linhas 5 a 7 do código.

```
Se (T_atual > (T_ultsinc + Δsinc))
  T_ultsinc = T_atual
  Δsinc = rand()*media_sinc + media_sinc/2
  almtf_header.sincjoin = sincjoin // inclui no cabeçalho ALMTF
  sincjoin = sincjoin + 1
  Se (sincjoin > total_camadas)
    sincjoin = 2
```

Foram identificados alguns mecanismos adicionais a serem utilizados entre receptores na mesma sub-rede para melhoria na estabilidade do algoritmo ALMTF, entretanto, os mesmos não foram implementados, pois o código do simulador NS2 não permite uma implementação simples dos mecanismos descritos. Os mecanismos são os seguintes:

- **Aprendizado de *join***: dentro da sua sub-rede (TTL = 1), o receptor monitora as mensagens IGMP de *join* dos outros receptores. Caso algum receptor faça *join* em um grupo multicast correspondente a duas ou mais camadas acima da que ele está cadastrado, o mesmo se cadastra em todas camadas até a anterior à qual foi feito *join*. Por exemplo, o receptor está na camada C2 e outro receptor faz *join* na camada C4. Imediatamente o receptor vai para camada C3;
- **Aprendizado de *leave***: se a rede está congestionada e tem dois receptores, um na camada C3 e outro na camada C2, o receptor da camada C2 não deve descer nível para C1 até o receptor da camada C3 descer para a camada C2. Assim, o máximo de diferença de camadas entre receptores é um.

6.1.9 Dessincronização de receptores de sessões diferentes

No item 2.8.2 foi explicado que receptores localizados em sessões diferentes deveriam efetuar tentativas de *join* em momentos diferentes, a fim de evitar excesso de tráfego num determinado canal, que poderia fazer com que, mesmo havendo banda

suficiente para um dos receptores subir de camada, todos falhassem na tentativa de *join*.

A implementação desse mecanismo no ALMTF foi detalhada no item 6.1.8, que serve tanto para sincronizar receptores de mesma sessão como dessincronizar receptores de sessões diferentes.

6.2 Simulações do ALMTF

A metodologia das simulações foi definida no item 4.7, e as simulações a seguir visam analisar o algoritmo nos experimentos propostos no item 4.8. Nem todas as simulações serão mostradas neste capítulo por questões de espaço, porém, todas as simulações podem ser encontradas em http://www.inf.unisinos.br/~roesler/tese e no CD anexo a esta Tese.

As simulações foram divididas conforme as métricas estabelecidas no item 4.7, que são detalhadas em cada um dos itens a seguir, buscando analisar o algoritmo em relação a: adaptabilidade, escalabilidade, tempo de adaptação, estabilidade, eqüidade de tráfego e taxa de perdas.

6.2.1 Análise da adaptabilidade do ALMTF em ambientes heterogêneos

Foram efetuadas todas as simulações de adaptabilidade definidas no item 4.8, e as mais significativas estão descritas a seguir. Para as simulações, utilizou-se a topologia 1 da figura 4.3, que é repetida na figura 6.11 já com os parâmetros específicos utilizados para testar a adaptabilidade do algoritmo em ambientes heterogêneos. Os parâmetros configurados para a primeira simulação estão descritos a seguir, e serão utilizados como base para as outras simulações, que vão modificar alguns desses parâmetros:

- Atraso A1 a A4 = 10ms;
- Camadas exponenciais padrão (30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s e 960 kbit/s);
- Tamanho do pacote = 500 bytes;
- Tipo de fila = *droptail* com 20 pacotes;
- Randomização alta (±0,5s);
- Sem uso de pares de pacotes.

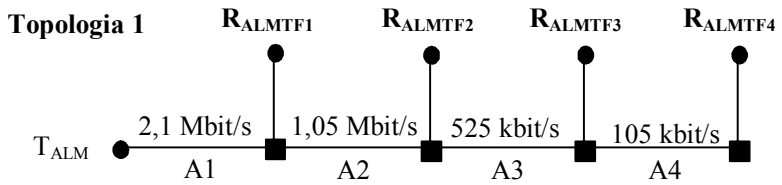


FIGURA 6.11 – Topologia para simulações de adaptabilidade

A figura 6.12 mostra o resultado da simulação. Pode-se constatar que todos os receptores adaptaram-se de acordo com a largura de banda disponível. O receptor ALMTF1 se inscreveu em 6 camadas (total de 1890 kbit/s), o que é consistente com o enlace de 2,1 Mbit/s. O receptor ALMTF2 se inscreveu em 5 camadas (930 kbit/s),

coerente com o enlace de 1,05 Mbit/s. O receptor ALMTF3 se inscreveu em 4 camadas (450 kbit/s), o que é adequado para o enlace de 525 kbit/s. Já o receptor ALMTF4 se inscreveu em 2 camadas (90 kbit/s), pois a banda do enlace é somente 105 kbit/s.

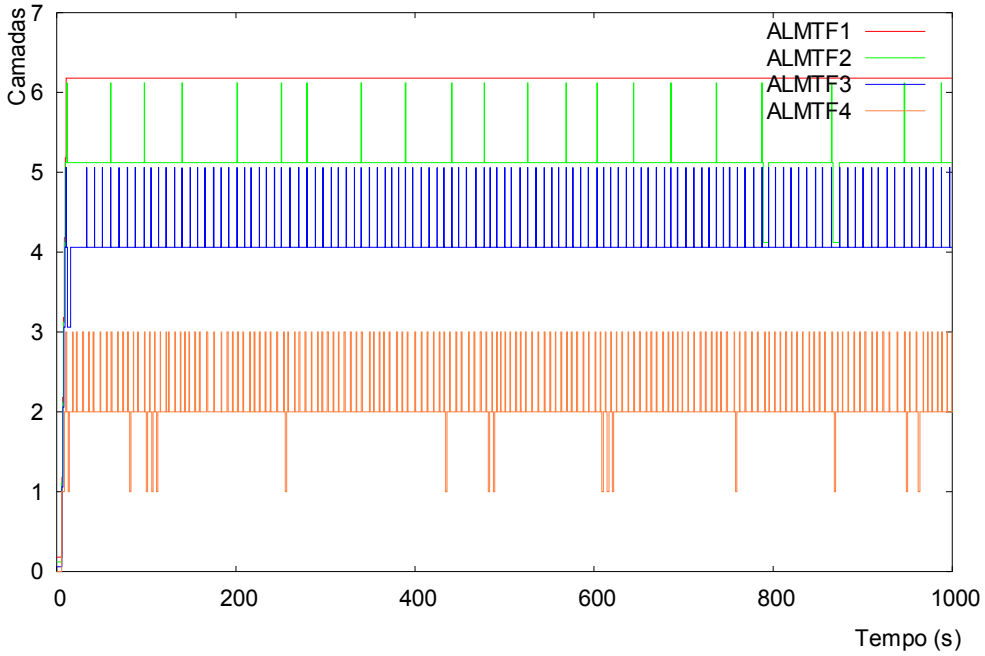


FIGURA 6.12 – Camadas exponenciais padrão sem utilizar pares de pacotes

O número maior de tentativas de subir camada do receptor ALMTF4 é explicado pelo fato de que, quanto menor a camada em que o receptor está inscrito, maior o número de vezes que o mesmo pode efetuar *join*, devido à variável *sincjoin*, conforme explicado no detalhamento. Além disso, como as camadas são exponenciais, o intervalo de banda até a próxima camada é menor do que nos receptores inscritos em mais camadas.

A vazão para cada receptor está descrita a seguir e ilustrada graficamente na figura 6.13, e foi obtida através da saída da simulação, conforme descrição no item 4.7.5 (Metodologia para análise de resultados no simulador). Como pode ser visto pelos resultados, o valor obtido é bastante próximo do ideal. O resultado é considerado ideal se a vazão é igual à soma das camadas possíveis do receptor se inscrever. Devido à granularidade das camadas, não é possível obter um resultado igual à largura de banda do enlace, conforme explicado no item 2.5 (Eqüidade de tráfego). O resultado mostra uma vazão próxima a 100% da vazão ideal que poderia ser conseguida para um algoritmo baseado em camadas.

- Receptor ALMTF1: vazão:1885 kbit/s. Vazão ideal: 1890 kbit/s (camadas 1 a 6);
- Receptor ALMTF2: vazão: 927 kbit/s. Vazão ideal: 930 kbit/s (camadas 1 a 5);
- Receptor ALMTF3: vazão: 451 kbit/s. Vazão ideal: 450 kbit/s (camadas 1 a 4);
- Receptor ALMTF4: vazão: 92 kbit/s. Vazão ideal: 90 kbit/s (camadas 1 a 2).

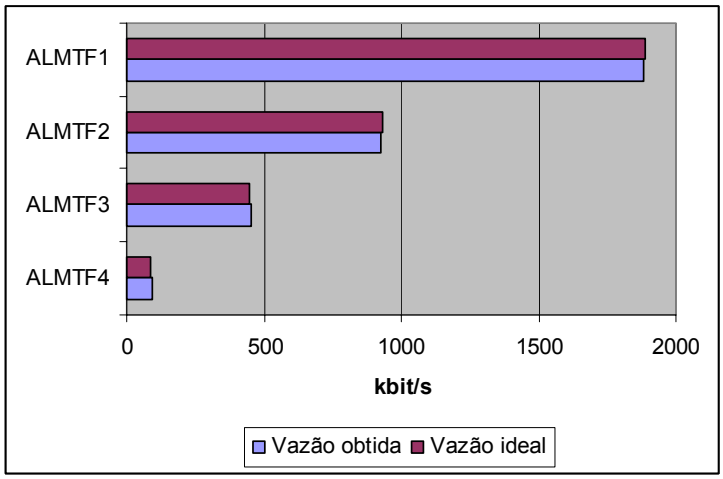


FIGURA 6.13 – Comparação entre vazão obtida e ideal para 4 receptores

A figura 6.14 mostra a mesma simulação, porém com a utilização de pares de pacotes. Nesse caso, como os pares de pacotes permitem a cada receptor o cálculo da banda máxima disponível na rede, e os receptores são os únicos usuários do enlace, praticamente não existem tentativas de subir camada, o número de perdas é zero e a adaptabilidade é consistente com a banda disponível.

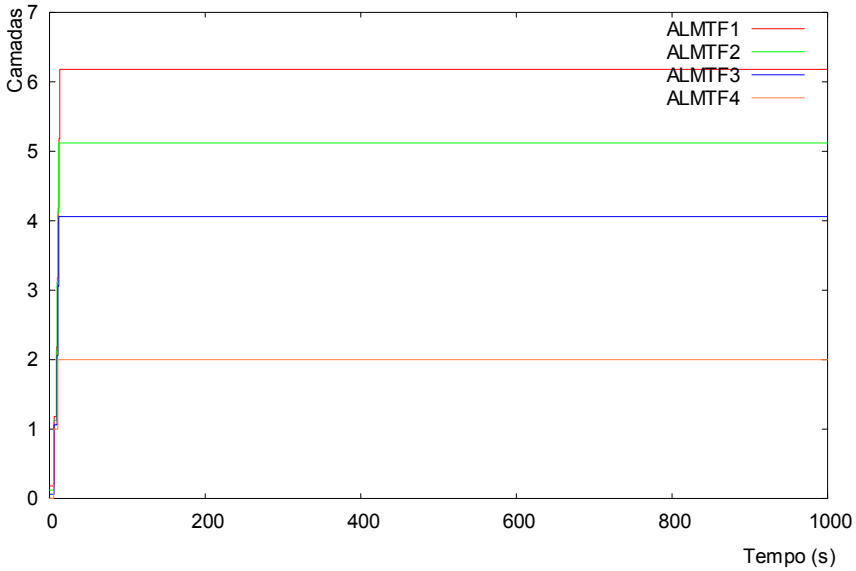


FIGURA 6.14 – Camadas exponenciais padrão utilizando pares de pacotes

A vazão para cada receptor está descrita a seguir e ilustrada graficamente através da figura 6.15. A pequena diferença de vazão em relação à simulação anterior é devido a dois fatores: a) o arredondamento do tempo de início dos receptores (que começam de forma aleatória a partir de um instante pré-definido, podendo gerar uma diferença de até um segundo); b) cada tentativa de subir camada faz com que o receptor receba um excedente de tráfego enquanto a fila não esgotar, pois o valor nominal do enlace é maior do que o tráfego exigido pelo receptor. Por exemplo, o enlace do receptor ALMTF4 é de 100 kbit/s, e se cadastrando em duas camadas o receptor consome 90 kbit/s, ou seja, menos que o valor nominal do enlace.

- Receptor ALMTF1: vazão: 1836 kbit/s. Vazão ideal: 1890 kbit/s (camadas 1 a 6);
- Receptor ALMTF2: vazão: 907 kbit/s. Vazão ideal: 930 kbit/s (camadas 1 a 5);
- Receptor ALMTF3: vazão: 438 kbit/s. Vazão ideal: 450 kbit/s (camadas 1 a 4);
- Receptor ALMTF4: vazão: 88 kbit/s. Vazão ideal: 90 kbit/s (camadas 1 a 2).

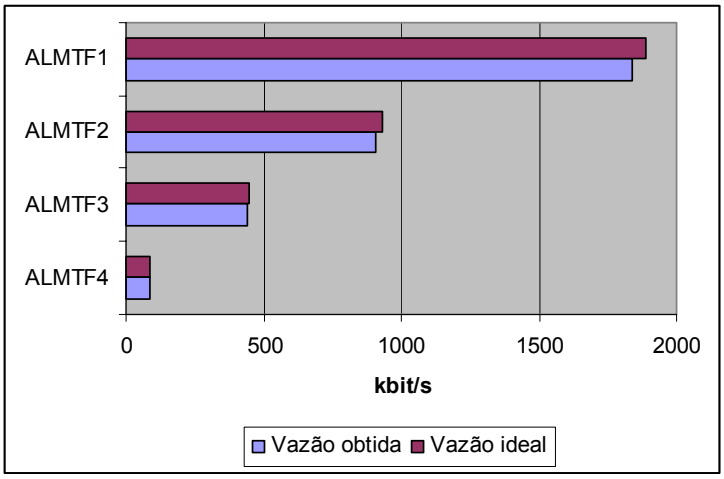


FIGURA 6.15 – Comparação entre vazão obtida e ideal para 4 receptores

A agressividade do ALMTF é dependente do RTT, tal qual o protocolo TCP, assim, a simulação com o atraso dos enlaces em 1ms ao invés de 10ms provoca um maior número de tentativas de subir camada, como pode ser constatado através da figura 6.16. Quando se utilizam pares de pacotes, o receptor descobre a banda máxima e não faz tentativas acima deste patamar, fazendo com que o gráfico tenha uma característica igual à da figura 6.14.

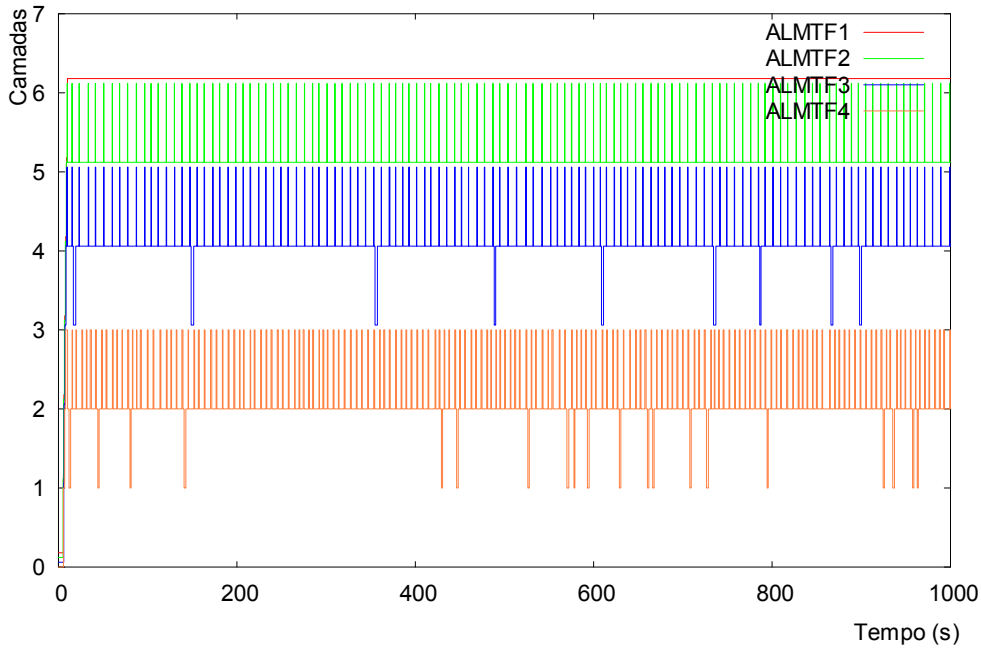


FIGURA 6.16 – Camadas exponenciais, sem pares de pacotes e atraso 1ms

Passando para tipo de camadas iguais ao do VEBIT (15 kbit/s, 45 kbit/s, 80 kbit/s, 150 kbit/s e 385 kbit/s), pode-se verificar que o sistema também se adapta

corretamente, como ilustra a figura 6.17. Os receptores ALMTF1 e ALMTF2 se cadastraram em todas as camadas, pois as mesmas consomem 675 kbit/s de banda, e ambos tem mais do que isso. Já o receptor ALMTF3 se cadastrou somente nas quatro primeiras camadas (consumindo 290 kbit/s de um enlace com 525 kbit/s), e o receptor ALMTF4 se cadastrou somente nas primeiras duas camadas (total de 60 kbit/s de um enlace com 105 kbit/s).

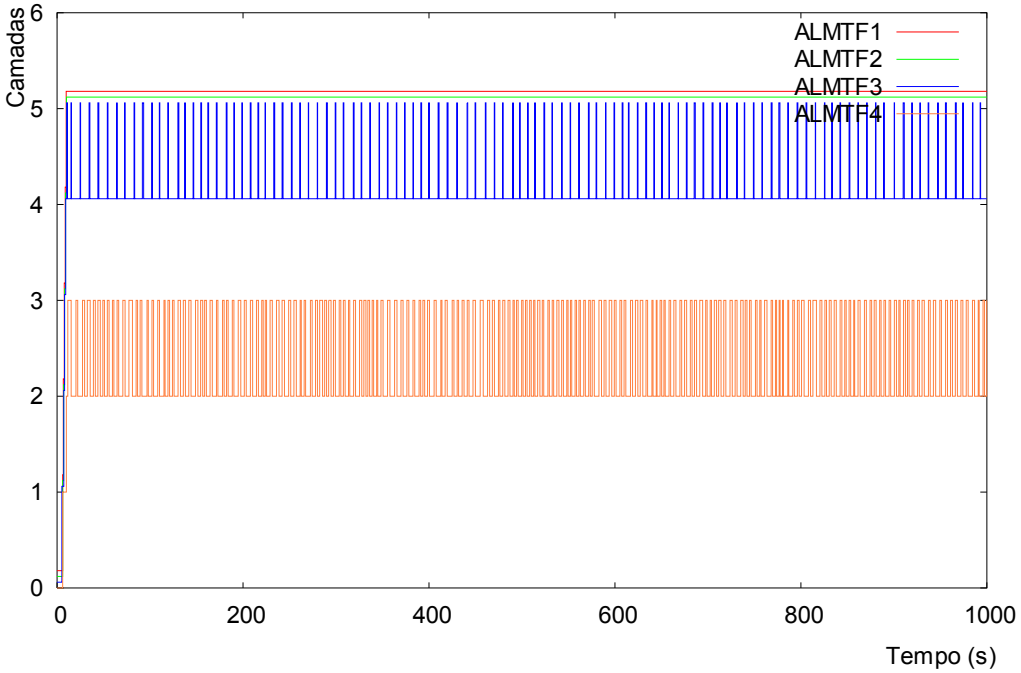


FIGURA 6.17 – Camadas exponenciais iguais ao VEBIT sem pares de pacotes

A vazão obtida e ideal para cada receptor é descrita nos itens a seguir e representada graficamente na figura 6.18.

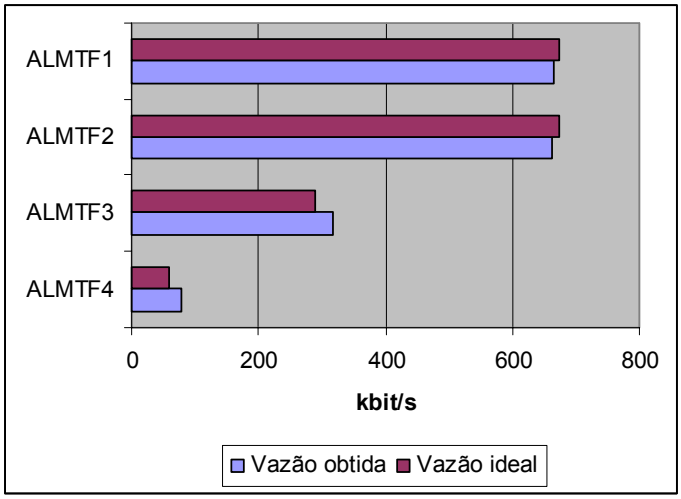


FIGURA 6.18 – Comparação entre vazão obtida e ideal para 4 receptores

Os receptores ALMTF1 e ALMTF2 atingiram praticamente 100% do valor ideal. Já os receptores ALMTF3 e ALMTF4 atingiram mais de 100%, e isso se deve às

tentativas de *join* na camada superior. No ALMTF4, por exemplo, o receptor está cadastrado normalmente na camada 2, consumindo 60 kbit/s de banda. Cada tentativa de *join* na camada 3 cria uma demanda de 140 kbit/s, entretanto, como o enlace tem 105 kbit/s, demora certo tempo até que o roteador comece a descartar pacotes. Durante esse tempo, o receptor acumula pacotes da camada 3, fazendo com que o total de pacotes recebidos seja superior ao esperado.

- Receptor ALMTF1: vazão: 665 kbit/s. Vazão ideal: 675 kbit/s (5 camadas);
- Receptor ALMTF2: vazão: 663 kbit/s. Vazão ideal: 675 kbit/s (5 camadas);
- Receptor ALMTF3: vazão: 318 kbit/s. Vazão ideal: 290 kbit/s (4 camadas);
- Receptor ALMTF4: vazão: 78 kbit/s. Vazão ideal: 60 kbit/s (2 camadas).

Repetindo a simulação com o uso de pares de pacotes, o resultado é semelhante ao obtido com camadas exponenciais, conforme ilustrado na figura 6.19, ou seja, sem tentativas de subir camada além da banda máxima.

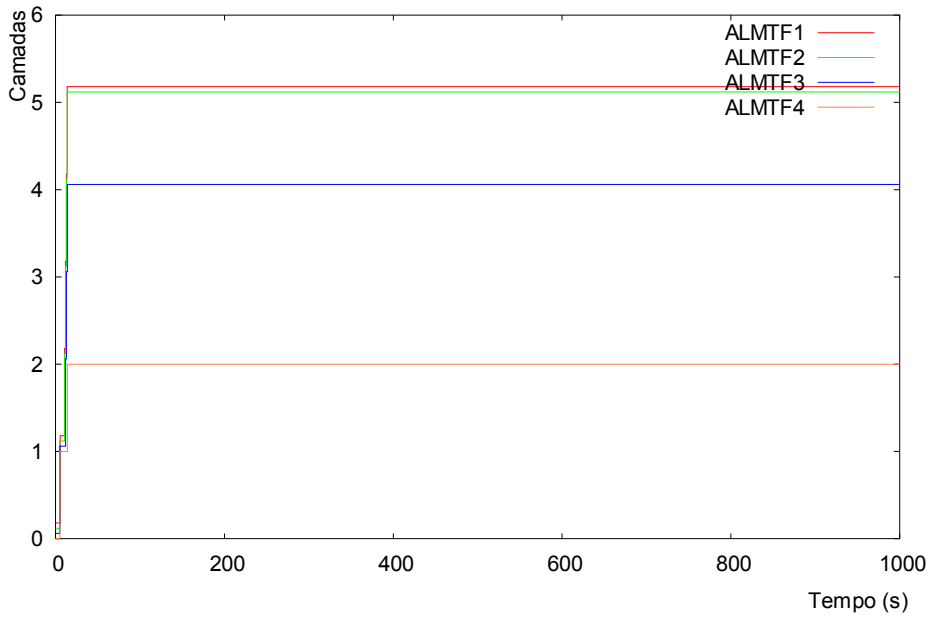


FIGURA 6.19 – Camadas exponenciais iguais ao VEBIT com pares de pacotes

Quando se utilizam camadas de 50 kbit/s sem pares de pacotes, o sistema também se adapta de acordo com a banda disponível nos receptores, conforme ilustra a figura 6.20. O receptor ALMTF1 adaptou-se em 40 camadas (2 Mbit/s), o ALMTF2 em 20 camadas (1 Mbit/s), o ALMTF3 em 10 camadas (500 kbit/s) e o ALMTF4 em 2 camadas (100 kbit/s). Pode-se observar nesse gráfico o impacto da sincronização de *join* (variável *sincjoin*, descrita anteriormente). Quanto maior a camada, mais tempo leva até a variável *sincjoin* permitir ao receptor fazer tentativa de *join*, como pode ser visto no receptor ALMTF1, que efetua menos tentativas de *join* que os outros receptores.

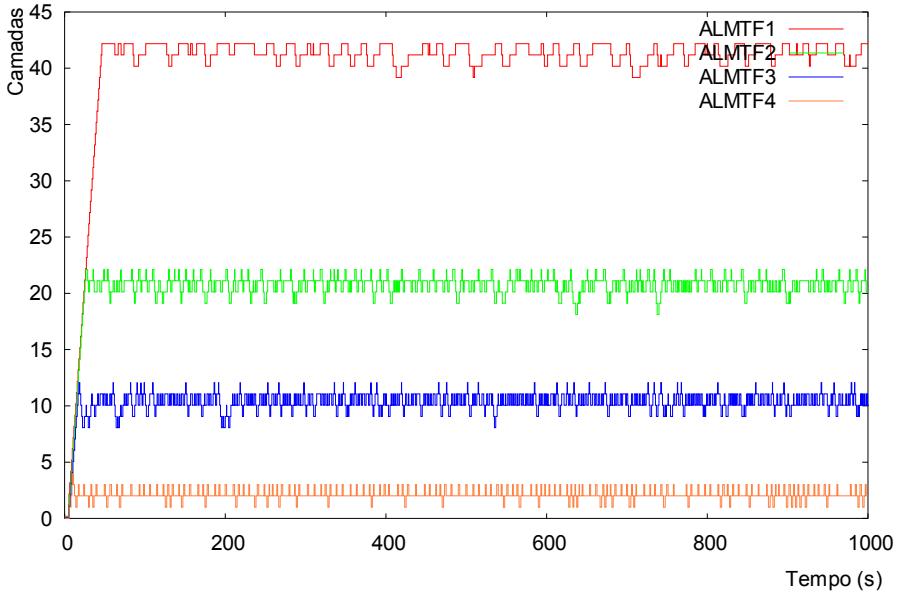


FIGURA 6.20 – Camadas de 50 kbit/s sem pares de pacotes

Com pares de pacotes a situação muda um pouco devido ao elevado número de fluxos (42 para uma banda de 2,1 Mbit/s). Isso causa o mesmo problema visto no ALMP, que é a fila ser insuficiente para conter todos os fluxos simultaneamente, ou seja, quando o transmissor envia os pacotes randomicamente, muitos são enviados ao mesmo tempo, causando um acúmulo de pacotes na fila do roteador. A conseqüência disso é a perda de pacotes, que pode ser observada no receptor ALMTF1 do gráfico da figura 6.21, que adaptou-se na camada 39. Passando a fila para 60 pacotes, o sintoma desaparece, da mesma forma que visto nas simulações do ALMP, no capítulo 5.

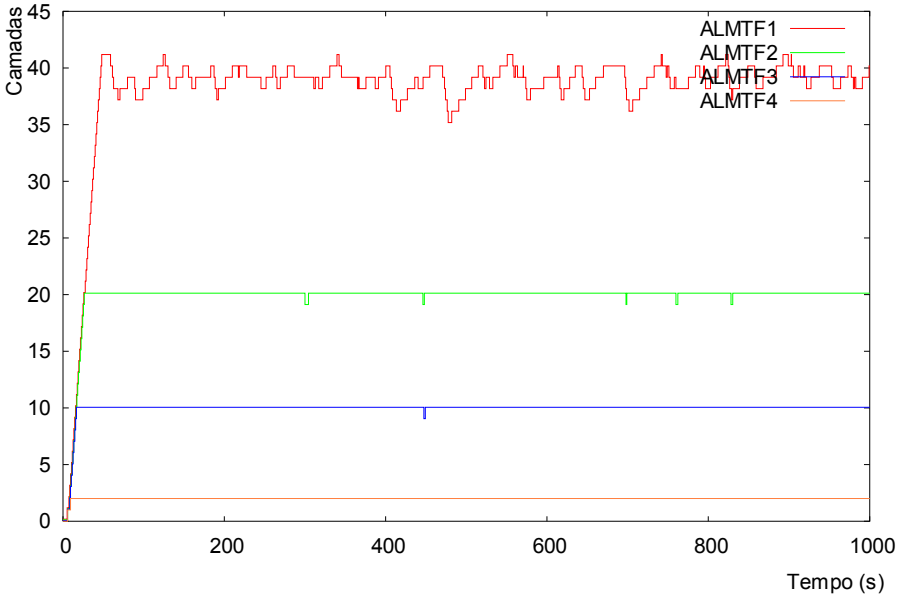
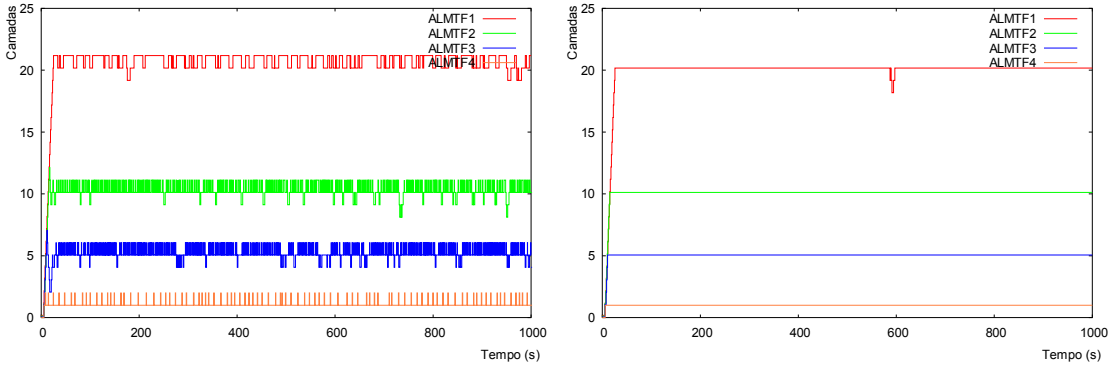


FIGURA 6.21 – Camadas de 50 kbit/s com pares de pacotes

Os outros receptores ilustrados na simulação da figura 6.21 adaptaram-se de acordo com a banda disponível: o receptor ALMTF2 na camada de número 20 (correto

para banda de 1,05 Mbit/s), o receptor ALMTF3 na camada 10 (enlace de 525 kbit/s) e o receptor ALMTF4 na camada 2 (correto para 105 kbit/s).

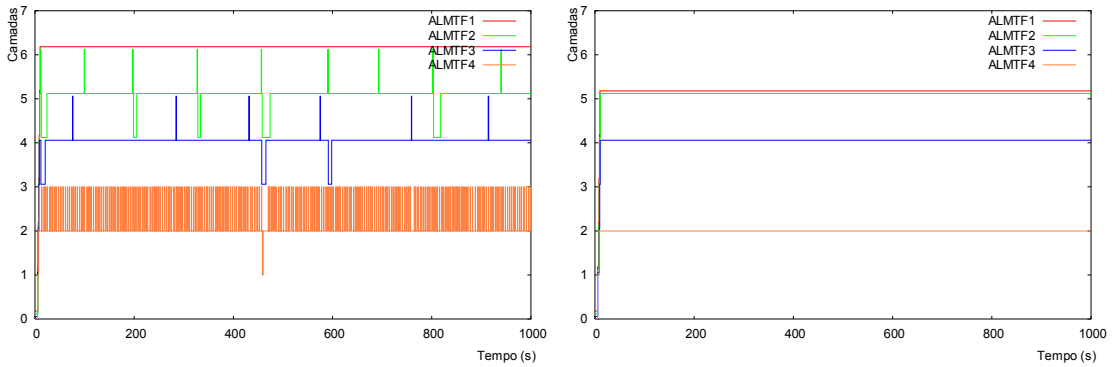
Modificando o tipo das camadas para de 100 kbit/s, o sistema se adapta de forma coerente com a banda disponível, como é ilustrado na figura 6.22, onde o gráfico "a" representa a adaptação sem a utilização de pares de pacotes, e o gráfico "b" utilizando pares de pacotes. Os receptores ALMTF1 a ALMTF4 adaptaram-se ao correto número de camadas (respectivamente 20, 10, 5 e 1 camada).



a) Sem pares de pacotes b) Com pares de pacotes

FIGURA 6.22 – ALMTF com camadas de 100 kbit/s

Diminuindo o tamanho do pacote de 500 bytes para 250 bytes, o número de pacotes por segundo enviado pelo transmissor deve ser duplicado, a fim de que a taxa da camada seja mantida. Isso provoca uma maior utilização das filas no simulador, que é baseado em número de pacotes e não em bytes. O resultado é ilustrado na figura 6.23.



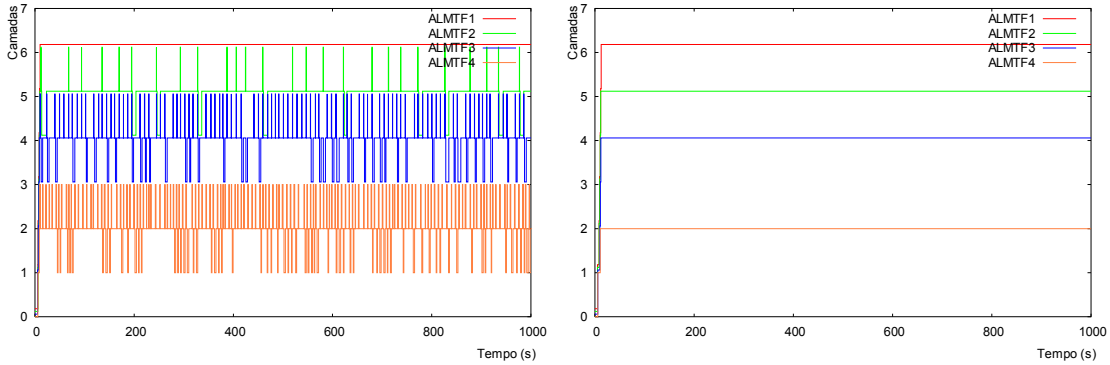
a) Sem pares de pacotes b) Com pares de pacotes

FIGURA 6.23 – Camadas exponenciais e pacote de 250 bytes

A figura 6.23a representa a simulação sem pares de pacotes, onde todos os receptores adaptaram-se adequadamente. No gráfico da figura 6.23b, com pares de pacotes, o receptor ALMTF1 não atingiu sua camada correta devido ao tamanho da janela de congestionamento, pois o cálculo da banda máxima (para uma janela de 30 pacotes) foi menor do que a banda máxima da rede. A banda necessária para subir para a camada número seis é 1,89 Mbit/s, porém, para um RTT de 42ms (obtido na simulação), *bwshare = (30*250*8) / 0,042*, ou seja, *bwshare* = 1,42 Mbit/s, que é menor do que os 1,89 Mbit/s. A alternativa para eliminar esse sintoma seria aumentar o valor da variável "*windowsize*" do algoritmo ALMTF. No gráfico "a" o sintoma não

aconteceu, pois não se está utilizando pares de pacotes e não houve perdas. Nesse caso, a banda da equação atinge a banda máxima estabelecida para a rede, ou seja, 10 Mbit/s. Como *bwshare* deve ser, no mínimo, metade da banda da equação, ele fica fixo em 5 Mbit/s, porém, como está na última camada, não tem mais o que subir. Caso houvesse perdas, a banda da equação e *bwshare* se ajustariam.

A simulação ilustrada na figura 6.24 modifica o tipo de fila para RED 10/20, voltando o tipo das camadas para exponenciais padrão. O gráfico "a" representa a simulação sem pares de pacotes, enquanto o gráfico "b" com pares de pacotes. No gráfico "a", observa-se um aumento na instabilidade em relação à mesma simulação com fila *droptail* de 20 pacotes (vista na figura 6.12). Isso pode ser explicado através do funcionamento do RED, que descarta pacotes com determinada probabilidade quando a fila atinge 10 posições (*threshold* mínimo), chegando a descartar 100% dos pacotes quando a fila atinge a posição 20 (*threshold* máximo). O resultado é um maior número de perdas, fazendo o receptor responder mais rápido ao congestionamento, porém, apresentando menor estabilidade. O gráfico "b" mostra que, com pares de pacotes, o receptor não tenta subir camadas além da sua banda disponível, gerando uma maior estabilidade ao sistema.



a) Sem pares de pacotes b) Com pares de pacotes

FIGURA 6.24 – ALMTF com fila RED 10/20 sem e com pares de pacotes

Foram feitas outras simulações, conforme descrição no item 4.8 (Definição das simulações), entretanto, elas não geraram alterações significativas nos resultados, e somente serão comentadas brevemente a seguir, entretanto, estão disponíveis em http://www.inf.unisinos.br/~roesler/tese e no CD anexo a esta Tese.

- **Tamanho de pacote de 1000 bytes ao invés de 500 bytes**: quando se utilizam pacotes de 1000 bytes, o número de pacotes enviados por segundo é menor, pois a taxa de transmissão da camada é constante. Isso onera menos a fila, e observa-se um comportamento semelhante às simulações com uma fila de tamanho maior;
- **Fila RED 10/20 e camadas de 100 kbit/s**: o resultado foi bastante semelhante ao obtido com fila *droptail*, visto na figura 6.22;
- **Fila RED 20/60 com camadas exponenciais**: o resultado foi muito parecido com o obtido com filas RED 10/20, mostrado na figura 6.24;
- **Sem randomização (CBR puro)**: nesse caso, o gráfico gerado é praticamente o mesmo do que com randomização na transmissão dos pacotes (±0,5s), visto na figura 6.12. Valem as observações efetuadas para a mesma simulação do ALMP.

6.2.2 Análise de escalabilidade do ALMTF

Para analisar a escalabilidade do algoritmo, a metodologia prevê aumentar o número de receptores buscando verificar mudanças no comportamento dos fluxos e o impacto para um grande número de usuários. Nas simulações de adaptabilidade, foram feitos os testes para quatro receptores, cada um deles localizado num enlace com largura de banda diferente. Nos testes de escalabilidade, esse número será aumentado, e serão analisados os resultados para um total de 20 e 100 receptores. Com essa informação, pode-se analisar o algoritmo em três escalas: 1 receptor por bloco (visto no item 6.2.1), 5 receptores por bloco e 25 receptores por bloco, ou seja, um total de 4 receptores (visto no item 6.2.1), 20 receptores e 100 receptores.

As simulações de escalabilidade também visam testar a capacidade do algoritmo em adaptar-se a ambientes heterogêneos, no caso da existência de vários receptores localizados atrás de cada roteador.

As simulações tomarão como base os mesmos parâmetros utilizados nos testes de adaptabilidade, e a topologia utilizada também é a topologia 1 da figura 4.3, que é repetida na figura 6.25 já com os parâmetros específicos utilizados para testar a escalabilidade do algoritmo em ambientes heterogêneos. Os parâmetros configurados para a primeira simulação estão descritos a seguir, e serão utilizados como base para as outras simulações, que vão modificar alguns desses parâmetros:

- Número de receptores por bloco: r = 5, totalizando 20 receptores e r = 25, totalizando 100 receptores;
- Camadas exponenciais padrão (30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s e 960 kbit/s);
- Tamanho do pacote = 500 bytes;
- Tipo de fila = *droptail* com 20 pacotes;
- Randomização alta (±0,5s);
- Sem uso de pares de pacotes.

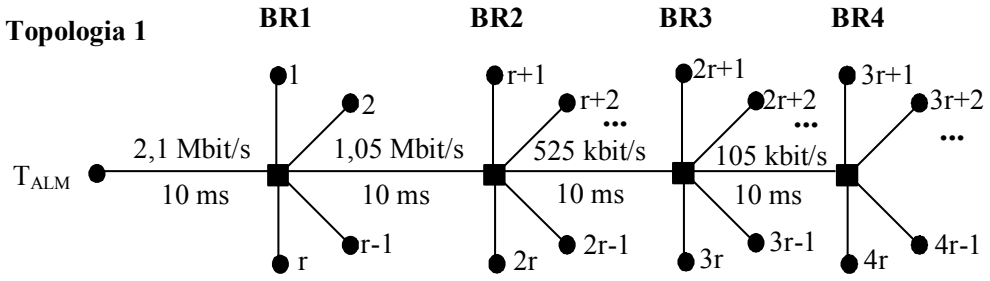


FIGURA 6.25 – Topologia para simulações de escalabilidade

Um fator importante é que todos receptores em uma determinada sub-rede devem chegar às mesmas conclusões sobre o número de camadas a se inscrever, caso contrário, o algoritmo não vai funcionar adequadamente numa situação de congestionamento, visto que um único receptor cadastrado em determinada camada (grupo multicast) mantém o tráfego dos pacotes para toda sub-rede.

O gráfico da figura 6.26 mostra o resultado das simulações para 20 receptores sem a utilização de pares de pacotes. Utilizou-se gráfico por fluxo ao invés de camadas

para melhorar a legibilidade.

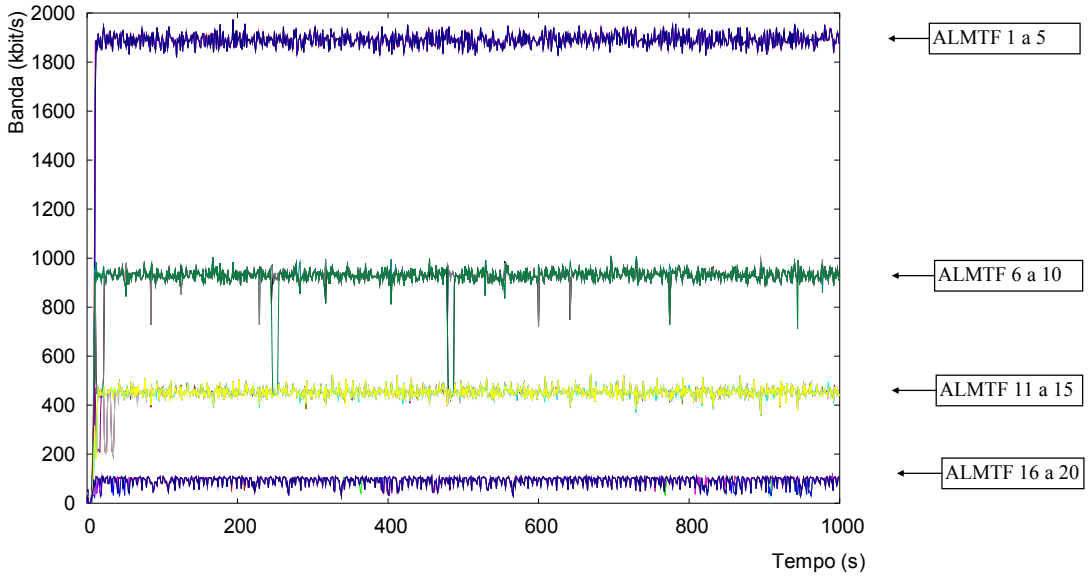


FIGURA 6.26 – Escalabilidade do ALMTF - 20 receptores sem pares de pacotes

Pode-se ver que todos receptores adaptaram-se de acordo com a largura de banda disponível. Os receptores ALMTF1 a ALMTF5 se inscreveram em 6 camadas (1890 kbit/s cada), que é consistente com o limite de 2,1 Mbit/s. Os receptores ALMTF6 a ALMTF10 se inscreveram em 5 camadas (930 kbit/s), que é consistente com o gargalo de 1,05 Mbit/s. Os receptores ALMTF11 a ALMTF15 se inscreveram em 4 camadas (450 kbit/s), coerente com o gargalo de 525kbit/s, e os receptores ALMTF16 a ALMTF20 se inscreveram em 2 camadas (90 kbit/s), que é adequado ao gargalo de 105 kbit/s.

Existe uma maior instabilidade nas camadas inferiores, devido à característica do algoritmo de permitir mais tentativas de *join* nas camadas inferiores, conforme descrição no item 6.1.8. A vazão obtida e ideal para cada bloco de receptores é ilustrada graficamente na figura 6.27.

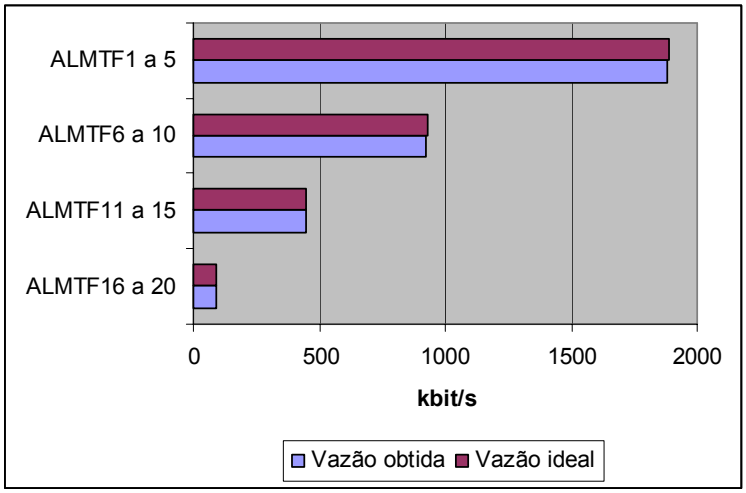


FIGURA 6.27 – Comparação entre vazão obtida e ideal para 20 receptores

A maior parte dos receptores atingiu uma vazão próxima a 100%, como pode-se ver pelos itens a seguir, que detalham o resultado ilustrado na figura 6.27.

- Receptores ALMTF 1 a 5: vazão: 1884 a 1885 kbit/s. Vazão ideal: 1890 kbit/s (camadas 1 a 6);
- Receptores ALMTF 6 a 10: vazão: 917 a 928 kbit/s. Vazão ideal: 930 kbit/s (camadas 1 a 5);
- Receptores ALMTF 11 a 15: vazão: 448 a 450 kbit/s. Vazão ideal: 450 kbit/s (camadas 1 a 4);
- Receptores ALMTF 16 a 20: vazão: 89 a 91 kbit/s. Vazão ideal: 90 kbit/s (camada 1 + camada 2).

A figura 6.28 mostra a escalabilidade do ALMTF para 100 receptores, sem a utilização de pares de pacotes. Pode-se constatar que todos adaptaram-se de forma coerente com a largura de banda disponível, da mesma forma que a simulação com 20 receptores. Algumas tentativas de subir camada também podem ser observadas, principalmente nas camadas de baixo, onde o número permitido de tentativas de *join* é maior, conforme detalhado anteriormente. Pode-se observar também que, quando um receptor sobe ou desce de camada, muitos receptores fazem a mesma coisa, pois os mesmos estão sincronizados.

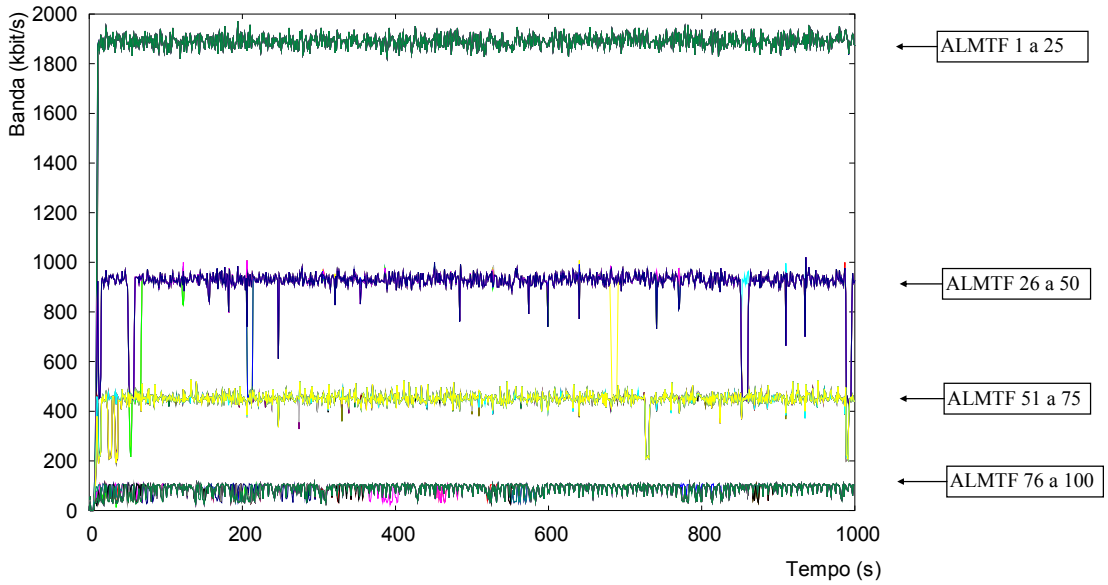


FIGURA 6.28 – Escalabilidade do ALMTF – 100 receptores sem pares de pacotes

A vazão obtida e ideal para cada bloco de receptores é descrita nos itens a seguir e ilustrado graficamente na figura 6.29. Todos os receptores atingiram uma vazão próxima a 100% da vazão ideal.

- Receptores ALMTF 1 a 25: vazão: 1875 a 1878 kbit/s. Vazão ideal: 1890 kbit/s;
- Receptores ALMTF 26 a 50: vazão: 901 a 925 kbit/s. Vazão ideal: 930 kbit/s;
- Receptores ALMTF 51 a 75: vazão: 427 a 445 kbit/s. Vazão ideal: 450 kbit/s;
- Receptores ALMTF 76 a 100: vazão: 82 a 89 kbit/s. Vazão ideal: 90 kbit/s.

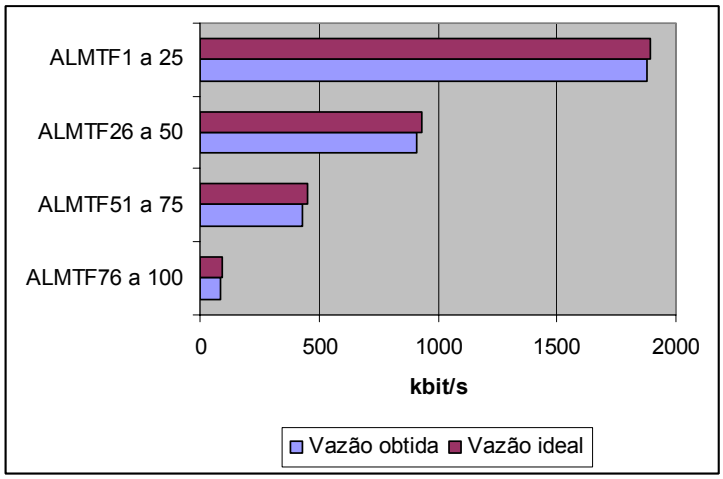


FIGURA 6.29 – Comparação entre vazão obtida e ideal para 100 receptores

A figura 6.30 mostra a mesma simulação de 100 receptores com o uso de pares de pacotes. Pode-se perceber que o sistema repetiu o comportamento mostrado nas simulações anteriores com pares de pacotes, ou seja, adaptou-se de acordo com a banda disponível sem efetuar tentativas de subir camadas acima da banda nominal do enlace.

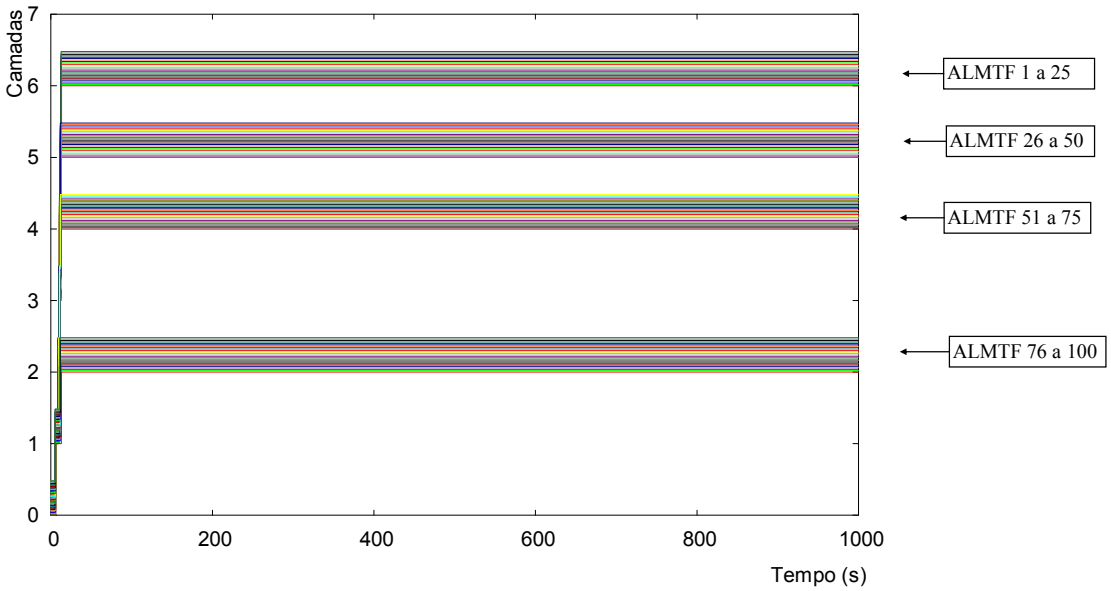


FIGURA 6.30 – Escalabilidade do ALMTF – 100 receptores com pares de pacotes

A tabela 6.1 mostra uma comparação entre estabilidade, perdas e vazão do algoritmo ALMTF para um único transmissor e vários receptores, onde r = 1, 5 e 25, ou seja, um total de 4, 20 e 100 receptores. A taxa de cada camada é exponencial, e as simulações são sem pares de pacotes. Os valores para 20 e 100 receptores são os valores médios obtidos para cada bloco de receptores. O valor e significado de VCM (Variações de Camada por Minuto) foi explicado no item 4.7.5, e será detalhado para o ALMTF no item 6.2.4 (Análise da estabilidade do ALMTF). O método de obtenção da taxa de perdas foi explicado no item 4.7.5, e será detalhado para o ALMTF no item 6.2.6 (Análise da taxa de perdas do ALMTF).

Através da comparação da estabilidade (coluna VCM na tabela 6.1) linha a linha entre as três escalas do algoritmo, percebe-se que a estabilidade de cada fluxo não é modificada significativamente quando se aumenta o número de receptores. Por exemplo, no pior caso, o fluxo ALMTF4 teve um VCM de 21,6, enquanto que a média dos fluxos ALMTF76 a 100 foi de 21,4. Isso se explica pois todos receptores efetuam tentativas de subir camada de forma sincronizada, através da variável *sincjoin*, explicada no início do capítulo.

Analisando a taxa de perdas entre as três escalas do algoritmo (coluna "taxa média de perdas" na tabela 6.1), percebe-se que a taxa de perdas também não é influenciada de forma significativa com o aumento do número de receptores, e isso também se deve graças ao sincronismo entre os receptores.

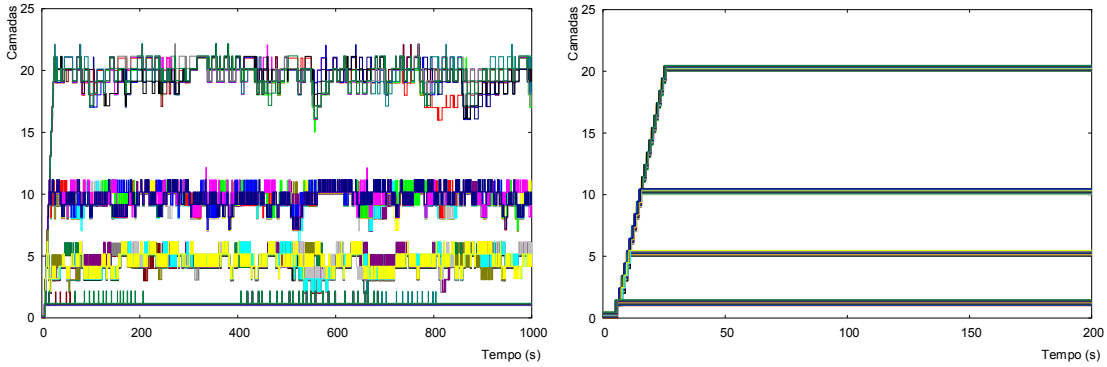
Analisando a vazão (coluna "vazão média" na tabela 6.1), observa-se que o algoritmo diminui um pouco a vazão com o aumento no número de receptores. Por exemplo, no pior caso, o receptor ALMTF4 obteve uma vazão de 92 kbit/s. Cada um dos 25 receptores mostrados na última linha da tabela (ALMTF76 a 100) obtiveram uma vazão de 85 kbit/s, ou seja, aproximadamente 5% menor. A diminuição foi menor para os outros receptores, como pode ser visto na tabela.

A coluna de vazão também serve para mostrar a adaptabilidade do algoritmo com o aumento no número de receptores. De acordo com os resultados observados, percebe-se que o algoritmo é adaptável até 100 receptores, e existe uma tendência que consiga se adaptar com mais receptores.

TABELA 6.1 – Análise de estabilidade, perdas e vazão para o algoritmo ALMTF

Total de receptores	VCM médio	Taxa média de perdas	Vazão média (kbit/s)
4 receptores	ALMTF1: 0,00 ALMTF2: 2,00 ALMTF3: 13,9 ALMTF4: 21,6	ALMTF1: 0,00% ALMTF2: 0,72% ALMTF3: 1,18% ALMTF4: 3,33%	ALMTF1: 1885 ALMTF2: 927 ALMTF3: 451 ALMTF4: 92
20 receptores	ALMTF1 a 5: 0,00 ALMTF6 a 10: 1,80 ALMTF11 a 15: 13,7 ALMTF16 a 20: 20,4	ALMTF1 a 5: 0,00% ALMTF6 a 10: 0,55% ALMTF11 a 15: 1,20% ALMTF16 a 20: 3,58 %	ALMTF1 a 5: 1884 ALMTF6 a 10: 921 ALMTF11 a 15: 449 ALMTF16 a 20: 90
100 receptores	ALMTF1 a 25: 0,00 ALMTF26 a 50: 1,50 ALMTF51 a 75: 13,5 ALMTF76 a 100: 20,4	ALMTF1 a 25: 0,00% ALMTF26 a 50: 0,62% ALMTF51 a 75: 1,47% ALMTF76 a 100: 3,70%	ALMTF1 a 25: 1877 ALMTF26 a 50: 910 ALMTF51 a 75: 432 ALMTF76 a 100: 85

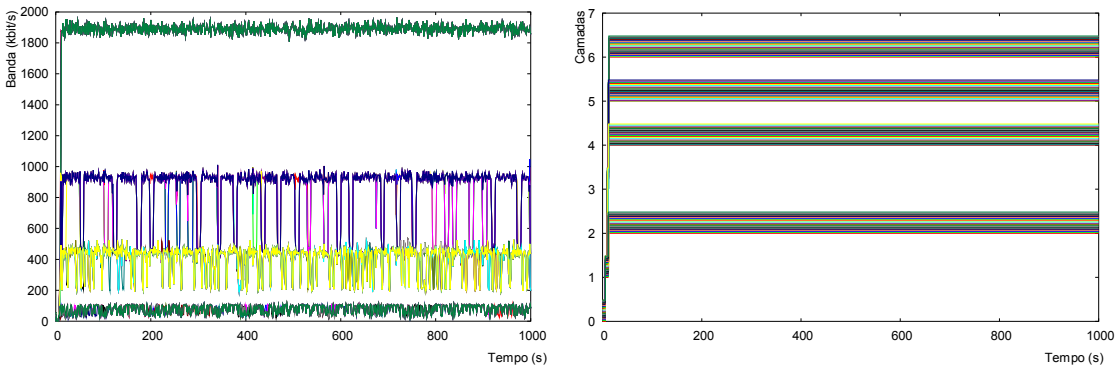
Com camadas de 100 kbit/s o sistema também adaptou-se corretamente, conforme ilustrado na figura 6.31, onde o gráfico "a" representa a simulação por fluxo (a fim de facilitar a legibilidade) e sem pares de pacotes. Como nem todos receptores estão exatamente com a mesma banda no momento que chega o *sincjoin*, alguns sobem de camada e outros não, provocando certa diferença na banda utilizada, conforme pode ser visto no gráfico "a". O gráfico "b" representa a simulação com pares de pacotes, mostrando a adaptação por camadas. Pode-se verificar que todos os receptores se mantiveram na camada correta e estáveis ao longo do tempo.



a) Sem pares de pacotes b) Com pares de pacotes

FIGURA 6.31 – 100 receptores: camadas de 100 kbit/s

Outra simulação efetuada foi com fila RED 10/20 e 100 receptores, conforme ilustra a figura 6.32, onde o gráfico "a" representa a simulação por fluxo (a fim de facilitar a legibilidade) e sem pares de pacotes. Observa-se uma maior instabilidade em relação aos gráficos anteriores, principalmente aos receptores inscritos na camada 5 (ALMTF 26 a 50 – correspondente a uma banda de 930 kbit/s) e aos inscritos na camada 4 (ALMTF 51 a 75 – correspondente a uma banda de 450 kbit/s). O gráfico "b" mostra o resultado da simulação com pares de pacotes, onde o sistema ficou estável, assemelhando-se aos resultados já vistos anteriormente.



a) Sem pares de pacotes b) Com pares de pacotes

FIGURA 6.32 – 100 receptores: camadas exponenciais, RED 10/20

A partir das simulações de escalabilidade obtidas, pode-se concluir que o sistema aceita um grande número de usuários sem perder suas características. Uma das razões para isso é a utilização de sincronismo de *join* entre receptores cadastrados na mesma sessão. Entretanto, mesmo com o uso de tal mecanismo, acontece um aumento no número de tentativas de subir camada, provocando um aumento no número de perdas e maior instabilidade, quando comparada a apenas um receptor.

6.2.3 Análise de eqüidade de tráfego do ALMTF

As simulações de análise de eqüidade de tráfego têm por objetivo analisar o comportamento do algoritmo com tráfego concorrente, verificando se o mesmo tem capacidade para compartilhar tráfego igualmente com outros fluxos. No caso do ALMTF, o tráfego concorrente é composto de: a) outras sessões ALMTF concorrentes;

b) tráfego TCP; c) tráfego CBR.

A topologia utilizada para as simulações foi a topologia 2 da figura 4.3, que é reproduzida na figura 6.33 já com os parâmetros específicos utilizados para testar a eqüidade do algoritmo. Na topologia, existe um número *m* de fluxos ALMP e um número *n* de fluxos TCP concorrendo através de um enlace único. Os parâmetros configurados para a primeira simulação estão descritos a seguir, e serão utilizados como base para as outras simulações, que vão modificar alguns desses parâmetros:

- Banda "B" no enlace central especificada como *(m+n)*500kbit/s*, onde *m* é o número de transmissores ALM no momento, e *n* é o número de transmissores TCP concorrentes. Isso foi feito para acomodar 4 camadas exponenciais para cada sessão ALM (que demandam um total de 450 kbit/s: 30+60+120+240);
- Atraso A = 10ms;
- Camadas exponenciais padrão (30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s e 960 kbit/s);
- Tamanho do pacote = 500 bytes;
- Tipo de fila = *droptail* com 20 pacotes;
- Randomização alta (±0,5s);
- Com uso de pares de pacotes.

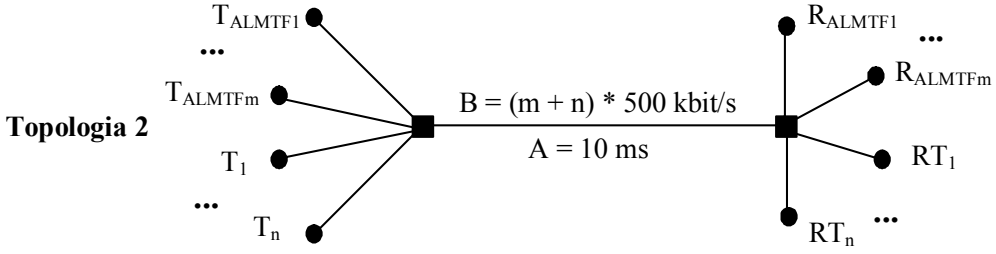


FIGURA 6.33 – Topologia para simulações de eqüidade

A seguir serão analisados os resultados obtidos para as simulações especificadas no item 4.8 (Definição das simulações) para eqüidade de tráfego.

A simulação da figura 6.34 mostra dois fluxos ALMTF concorrendo por 1 Mbit/s de banda no enlace central.

Pode-se constatar que os dois receptores adaptaram-se na camada correta, dividindo igualmente a banda. A vazão do receptor ALMTF1 ficou em 444 kbit/s, enquanto a vazão do receptor ALMTF2 foi de 445 kbit/s, ou seja, praticamente a mesma, mostrando que o algoritmo ALMTF é altamente eqüitativo entre sessões do próprio protocolo.

Da mesma forma que explicado para as simulações de eqüidade do algoritmo ALMP, a partir desse momento todos os gráficos serão gerados com o mecanismo de pares de pacotes, pois o mesmo é parte integrante do algoritmo ALMTF.

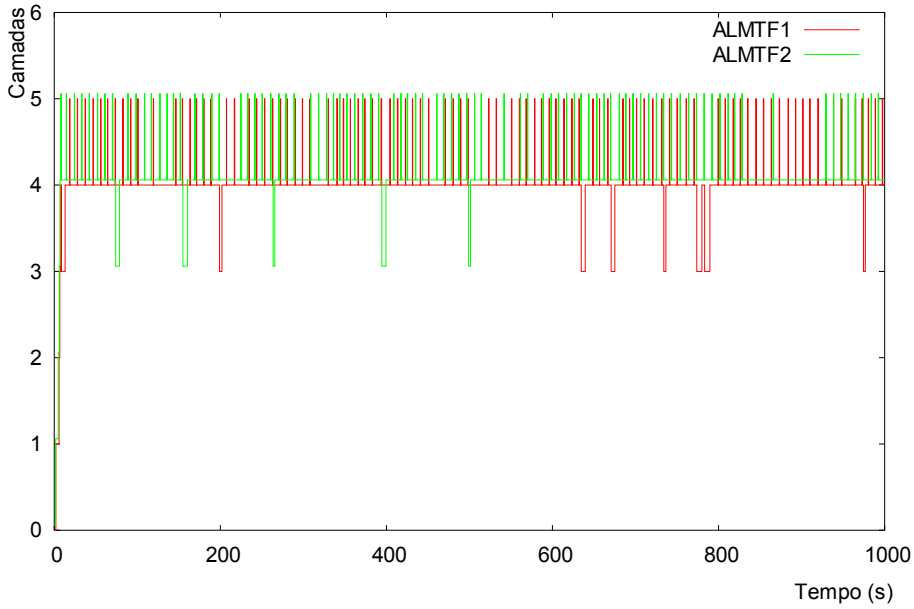


FIGURA 6.34 – Eqüidade do ALMTF para 2 fluxos concorrentes

Acrescentando dois fluxos TCP e aumentando a banda no enlace central para 2 Mbit/s, o sistema mantém a estabilidade, como mostra a figura 6.35. Pode-se observar que os dois fluxos ALMTF ficaram estáveis na camada 4 (aproximadamente 450 kbit/s), enquanto os dois fluxos TCP variavam bastante no entorno da banda eqüitativa, que é de 500 kbit/s. A vazão dos dois fluxos ALMTF foi de 405 e 419 kbit/s, para o ideal de 450 kbit/s, e a vazão para os dois fluxos TCP foi de 507 e 529 kbit/s, respectivamente. O tempo de simulação foi reduzido para 200s a fim de permitir uma melhor visualização do gráfico.

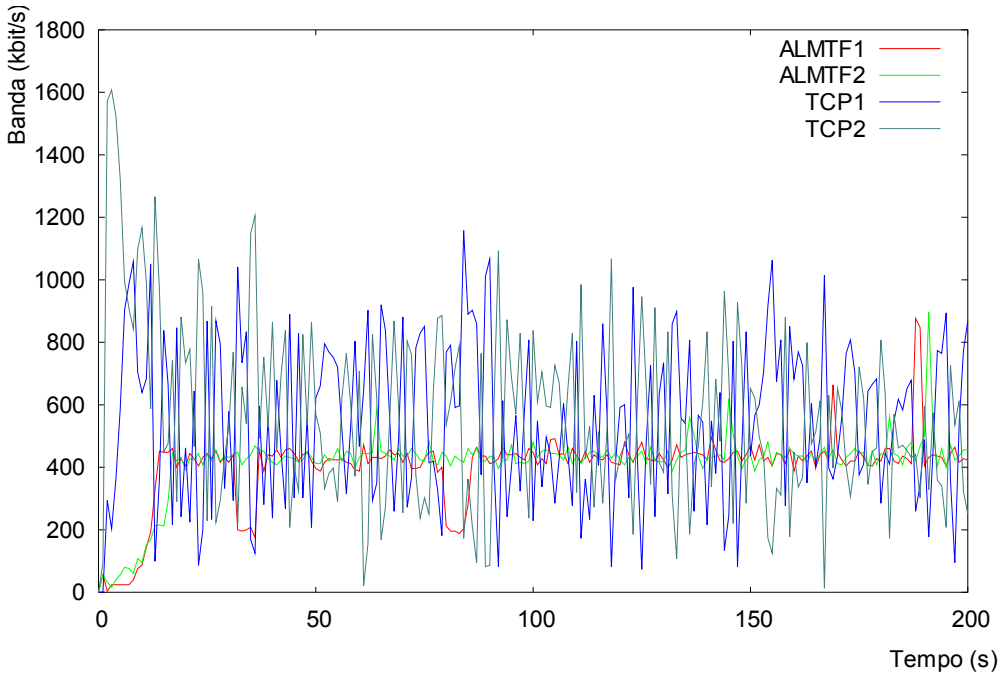


FIGURA 6.35 – Eqüidade para 2 fluxos ALMTF e dois fluxos TCP concorrentes

O gráfico da figura 6.36 mostra o compartilhamento de banda entre 10 fluxos ALMTF concorrentes, e banda no enlace central de 5 Mbit/s. Observa-se que todos os receptores adaptaram-se de acordo com a banda disponível, aproximadamente na camada 4. A vazão entre os dez receptores pode ser resumida da seguinte forma: mínima = 467 kbit/s (*fairness index* FI = 1,03); máxima = 496 kbit/s (FI = 1,1); média = 479,3 kbit/s (FI = 1,06).

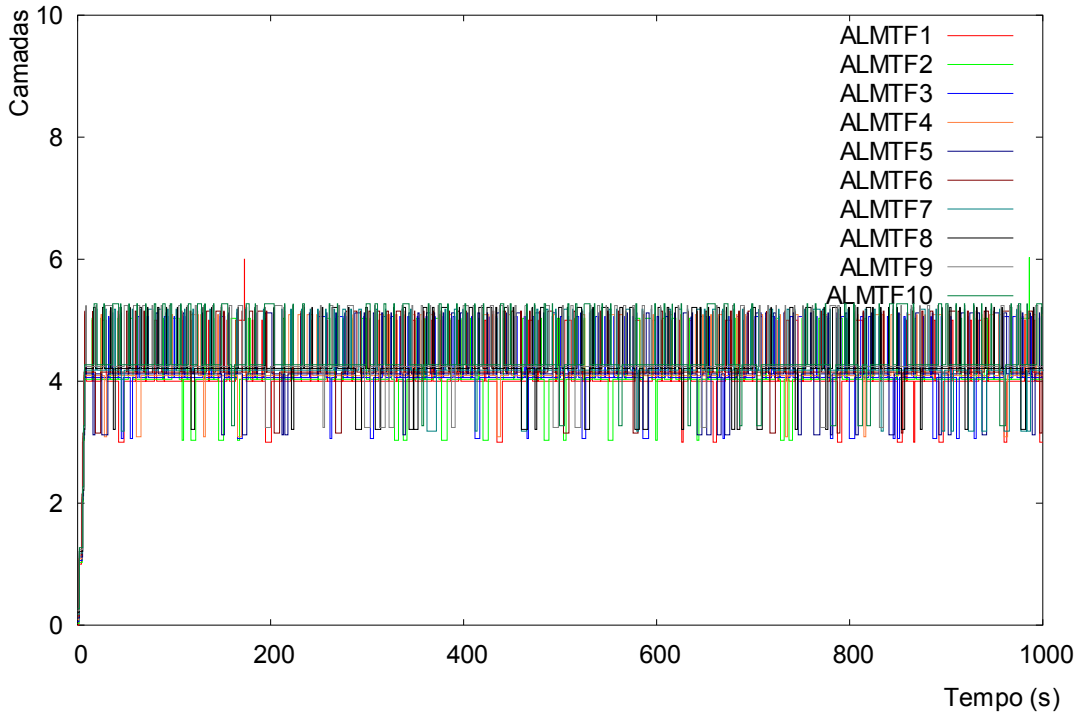
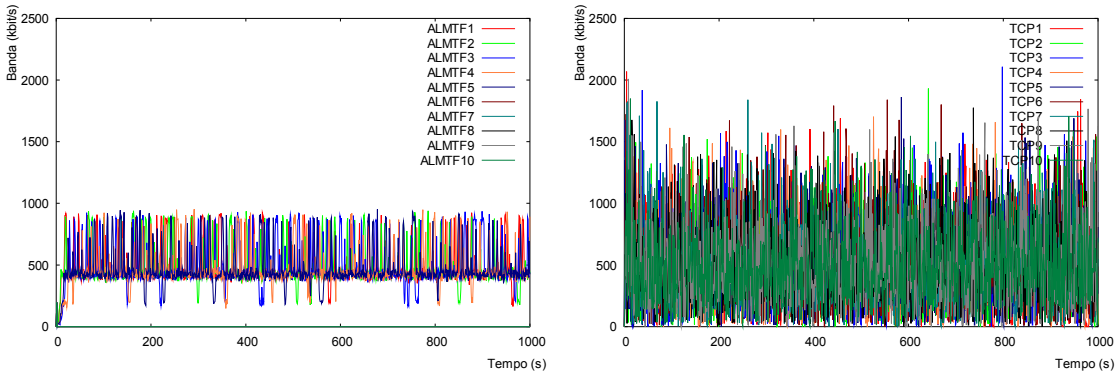


FIGURA 6.36 – Eqüidade para 10 fluxos ALMTF concorrentes e camadas exponenciais

A figura 6.37 mostra 20 fluxos concorrentes, sendo dez deles do tipo ALMTF e os outros dez TCP, todos compartilhando um enlace de 10 Mbit/s. O gráfico "a" mostra os fluxos ALMTF, enquanto o gráfico "b" mostra os fluxos TCP.



a) Fluxos ALMTF b) Fluxos TCP

FIGURA 6.37 – Eqüidade para 10 fluxos ALMTF e 10 fluxos TCP concorrentes

Observa-se que, apesar de não possuírem uma estabilidade perfeita, os fluxos ALMTF são bem mais estáveis que os fluxos TCP. A vazão dos dez fluxos ALMTF foi: mínima = 462 kbit/s; máxima = 483 kbit/s; média = 475 kbit/s. A vazão para os dez

fluxos TCP foi de: mínima = 440 kbit/s; máxima = 480 kbit/s; média = 460,5 kbit/s. O resultado da figura 6.37 mostra que o ALMTF, além de transmitir de forma eqüitativa com seu próprio tráfego, também transmite de forma eqüitativa com fluxos TCP concorrentes.

O gráfico da figura 6.38 mostra a eqüidade de tráfego do ALMTF para cinco receptores e camadas exponenciais iguais às geradas pelo codificador VEBIT. As camadas do VEBIT foram definidas como 15 kbit/s, 45 kbit/s, 80 kbit/s, 150 kbit/s e 385 kbit/s. Como explicado no capítulo do ALMP, é melhor utilizar 1,75 Mbit/s no enlace central, ou 350 kbit/s por receptor, para existir um valor ótimo para que todos se adaptem. No caso, o valor ótimo é a camada 4, ou 290 kbit/s. A vazão dos cinco fluxos ALMTF foi: mínima = 305 kbit/s; máxima = 321 kbit/s; média = 310,5 kbit/s.

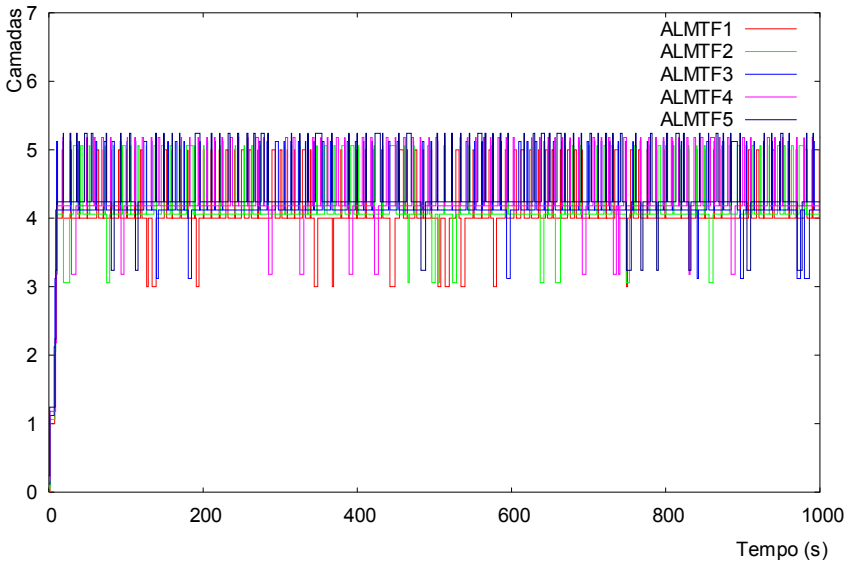
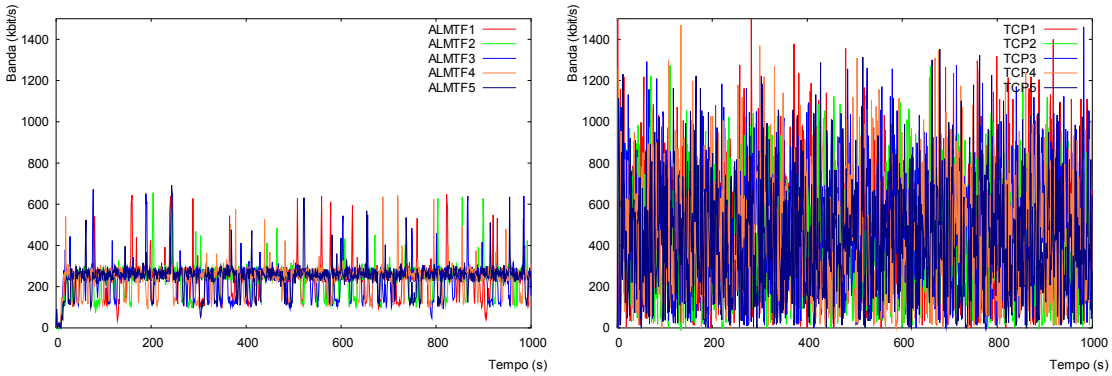


FIGURA 6.38 – Eqüidade para camadas iguais ao VEBIT – enlace de 1,75 Mbit/s

Refazendo a simulação para cinco receptores ALMTF e cinco TCP compartilhando um enlace de 3,5 Mbit/s, o equivalente a 350 kbit/s por receptor, tem-se o resultado mostrado na figura 6.39.



a) Fluxos ALMTF b) Fluxos TCP

FIGURA 6.39 – 5 ALMTF com camadas iguais ao VEBIT e 5 TCP concorrentes

A vazão dos cinco fluxos ALMTF (gráfico “a”) foi: mínima = 236 kbit/s; máxima = 249 kbit/s; média = 243 kbit/s. A vazão para os cinco fluxos TCP (gráfico

"b") foi de: mínima = 399 kbit/s; máxima = 431 kbit/s; média = 413,4 kbit/s. Nessa simulação, o TCP foi mais agressivo que o ALMTF.

Quando se utilizam camadas de 100 kbit/s, o espaçamento entre a camada atual e a próxima é menor do que com camadas exponenciais, tornando o sistema menos estável, como pode ser observado na figura 6.40. Pode-se observar que todos receptores ficaram a maior parte do tempo entre as camadas 4 e 6, que correspondem à banda eqüitativa para o enlace central com 5 Mbit/s. A vazão ideal eqüitativa para os receptores é de 500 kbit/s, e a vazão obtida para os cinco fluxos ALMTF foi: mínima = 484 kbit/s; máxima = 503 kbit/s; média = 493,4 kbit/s. Esse resultado mostra que o ALMTF transmite de forma eqüitativa com seu próprio tráfego, mesmo com camadas de 100 kbit/s.

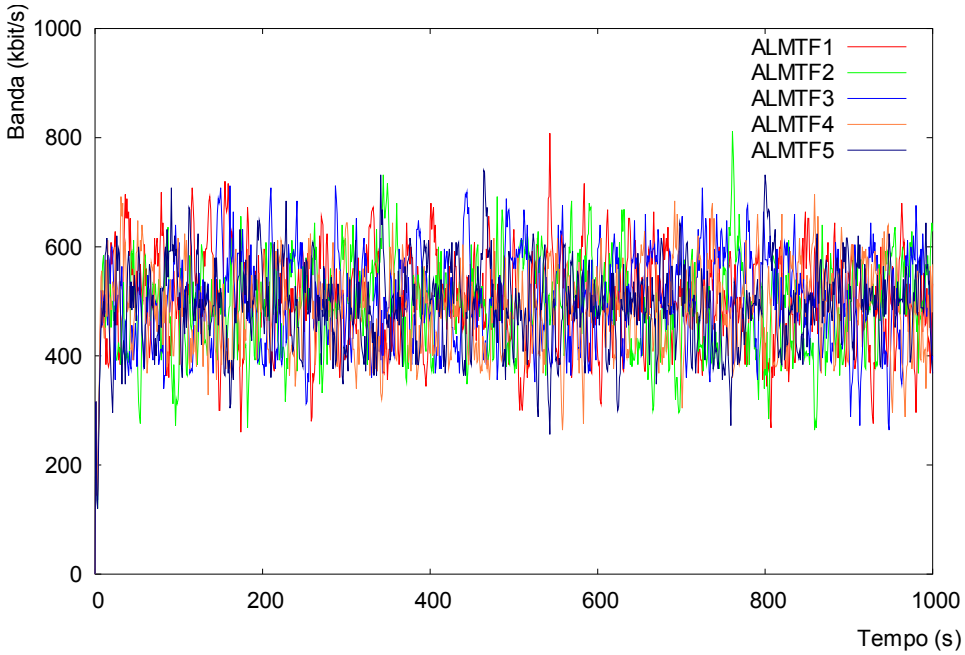
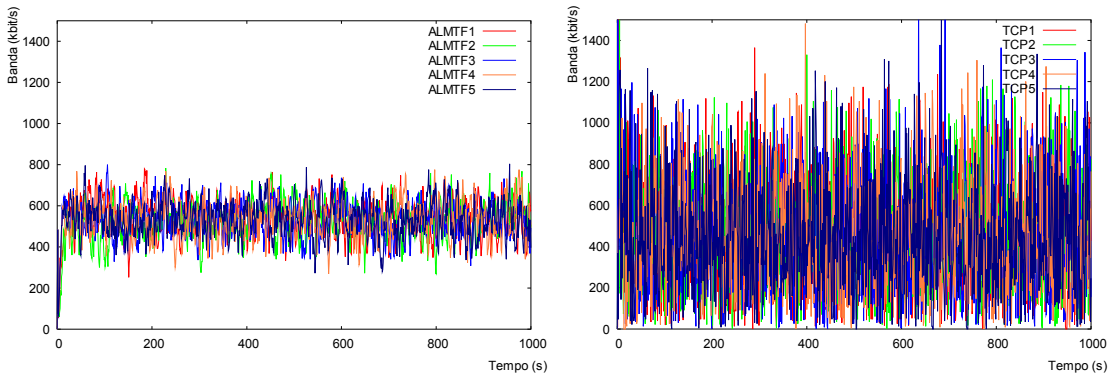


FIGURA 6.40 – Eqüidade do ALMTF para 5 fluxos e camadas de 100 kbit/s

Adicionando cinco fluxos TCP pode-se ver o resultado na figura 6.41, onde o gráfico "a" mostra os fluxos ALMTF e o gráfico "b" mostra os fluxos TCP. Os gráficos foram separados para uma melhor visualização.



a) Fluxos ALMTF b) Fluxos TCP

FIGURA 6.41 – 10 fluxos, sendo 5 ALMTF com camadas de 100kbit/s e 5 TCP

Os fluxos ALMTF são mais estáveis que os fluxos TCP, como pode ser visto na figura 6.41a, onde a variação de banda foi praticamente entre 400 e 600 kbit/s. Na figura 6.41b percebe-se um aumento considerável na variação de uso de banda dos fluxos TCP, que variaram entre 0 e 1.400 kbit/s. A vazão dos cinco fluxos ALMTF (gráfico "a") foi: mínima = 525 kbit/s; máxima = 538 kbit/s; média = 531,6 kbit/s. A vazão para os cinco fluxos TCP (gráfico "b") foi de: mínima = 407 kbit/s; máxima = 423 kbit/s; média = 415,4 kbit/s. Nessa simulação, o ALMTF foi mais agressivo que o TCP.

A simulação ilustrada na figura 6.42 mostra a capacidade do algoritmo reduzir sua banda para dividir eqüitativamente o tráfego após estar em regime permanente. O enlace central possui 1 Mbit/s, portanto, o fluxo ALMTF1 já estava estabilizado na camada 5, ou seja, utilizando 930 kbit/s, quando entrou o fluxo ALMTF2, no instante 100s. Com a entrada do segundo fluxo, ocorreram perdas e os dois fluxos adaptaram-se rapidamente na camada de número 4, ou seja, cada um utilizando 450 kbit/s do total de 1 Mbit/s.

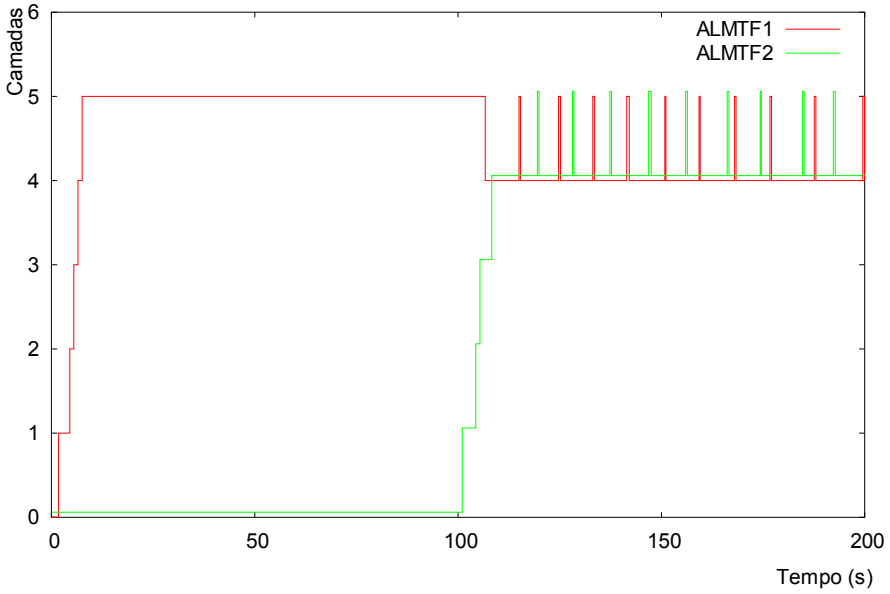


FIGURA 6.42 – Eqüidade do ALMTF para dois fluxos, um iniciando 100s após o outro

O resultado visto na figura 6.42 mostra que o algoritmo ALMTF se adapta, dividindo eqüitativamente a banda, mesmo quando já se encontra em regime permanente e entram outros fluxos na rede.

Com cinco fluxos o algoritmo também se adapta atinge a eqüidade de tráfego, como mostra a figura 6.43, onde cada fluxo inicia 100s após o outro, ou seja:

- ALMTF1: início no instante 1s;
- ALMTF2: início no instante 100s;
- ALMTF3: início no instante 200s;
- ALMTF4: início no instante 300s;
- ALMTF5: início no instante 400s.

Pode-se perceber a convergência do algoritmo na direção de uma divisão eqüitativa de banda, que é alcançada na camada 4 (total de 450 kbit/s), pois o total de

banda no enlace central é de 2,5 Mbit/s.

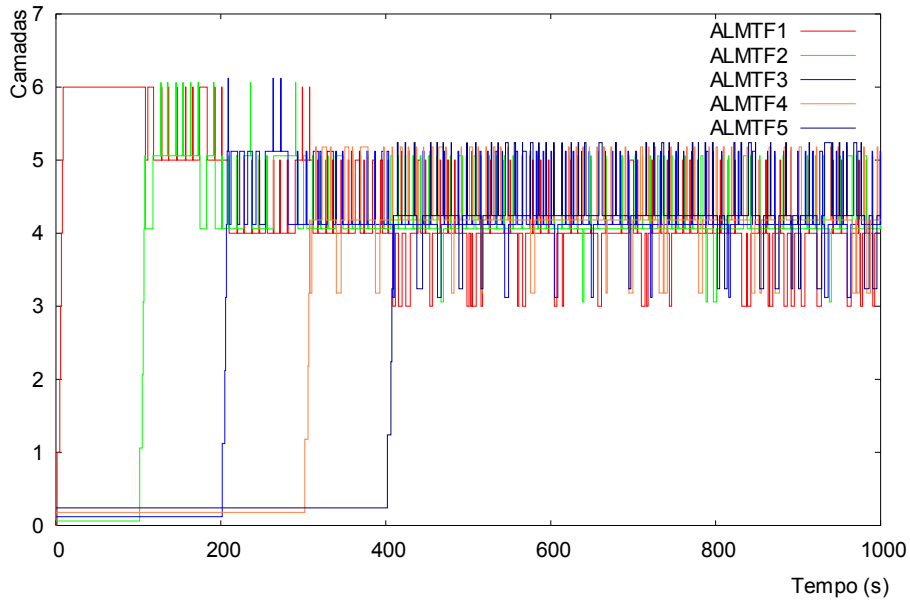
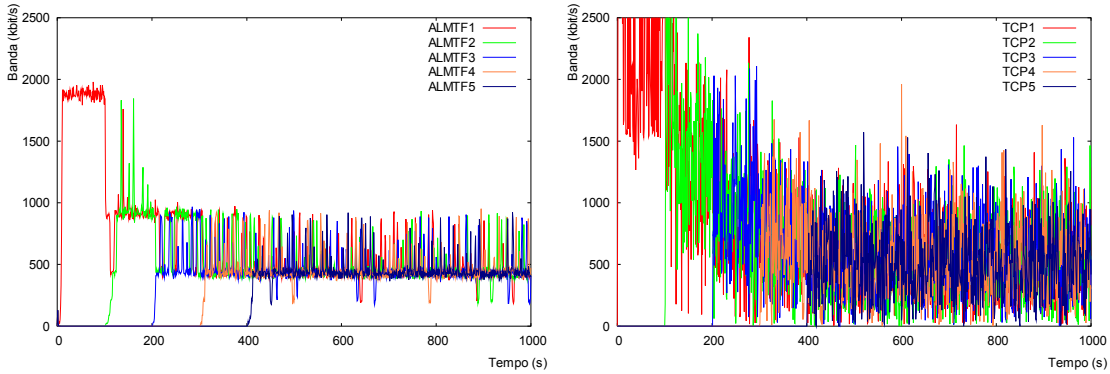


FIGURA 6.43 – Eqüidade para cinco fluxos, cada um iniciando 100s após o outro

A figura 6.44 mostra uma simulação com um total de 10 fluxos concorrentes: 5 ALMTF com camadas exponenciais (gráfico “a”) e 5 TCP (gráfico “b”), onde os fluxos de cada algoritmo iniciam 100s após o outro, ou seja:

- ALMTF1 e TCP1: início no instante 1s;
- ALMTF2 e TCP2: início no instante 100s;
- ALMTF3 e TCP3: início no instante 200s;
- ALMTF4 e TCP4: início no instante 300s;
- ALMTF5 e TCP5: início no instante 400s.



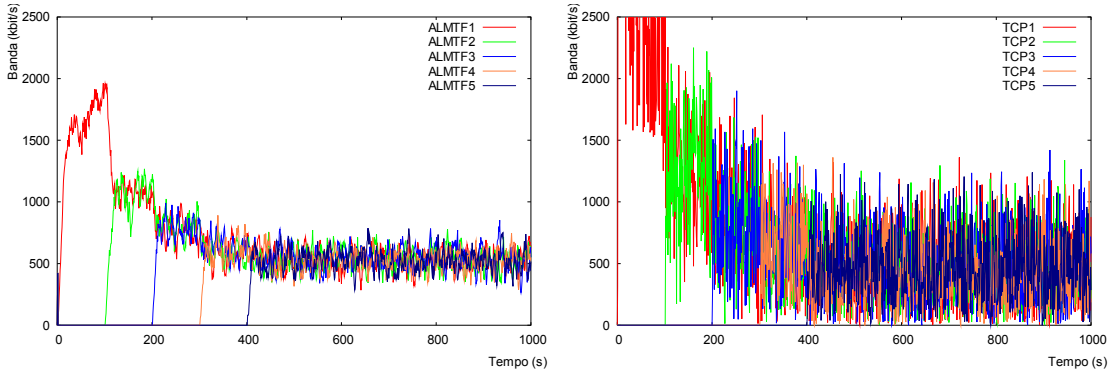
a) Fluxos ALMTF b) Fluxos TCP

FIGURA 6.44 – 10 fluxos (5 ALMTF, e 5 TCP) iniciando a cada 100s

A largura de banda no enlace central é de 10 Mbit/s, perfazendo um total de 500 kbit/s para cada fluxo. Novamente percebe-se que os fluxos ALMTF dividem a banda de forma aproximadamente eqüitativa com os fluxos TCP, porém com maior estabilidade. Os fluxos ALMTF tiveram a seguinte vazão, respectivamente: 604 kbit/s; 459 kbit/s; 342 kbit/s; 315 kbit/s e 271 kbit/s. Os fluxos TCP mostraram uma vazão

superior, pois a adaptação no ALMTF é em 450 kbit/s e não em 500 kbit/s. A vazão dos receptores TCP foi de: 857 kbit/s; 610 kbit/s; 455 kbit/s, 316 kbit/s e 267 kbit/s.

A figura 6.45 mostra a mesma simulação, porém com camadas de 100 kbit/s, cada fluxo iniciando 100s após o anterior, da mesma forma que a simulação anterior. Pode-se observar que os fluxos ALMTF (gráfico "a") dividem a banda de forma aproximadamente eqüitativa com os fluxos TCP (gráfico "b"). Os fluxos ALMTF tiveram a seguinte vazão, respectivamente: 696 kbit/s; 551 kbit/s; 420 kbit/s; 380 kbit/s e 313 kbit/s. Os fluxos TCP mostraram uma vazão semelhante: 821 kbit/s; 505 kbit/s; 368 kbit/s, 317 kbit/s e 266 kbit/s.



a) Fluxos ALMTF b) Fluxos TCP

FIGURA 6.45 – 10 fluxos (5 ALMTF, camadas 100k, e 5 TCP), iniciando a cada 100s

Em relação ao compartilhamento de banda com tráfego CBR, o algoritmo efetuou uma divisão eqüitativa da banda, como mostra a figura 6.46, onde um fluxo CBR de 500 kbit/s foi transmitido juntamente com dois fluxos ALMTF num enlace de 1 Mbit/s. O fluxo CBR não se adapta, forçando os dois fluxos ALMTF a dividirem o restante da banda, ou seja, 500 kbit/s (equivalente a 3 camadas, ou 210 kbit/s cada um). Quando o fluxo CBR termina, no instante 200s, os fluxos ALMTF se adaptam e utilizam a banda total do enlace principal, ou seja, 1 Mbit/s (equivalente a 4 camadas, ou 450 kbit/s por fluxo). No instante 600s inicia um segundo fluxo CBR, forçando novamente os dois fluxos ALMTF a adaptarem-se, voltando a dividir 500 kbit/s, ou 3 camadas.

Foram feitas outras simulações, conforme descrição no item 4.8 (Definição das simulações), entretanto, elas não geraram alterações significativas nos resultados, e somente serão comentadas brevemente a seguir, entretanto, estão disponíveis em http://www.inf.unisinos.br/~roesler/tese e no CD anexo a esta Tese.

- **Pacote com tamanho 250 bytes ou 1000 bytes ao invés de 500 bytes**: O resultado foi muito semelhante ao visto nas figuras anteriores;
- **Atraso de 1ms no enlace principal**: O resultado mostrou que o sistema possui semelhança ao protocolo TCP, pois com RTT menor, o algoritmo é executado mais vezes, ficando mais agressivo e aumentando o número de tentativas de subir camada, incrementando o número de perdas;
- **Windowsize de 60 no ALMTF e no TCP**: O resultado foi muito semelhante ao visto nas figuras anteriores.

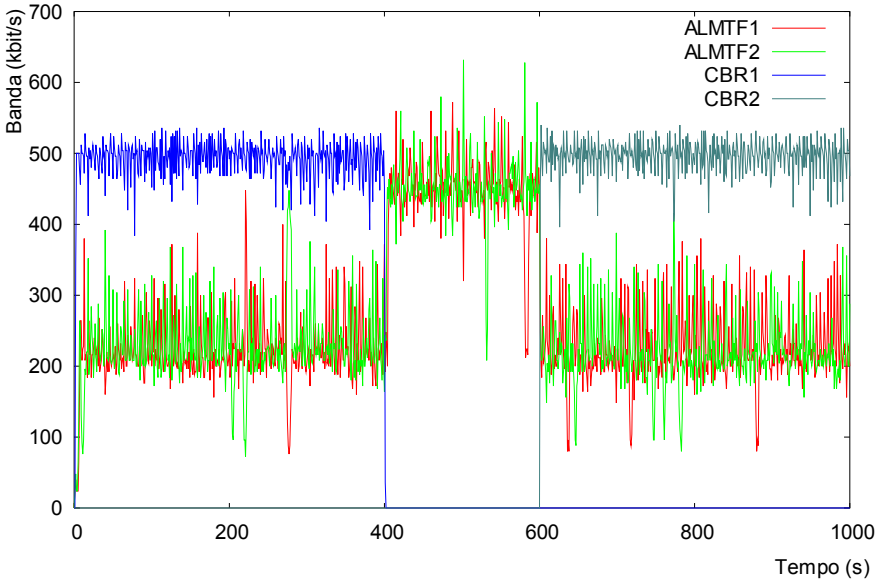


FIGURA 6.46 – Eqüidade de tráfego do ALMTF com CBR

6.2.4 Análise da estabilidade do ALMTF

A estabilidade do algoritmo define a quantidade de variações de camada que aconteceram por minuto após a fase inicial (*start-state*), conforme definição no item 4.8.4. Assim, pode-se construir uma tabela com os principais resultados de estabilidade obtidos nas simulações anteriores do ALMTF. Vale lembrar que, quando um algoritmo sobe de camada, gera perdas e desce de camada, conta como duas variações.

A tabela 6.2 mostra o número de variações de camada por minuto (VCM) de várias simulações anteriores. A primeira coluna fornece uma descrição sucinta da simulação, enquanto a segunda coluna associa a figura referente à simulação em questão. A terceira coluna mostra o total de variações por minuto dos fluxos participantes. No caso onde existem vários receptores, é fornecido o pior resultado, o melhor resultado e a média. Para o cálculo do VCM, utilizou-se o programa em linguagem *perl vcm.pl*, descrito no item 4.8.4, que fornece o VCM do fluxo que foi especificado no parâmetro de entrada. Em linhas gerais, o programa calcula todas as trocas de camada do fluxo em regime permanente e divide pelo total de minutos da simulação (no caso, 16,66, pois as simulações são de 1000 segundos). As seguintes conclusões podem ser obtidas a partir da tabela 6.2 e dos gráficos associados:

- Fluxos com menos banda possuem, geralmente, um VCM mais alto, e isso se deve ao maior número de tentativas de subir camada, pois a variável *sincjoin* permite um maior número de tentativas de *join* para os receptores inscritos nas camadas mais baixas. Isso pode ser constatado pela primeira linha da tabela, referente à figura 6.12, onde o fluxo ALMTF1 ficou constante, sem variações, enquanto o fluxo ALMTF4 sofreu 21 variações de camada por minuto;
- A utilização de pares de pacotes impede que o algoritmo tente subir camadas além da sua banda máxima, provocando uma maior estabilidade ao sistema (visto através de todas figuras que tem as duas simulações, sem e com pares de pacotes);
- Quanto mais próximas estiverem as camadas, menor a estabilidade, pois o algoritmo

vai efetuar tentativas de subir camada de forma mais freqüente. Isso pode ser visto pela comparação entre as simulações com camadas exponenciais e de 100k;

- A variável *sincjoin* sincroniza os receptores de forma que eles façam tentativas de subir camada de forma sincronizada. Isso pode ser visto através do gráfico ilustrado na figura 6.28, por exemplo;
- A estabilidade do ALMTF é bem maior do que a de fluxos TCP equivalentes, conforme pode ser observado nas figuras 6.35, 6.37 e 6.41;
- A estabilidade do ALMTF diminui com o aumento de receptores, conforme pode ser observado nas linhas da tabela referentes à escalabilidade, quando comparadas com apenas um receptor por bloco.

TABELA 6.2 – Estabilidade do ALMTF – variações de camada por minuto (VCM)

Descrição (adaptabilidade)	**Figura**	**VCM** **ALM1, ALM2, ALM3, ALM4**
Camadas exponenciais sem PP[1] (atraso 10ms)	Figura 6.12	0,0 2,0 13,9 21,6
Camadas exponenciais com PP	Figura 6.14	0,0 0,0 0,0 0,0
Camadas exponenciais sem PP (atraso 1ms)	Figura 6.16	0,0 13,9 15,2 22,5
Camadas VEBIT sem PP	Figura 6.17	0,0 0,0 12,9 21,0
Camadas VEBIT com PP	Figura 6.19	0,0 0,0 0,0 0,0
Camadas 100 kbit/s sem PP	Figura 6.22	8,0 34,0 39,7 11,2
Camadas 100 kbit/s com PP	Figura 6.22	0,0 0,0 0,0 0,0
Camadas exponenciais fila RED sem PP	Figura 6.24	10,0 4,19 16,9 25,0
Camadas exponenciais fila RED com PP	Figura 6.24	0,0 0,0 0,0 0,0
Descrição (escalabilidade)	**Figura**	**VCM: Bloco1 a Bloco4. (Pior, melhor e média)**
Camadas exponenciais (100 receptores) sem PP	Figura 6.28	B1: 0,0 0,0 0,0 B2: 2,2 0,8 1,5 B3: 13,6 13,4 13,5 B4: 20,68 19,28 20,48
Camadas exponenciais (100 receptores) com PP	Figura 6.30	VCM = 0 em todos casos
Camadas de 100k (100 receptores) sem PP	Figura 6.31	B1: 11,9 1,1 9,5 B2: 37,0 35,1 36,2 B3: 40,3 37,6 39,5 B4: 7,9 0,0 1,2
Camadas de 100k (100 receptores) com PP	Figura 6.31	VCM = 0 em todos casos
Descrição (eqüidade)	**Figura**	**VCM (Pior, melhor e média)**
Eqüidade com 2 ALMTF – camadas exponenciais	Figura 6.34	12,0 11,1 11,5
Eqüidade com 4 fluxos: - 2 ALMTF - 2 TCP	Figura 6.35	3,4 3,2 3,3 48,5 44, 6 46,5
Eqüidade com 10 ALMTF – camadas expon.	Figura 6.36	12,4 10,4 11,5
Eqüidade com 20 fluxos: - 10 ALMTF - 10 TCP	Figura 6.37	10, 6 9,4 10,0 69,0 63,0 66,0
Eqüidade com 5 fluxos – camadas de 100k	Figura 6.40	39,2 37,0 38,0
Eqüidade com 10 fluxos (100k): - 5 ALMTF - 5 TCP	Figura 6.41	37,7 35,9 36,6 169,0 161,0 163,0

[1] Pares de pacotes

6.2.5 Análise do tempo de adaptação do ALMTF

O algoritmo do ALMTF é dividido em duas fases: *start-state* e *steady-state*. A fase *start-state* é utilizada para tentar fazer o algoritmo adaptar-se rapidamente à sua banda eqüitativa, conforme detalhado anteriormente, e pode ser observada claramente na figura 6.12, cujos primeiros 15s de simulação foram reproduzidos na figura 6.47 a fim de detalhar o processo de adaptação.

Observa-se, através da figura 6.47, que todos os receptores ingressaram na sessão aproximadamente no instante 5s (de forma randômica, conforme explicado no item 4.7). Seis segundos após a entrada, todos os receptores tinham saído da fase *start-state*, atingindo o regime permanente. Os receptores que se adaptam nas menores camadas chegam ao regime permanente de forma mais rápida, pois detectam perdas antes. O tempo mínimo para subir uma camada é um segundo, que é o tempo de estabilização estabelecido no algoritmo, conforme detalhado no início do capítulo.

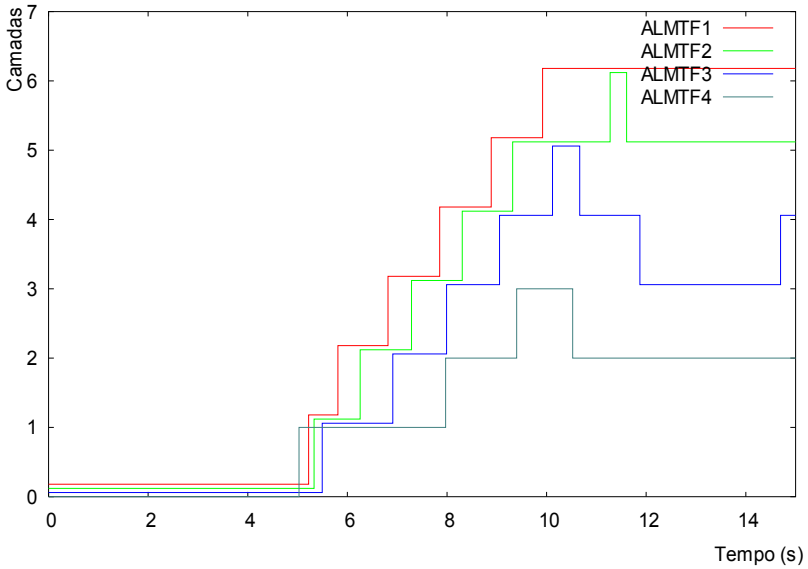


FIGURA 6.47 – Fase *start-state* do ALMTF com camadas exponenciais padrão

O algoritmo possui uma proteção para que o receptor suba no máximo uma camada por vez, da mesma forma que o ALMP, portanto, normalmente quanto maior o número de camadas que o receptor deve se inscrever, maior o tempo de adaptação. Isso é ilustrado na figura 6.48 para camadas de 50 kbit/s com pares de pacotes, onde o algoritmo levou aproximadamente 40s para subir 40 camadas. Isso pode ser observado através do receptor ALMTF1 da figura 6.48.

A fase *start-state* tem por objetivo levar o receptor a utilizar uma banda próxima à eqüitativa, e caso necessário, ele deve atingir o restante da adaptação através da fase *steady-state*, que possui uma taxa de subida e descida dez vezes mais lenta que o TCP, conforme detalhado no início do capítulo.

Apesar do ALMTF possuir uma taxa de subida e descida mais lenta que o TCP, ainda se obtém uma rápida taxa de adaptação em regime permanente, bem superior se comparado ao ALMP.

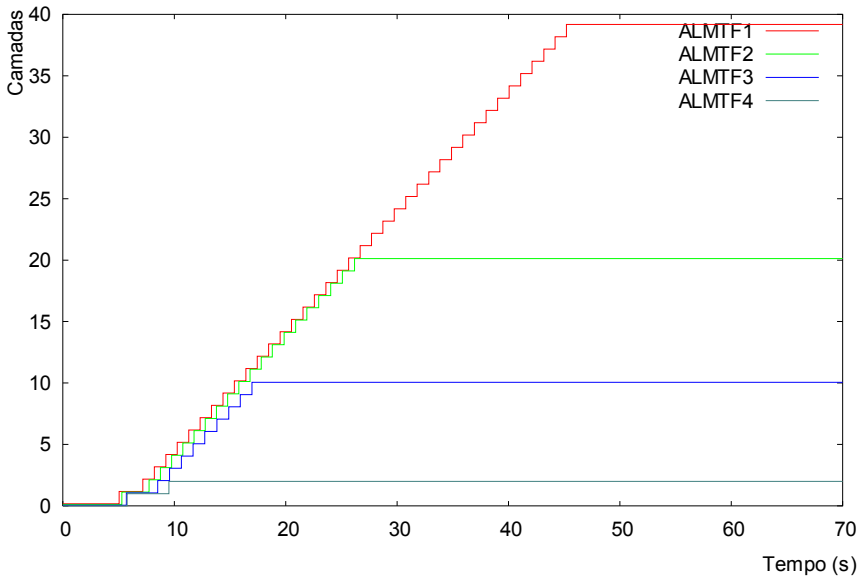


FIGURA 6.48 – Fase *start-state* do ALMTF com camadas de 50 kbit/s com PP

6.2.6 Análise da taxa de perdas do ALMTF

A saída de todas as simulações mostra, entre outras informações, o total de pacotes recebidos e a taxa de perdas, conforme foi descrito no item 4.7.5 (Metodologia para análise de resultados no simulador). Os parágrafos a seguir vão analisar alguns gráficos com base na taxa de perdas obtida.

A tabela 6.3 mostra um resumo da taxa de perdas para as principais simulações efetuadas. A primeira coluna fornece uma descrição sucinta da simulação, enquanto a segunda coluna associa a figura referente à simulação em questão. A terceira coluna mostra a taxa de perdas dos fluxos participantes. No caso onde existem vários receptores, é fornecido o pior resultado, a média e o melhor resultado.

As seguintes conclusões podem ser obtidas a partir da tabela 6.3 e os gráficos associados:

- O número de perdas com pares de pacotes tende a ser menor, devido ao menor número de tentativas de subida de camada quando o limite da banda é atingido (visto através das figuras 6.12 e 6.14, por exemplo);
- Quanto menor a banda, maior o número de pontos de sincronização permitindo que seja feita tentativa de *join*, portanto, maior a taxa de perdas (visto através das figuras 6.12 e 6.17);
- Da mesma forma que no protocolo TCP, os fluxos localizados em enlaces mais rápidos, ou com menor RTT, tem seus algoritmos executados mais vezes, sendo mais agressivos e gerando mais perdas (visto através da figura 6.16);
- A taxa de perdas do ALMTF é superior à taxa de perdas de fluxos TCP concorrentes, pois o TCP pára de transmitir assim que sente perda, tendo um caráter de descida multiplicativo (AIMD), provocando uma grande variabilidade de fluxo, porém, menor número de perdas. O ALMTF decrementa somente 5% da banda do fluxo, gerando essa conseqüência.

TABELA 6.3 – Taxa de perdas para simulações do ALMTF

Descrição (adaptabilidade)	Figura	Taxa de perdas: ALMTF1 a ALMTF4
Camadas exponenciais sem PP (atraso 10ms)	Figura 6.12	0,00%; 0,72%; 1,18%; 3,33%
Camadas exponenciais com PP	Figura 6.14	0,00%; 0,00%; 0,00%; 0,00%
Camadas exponenciais sem PP (atraso 1ms)	Figura 6.16	0,00%; 1,33%; 2,12%; 4,20%
Camadas VEBIT sem PP	Figura 6.17	0,00%; 0,00%; 0,70%; 1,20%
Camadas VEBIT com PP	Figura 6.19	0,00%; 0,00%; 0,00%; 0,00%
Camadas 100 kbit/s sem PP	Figura 6.22	0,10%; 0,25%; 0,47%; 2,20%
Camadas 100 kbit/s com PP	Figura 6.22	0,02%; 0,03%; 0,03%; 0,02%
Camadas exponenciais fila RED sem PP	Figura 6.24	0,00%; 0,50%; 0,60%; 1,10%
Camadas exponenciais fila RED com PP	Figura 6.24	0,00%; 0,00%; 0,00%; 0,00%
Descrição (escalabilidade)	**Figura**	**Perdas: Bloco1 a Bloco4. (Pior, melhor e média)**
Camadas exponenciais (100 receptores) sem PP	Figura 6.28	B1: 0,00% 0,00% 0,00% B2: 0,70% 0,54% 0,62% B3: 1,50% 1,43% 1,47% B4: 3,80% 3,60% 3,70%
Camadas exponenciais (100 receptores) com PP	Figura 6.30	VCM = 0 em todos os casos
Camadas cont. 100k (100 receptores) sem PP	Figura 6.31	B1: 0,18% 0,17% 0,17% B2: 0,31% 0,30% 0,30% B3: 0,70% 0,65% 0,68% B4: 2,11% 2,04% 2,06%
Camadas cont. 100k (100 receptores) com PP	Figura 6.31	VCM = 0 em todos os casos
Descrição (eqüidade)	**Figura**	**Perdas (Pior, mehor e média)**
Eqüidade com 2 ALMTF – camadas exp.	Figura 6.34	1,99% 1,92% 1,95%
Eqüidade com 4 fluxos: - 2 ALMTF - 2 TCP	Figura 6.35	3,72% 3,71% 3,71% 1,06% 1,06% 1,06%
Eqüidade com 10 ALMTF – camadas exp.	Figura 6.36	11,9% 10,4% 11,3%
Eqüidade com 20 fluxos: - 10 ALMTF - 10 TCP	Figura 6.37	6,62% 6,40% 6,65% 1,70% 1,60% 1,68%
Eqüidade com 5 fluxos – camadas cont. 100k	Figura 6.40	4,45 4,38 4,41
Eqüidade com 10 fluxos (100k): - 5 ALMTF - 5 TCP	Figura 6.41	7,23% 7,06% 7,10% 1,78% 1,70% 1,74%

6.3 Conclusões sobre o ALMTF

O algoritmo ALMTF é destinado à transmissão de fluxos multimídia em multicast através da Internet. O diferencial em relação ao algoritmo ALMP é que o ALMTF transmite de forma eqüitativa com fluxos TCP concorrentes.

Para transmitir de forma eqüitativa com fluxos TCP concorrentes, é necessário seguir o modelo de incremento e decremento de banda utilizado pelo TCP, entretanto, com suavizações onde possível a fim de melhorar a estabilidade do algoritmo. Para tanto, o ALMTF utiliza dois mecanismos de controle de fluxo:

- O primeiro é baseado em “janela de congestionamento”, que segue o modelo do

TCP, porém diminui em dez vezes sua agressividade;
- O segundo é baseado na equação do TCP, que procura inferir o comportamento dos fluxos TCP através de uma equação.

Em relação à adaptação em redes heterogêneas, foi visto que o algoritmo se adapta de forma coerente com a topologia existente, entretanto, os receptores que utilizam menos banda, se adaptando em menos camadas, possuem uma maior taxa de tentativas de subir camada (devido à variável *sincjoin*) e recebem uma característica de tráfego menos estável quando comparada aos receptores localizados em redes mais rápidas.

A utilização de pares de pacotes mostrou aumentar a estabilidade do algoritmo, entretanto, seu uso é particularmente útil quando a inscrição na próxima camada gera uma banda superior à do enlace. Nos casos onde há concorrência entre vários fluxos ALMP diferentes, percebe-se pouca diferença em relação à não utilização dos pares de pacotes.

Em relação à escalabilidade, os resultados das simulações para até 100 receptores, vistos através das figuras 6.26 a 6.32, mostram que o algoritmo suporta um aumento de receptores com leve diminuição de estabilidade, devido principalmente ao aumento no número de tentativas de subir camada, que é minimizado com a utilização do sincronismo via variável *sincjoin*. O impacto das mensagens necessárias para cálculo de RTT também se mostrou desprezível em relação ao tráfego gerado pelos pacotes multimídia sendo transmitidos.

Em termos de eqüidade com o próprio tráfego, as figuras 6.34 e 6.36, entre outras, mostraram que o algoritmo funciona de forma eqüitativa, e os receptores utilizam um tráfego muito semelhante.

Em relação à concorrência com tráfego TCP, as figuras 6.35 e 6.37, entre outras, ilustraram a capacidade de compartilhamento de banda do algoritmo, que se mostrou bastante eficiente.

Um fator que sempre deve ser levado em consideração na análise de eqüidade de um algoritmo é sua capacidade de adaptar-se após o algoritmo ter atingido a fase de regime permanente. As figuras 6.42 a 6.45, mostram que o ALMTF se adapta corretamente nesse caso, tanto quando entram novos fluxos do próprio algoritmo como quando entram novos fluxos TCP.

Com base em todas as simulações realizadas, conclui-se que o ALMTF cumpriu seus objetivos, que são a adaptação em ambientes heterogêneos, permitindo escalabilidade e transmissão de forma eqüitativa com seu próprio tráfego e com tráfego TCP concorrente.

7 Comparações com outros algoritmos

Este capítulo tem como objetivo comparar os algoritmos desenvolvidos para esta Tese com outros trabalhos relacionados à transmissão multicast com adaptação no receptor. Para fins de comparação com o ALMP e o ALMTF, escolheu-se os seguintes algoritmos:

- RLM (*Receiver-Driven Layered Multicast* – item 3.1 [MCC 96]), por ele utilizar transmissão em camadas com multicast e adaptação no receptor, ou seja, possui a mesma filosofia do ALMP e do ALMTF. Além disso, ele é o algoritmo que serve como base de comparação da maioria dos outros algoritmos, por ter sido o primeiro desenvolvido para esse fim;
- RLC (*Receiver-Driven Layered Congestion Control* – item 3.2 [VIC 98]), por ele também utilizar transmissão em camadas com multicast e adaptação no receptor. Além disso, tem como objetivo a eqüidade de tráfego com o TCP. No RLC, foi necessária a utilização da versão *2.1b3* do NS2, pois o algoritmo disponibilizado só funcionava nela;
- TFMCC (*TCP-Friendly Multicast Congestion Control* – item 3.9 [WID 2001b]), por utilizar a equação do TCP como método de adaptação. O algoritmo TFMCC não trabalha com transmissão em camadas, se adaptando ao receptor mais lento entre todos os receptores que estão assistindo a transmissão, entretanto, ganha importância por ser o único algoritmo multicast baseado na equação do TCP disponível para inclusão no NS2. Além disso, todas as simulações da topologia 2 da figura 4.3 (voltada para eqüidade de tráfego) trabalham com um transmissor e um receptor, tornando o algoritmo TFMCC comparável ao ALM, pois quando existe somente um receptor, este é o mais lento.

Uma descrição mais detalhada de cada um dos protocolos comparados pode ser encontrada no capítulo 3 (Trabalhos Relacionados), onde foram analisados também outros algoritmos relacionados, tal como mostra o diagrama da figura 3.1, que é um resumo de todos os trabalhos relacionados detalhados nesta Tese.

O algoritmo do PLM (descrito no item 3.5) e o baseado em redes ativas (item 3.6) não serão utilizados como critério de comparação, pois necessitam de roteadores com características especiais, como suporte a fila WF2Q ou suporte a redes ativas, e portanto não são comparáveis ao ALMP nem ao ALMTF, que utilizam políticas existentes atualmente nos roteadores centrais.

Outros algoritmos são baseados em unicast, e outros não foram implementados no NS2, dificultando a comparação. Além disso, alguns algoritmos não possuem o código-fonte para o NS2 disponível. Entretanto, julgou-se que os três algoritmos escolhidos para fins de comparação são suficientes para dar uma visão geral entre as diferenças dos algoritmos desenvolvidos nesta Tese e outros algoritmos relacionados.

Os algoritmos comparados foram inseridos no código do simulador NS2, e as simulações utilizadas foram um subconjunto das simulações definidas no item 4.8 (Definição das simulações). Assim, as mesmas topologias e parâmetros utilizados nos

algoritmos desta Tese foram aplicados aos outros algoritmos.

Os próximos itens comparam o RLM, o RLC e o TFMCC ao ALMP e ao ALMTF, propostos nesta Tese. Os algoritmos foram comparados em termos de adaptabilidade em ambientes heterogêneos, eqüidade com o próprio tráfego, eqüidade com tráfego TCP concorrente, velocidade de adaptação na fase inicial, velocidade de adaptação em regime permanente e estabilidade.

Para comparar os algoritmos em termos de estabilidade, o critério utilizado é o valor de VCM, ou Variações de Camada por Minuto. Para isso utilizou-se dois programas descritos no item 4.7.5, um implementado em linguagem *perl* (*vcm.pl*) e outro em linguagem C (*vcmtcp.exe*). O primeiro analisa as variações de camadas dos fluxos, e o segundo analisa o equivalente às variações de camadas para um fluxo compartilhando o enlace, como o TCP, por exemplo.

Foram efetuadas diversas simulações além das incluídas no volume desta Tese. A relação completa está acessível em http://www.inf.unisinos.br/~roesler/tese e no CD anexo a esta Tese.

O item 7.1 compara os algoritmos em termos de adaptação em ambientes heterogêneos e estabilidade. No item 7.2 são efetuadas comparações de escalabilidade até 100 receptores em ambientes heterogêneos. O item 7.3 compara o tempo de adaptação inicial dos algoritmos, ou seja, o tempo que ele demora até detectar perdas e passar à fase de regime permanente. No item 7.4 é feita uma comparação do tempo de adaptação em regime permanente, bem como da estabilidade e eqüidade com o próprio tráfego. O item 7.5 compara os algoritmos em termos de estabilidade e eqüidade com tráfego TCP concorrente, onde o fluxo do algoritmo inicia antes e depois do TCP. Finalmente, o item 7.7 traça algumas conclusões sobre as comparações efetuadas, apresentando uma tabela-resumo dos resultados obtidos.

7.1 Comparações dos algoritmos em termos de adaptação em ambientes heterogêneos e estabilidade

Para comparar os algoritmos em termos de adaptação em ambientes heterogêneos, foi utilizada como base a topologia 1 da figura 4.3, que é repetida na figura 7.1.

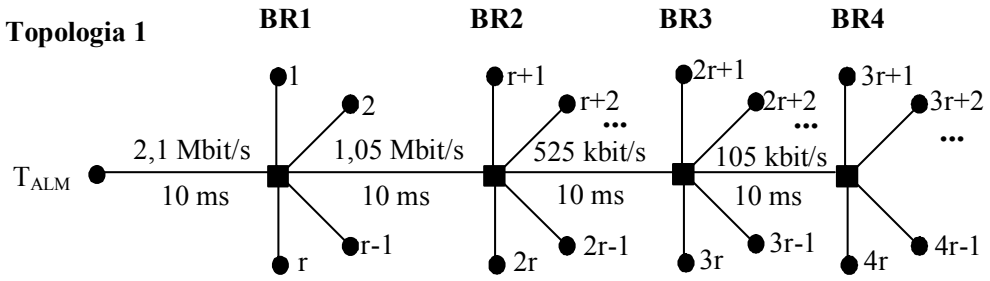
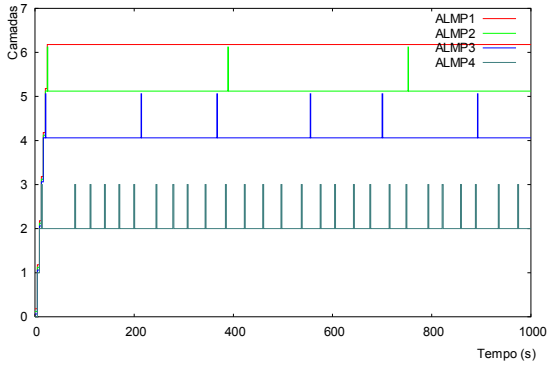


FIGURA 7.1 – Topologia para as comparações

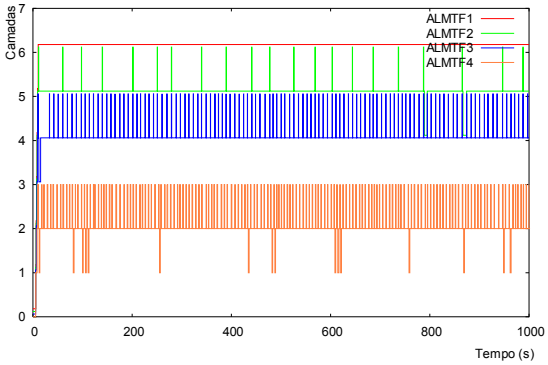
Os parâmetros configurados para a primeira simulação estão descritos a seguir, e serão utilizados como base para as outras simulações, que vão modificar alguns desses parâmetros:

- Número de receptores por bloco: r = 1, totalizando 4 receptores;
- Camadas exponenciais padrão (30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s e 960 kbit/s). No caso do TFMCC, esse dado não se aplica;
- Tamanho do pacote = 500 bytes;
- Tipo de fila = *droptail* com 20 pacotes;
- Randomização alta (±0,5s);
- Sem uso de pares de pacotes (válido para o ALMP e ALMTF).

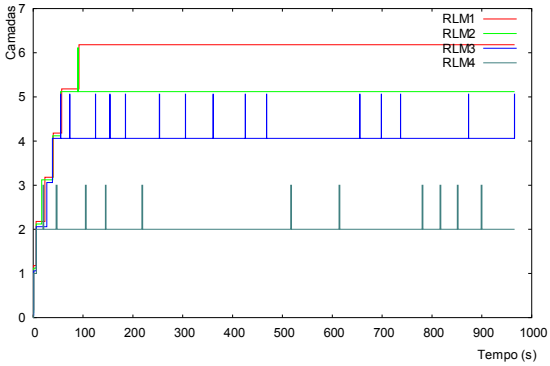
A figura 7.2 mostra os resultados das simulações para todos os algoritmos comparados. O gráfico "a" refere-se ao ALMP, o "b" ao ALMTF, o "c" ao RLM, o "d" ao RLC e o "e" ao TFMCC, como também mostram as legendas dos gráficos.



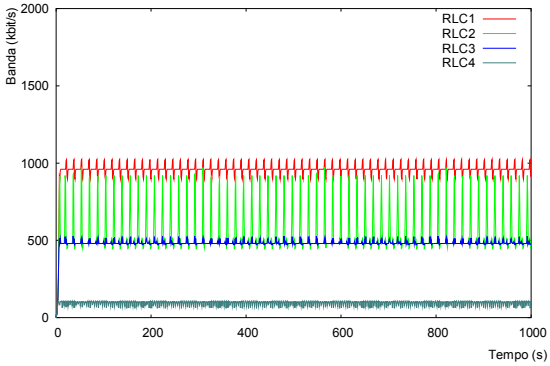
a) algoritmo ALMP



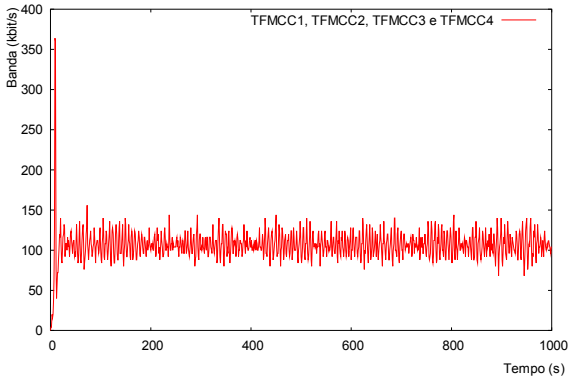
b) algoritmo ALMTF



c) algoritmo RLM



d) algoritmo RLC



e) algoritmo TFMCC

FIGURA 7.2 – Adaptação dos algoritmos em redes heterogêneas

Para facilitar a análise da simulação apresentada na figura 7.2, foi criada a tabela 7.1, que compara os algoritmos em termos de estabilidade, mostrando o valor de VCM para cada um dos quatro receptores.

TABELA 7.1 – Estabilidade dos algoritmos para a simulação de adaptabilidade

Algoritmo	VCM			
	Rec1	Rec2	Rec3	Rec4
ALMP	0,00	0,24	0,60	3,12
ALMTF	0,00	2,00	13,9	21,6
RLM	0,00	0,00	1,80	0,90
RLC	0,00	7,10	0,00	0,00
TFMCC	0,00	0,00	0,00	0,00

A partir da análise da figura 7.2, percebe-se que o ALMP (gráfico “a”), o ALMTF (gráfico “b”) e o RLM (gráfico “c”) adaptaram-se ao correto número de camadas, permitindo que receptores localizados em enlaces de diferentes larguras de banda obtenham uma qualidade compatível com a sua possibilidade de tráfego. Os fluxos ALMP1, ALMTF1 e RLM1 adaptaram-se em 6 camadas (demandando 1.890 kbit/s), que é compatível com o enlace de 2.100 kbit/s. Os fluxos ALMP2, ALMTF2 e RLM2 adaptaram-se em 5 camadas (demandando 930 kbit/s), que é compatível com o enlace de 1.050 kbit/s. Os fluxos ALMP3, ALMTF3 e RLM3 adaptaram-se em 4 camadas (demandando 450 kbit/s), que é compatível com o enlace de 525 kbit/s. Os fluxos ALMP4, ALMTF4 e RLM4 adaptaram-se em 2 camadas (demandando 90 kbit/s), que é compatível com o enlace de 100 kbit/s.

O RLC (gráfico “d”) teve problemas de adaptação nas camadas mais altas. O fluxo RLC1, que deveria ter estabilizado em 1.890 kbit/s, ficou próximo a 1.000 kbit/s, e o fluxo RLC2, que deveria ter estabilizado em 930 kbit/s, ficou variando entre a quarta e quinta camadas. Já os fluxos RLC3 e RLC4 adaptaram-se ao correto número de camadas.

O transmissor do TFMCC (gráfico “e”) adaptou-se corretamente segundo sua filosofia, ou seja, transmitiu um único fluxo multicast de aproximadamente 100 kbit/s, compatível com a largura de banda disponível no enlace mais lento da topologia. Esse fluxo multicast foi destinado a todos os receptores.

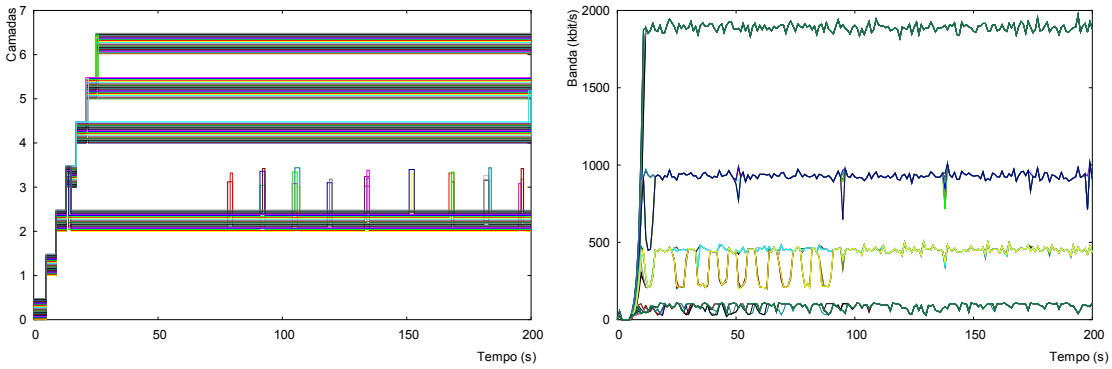
Em termos de estabilidade, pode-se observar, a partir da tabela 7.1, o que já foi visualizado no gráfico, ou seja, TFMCC teve VCM de zero, pois seu fluxo único de 100 kbit/s é bastante estável. Dos outros algoritmos, o RLM teve o menor VCM, seguido do ALMP, do RLC e, por último, o ALMTF. O ALMTF necessita ser mais agressivo devido à sua característica de compartilhamento de banda com o TCP.

Observa-se que todos os receptores de todos os algoritmos adaptaram-se nas camadas corretas, de acordo com a banda passante, entretanto, o ALMP e o RLM tiveram o menor número de tentativas para subir de camada. O ALMTF teve o maior número de tentativas, e isso é devido à necessidade de gerar tráfego de forma competitiva com o TCP.

7.2 Comparações de escalabilidade dos algoritmos

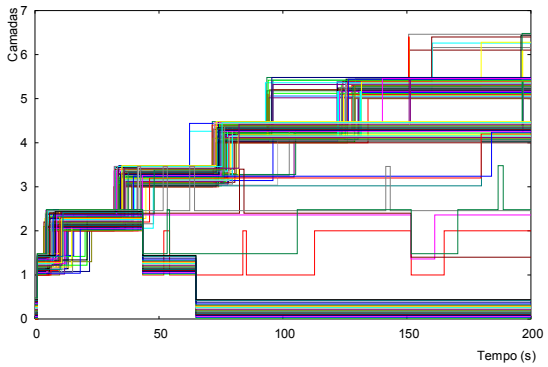
Este item tem por objetivo comparar os algoritmos em termos de escalabilidade. Para tanto, será utilizada a mesma topologia da simulação anterior, porém com r = 25, ou seja, 100 receptores.

A figura 7.3 ilustra o comportamento de cada algoritmo. O ALMP e o ALMTF foram simulados sem pares de pacotes a fim de permitir uma comparação em qualquer tipo de topologia, independente se atinge o limite da banda ou não.

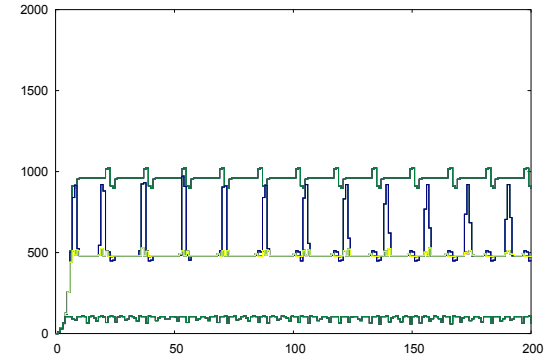


a) algoritmo ALMP

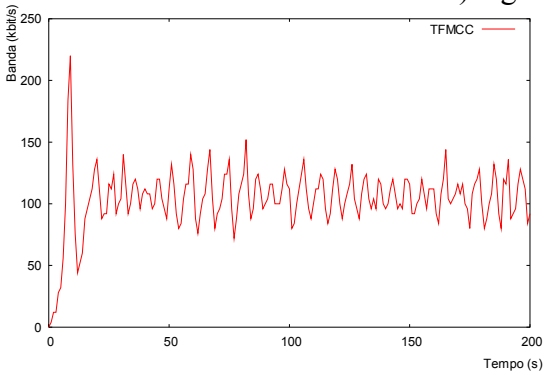
b) algoritmo ALMTF



c) algoritmo RLM



d) algoritmo RLC



e) algoritmo TFMCC

FIGURA 7.3 – Comparação de escalabilidade para 100 receptores

O ALMP (gráfico “a”) obteve a melhor escalabilidade entre todos os algoritmos analisados, com poucas variações de camada. Pode-se observar pelo gráfico que todos os receptores adaptaram-se ao correto número de camadas. Os receptores ALMP1 a ALMP25 adaptaram-se na camada 6 (demandando 1.890 kbit/s, ou seja, taxas da

camada 1 à camada 6, que equivale a 30 kbit/s + 60 kbit/s + 120 kbit/s + 240 kbit/s + 480 kbit/s + 960 kbit/s). Os receptores ALMP26 a ALMP50 adaptaram-se na camada 5 (utilizando 930 kbit/s). Os receptores ALMP51 a ALMP75 adaptaram-se na camada 4 (taxa de 450 kbit/s), e os receptores ALMP76 a ALMP100 adaptaram-se na camada 2 (taxa de 90 kbit/s).

O ALMTF (gráfico "b"), é mais agressivo, fazendo com que o número de tentativas de subir de camada seja superior ao ALMP. Assim, para facilitar a visualização, utilizou-se o gráfico por fluxo ao invés de por camadas. Como pode-se observar, todos os receptores (ALMTF1 a ALMTF100) adaptaram-se ao correto número de camadas, recebendo uma característica de tráfego muito semelhante entre si.

O RLM (gráfico "c") mostrou certo problema na escalabilidade, pois alguns receptores não conseguiram adaptar-se e abandonaram a sessão aproximadamente no instante 60s.

O RLC (gráfico "d") mostrou uma característica para 100 receptores muito semelhante do resultado visto na figura com quatro receptores (figura 7.2), mostrando que esse algoritmo possui uma boa escalabilidade, e os receptores agem de forma sincronizada. Os problemas de adaptação foram exatamente os mesmos vistos na simulação da figura 7.2.

O TFMCC (gráfico "e") se adapta sempre ao receptor com a menor banda, portanto, adaptou-se ao enlace de 100 kbit/s, e o resultado mostra que todos os 100 receptores, independente do enlace, receberam uma transmissão de aproximadamente 100 kbit/s.

Pode-se concluir, a partir da simulação apresentada, que o ALMP é o algoritmo que melhor suporta escalabilidade no modelo de camadas, seguido do ALMTF. O RLM não suporta escalabilidade, o RLC suporta escalabilidade porém não se adapta corretamente nas camadas altas, e o TFMCC suporta uma grande escalabilidade, porém, dentro do seu modelo, que não possui adaptabilidade em ambientes heterogêneos, ou seja, utiliza a taxa de transmissão do receptor com a menor banda.

7.3 Comparações do tempo de adaptação inicial dos algoritmos

O objetivo deste item é comparar os diferentes algoritmos em relação ao seu tempo de adaptação inicial, pois alguns algoritmos são divididos em duas fases:

- **Fase inicial**: o algoritmo aumenta rapidamente a sua banda, buscando chegar no regime permanente o mais rápido possível;
- **Fase de regime permanente**: o algoritmo descobre o ponto de congestionamento da rede e aumenta a banda de forma menos agressiva, a fim de evitar um número excessivo de perdas na rede.

A figura 7.4 mostra uma comparação entre o tempo de adaptação inicial dos algoritmos ALMP, ALMTF e RLM, enquanto a figura 7.5 compara o tempo de adaptação inicial dos algoritmos RLC e TFMCC.

Através da figura 7.4, observa-se que o tempo de adaptação do ALMP e do

ALMTF são compatíveis na fase inicial, ficando em torno de 25s para o ALMP e 20s para o ALMTF. Pode-se ver também que o RLM possui uma adaptação bastante lenta, pois levou aproximadamente 300s para subir 20 camadas. Uma causa para a lenta adaptação inicial do RLM é porque ele utiliza somente um método para aumentar banda, ou seja, seu algoritmo não possui uma fase inicial (mais rápida) e uma fase de regime permanente (mais lenta), como o TCP, o ALMP e o ALMTF.

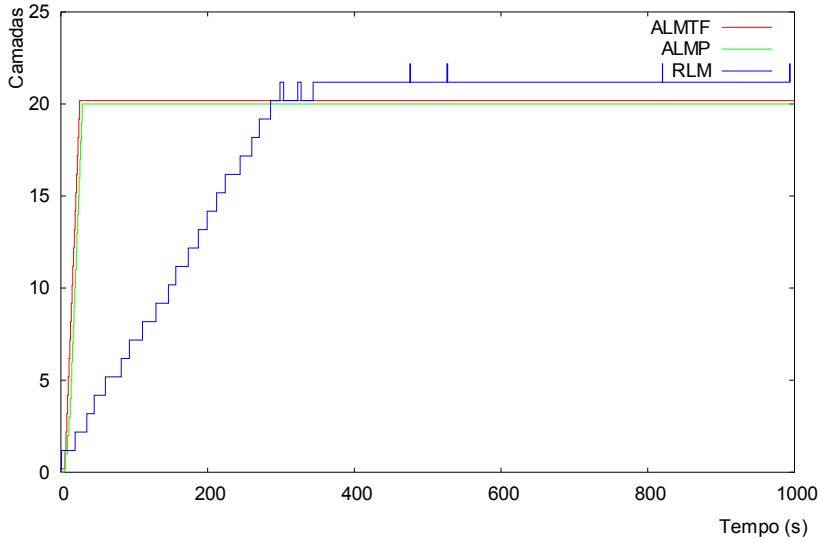
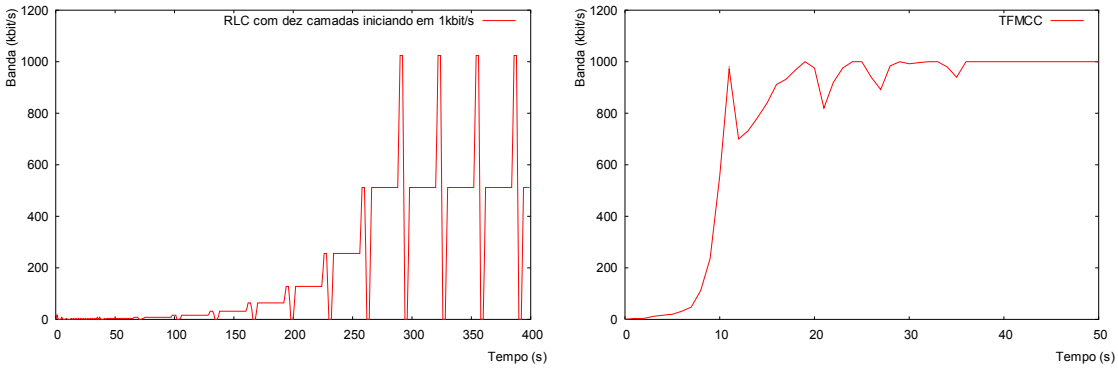


FIGURA 7.4 – Tempo de adaptação inicial do ALMTF, ALMP e RLM

Através da figura 7.5 pode-se observar que o RLC (gráfico "a") também é um protocolo de adaptação lenta, pois levou cerca de 260s para adaptar-se em dez camadas. Como as camadas do RLC necessitam obrigatoriamente ser exponenciais, e uma sendo o dobro da anterior, a simulação foi feita de tal forma que a primeira camada iniciava em 1 kbit/s, e o máximo que se conseguiu atingir foram dez camadas.

Já o TFMCC se adapta de forma rápida, conforme pode ser visto no gráfico "b" da figura 7.5, levando cerca de dez segundos para atingir o regime permanente.



a) algoritmo RLC b) algoritmo TFMCC

FIGURA 7.5 – Tempo de adaptação inicial do RLC e do TFMCC

7.4 Comparações de eqüidade com o próprio tráfego, estabilidade e tempo de adaptação em regime permanente

Para análise de eqüidade entre tráfegos concorrentes, foi utilizada a topologia 2 da figura 4.3, que é reproduzida na figura 7.6 já com os parâmetros específicos utilizados para testar a eqüidade dos algoritmos. Na topologia, existe um número *m* de fluxos do algoritmo em questão e um número *n* de fluxos TCP concorrendo através de um enlace único. Os parâmetros configurados para a primeira simulação estão descritos a seguir, e serão utilizados como base para as outras simulações, que vão modificar alguns desses parâmetros:

- Banda "B" no enlace central especificada como *(m+n)*500kbit/s*, onde *m* é o número de transmissores do algoritmo no momento, e *n* é o número de transmissores TCP concorrentes. Isso foi feito para acomodar 4 camadas exponenciais para cada sessão do algoritmo em camadas (que demandam um total de 450 kbit/s: 30+60+120+240);
- *n = 0*, pois este item não trata com fluxos TCP concorrentes;
- Atraso A = 10ms;
- Camadas exponenciais padrão (30 kbit/s, 60 kbit/s, 120 kbit/s, 240 kbit/s, 480 kbit/s e 960 kbit/s);
- Tamanho do pacote = 500 bytes;
- Tipo de fila = *droptail* com 20 pacotes;
- Randomização alta (±0,5s);

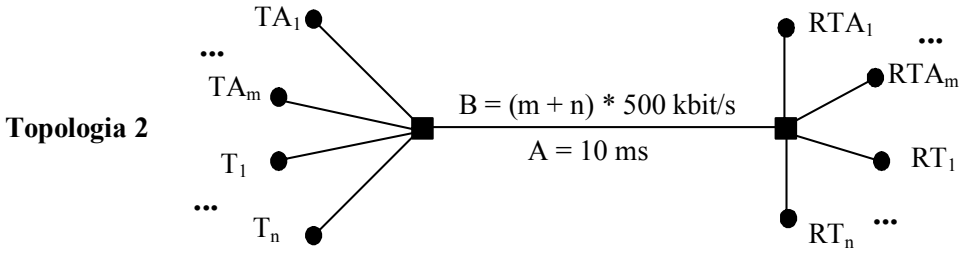


FIGURA 7.6 – Topologia para simulações de eqüidade

As simulações mostradas na figura 7.7 serão utilizadas com o objetivo de comparar a eqüidade para dois fluxos do mesmo algoritmo, bem como a estabilidade e o tempo de adaptação em regime permanente. A simulação mostra dois fluxos de mesmo tipo compartilhando um enlace de 1 Mbit/s. O primeiro fluxo inicia no instante zero, e o segundo fluxo no instante 100s.

Em termos de eqüidade de tráfego e tempo de adaptação em regime permanente, pode-se ver que o ALMP (gráfico "a") e o ALMTF (gráfico "b") adaptaram-se rapidamente à nova condição de tráfego após a entrada do segundo receptor. Além disso, após a adaptação, eles compartilharam tráfego de forma eqüitativa. O segundo fluxo do ALMP necessitou aproximadamente 25s para adaptar-se, enquanto que o segundo fluxo do ALMTF necessitou em torno de 10s. Caso o ALMP gerasse perdas antes de adaptar-se, ele demoraria bem mais para atingir a eqüidade de tráfego, como pode ser visto pela figura 5.32, onde o ALMP sofreu perdas quanto tentou subir para a camada dois, passando para o regime permanente e demorando cerca de 250s para

adaptar-se.

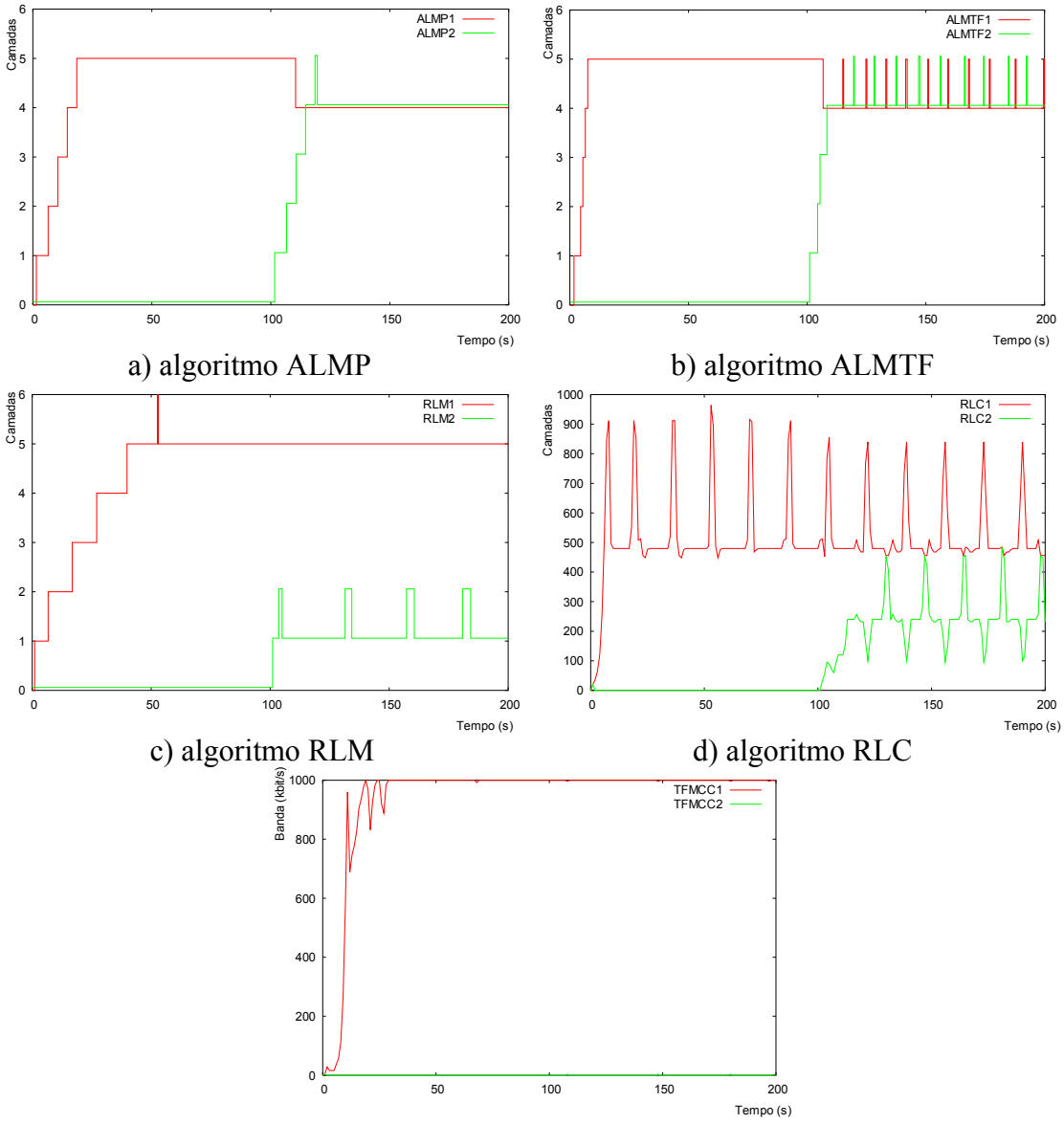


FIGURA 7.7 – Tempo de adaptação em regime permanente e eqüidade para dois fluxos

O RLM (gráfico “c”) não conseguiu adaptar-se, pois possui problemas de eqüidade com o próprio tráfego quando os fluxos iniciam em instantes diferentes. Como o receptor se baseia no tempo para efetuar nova tentativa de subir camada, ele é extremamente conservador em termos de banda, dificultando a entrada de novos fluxos.

O RLC (gráfico “d”) também não conseguiu adaptar-se e, além disso, não utilizou toda a banda disponível. Um fluxo ficou utilizando aproximadamente 500 kbit/s, e o outro fluxo utilizou aproximadamente 200 kbit/s. Isso mostra que o RLC também tem problemas para compartilhar banda com novos fluxos.

O algoritmo do TFMCC (gráfico “e”) utiliza a equação da banda no receptor para determinar a quantidade de tráfego que pode ser transmitida, entretanto, o uso da

equação mostrou dificuldades para diminuir a banda quando entra outro fluxo semelhante no instante 100s, e não permitiu a entrada do segundo fluxo. Isso mostra que o uso da equação do TCP (detalhada no item 2.6) possui problemas de adaptabilidade em regime permanente. Outros testes foram realizados para detalhar o problema, e descobriu-se que, quando os fluxos iniciam com pouca diferença de tempo, eles conseguem adaptar-se corretamente.

Pode-se concluir que os algoritmos ALMP e ALMTF conseguem adaptar-se e compartilhar banda com outros tráfegos de mesmo tipo, mesmo quando eles iniciam em instantes diferentes de tempo. Entretanto, os algoritmos RLM, RLC e TFMCC possuem problemas de adaptação para fluxos que iniciam em instantes diferentes de tempo.

Para analisar a eqüidade para diferentes sessões iniciando ao mesmo tempo, efetuou-se a simulação ilustrada na figura 7.8, onde existem 10 fluxos do algoritmo compartilhando uma banda de 5 Mbit/s, ou seja, destinando 500 kbit/s para cada um dos fluxos, e todos os fluxos começando a simulação praticamente ao mesmo tempo (randomicamente dentro de um segundo). Os gráficos foram gerados por fluxo a fim de facilitar a análise entre todos algoritmos e melhorar a legibilidade de alguns gráficos.

Para facilitar a análise, desenvolveu-se também a tabela 7.2, que mostra os resultados de vazão para cada um dos gráficos da simulação. Na última coluna, é feito um resumo dos resultados de vazão, mostrando a maior vazão obtida, a menor vazão obtida, e a média.

TABELA 7.2 – Estabilidade e vazão para dez fluxos concorrendo entre si

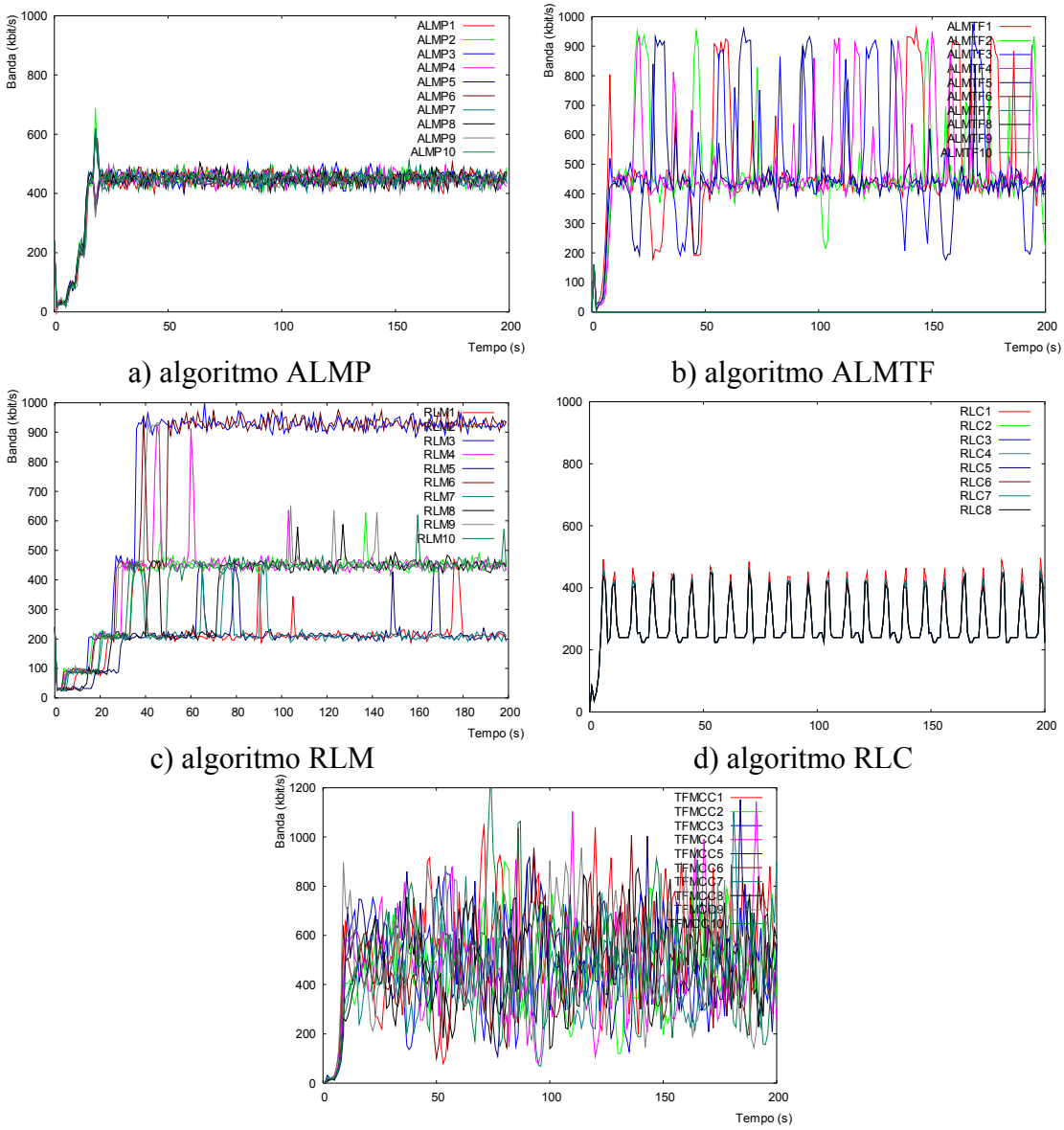
Algoritmo	Vazão (detalhamento)	Vazão (kbit/s)		
		Maior	Menor	Média
ALMP	426; 427; 427; 428; 428; 428; 429; 430; 430; 432	432	426	428,5
ALMTF	420; 444; 469; 472; 472; 482; 486; 487; 488; 519	519	420	473,9
RLM	200; 208; 212; 354; 387; 400; 414; 424; 773; 801	801,5	200,4	407,5
RLC (8 fluxos)	281; 282; 282; 282; 283; 283; 283; 287	287,7	281,3	283,5
TFMCC	437; 451; 466; 471; 471; 472; 479; 508; 509; 545	545,1	437,2	481,3

O ALMP (gráfico “a”) mostrou uma adaptação ao número correto de camadas, ou seja, quatro camadas, correspondendo a 450 kbit/s por receptor. Através da tabela 7.2, observa-se que todos os fluxos dividiram eqüitativamente a banda, atingindo uma média geral de 428,5 kbit/s.

O ALMTF (gráfico “b”) obteve uma maior utilização da banda, porém com menor estabilidade que o ALMP. Em termos de eqüidade, observa-se que o ALMTF conseguiu compartilhar adequadamente o enlace, sendo que somente um receptor utilizou um valor elevado de banda (519 kbit/s), e os outros ficaram próximos a 470 kbit/s. A utilização de mais do que 450 kbit/s se explica pelas tentativas de subir camada, que forçam uma maior utilização no enlace.

O RLM (gráfico “c”) não conseguiu dividir o tráfego de forma eqüitativa, e somente cinco dos dez fluxos adaptaram-se na camada correta (camada quatro, demandando 450 kbit/s). Dos outros cinco fluxos, três ficaram na camada três (210 kbit/s) e dois ficaram na camada cinco (930 kbit/s), mostrando sua dificuldade de

compartilhar tráfego.



a) algoritmo ALMP

b) algoritmo ALMTF

c) algoritmo RLM

d) algoritmo RLC

e) algoritmo TFMCC

FIGURA 7.8 – Eqüidade entre 10 fluxos iniciando ao mesmo tempo

No RLC (gráfico “d”), observa-se através do gráfico e da tabela 7.2 que todos os fluxos compartilharam eqüitativamente o enlace, porém, na camada errada. Os fluxos adaptaram-se na camada três (que demanda 210 kbit/s), e não na camada quatro. Um problema que ocorreu nas simulações do RLC é que o algoritmo implementado por Vicisano não suportava mais do que oito fluxos simultaneamente, e por isso os resultados foram feitos para oito fluxos compartilhando um enlace de 4 Mbit/s, em vez de dez fluxos, como nos outros algoritmos.

No TFMCC (gráfico “e”), observa-se que, quando os fluxos começam juntos, ocorre um equilíbrio no uso da equação de banda, e os fluxos tornam-se competitivos. A média de utilização de cada fluxo foi de 481,3 kbit/s (conforme tabela 7.2). O resultado ideal para o TFMCC seria de 500 kbit/s, pois o mesmo não trabalha com camadas.

A tabela 7.3 mostra o resultado em termos de estabilidade e vazão para a simulação da figura 7.8. Observa-se que o ALMP e o RLM são os mais estáveis. O RLM é estável, entretanto, alguns fluxos não conseguiram adaptar-se na camada correta, como visto anteriormente.

O ALMTF mostrou que é bem mais agressivo que o ALMP, tendo uma média de aproximadamente 10 variações de camadas por minuto, porém, isso é necessário para manter a eqüidade de tráfego com fluxos TCP.

O RLC teve um grau de estabilidade intermediário, com uma média de 4,2 variações de camada por minuto. O algoritmo que mostrou a pior estabilidade foi o TFMCC, que apresentou uma média próxima a 20 variações de camada por minuto.

TABELA 7.3 – Detalhe da estabilidade para os dez fluxos

Algoritmo	VCM		
	Maior	Menor	Média
ALMP	0,6	0,0	0,2
ALMTF	12,2	8,6	10,1
RLM	3,6	1,8	2,8
RLC (oito fluxos)	12,9	2,7	4,2
TFMCC	21,6	14,4	19,02

7.5 Comparações de eqüidade com tráfego TCP concorrente

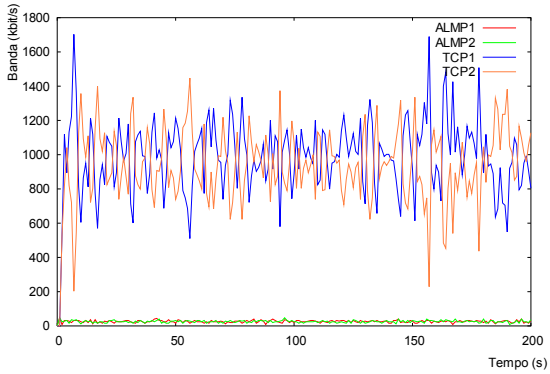
O objetivo deste item é comparar os algoritmos em termos de eqüidade com tráfego TCP concorrente. Para tanto, será utilizada a topologia 2, ilustrada na figura 7.6.

A figura 7.9 mostra uma simulação de dois fluxos do algoritmo em questão concorrendo com dois fluxos TCP, num enlace central de 2 Mbit/s, ou seja, deixando idealmente a banda de 500 kbit/s por receptor. A tabela 7.4 mostra os resultados de vazão para a simulação referente à figura 7.9. Na tabela pode-se verificar detalhadamente os valores de vazão obtidos nas simulações.

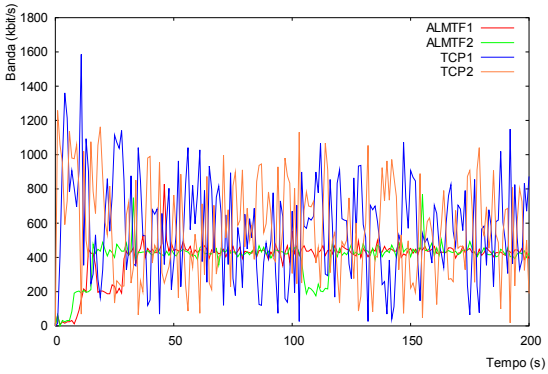
O ALMP (gráfico “a”), não é adequado para competir com o TCP, conforme descrição no capítulo 5. Isso fica claro no gráfico, onde o tráfego ALMP ficou sempre na camada um, pois o TCP é muito mais agressivo. A tabela reflete esse comportamento, mostrando que os fluxos ALMP utilizaram 25 kbit/s de banda, enquanto os fluxos TCP utilizaram 911 kbit/s.

O ALMTF (gráfico “b”) adaptou-se corretamente, conforme descrito no capítulo 6. No ALMTF, a vazão ideal seria de 450 kbit/s (equivalente a 4 camadas), e a vazão ideal do TCP seria de 550 kbit/s (ocupando o restante da banda). Através da tabela, observa-se que o ALMTF utilizou aproximadamente 410 kbit/s, enquanto o TCP utilizou aproximadamente 510 kbit/s.

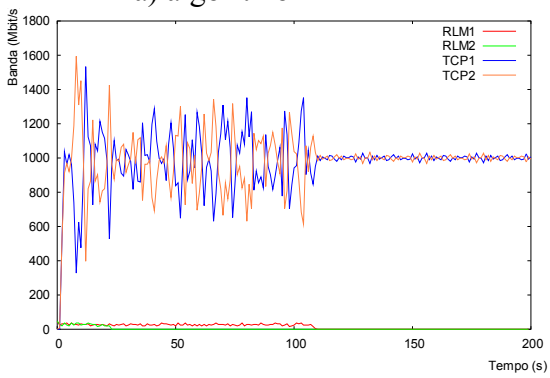
No caso do RLM (gráfico “c”), não aconteceu adaptação, pois os fluxos TCP tomaram conta da banda e os fluxos RLM não conseguiram adaptar-se, ficando inicialmente na primeira camada. Após certo tempo, os receptores abandonaram a sessão, e a banda ficou exclusivamente para o TCP.



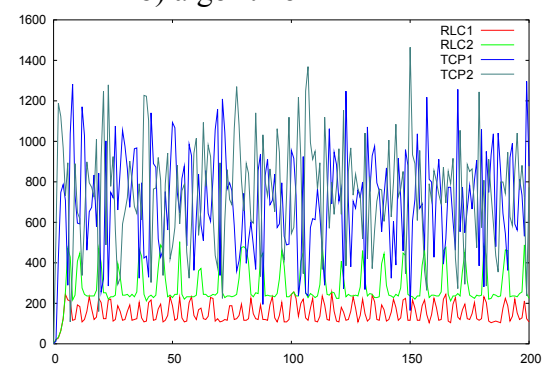
a) algoritmo ALMP



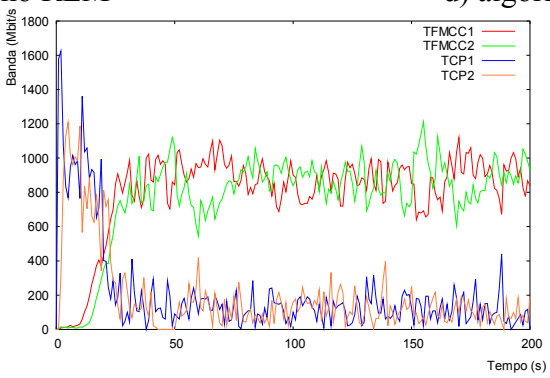
b) algoritmo ALMTF



c) algoritmo RLM



d) algoritmo RLC



e) algoritmo TFMCC

FIGURA 7.9 – Comparação com dois fluxos TCP concorrentes

TABELA 7.4 – Vazão para dois fluxos TCP concorrendo com dois do algoritmo

Algoritmo	**Vazão do algoritmo (kbit/s)**		**Vazão do TCP (kbit/s)**	
	Fluxo1	**Fluxo2**	**Fluxo1**	**Fluxo2**
ALMP	25	25	911	911
ALMTF	419	405	507	529
RLM	14,3	3,1	984	981
RLC	154	284	703	745
TFMCC	797,5	782,3	181	218

O RLC (gráfico “d”) se mostrou muito menos agressivo que o TCP, como pode ser visto no gráfico e na tabela, utilizando aproximadamente 200 kbit/s quando os fluxos

TCP utilizaram aproximadamente 700 kbit/s.

O TFMCC (gráfico “e”), utilizou mais banda do que o ideal para uma divisão eqüitativa da banda, conforme pode ser visto no gráfico e na tabela. A banda eqüitativa deveria ser 500 kbit/s, e os fluxos TFMCC utilizaram aproximadamente 790 kbit/s, deixando o TCP com aproximadamente 200 kbit/s.

Pode-se concluir que o único algoritmo que se comportou de forma eqüitativa com o TCP foi o ALMTF. Todos os outros falharam em algum aspecto, ou transmitindo muito a mais que o TCP ou muito a menos que o TCP.

A tabela 7.5 mostra os resultados de estabilidade para a simulação referente à figura 7.9. Como o ALMP e o RLM praticamente não saíram da camada um, eles não foram incluídos. Assim, a tabela 7.5 só trata o ALMTF, o RLC e o TFMCC. Observa-se que o ALMTF é bem mais estável que o TCP, sofrendo variações de camada 20 a 30 vezes menos que um fluxo TCP. Já o RLC e o TFMCC são mais estáveis que o TCP, porém, ainda assim são bem menos estáveis em relação ao ALMTF.

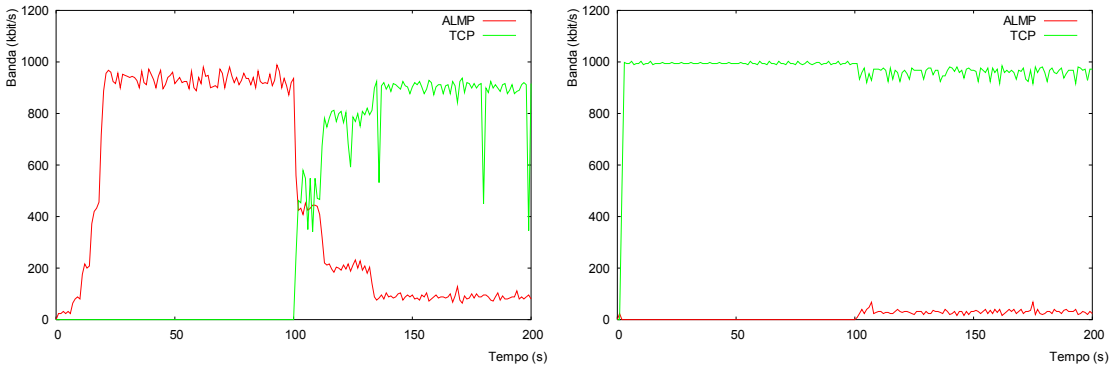
TABELA 7.5 – Estabilidade para dois fluxos TCP concorrendo com dois do algoritmo

Algoritmo	VCM do algoritmo		VCM do TCP	
	Fluxo1	Fluxo2	Fluxo1	Fluxo2
ALMTF	1,49	2,68	46,5	50,4
RLC	18	10	34	34
TFMCC	15	11	45	43

A partir da análise da tabela 7.4 e da tabela 7.5, conclui-se que o ALMTF é competitivo com o tráfego TCP, porém com muito mais estabilidade. Essa conclusão também pode ser obtida observando o gráfico referente ao ALMTF na figura 7.9.

7.6 Fluxo do algoritmo iniciando antes e depois do TCP

Este item tem como objetivo analisar se o algoritmo consegue adaptar-se com fluxos TCP concorrentes iniciando antes e depois do algoritmo. Para tanto, foi utilizada a topologia 2, vista na figura 7.6, com um fluxo do algoritmo concorrendo com um fluxo TCP, ambos compartilhando um enlace de 1 Mbit/s.

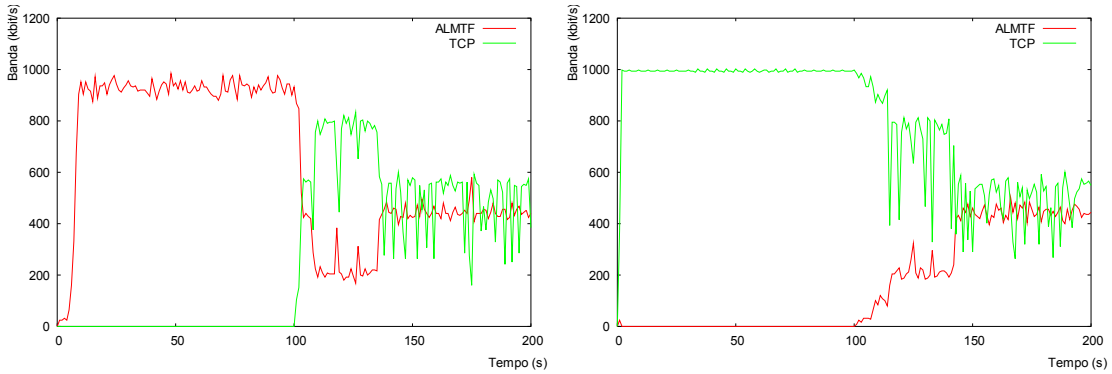


a) ALMP inicia antes b) ALMP inicia depois

FIGURA 7.10 – Eqüidade do ALMP começando antes e depois do TCP

O ALMP não é competitivo com tráfego TCP, como pode ser visto através da figura 7.10. Quando o ALMP inicia antes do fluxo TCP (gráfico “a”) ou depois (gráfico “b”), o fluxo ALMP vai para a camada um, pois é muito menos agressivo.

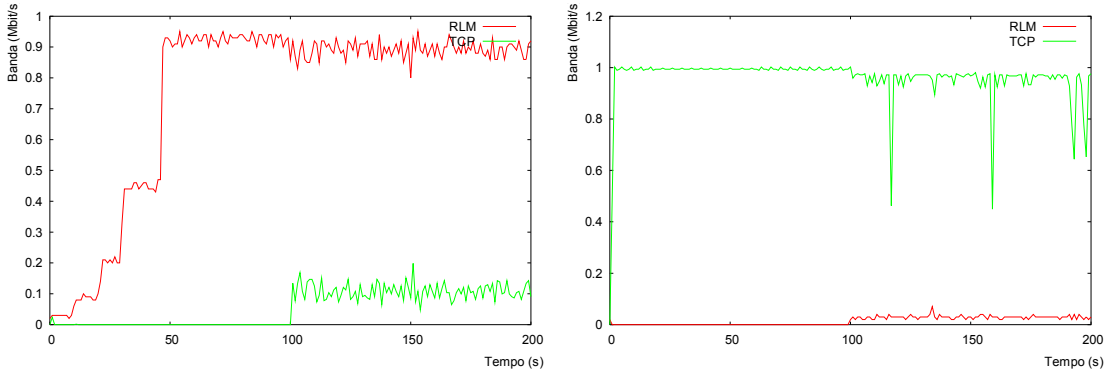
Efetuando a mesma simulação com o ALMTF, pode-se observar, através da figura 7.11, que o ALMTF consegue obter eqüidade de tráfego com o TCP. Tanto quando o ALMTF inicia 100s antes do TCP como quando inicia 100s depois, a adaptação ocorre no instante 150s, ou seja, após a entrada de um fluxo TCP, o ALMTF com camadas exponenciais demora cerca de 50s para adaptar-se.



a) ALMTF inicia antes b) ALMTF inicia depois

FIGURA 7.11 – Eqüidade do ALMTF começando antes e depois do TCP

O RLM possui problemas de adaptação com o TCP, tanto quando seu fluxo inicia antes do TCP como quando inicia após, conforme mostra a figura 7.12. Quando o RLM inicia antes (gráfico “a”), o RLM não deixa o TCP aumentar de banda, e toma conta do enlace. No caso do RLM iniciar depois do TCP (gráfico “b”), o RLM não consegue forçar o TCP a diminuir sua taxa.



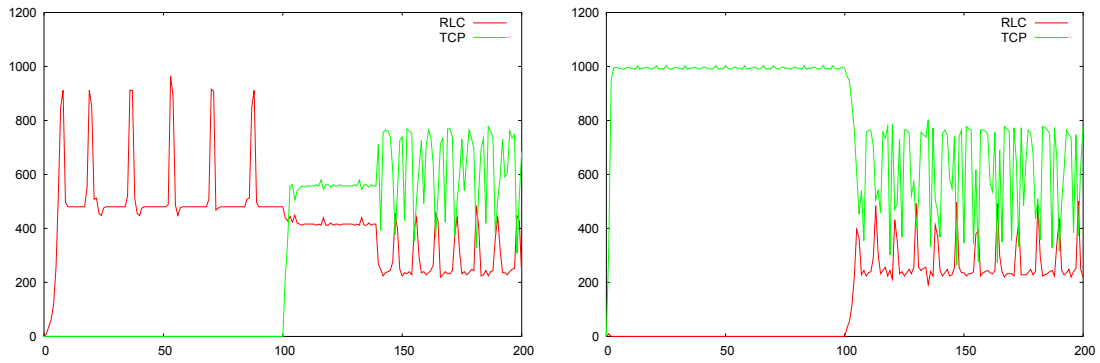
a) RLM inicia antes b) RLM inicia depois

FIGURA 7.12 – Eqüidade do RLM começando antes e depois do TCP

O RLC se adapta com o TCP para essa quantidade de banda no enlace. A figura 7.13 mostra o resultado da simulação para o RLC iniciando antes e depois do TCP. Quando o RLC inicia antes (gráfico “a”), ele permite ao TCP uma adaptação no instante 100s. Nesse ponto, o TCP passa a utilizar uma banda de mais que o dobro da banda do RLC (RLC com aproximadamente 200 kbit/s e TCP com aproximadamente 500 kbit/s).

No caso do RLC começar depois (gráfico “b”), o resultado é muito similar, e o RLC consegue fazer o TCP diminuir sua banda para os mesmos níveis mostrados no

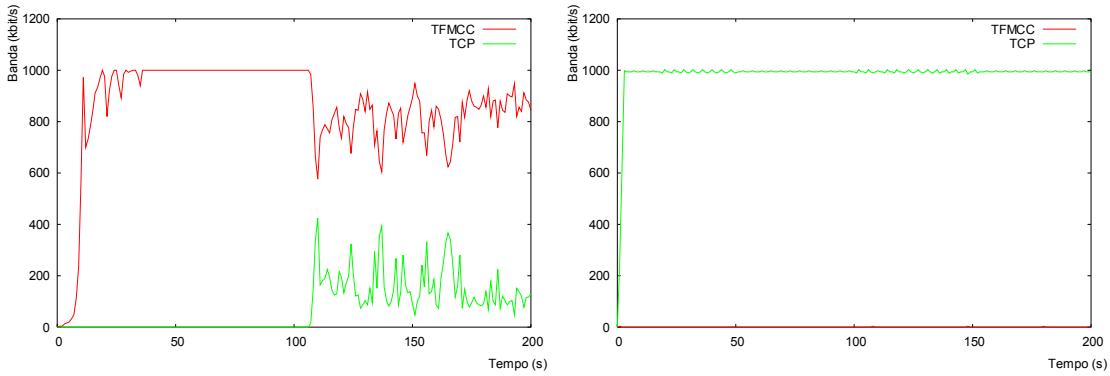
gráfico “a”.



a) RLC inicia antes b) RLC inicia depois

FIGURA 7.13 – Eqüidade do RLC começando antes e depois do TCP

O TFMCC também mostrou problemas de adaptação semelhantes ao RLM, ou seja, quando o TFMCC inicia antes do TCP, o TCP tem dificuldades de subir de banda, como mostra a figura 7.14a. Quando o TFMCC inicia depois, ele não consegue aumentar seu tráfego, conforme ilustra a figura 7.14b.



a) TFMCC inicia antes b) TFMCC inicia depois

FIGURA 7.14 – Eqüidade do TFMCC começando antes e depois do TCP

A partir dos resultados vistos nas figuras 7.10 a 7.14, pode-se concluir que o único algoritmo que consegue uma transmissão eqüitativa com o TCP é o ALMTF.

7.7 Conclusões sobre as comparações efetuadas

Neste capítulo, efetuou-se uma série de comparações entre os algoritmos desenvolvidos no âmbito desta Tese e outros algoritmos relacionados. Utilizou-se o RLM, o RLC e o TFMCC para efeitos de comparação.

A tabela 7.6 apresenta um resumo das comparações efetuadas, identificando os pontos fortes e fracos de cada algoritmo. Cada linha da tabela será analisada a seguir.

Na primeira linha da tabela 7.6 (Adaptação em ambientes heterogêneos) considerou-se que todos algoritmos adaptaram-se corretamente, entretanto, o resultado da figura 7.2 mostrou que o algoritmo RLC teve dificuldades de adaptação com bandas elevadas, porém, adaptou-se corretamente com bandas menores. O TFMCC se adapta ao

receptor mais lento, portanto, não suporta ambientes heterogêneos.

Na segunda linha da tabela, considerou-se que o RLM não suporta escalabilidade, pois muitos receptores não conseguiram entrar na sessão com 100 receptores, conforme ilustrado na figura 7.3.

A terceira linha da tabela analisa o tempo de adaptação inicial, até a primeira perda. Observa-se que os algoritmos com tempo de adaptação mais rápido são o ALMTF e o TFMCC.

A quarta linha da tabela analisa o tempo de adaptação em regime permanente, ou seja, após perdas detectadas. Nesse caso, o ALMTF continua bastante rápido. O ALMP é lento nesse caso, pois tem seu foco principal na estabilidade. O RLM não consegue adaptar-se, por isso foi incluído o valor N/A. O TFMCC utiliza a equação do TCP, e é bastante rápido para adaptar-se em regime permanente.

A quinta linha da tabela analisa os algoritmos em termos de estabilidade para dez sessões concorrentes. Observa-se que o ALMP tem o resultado mais baixo, ou seja, é o algoritmo mais estável dos analisados. O pior é o TFMCC, com 19,02 variaçoes de camada por minuto.

A sexta linha da tabela analisa se o algoritmo permite adaptação com tráfego do mesmo tipo iniciando em instantes diferentes. Observa-se que somente os algoritmos desenvolvidos no âmbito desta Tese conseguem adaptar-se.

As últimas quatro linhas da tabela analisam a eqüidade dos algoritmos. Para tanto, foi utilizado o conceito de *fairness index* (FI), definido no item 4.7.5, que é a relação entre a taxa que o receptor está utilizando (TU) frente à taxa eqüitativa para este receptor (TE), ou seja, FI = TU / TE, e quanto mais próximo do valor "um", mais eqüitativo é o fluxo.

A sétima linha da tabela analisa o FI para a simulação da figura 7.8, com dez fluxos do mesmo algoritmo iniciando ao mesmo tempo e concorrendo entre si. Utilizou-se a média dos resultados para cálculo do FI. Observa-se que o ALMP, ALMTF e TFMCC atingiram níveis próximos a "um". Para o RLM, foi utilizado N/A, pois a divisão de banda entre os dez fluxos não foi próxima.

A oitava linha mostra o índice FI para a simulação da figura 7.9, com dois fluxos TCP concorrendo com dois fluxos do algoritmo. Pode-se observar que o único algoritmo que conseguiu um índice de *fairness* próximo a "um" foi o ALMTF, que ficou com FI = 0,91. O TFMCC ficou com um índice FI = 1,57, significando que seus fluxos utilizaram 57% de banda a mais do que deveriam.

As linhas nove e dez da tabela mostram o índice FI para o fluxo do algoritmo iniciando antes e depois de um fluxo TCP. Novamente verifica-se que o único algoritmo que atingiu níveis próximos a "um" em ambos os casos foi o ALMTF, provando que ele é realmente TCP-friendly.

Em linhas gerais, pode-se concluir que o ALMP seria o algoritmo mais indicado para utilização em redes privativas (sem a presença do TCP concorrendo), devido à sua característica de adaptação em ambientes heterogêneos, aliada à sua alta estabilidade, escalabilidade e eqüidade com tráfego concorrente do seu próprio tipo, independente do mesmo iniciar em momentos diferentes. O uso do RLM fica comprometido

principalmente devido à sua falta de habilidade em adaptar-se quando entram novos fluxos em instantes diferentes de tempo.

Caso seja necessário utilizar a transmissão em ambientes que necessitem concorrer com tráfego TCP, o mais indicado seria o ALMTF, principalmente devido à sua capacidade de dividir banda de forma eqüitativa com tráfego TCP, não importando se o mesmo inicia antes ou depois. Além disso, o ALMTF mostrou possibilidade de adaptação em ambientes heterogêneos, tempo de adaptação rápido, alta escalabilidade e eqüidade com seu próprio tráfego. O uso do TFMCC é problemático devido à sua falta de habilidade em adaptar-se quando entram novos fluxos em instantes diferentes de tempo.

TABELA 7.6 – Resumo das comparações entre os algoritmos

Descrição	ALMP	ALMTF	RLM	RLC	TFMCC
Adaptação em ambientes heterogêneos	sim	sim	sim	sim	não
Escalabilidade para 100 receptores	sim	sim	não	sim	sim
Tempo aproximado de adaptação para 20 camadas até primeira perda	25s	20s	300s	250s	9s
Tempo aproximado de adaptação em regime permanente para subir 4 camadas com dois fluxos concorrentes	200s	7s	N/A	30s	N/A
Estabilidade com dez fluxos concorrentes (valor de VCM)	0,26	10,09	2,8	4,23	19,02
Adaptação com tráfego iniciando em instantes diferentes	sim	sim	não	não	não
FI com 10 fluxos do próprio tráfego iniciando ao mesmo tempo	0,95	1,05	N/A	0,63	0,96
FI com dois TCP concorrentes (iniciando ao mesmo tempo)	0,05	0,91	0,03	0,48	1,57
FI com um fluxo TCP concorrente (algoritmo iniciando antes)	0,05	0,92	1,91	0,63	1,71
FI com um fluxo TCP concorrente (algoritmo iniciando depois)	0,03	1,00	0,06	0,60	0,0001

8 Codificação de Vídeo Multi-Taxas

No âmbito do SAM, descrito no capítulo 4, um módulo fundamental é o codificador de vídeo utilizado, que forma a base da transmissão, como é visto nos módulos "codificador em camadas" e "decodificador em camadas" da figura 8.1, que foi repetida aqui para simplificar o entendimento do objetivo deste capítulo. O módulo "Controle de congestionamento e camadas (ALM)" é a principal contribuição desta Tese, e os algoritmos desenvolvidos para esse módulo foram descritos nos capítulos anteriores, e são o ALMP e ALMTF.

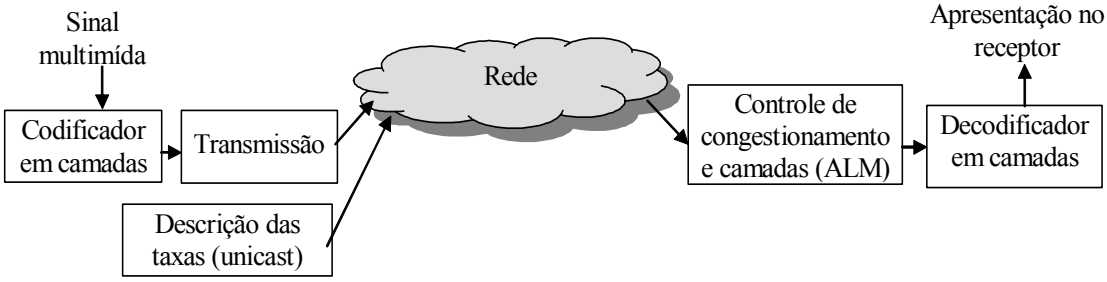


FIGURA 8.1 – Modelo do SAM

Entretanto, para que o sistema SAM fique completo, é necessário descrever as possibilidades de codificadores que o sistema suporta, portanto, este capítulo tem por objetivo analisar de forma breve alguns codificadores em camada existentes, bem como as etapas envolvidas no processo de codificação e decodificação de um sinal de vídeo, abordando em maiores detalhes o codificador VEBIT. Uma simplificação deste capítulo foi publicada em [BRU 2003b].

Do ponto de vista do codificador de vídeo, a transmissão multi-taxa pode ser produzida através de transmissão em camadas, replicação em várias taxas ou através do uso de transcodificadores intermediários, conforme detalhamento no item 2.3 (Ambientes heterogêneos e codificação multi-taxas).

Para os objetivos desta Tese, se julgou que o método mais adequado de transmissão seria através de camadas, pois elas permitem uma maior economia global dos recursos da rede com moderada complexidade. Em paralelo com esta Tese, Bruno [BRU 2003] produziu uma dissertação de mestrado cujo foco foi o desenvolvimento de um codificador em camadas, denominado VEBIT (Vídeo Escalável por Bitplanes). Esse codificador será integrado ao sistema SAM, proposto nesta Tese.

Na área de codificação de vídeo, transmissão de vídeo em camadas é freqüentemente referida como "codificação escalável" [LI 2003] ou "codificação hierárquica", e pode ser conseguida de diversas formas, como será explicado no item 8.1. O restante deste documento vai continuar utilizando o termo "em camadas" se referindo à escalabilidade do vídeo, entretanto, os nomes originais dos padrões serão mantidos na forma como foram concebidos, ou seja, os termos "escalável" e "escalabilidade" serão utilizados quando necessário.

O restante deste capítulo está dividido da seguinte forma: o item 8.1 (Conceitos

sobre codificação de vídeo multi-taxa) fornece conceitos básicos importantes para o entendimento da codificação de vídeo. O item 8.2 (Trabalhos relacionados) mostra alguns trabalhos já desenvolvidos sobre codificação em camadas. O item 8.3 (Detalhamento do VEBIT) descreve o funcionamento do VEBIT e sua implementação, enquanto o item 8.4 (Resultados obtidos com o codificador VEBIT) mostra os resultados obtidos. No item 8.5 (Extensão do VEBIT para suporte à transmissão ao vivo) se colocam alguns resultados de uma extensão ao VEBIT para que o mesmo pudesse suportar captura direto da placa de vídeo. O item 8.6 (Integração no SAM) aborda algumas considerações sobre a integração do codificador VEBIT com o SAM, e finalmente o item 8.7 (Conclusões e melhorias futuras) apresenta as conclusões do capítulo.

8.1 Conceitos sobre codificação de vídeo multi-taxa

Quando se está transmitindo vídeo em várias taxas, é importante ter em mente os seguintes conceitos:

- **Economia de banda**: as técnicas de geração de vídeo multi-taxa devem ser comparadas em relação à quantidade de banda que geram na rede, e as possibilidades são transmissão em camadas, codificação com replicação e transcodificadores intermediários, conforme descrito anteriormente;
- **Granularidade de adaptação**: diz respeito à diferença de taxa de transmissão entre os vários fluxos do mesmo vídeo, conforme visto no item 2.5 (Eqüidade de tráfego). Se a granularidade é muito grossa, cria dificuldades para se obter eqüidade com tráfego concorrente. Se a granularidade é muito fina, gera uma estrutura de controle multicast complexa de gerenciar;
- **Complexidade da decodificação**: para transmitir multicast em tempo real, o sinal deve ser codificado uma vez e decodificado por todos os receptores, portanto, é importante que o decodificador seja mantido simples, a fim de que o sinal consiga atingir uma maior diversidade de receptores.

As normas H.263, MPEG-2 e MPEG-4 possuem métodos padronizados para codificar vídeos em camadas. Em geral, cada método permite a transmissão em apenas duas camadas, uma de baixa qualidade e outra de alta qualidade, com exceção da escalabilidade granular fina, que permite mais camadas, mas foi padronizada somente no MPEG-4.

Os seguintes itens descrevem os principais métodos de obtenção de vídeos multi-taxa, que são: mudança na taxa de quadros por segundo (escalabilidade temporal – item 8.1.1), tamanho do quadro (escalabilidade espacial – item 8.1.2), qualidade do quadro (escalabilidade de qualidade ou SNR (*Signal Noise Ratio*) – item 8.1.3), freqüência do quadro (escalabilidade por freqüência ou particionamento – item 8.1.4) ou pela utilização de *bitplanes* (escalabilidade granular fina – item 8.1.5).

8.1.1 Escalabilidade temporal

A escalabilidade temporal consiste em se obter múltiplas camadas através da divisão da informação contida no domínio tempo dos quadros sendo transmitidos, como

por exemplo a modificação da taxa de quadros por segundo, ou através de métodos de compressão temporal com posterior divisão em camadas. Segundo Li [LI 2003], essa é a forma de escalabilidade mais comum existente atualmente, sendo adotada em diversos padrões de compressão de vídeo, como o H.263 e a família MPEG.

Uma forma de obter escalabilidade temporal, no caso do MPEG, é dividir os quadros I (*Intraframe*), P (*Predictive*) e B (*Bidirecionally-predictive*) de forma que cada um deles seja transmitido como uma camada separada, sendo que a camada base é composta somente dos quadros I, pois são os mais importantes. Um problema nesta técnica é a granularidade de adaptação ao vídeo, pois existem somente três camadas, e taxas típicas para o MPEG-1 são [LI 2003]: 256 kbit/s para os quadros I, 900 kbit/s para quadros I + P e 1,5 Mbit/s para quadros I + P + B. Nessas condições, a granularidade é muito grossa, sendo difícil obter eqüidade com tráfego concorrente.

Outra forma é dividir os quadros a serem transmitidos em quadros pares e quadros ímpares, enviando cada conjunto de quadros numa camada separada. Nesse caso, como não existe camada mais importante, os receptores podem se cadastrar em qualquer uma delas, entretanto, deve-se definir qual é a camada um e qual é a camada dois, a fim de otimizar a rede. Assim, caso nenhum receptor tenha banda para se cadastrar nas duas camadas, o transmissor só vai transmitir um grupo multicast.

Outra forma de codificação temporal em camadas pode ser obtida via utilização da transformada 3D de *Wavelet* [TAU 94], [BRU 2003]. Nesse caso, se aplica a transformada de *Wavelet* no sentido temporal para um determinado número de quadros, dividindo o sinal em freqüências altas e baixas. Como as freqüências no domínio temporal são dadas pela quantidade de alteração que existe entre os quadros, as freqüências altas vão significar grande alteração entre quadros, e as freqüências baixas representam pouca variação. Nesse caso, pode-se enviar uma camada base correspondendo às freqüências baixas, e uma adicional representando as freqüências altas. Esse método é bom para vídeos com pouco movimento, pois em filmes de ação, com muitas variações de cena, a informação dessas variações fica na camada adicional, e não é vista para quem se cadastrou apenas na camada base.

8.1.2 Escalabilidade espacial

Na escalabilidade espacial, as várias camadas são obtidas através de imagens com resolução mais baixa nas camadas inferiores, e a resolução aumenta à medida que o receptor decodifica mais camadas [BRU 2003] [PUR 93].

Os principais algoritmos utilizados para efetuar escalabilidade espacial são o de pirâmide laplaciana [BUR 83] e as transformadas de Wavelet [GHA 99].

Na pirâmide laplaciana, a imagem é reduzida através da média aritmética dos seus *pixels*, como mostra a figura 8.2. O gráfico da esquerda mostra a imagem original, que é reduzida conforme mostra o gráfico central, onde é feita a média aritmética de grupos de quatro *pixels* ((4+9+1+2)/4), gerando uma imagem menor (Média 1), e assim sucessivamente até a obtenção de um único valor, que representa o valor médio da imagem, ou seu valor DC. O gráfico da direita mostra as três camadas geradas pelo sistema: a camada 1 ou camada base (valor DC), a camada 2 (formada pela diferença entre a "Média 1", vista no gráfico central, e o valor DC) e a camada 3 (formada pela

diferença entre a imagem original, vista no primeiro gráfico, e a “Média 1”, vista no gráfico central). A transmissão é feita pela diferença em relação à média, pois necessita menos bits do que a transmissão do valor completo.

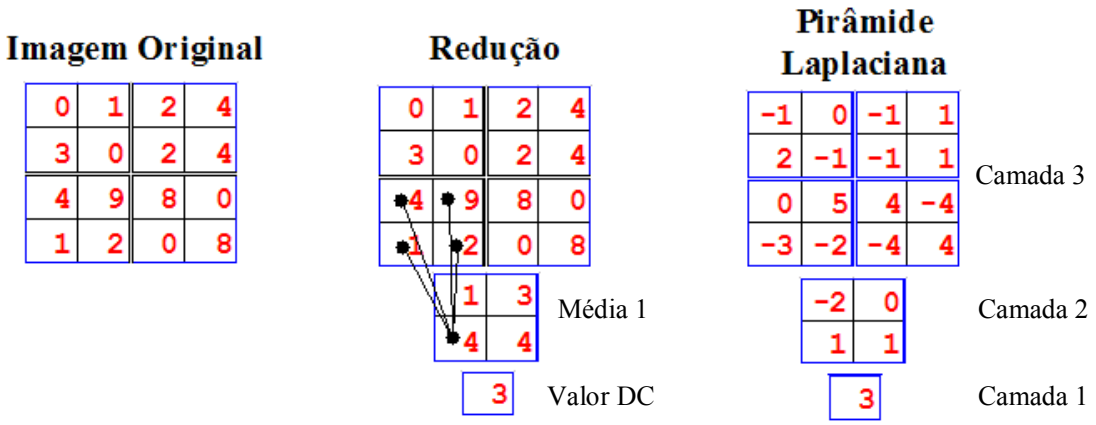


FIGURA 8.2 – Redução espacial baseada em pirâmide laplaciana

No algoritmo de Wavelet, se utiliza a transformada de Wavelet, que separa a imagem em freqüências altas e baixas, como pode ser visto na figura 8.3. A região de freqüências baixas, localizada no canto superior esquerdo, forma a camada base, enquanto as demais freqüências formam a camada adicional.

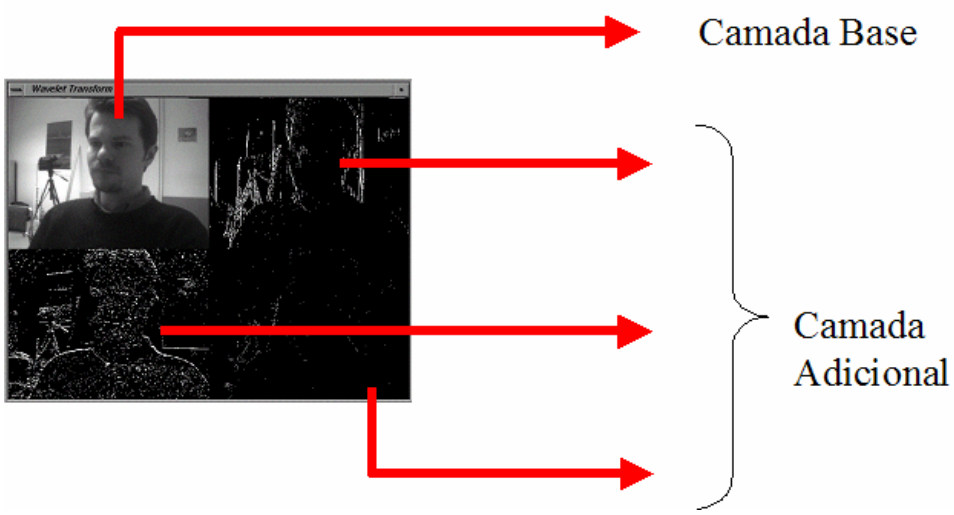


FIGURA 8.3 – Redução espacial baseada em Wavelet

8.1.3 Escalabilidade de qualidade ou SNR

Na escalabilidade de qualidade, também conhecida como escalabilidade SNR (*Signal-Noise Ratio*), as múltiplas taxas são obtidas através da variação no grau de precisão dos quantizadores, que provocam uma variação na qualidade da imagem gerada [BRU 2003], [WIL 97], [FOG 2003]. O grau de precisão utilizado nos quantizadores é dependente da taxa que se deseja obter na camada base. A distorção introduzida nessa primeira quantização é novamente quantizada com uma precisão superior, gerando uma camada adicional. O processo se repete pelo número de camadas

desejado pelo codificador. Na camada base também se transmite informações necessárias para a decodificação, como os vetores de movimento.

8.1.4 Escalabilidade por freqüência ou particionamento de dados

Na escalabilidade por freqüência, também conhecida como escalabilidade por particionamento de dados, as informações do domínio freqüência da imagem são divididas em camadas distintas. Tais informações são o resultado de uma transformada, normalmente a DCT (*Discrete Cosine Transform*), que é aplicada aos valores de entrada [BRU 2003].

Como a DCT costuma ser dividida em blocos 8x8, a saída possui 64 valores no domínio freqüência, que são organizados de forma a deixar as freqüências mais baixas à frente num vetor, conforme mostra a figura 8.4. O ponto de particionamento é definido de acordo com a quantidade de camadas a serem criadas. A camada base contém os valores mais críticos, compostos pelas freqüências baixas, pela componente DC e os ACs mais significativos. Os demais valores AC são transmitidos em camadas adicionais.

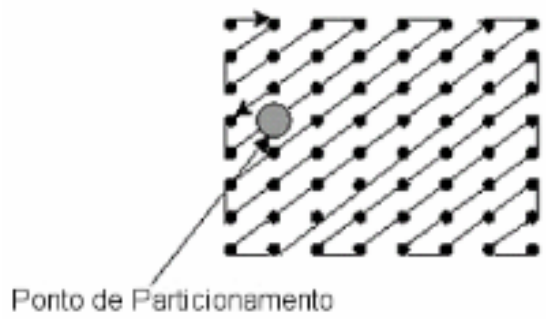


FIGURA 8.4 – Ponto de particionamento na escalabilidade por freqüência

Essa técnica é utilizada pelo JPEG progressivo a fim de visualizar progressivamente uma imagem. Inicialmente os valores DC são enviados, resultando numa imagem nebulosa, entretanto, dando ao usuário uma idéia da imagem final. À medida que os demais valores chegam, a imagem torna-se mais nítida.

O padrão MPEG-2 define a criação de, no máximo, duas camadas para essa técnica, que normalmente é utilizada juntamente com outras, como escalabilidade espacial e de qualidade.

8.1.5 Escalabilidade granular fina por bitplanes

Um problema nos métodos de escalabilidade vistos anteriormente, na forma como foram padronizados no MPEG, é que geram no máximo duas camadas. O modelo de escalabilidade espacial do MPEG-2 apresenta três camadas, entretanto, isso é porque ele utiliza a escalabilidade espacial mais a de qualidade. O objetivo da escalabilidade granular fina, ou FGS (*Fine Granular Scalability*) [BRU 2003], [LI 2001] é permitir um maior número de camadas, resultando numa menor granularidade e conseqüente transmissão mais eqüitativa de tráfego, conforme discutido nos conceitos iniciais deste capítulo.

Uma forma de atingir a escalabilidade granular fina é através de *bitplanes*, sendo um método padrão no MPEG-4. A camada base é codificada de forma que o vídeo possa ser decodificado por qualquer dispositivo MPEG-4, independente do mesmo possuir recurso de escalabilidade. Para tanto, se utiliza uma das codificações vistas nos itens anteriores para gerar uma camada base e uma camada adicional. A camada adicional é dividida em várias camadas de refinamento, através do algoritmo de *bitplanes*.

Inicialmente, é feita a codificação normal baseada na transformada DCT, Os valores obtidos são divididos em *bitplanes*, como mostra o exemplo abaixo, onde um bloco 2D-DCT de 8x8 foi codificado no seguinte vetor de 64 posições [BRU 2003]:

"10, 0, 6, 0, 0, 3, 0, 2, 2, 0, 0, 2, 0, 0, 1, 0, ... , 0, 0"

O maior número do vetor é "10", que pode ser representado com 4 bits (1010), portanto, é possível gerar até 4 diferentes camadas, que são mostradas na figura 8.5. A camada 1 é a mais importante, pois carrega o bit mais significativo (MSB) do vetor original. Uma possibilidade utilizada no VEBIT foi de utilizar dois bits por camada ao invés de um, reduzindo o número de camadas. Em geral, a escolha do número de camadas depende da taxa desejada de bits por segundo, da granularidade desejada nas camadas e da qualidade escolhida, entre outros fatores.

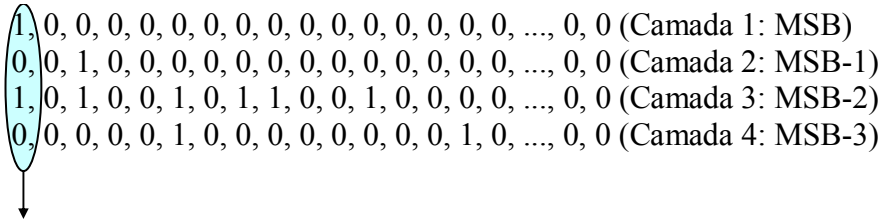


FIGURA 8.5 – Divisão de vetor DCT em *bitplanes*

Após a divisão, cada *bitplane* pode ser comprimido utilizando alguma técnica de entropia, como RLE (*Run-Length Encoding*) ou Huffman. O MPEG-4 se aproveita do fato do vetor ser binário e define uma variação no método de compressão RLE, utilizando um valor que indica o número de zeros consecutivos antes do próximo bit "1". Por exemplo, o terceiro *bitplane* (MSB-2) pode ser comprimido como: "0, 1, 2, 1, 0, 2".

Caso o receptor esteja cadastrado somente na camada básica, a qualidade percebida da imagem será menor do que a original. Por exemplo, o primeiro coeficiente do vetor será lido como "8" em vez de "10". O terceiro será lido como "0" em vez de "6", e assim por diante. Entretanto, segundo [LI 2001], a qualidade da imagem utilizando a técnica FGS é superior quando comparada com a técnica SNR para a mesma taxa de bits.

8.2 Trabalhos relacionados

Lombardo [LOM 2003] propõe a utilização de uma característica de controle de taxa presente no padrão MPEG a fim de obter um número de bits por segundo constante

na rede, resultando em tráfego de vídeo CBR. Seu objetivo é conformar o tráfego de saída de acordo com a variável *Tspec*, presente nas transmissões com qualidade de serviço do tipo Intserv.

O codificador *WaveVideo* é um trabalho desenvolvido pelo laboratório de redes "TIK", em Zurique, na Suíça. Fankhauser [FAN 98] descreve um codificador voltado para redes *wireless* heterogêneas, com diferentes qualidades e diferentes taxas de erro. O codificador utiliza transformada Wavelet com extensão ao eixo temporal, ou seja, em 3D. O nível de redundância e a qualidade do sinal podem ser bastante flexíveis, e os parâmetros são controlados pelo receptor. O sinal de vídeo original é enviado em alta qualidade via transmissão multicast. Outros terminais móveis na mesma sub-rede podem receber a mesma qualidade. Terminais móveis localizados fora da sub-rede podem estar em enlaces com alta taxa de erro ou com baixa largura de banda. Para atender esses terminais, um filtro é inserido no *switch* de saída da rede, adaptando a qualidade do sinal para os receptores abaixo dele. Esse codificador é adequado para sistemas de transmissão multicast em camadas a partir de transcodificadores intermediários, conforme visto no capítulo 3. No caso do *WaveVideo*, a utilização de redes *wireless* facilita o processo, pois existem pontos bem definidos para inserir o filtro.

Um modelo de codificador não cumulativo bastante utilizado é conhecido como MDC (*Multiple Description Coding*), que propõe a transmissão de um sinal em múltiplos canais. Na sua forma mais simples para vídeo, ele considera dois canais em paralelo, onde cada um carrega metade dos quadros do vídeo (que é dividido em quadros pares e ímpares). O problema dessa abordagem é em enlaces com altas taxas de erro [WAN 2002], pois, devido à codificação, se o receptor recebe um quadro par, ele só pode analisar a diferença a partir de outro quadro par anterior. Para superar essa limitação, Wang [WAN 2002] propôs enviar uma redundância a mais em cada camada, ou seja, os quadros pares carregam, além da informação codificada dos quadros pares, uma redundância para poder recuperar a informação a partir de um quadro ímpar. O mesmo ocorre nos quadros ímpares. Esse processo permite uma maior recuperação do sinal no caso de pacotes perdidos.

A tabela 8.1 [LI 2003] lista as taxas de bit para duas seqüências de teste de vídeo padrão: *Foreman* e *Akiyo*, usando resolução QCIF e três diferentes codificadores: MPEG-4 em taxa simples, codificador FGS (definido no item 8.1.5) e codificador MDC. Pode ser visto que o *overhead* devido à codificação em camadas é superior à 20%.

TABELA 8.1 – Comparação entre MPEG-4 e diferentes codificadores em camada

	Taxa única	**FGS (*overhead*)**	**MDC (*overhead*)**
Foreman, PSNR = 33,3 dB	79,3 kbit/s	102,3 kbit/s (29%)	114,6 kbit/s (44%)
Akiyo, PSNR = 35,4 dB	19,3 kbit/s	23,8 kbit/s (23%)	25,7 kbit/s (33%)

8.3 Detalhamento do VEBIT

O objetivo principal do codificador em camadas VEBIT (Vídeo Escalável por *Bitplanes*) [BRU 2003] é permitir a transmissão em camadas de uma aula remota

síncrona na resolução QCIF (176x144) a 24 quadros por segundo. Além disso, especificou-se que o número de camadas não deve ser muito alto, ficando próximo a 5 camadas, o desempenho do codificador deve ser suficiente para que funcione em tempo real e a camada base deve ter condições de ser visualizada através de modem de 56 kbit/s.

Por motivos de simplicidade, utilizou-se uma transformada DCT tridimensional [SER 97], a fim de que o domínio tempo fosse também levado em consideração. Um problema da transformada utilizada é que possui tempos de codificação e decodificação semelhantes, onerando o processamento no receptor, entretanto, considerou-se o avanço na capacidade de processamento dos computadores atuais, e julgou-se que o tempo de decodificação estava apropriado, conforme descrição a ser vista no item 8.4.1.

O modelo de escalabilidade utilizado é baseado em *bitplanes*, conforme descrição no item 8.1.5, e foi escolhido por permitir escalabilidade granular fina, importante para o objetivo de obtenção de aproximadamente 5 camadas.

A figura 8.6 ilustra o modelo implementado. Cada quadro de vídeo é convertido de seu formato original para o formato YUV 4:2:2, e cada um dos sinais gerados (Y, U e V) é dividido em um bloco de oito quadros, que é utilizado como entrada no sistema. Cada bloco é dividido em blocos menores (de 8x8 *pixels*), resultando num cubo de 8x8x8 *pixels*, que é utilizado como entrada na codificação 3D-DCT. O vetor resultante possui 512 valores, que são quantizados e divididos em *bitplanes*, gerando as diversas camadas a serem transmitidas.

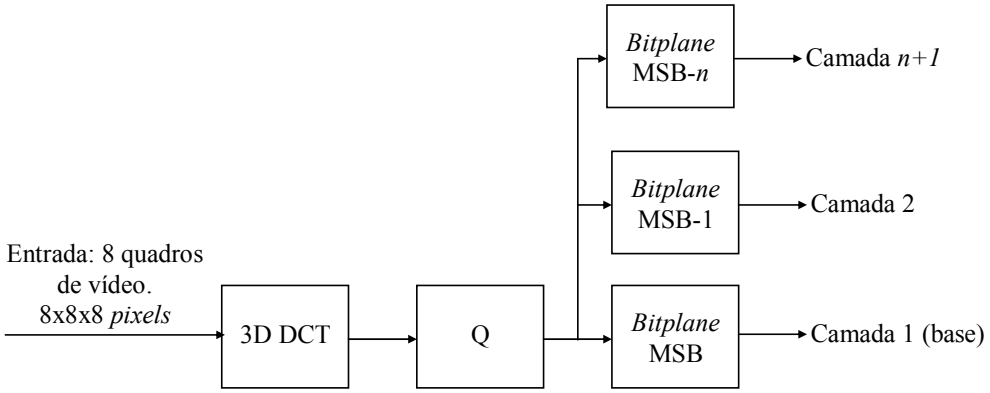


FIGURA 8.6 – Modelo de codificação utilizado no VEBIT

O decodificador faz o processo inverso, e está ilustrado na figura 8.7. Se estiver recebendo apenas uma camada, o receptor aplica diretamente a inversa da quantização (Q^{-1}) e a transformada 3D-DCT inversa, enviando para o usuário um vídeo com baixa qualidade. Caso esteja recebendo mais camadas, o receptor primeiro recupera o vetor de *bitplanes*, somando as informações recebidas das várias camadas, para então aplicar a inversa da quantização (Q^{-1}) e a transformada 3D-DCT inversa. A qualidade de saída é, portanto, proporcional ao número de camadas que o receptor está apto a receber naquele momento.

Os próximos itens analisam com maiores detalhes o funcionamento dos diversos blocos do codificador VEBIT, na forma como foram implementados.

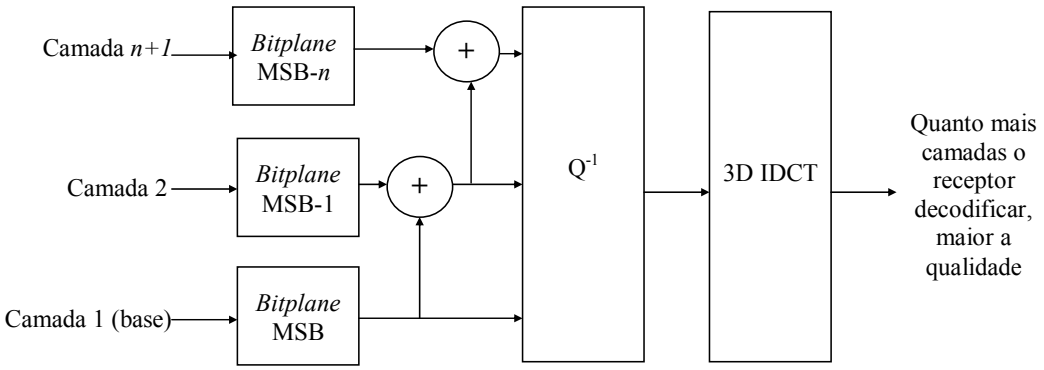


FIGURA 8.7 – Modelo de decodificação utilizado no VEBIT

8.3.1 Os arquivos de entrada e saída de vídeo

A entrada de dados no sistema é feita através de arquivos PPM (*Portable Pixmap*), cujo formato divide um vídeo em uma série de quadros, e cada quadro é representado como um arquivo separado. O formato permite um máximo de 10.000 quadros, totalizando aproximadamente sete minutos de vídeo a 24 quadros por segundo. Cada nome de arquivo possui um prefixo e o número do quadro a que se refere, assim, por exemplo, se um vídeo tem o prefixo "vid", os três primeiros quadros deste vídeo se chamam "vid0000.ppm", "vid0001.ppm" e "vid0002.ppm".

Cada arquivo PPM é dividido conforme mostra a figura 8.8. Os primeiros dois bytes contêm a identificação do formato (caracteres "P6"), seguida da altura H e largura W da imagem. Em seguida, encontra-se a seqüência de 3 bytes por *pixel* da imagem, no formato RGB.

1 2	3	4	5 6 7	8 9 10	11
P6	H	W	3 *pixels* RGB	3 *pixels* RGB	...

FIGURA 8.8 – Modelo de decodificação utilizado no VEBIT

O uso do formato PPM permite que se obtenha a imagem original, sem perdas, o que permite uma análise posterior da eficiência do algoritmo de compressão. Para se obter uma entrada de vídeo em outro formato, como MPEG, basta ler do dispositivo de captura e converter, conforme descrição no item 8.5.

Para converter os arquivos de entrada do formato RGB para o formato YUV 4:2:2, foi utilizada uma função de conversão existente no software *ffmpeg*[1]. Após a codificação, é gerado um arquivo de saída conforme ilustração na figura 8.9 [BRU 2003]. Cada item da figura é descrito a seguir:

- W (*Width*): largura da imagem em *pixels*;
- H (*Height*): altura da imagem em *pixels*;
- L (*Layers*): número de camadas codificadas no arquivo;
- FPms (*Frames per milisecond*): taxa de transmissão, em quadros por milisegundo. Esse valor vai direto para um *sleep* modificado da implementação;
- GOL (*Group of Layers*): contém informações de todas as camadas, sendo que cada

[1] ffmpeg.sf.net, acessado em março de 2003.

GOL é formado por 8 quadros. Por exemplo, se determinado vídeo foi codificado a 24 quadros por segundo e possui 5 segundos, ele tem 120 quadros, ou seja, 15 GOLs;

- S_{Li} (*Size of Layer i*): número de bytes de dados que a camada *Li* ocupa dentro dos 8 quadros do GOL atual. Essa informação é necessária, visto que a codificação é do tipo VBR (*Variable Bit Rate*);
- S*i* (*Size of Y, U ou V*): número de bytes de dados existentes na camada *Li* do GOP atual para cada componente Y, U ou V da imagem. Y, U e V representam os dados efetivamente codificados da imagem em questão;
- GOP (*Group of Pictures*): fornece os dados codificados dentro do GOL atual.

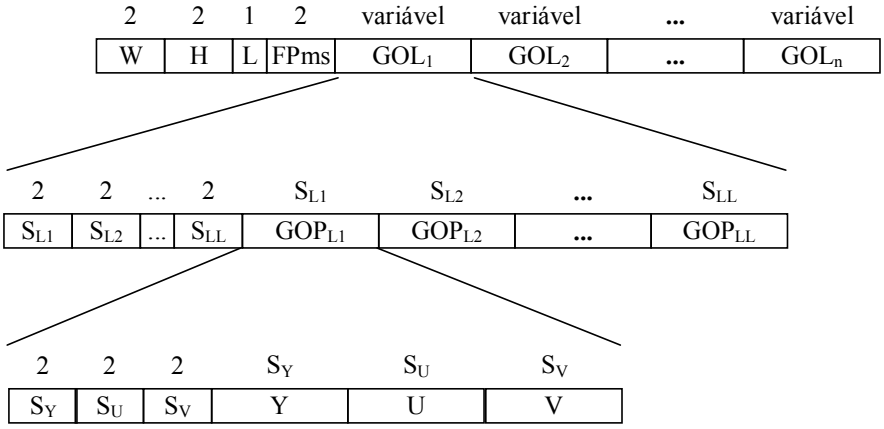


FIGURA 8.9 – Arquivo de saída gerado pelo VEBIT

8.3.2 A transformada 3D-DCT

Como foi visto na figura 8.6, a dimensão tempo é utilizada no codificador, que lê 8 quadros por vez, gerando um cubo de 8x8x8 *pixels* (vetor de 512 posições), e o aplica à entrada do módulo 3D-DCT. Este cubo contém 8 *pixels* horizontais, 8 verticais e 8 temporais para uma determinada primitiva, que pode ser Y, U ou V.

A transformada DCT é aplicada em cada conjunto de 8 bits, a cada uma das dimensões do cubo, conforme mostra a figura 8.10. Isso gera um total de 192 execuções da equação.

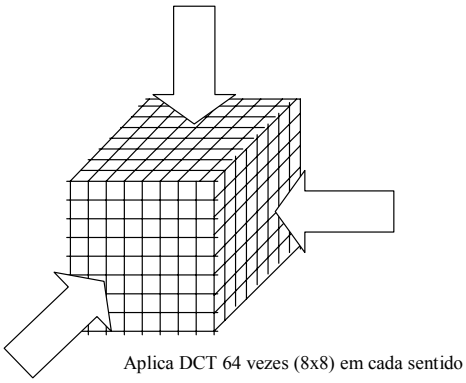


FIGURA 8.10 – Transformada 3D-DCT no VEBIT

Como resultado, obtém-se um vetor de 512 coeficientes DCT, sendo que a primeira posição do vetor contém o coeficiente DC. As primeiras posições do vetor gerado costumam ser de maior magnitude que os coeficientes de entrada, visto que a transformada condensa a energia nas primeiras posições do vetor. Na implementação, os valores atingem um máximo de 10 bits para o coeficiente DC, e 9 bits para os coeficientes AC, após a quantização.

Para a decodificação, o processo é o inverso, ou seja, a equação reversa é aplicada 192 vezes, gerando um tempo semelhante ao gerado para codificação. A otimização utilizada na 3D-DCT foi a proposta por Loeffler [LOE 89]. A fim de aumentar o desempenho, as operações foram realizadas utilizando-se números inteiros com um fator de aumento de escala para compensar o ponto flutuante. Mesmo assim, sempre é inserido algum erro no processo, que se propaga à medida que a transformada é executada repetidamente.

8.3.3 Quantização

Após a transformada, o vetor com os 512 coeficientes gerados pela DCT passa para a fase de quantização, que segue os princípios da quantização escalar, ou seja, utiliza uma tabela de referência para o processo de redução dos coeficientes. A tabela utilizada é semelhante à utilizada no MPEG-4 [BRU 2003], e efetua-se uma divisão do valor original vindo da DCT pelo valor contido na tabela. A divisão é um processo oneroso em termos de processamento, e na decodificação esse tempo não é gasto, fazendo com que, nessa etapa, o processo de decodificação seja mais rápido que o de codificação.

8.3.4 Codificação em bitplanes

O módulo de codificação em *bitplanes* divide os dados em camadas e efetua grande parte da compactação dos dados, conforme visto no item 8.1.5. Na implementação efetuada por Bruno [BRU 2003], é feito um tratamento diferenciado entre o coeficiente DC e os coeficientes AC.

O valor DC possui uma magnitude maior e não pode perder muito do seu valor pois ele representa a maior parte da qualidade da imagem. Os coeficientes AC têm uma magnitude relativamente menor (2 bits a menos em média) e possuem pontos que podem ser reduzidos sem prejudicar em muito a imagem (AC vai mais para o final do vetor),

Como o DC tem mais qualidade, ele deve ser inserido nas camadas mais baixas. Os coeficientes AC que representam a maior parte da qualidade (estão mais próximos do DC) e que possuem maior magnitude, também devem ser transmitidos nas primeiras camadas, pois representam maior qualidade.

Assim, o coeficiente DC corresponde à primeira posição do vetor gerado, sendo o maior valor existente no vetor. Nos vídeos analisados, o maior coeficiente DC em todos os casos poderia ser representado por 9 bits, portanto, utilizou-se 10 bits para seu armazenamento, dando uma margem de segurança de um bit. O número de bits do vetor atual é extremamente importante para o processo de codificação e decodificação, sendo armazenado no início da área de dados do arquivo de saída, que foi mostrado na figura

8.9. Assim, por exemplo, caso o vetor gerado pelo quantizador seja correspondente à componente de luminância Y do sinal YUV e tenha uma representação de 9 bits, o início da área de dados da componente Y terá 4 bits (1001) representando a precisão, ou quantidade de bits, necessária para transmitir o coeficiente DC. Na prática, esses 4 bits são comprimidos em 3 bits.

Os bits necessários para transmitir o coeficiente DC são distribuídos nas primeiras três camadas, de acordo com a tabela 8.2. Por exemplo, caso o número de bits necessários para representar a componente DC seja 5, os primeiros 2 bits após o MSB são transmitidos na camada 1, e os dois últimos bits são enviados na camada 2. O bit MSB nunca é transmitido, pois é a própria posição na tabela, que está armazenada no início da área de dados, conforme visto no parágrafo anterior. Essa posição já representa esse bit, não necessitando duplicar essa informação. No caso de 10 bits para representar a componente DC, o bit MSB não é transmitido pelo motivo explicado anteriormente. O segundo bit não é transmitido pois é sempre zero (visto através de medições que o valor nunca ultrapassa 1.500). Os bits 3 a 5 são transmitidos na camada 1, os bits 6 a 8 na camada 2, e os bits 9 e 10 na camada 3, conforme a tabela.

TABELA 8.2 – Distribuição dos bits DC nas diferentes camadas, sem contar o MSB

N. de bits para representar DC	N. de bits enviados na camada 1	N. de bits enviados na camada 2	N. de bits enviados na camada 3
2	1	-	-
3	2	-	-
4	2	1	-
5	2	2	-
6	3	2	-
7	3	3	-
8	3	3	1
9 e 10	3	3	2

Os coeficientes AC são representados pelas próximas 511 posições no vetor gerado pelo quantizador. Como o número de bits necessários para representar os componentes AC é menor do que para o componente DC, utilizou-se um máximo de 9 bits, cuja distribuição em cada camada é mostrada na tabela 8.3. Assim, por exemplo, caso uma determinada posição AC tenha o valor 454, que é igual a 111000110, a distribuição ficaria: camada 1: “1”; camada 2: “1”; camada 3: “10”; camada 4: “00”; camada 5: “110”. Num outro exemplo, caso o valor seja 9, que é igual a 1001, a distribuição ficaria: camada 1: “0”; camada 2: “0”; camada 3: “1”; camada 4: “0”; camada 5: “01”.

Analisou-se a distribuição dos bits nos *bitplanes*, constatando-se que a maior parte dos valores é zero. Implementou-se uma compressão *run-length* que permitiu uma maior compressão dos dados através de tabelas Huffman. Essa compressão reduziu bastante o tamanho do sinal, como será visto adiante, entretanto, o algoritmo ainda pode ser melhorado nesse aspecto [BRU 2003].

TABELA 8.3 – Distribuição dos bits AC nas diferentes camadas

N. bits para representar AC	N. de bits enviados na camada 1	N. de bits enviados na camada 2	N. de bits enviados na camada 3	N. de bits enviados na camada 4	N. de bits enviados na camada 5
1	-	-	-	-	1
2	-	-	-	-	2
3	-	-	-	1	2
4	-	-	1	1	2
5	-	1	1	1	2
6	1	1	1	1	2
7	1	1	1	2	2
8	1	1	2	2	2
9	1	1	2	2	3

8.4 Resultados obtidos com o codificador VEBIT

Em [BRU 2003] utilizou-se quatro vídeos para gerar os resultados, todos com resolução QCIF (176x144) a 24 quadros por segundo. Um dos vídeos (*Claire*) tem pouco movimento, outro (*Night*) tem muito movimento, e existem dois intermediários. A característica de cada um deles é descrita a seguir.

- *Carphone*: uma pessoa falando diretamente para a câmera em um carro em movimento. A pessoa gesticula e movimenta a cabeça constantemente, e pode-se ver, através da janela do carro, a paisagem se alterando. O vídeo possui um total de aproximadamente 15 segundos (376 quadros), e a paisagem se altera mais drasticamente dos quadros 190 em diante (últimos 7 segundos), quando o carro passa por uma seqüência de árvores, gerando uma atualização constante da região;
- *Forest*: uma clareira dentro da floresta, com uma pessoa centralizada na imagem. A câmera faz uma aproximação nesta pessoa, o que dura em torno de 200 quadros, ou 8 segundos. Após, a pessoa executa uma acrobacia, sendo acompanhada pela câmera. O vídeo possui 288 quadros, ou 12 segundos;
- *Claire*: ambiente típico de videoconferência, apresenta o rosto de uma pessoa falando, com pouco movimento da cabeça. O fundo não muda, e a maior ação é o movimento da boca da pessoa. Total de 488 quadros, ou aproximadamente 20 segundos;
- *Night*: vista aérea de uma cidade virtual, à noite, produzindo atualização de tela constantemente. Total de 200 quadros, ou aproximadamente 8 segundos.

Os resultados mostrados neste capítulo vão se referir somente a um ou dois dos vídeos analisados, dependendo da situação, pois julgou-se que traduzem adequadamente o funcionamento do codificador em camadas. Para um detalhamento de resultados em todos os vídeos, deve-se consultar [BRU 2003].

Uma visão geral da qualidade obtida do ponto de vista do usuário para o vídeo *Carphone* é ilustrada na figura 8.11. Pode-se ver que, se o usuário está recebendo somente a camada 1 (primeiro quadro), a imagem obtida é nebulosa, entretanto, é

possível distinguir o que está acontecendo. Observa-se que a qualidade vai aumentando à medida que decodifica-se mais camadas, e o melhor caso é quando decodifica-se todas as camadas, conforme ilustra o último quadro (camadas 1 a 5).

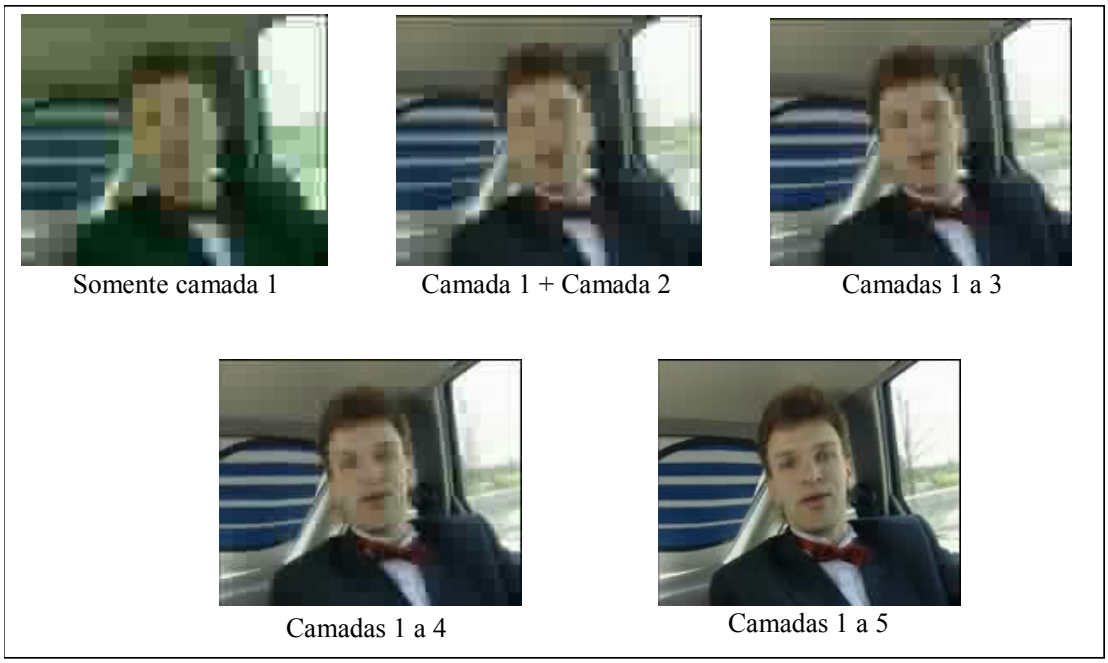


FIGURA 8.11 – Qualidade do vídeo *Carphone* do ponto de vista dos receptores

8.4.1 Rapidez de codificação e decodificação do VEBIT

O computador utilizado para análise da rapidez para codificação e decodificação do algoritmo foi um *Athlon* 1200 MHz, com 256 Mbytes de memória. A linha de comando utilizada pelo compilador GCC foi '-O3 –mmx', que habilita a utilização de máxima otimização para execução, e emprego de instruções MMX sempre que o compilador julgar necessário. Os resultados apresentados são a média de 10 execuções.

A tabela 8.4 mostra o desempenho do codificador para 80 quadros (3,3 segundos de vídeo considerando 24 quadros / segundo), comparando dois tipos de vídeo: *Forest*, com menos ação, e *Night*, que possui mais ação. Pode-se verificar que o vídeo *Forest* mostrou uma maior rapidez de codificação, exatamente pelo menor número de mudanças entre os quadros. Verifica-se também que, para o formato CIF (*Common Intermediate Format*), o tempo de codificação quase triplica, entretanto, ainda é suficiente para uma transmissão em tempo real. Transmissão em tempo real, no caso desta Tese, significa a habilidade de transmitir um sinal ao vivo, ou seja, o codificador e decodificador devem estar aptos a efetuar sua tarefa de forma igual ou mais rápida que a velocidade de entrada dos quadros na placa de captura de vídeo.

TABELA 8.4 – Desempenho da rotina de codificação para os vídeos Forest e Night

Resolução	***Forest***	***Night***
QCIF: 176x144	0,530 s	0,681 s
CIF: 352x288	1,665 s	2,078 s

A tabela 8.5 mostra o desempenho do decodificador para 3,3 segundos (80 quadros) do vídeo *Night*. Observa-se que o tempo de decodificação não muda muito nas diversas camadas, podendo-se concluir que a soma dos *bitplanes* consome pouco processamento (de fato, o maior processamento está na rotina IDCT (Inversa da DCT), como será visto adiante). Somente a última camada (camada 5) tem um tempo significativamente maior, pois a quantidade de dados necessários à sua decodificação é superior às demais. Outra conclusão é que o tempo de decodificação de todas as camadas é inferior ao tempo de codificação, e seria possível para um receptor decodificar todas as camadas desse vídeo em tempo real, pois levaria menos do que a duração do vídeo, que é 3,3 segundos. O vídeo apresentado na tabela 8.5 é o pior caso, e o desempenho é superior para os outros vídeos.

TABELA 8.5 – Desempenho da rotina de decodificação para o vídeo Night

Resolução	Camada 1	Camada 2	Camada 3	Camada 4	Camada 5
176x144	0,274	0,277	0,286	0,289	0,324
352x288	0,828	0,840	0,856	0,884	0,977

Analisando o desempenho das rotinas individualmente, chegou-se à conclusão que, para o vídeo *Night* com resolução QCIF (*Quarter CIF*), a FDCT 1D consome quase 50% do processamento. Na decodificação, a rotina IDCT 1D consome mais do que 60% do tempo.

8.4.2 Taxa de compressão do VEBIT

O objetivo da análise da taxa de compressão é descobrir quão eficiente é o algoritmo de compressão do VEBIT. A tabela 8.6 apresenta as taxas de compressão obtidas para as 5 camadas do vídeo *Carphone*. A coluna *bits/pixel* representa a quantidade de bits necessária para armazenar um *pixel*, e é obtida somando-se o número de *pixels* existente no vídeo e dividindo pelo total de bits gerados na camada; a taxa de compressão mostra a diferença entre o tamanho do vídeo original e do vídeo compactado.

TABELA 8.6 – Compressão do Vídeo Carphone

Camadas	Bits/pixel	Taxa compressão
1	0,0266	903:1
1 e 2	0,0654	367:1
1 a 3	0,1173	204:1
1 a 4	0,1995	120:1
1 a 5	0,5276	45:1

8.4.3 Taxa de transmissão por camada

O gráfico da figura 8.12 mostra a taxa de transmissão por camada necessária para o vídeo *Claire*, que é o que menos exige de banda. Caso um receptor se cadastre em todas camadas, ele vai necessitar aproximadamente 170 kbit/s.

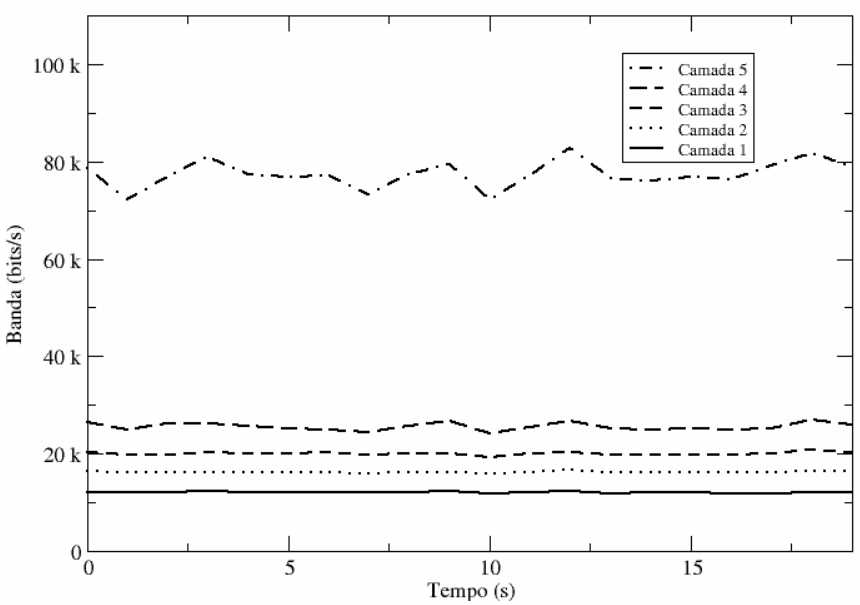


FIGURA 8.12 – Banda por camada do vídeo *Claire*

Na figura 8.13 pode-se ver a banda necessária para o vídeo *Night*, que exige a maior banda de todos os vídeos analisados. Nesse caso, o receptor que se cadastrar em todas camadas vai precisar de aproximadamente 650 kbit/s na rede.

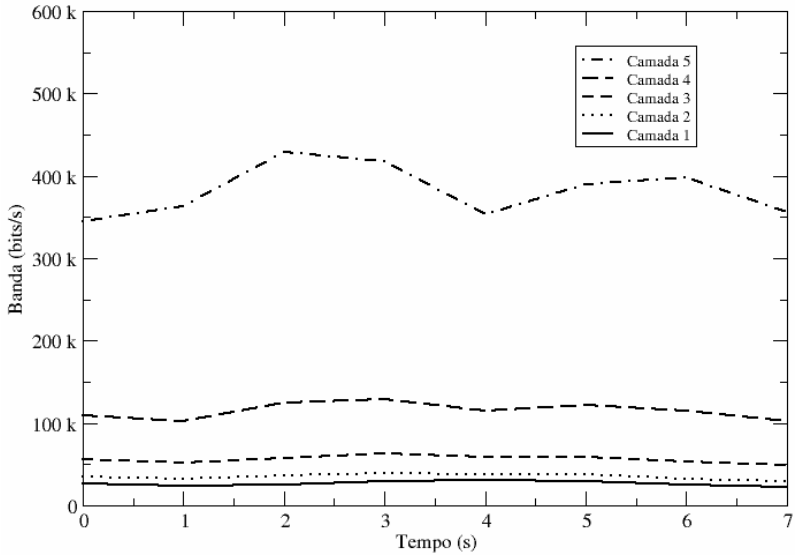


FIGURA 8.13 – Banda por camada do vídeo *Night*

O gráfico mostrado na figura 8.14 mostra uma comparação de necessidade de banda para todos os vídeos analisados. Pode-se ver que todos os vídeos possuem pelo menos uma camada abaixo do índice de 56 kbit/s, pois, conforme objetivos do projeto, o sistema deveria estar apto a trabalhar com receptores conectados via modem. No gráfico, efetuou-se um deslocamento de 0,05 pontos em cada ponto do eixo X, a fim de facilitar a visualização. Assim, a camada do vídeo "Carphone", ao invés de mudar em "1", "2", etc, muda em "0,95", "1,95", etc.

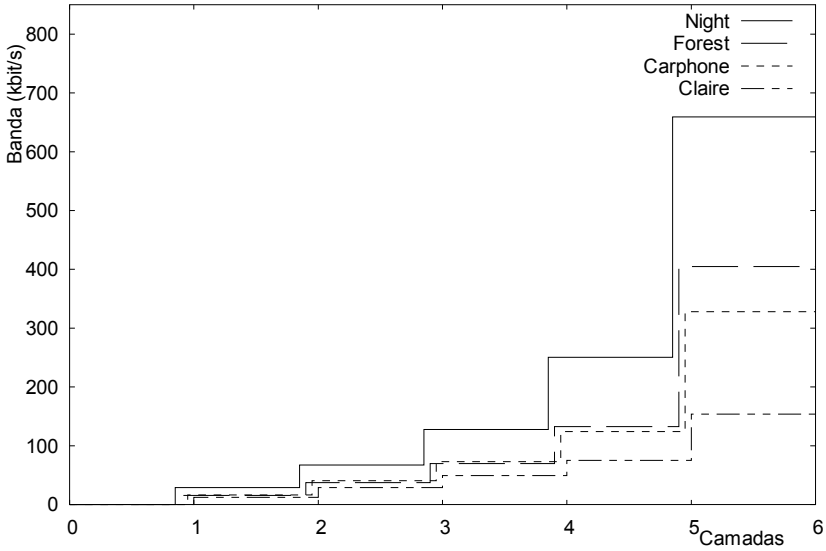


FIGURA 8.14 – Divisão de banda por camada em todos vídeos

8.4.4 Qualidade via PSNR

Foi efetuada uma análise de qualidade via PSNR (*Peak Signal to Noise Ratio*), comparando-se os quadros gerados pelo decodificador e os quadros originais. Cada *pixel* do quadro decodificado foi comparado com o *pixel* equivalente no quadro original, através da seguinte fórmula:

$$PSNR = 10\log_{10}\frac{255^2}{\frac{1}{Nl}\frac{1}{Nc}\sum_{x=0}^{Nc-1}[p'(y,x)-p(y,x)]^2}$$

Na fórmula, *Nl* e *Nc* representam o número de linhas e colunas de cada quadro, respectivamente. As expressões p(y,x) e p'(y,x) representam um *pixel* numa mesma posição do quadro original e do quadro decodificado.

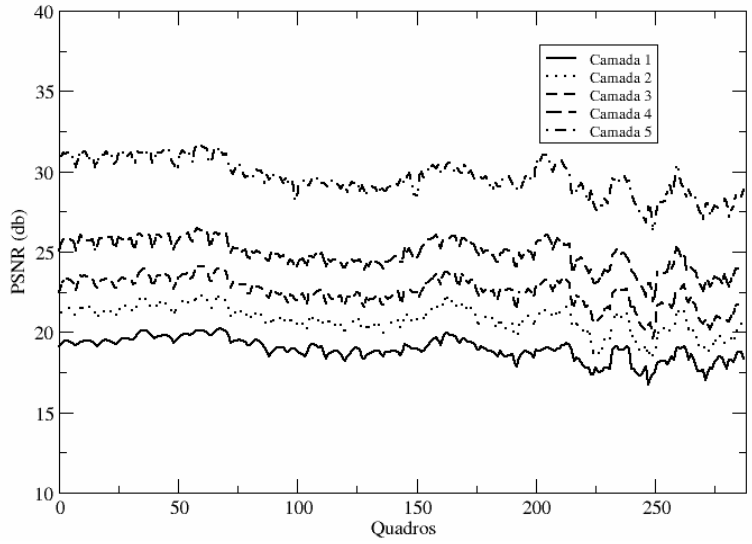


FIGURA 8.15 – PSNR do vídeo *Forest*

O valor resultante mostra a relação, em dB, entre o sinal gerado pelo quadro original e o quadro decodificado. Quanto mais parecidos forem os sinais, maior será o valor PSNR obtido. Quanto mais distorcido o sinal, maior será a diferença e menor será o valor resultante. A figura 8.15 mostra o cálculo PSNR para o vídeo *Forest*. Todos os vídeos atingiram, na média, mais do que 30dB para todas as camadas, com exceção do vídeo *Night*, que ficou em aproximadamente 25 dB.

8.5 Extensão do VEBIT para suporte à transmissão ao vivo

O VEBIT sofreu uma melhoria através de uma monografia de especialização do curso de Redes de Computadores e Internet da Unisinos. Na versão original, a entrada para o VEBIT era baseada somente em arquivos do tipo "PPM", conforme descrição anterior neste capítulo. Herbert [HER 2003] acrescentou suporte para que o algoritmo efetuasse captura direto de placas de vídeo existentes no computador, habilitando o seu funcionamento em tempo real. A figura 8.16 ilustra essa possibilidade, representada através do módulo desenvolvido na monografia, e denominado *VEBcapture*.

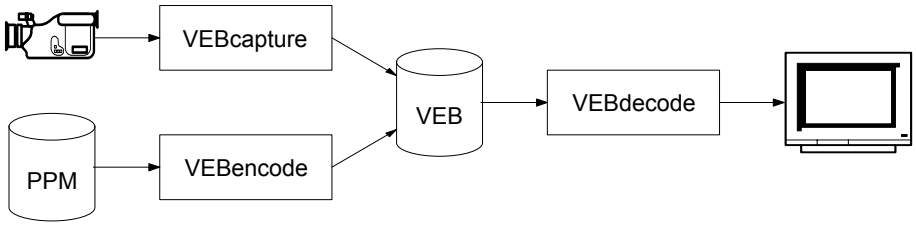


FIGURA 8.16 – Extensão ao VEBIT para suporte de vídeo ao vivo

Uma simplificação efetuada foi a utilização da biblioteca "Vídeo for Windows", o que criou certas dificuldades na hora de capturar os quadros da placa de vídeo. Alguns deles eram descartados por problemas de processamento e nem chegavam ao programa.

Outra dificuldade foi a necessidade de inverter os quadros lidos em RGB para ficar compatível com o PPM. A figura 8.17 mostra que o quadro lido em RGB possui o padrão de cor invertido, ou seja, no formato BGR. Além disso, as linhas de imagem estão invertidas. Para efetuar a compatibilização, foi desenvolvida uma rotina de conversão entre os formatos.

FIGURA 8.17 – Diferença de padrão RGB para PPM

Outro programa desenvolvido para permitir uma análise mais repetitiva de diversos vídeos, convertendo os mesmos para PPM, foi o "*avi2bmp*", que está

representado na figura 8.18. Existem diversas ferramentas na Internet para o processo de extração de DVD, gerando vídeos AVI, que podem ser convertidos para PPM através desse método.

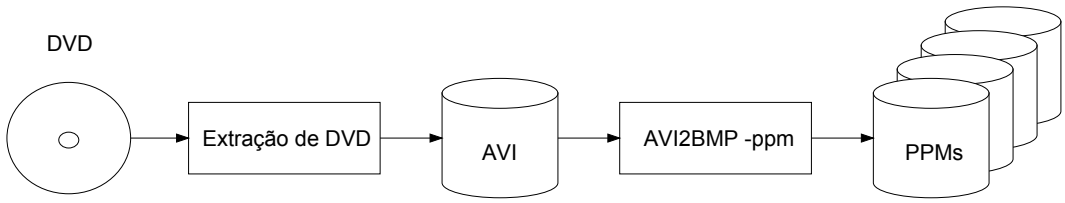


FIGURA 8.18 – Processo de obtenção dos vídeos e conversão para PPM

Diversos resultados foram obtidos, mostrando que é possível capturar ao vivo vídeos em resoluções de 176x144 e 352x288, a 24 quadros por segundo. Detalhes podem ser obtidos em Herbert [HER 2003].

8.6 Integração no SAM

Após a integração do VEBIT com o SAM, o algoritmo de controle de congestionamento ALM e o decodificador devem trabalhar em sintonia. Quando o receptor se cadastra numa determinada sessão de transmissão de vídeo, o algoritmo ALM determina em quantas camadas este receptor pode se inscrever. O número de camadas pode ser maior ou menor, de acordo com a topologia de rede deste usuário específico, bem como o nível de congestionamento existente na transmissão, conforme visto nos capítulos 4, 5 e 6.

Quando o receptor se inscreve em uma nova camada, passa a receber dados referentes ao *bitplane* correspondente. O decodificador utiliza esses dados para melhorar a qualidade do sinal no vídeo gerado. Similarmente, quando o receptor se desinscreve de alguma camada, o receptor deixa de ter as informações do *bitplane* correspondente, portanto, o vídeo gerado vai possuir menos qualidade.

Uma avaliação precisa do desconforto visual causado pela variação freqüente de qualidade de sinal (ou de camadas) deverá ser analisada após a integração. Na metodologia de teste do algoritmo, um dos parâmetros analisados é a estabilidade da transmissão, e esse parâmetro foi escolhido justamente para dar uma idéia da quantidade de variação por minuto das camadas (VCM), permitindo avaliar o impacto sobre o usuário.

8.6.1 Conseqüência dos erros

Caso aconteçam erros na transmissão para uma determinada camada, ou seja, um determinado pacote seja perdido, as informações referentes aos pixels do *bitplane* correspondente não chegam ao decodificador, diminuindo a qualidade do vídeo, entretanto, não inviabilizando a decodificação, pois o decodificador (caso esteja cadastrado em mais de uma camada) tem as informações dos outros *bitplanes* para utilizar no processo.

Como a codificação é feita em “cubos” 8x8x8, um pacote perdido vai afetar os 8

quadros subseqüentes, na região do vídeo onde os *pixels* foram perdidos. Assim que o próximo pacote do mesmo *bitplane* chegar corretamente, a decodificação segue normalmente e o decodificador passa a exibir o vídeo corretamente.

8.7 Conclusões e melhorias futuras

O presente capítulo apresentou o codificador em camadas VEBIT, que será integrado ao SAM. O algoritmo foi desenvolvido por Bruno [BRU 2003], sendo baseado em *bitplanes*, a fim de permitir escalabilidade granular fina. Outra característica é que o algoritmo utiliza a transformada 3D-DCT devido à sua simplicidade, analisando três dimensões do sinal através de cubos com 8x8x8 *pixels*. Cada cubo representa 8 *pixels* de largura, 8 de altura e 8 no domínio tempo.

Herbert [HER 2003] efetuou uma modificação no algoritmo original, inserindo suporte para captura diretamente a partir da placa de captura de vídeo, habilitando o VEBIT para funcionamento em tempo real.

Um dos requisitos do VEBIT foi a transmissão em camadas de uma aula remota síncrona na resolução QCIF (176x144) a 24 quadros por segundo, em um número não superior a 5 camadas. A implementação efetuada por [BRU 2003] dividiu o sinal em 5 *bitplanes*, ou 5 camadas.

Outro requisito era que o algoritmo funcionasse em tempo real, e essa possibilidade foi mostrada no item 8.4.1, onde se verificou a possibilidade tanto da codificação como da decodificação de um vídeo QCIF. Foram testados 5 diferentes vídeos, cada um com uma característica própria. Melhores resultados podem ser obtidos otimizando o algoritmo de DCT, pois essas rotinas são responsáveis por mais de 50% do consumo de tempo, tanto da codificação como da decodificação.

Em termos de adaptação para receptores heterogêneos, o objetivo era garantir a recepção em baixa velocidade, ou seja, para enlaces com menos de 56 kbit/s (conexão via modem). A demonstração dessa possibilidade ficou clara nos gráficos mostrados no item 8.4.3.

Evoluções ao algoritmo podem ser feitas, como a possibilidade de alteração dinâmica no número de camadas, permitindo adaptação única a determinado receptor, o que é importante em transmissões *unicast*. Além disso, o algoritmo de compressão pode ser otimizado, gerando vídeos com maior qualidade, nas mesmas taxas. Em termos de desempenho, o codificador 3D-DCT pode ser otimizado, pois responde pelo maior gasto de processamento. Para fins de portabilidade, pode ser desenvolvida uma API de programação, permitindo a fácil integração do algoritmo em outras aplicações.

Outro trabalho futuro visando aumentar o número de camadas poderia ser a divisão de camadas por *bitplanes* juntamente com taxas diferentes de quadros por segundo. Uma alternativa para fazer isso seria a execução do algoritmo primeiramente para os quadros pares e depois para os quadros ímpares, gerando 10 camadas. Caso o receptor se cadastrasse somente numa camada, receberia a transmissão a 12 quadros por segundo ao invés de 24.

9 Conclusões

Esta Tese descreveu em detalhes o SAM (Sistema Adaptativo para Multimídia), que permite aos receptores de uma transmissão multimídia adaptarem-se de acordo com o congestionamento na rede e possibilidades da máquina. Dessa forma, dois receptores localizados em redes com larguras de banda diferentes podem assistir à transmissão na melhor qualidade possível para os mesmos.

O método utilizado pelo SAM para obter adaptabilidade em redes heterogêneas é a transmissão multicast em camadas, onde cada camada é transmitida como um grupo multicast independente. O codificador em camadas utilizado foi criado através de uma dissertação de mestrado desenvolvida em paralelo com esta Tese, conforme detalhamento no capítulo 8.

O elemento central do SAM se localiza no receptor, sendo o algoritmo de controle de congestionamento definido como ALM (*Adaptive Layered Multicast*). Esse algoritmo determina o grau de estabilidade do sistema, bem como sua eqüidade com tráfego concorrente, tempo de adaptação e nível de perdas. No âmbito desta Tese, definiu-se dois algoritmos de controle de congestionamento: o ALMP (ALM para redes privativas) e o ALMTF (ALM TCP *friendly*), que foram detalhados nos capítulos 5 e 6, respectivamente.

Os principais problemas que os algoritmos tiveram que lidar foram os seguintes:

- Como fazer para que o sistema se adapte automaticamente às condições de rede e máquina do usuário?
- Como fazer para que o sistema utilize uma parcela eqüitativa de tráfego na Internet atual?
- Como fazer para que o sistema se mantenha estável ao longo do tempo?

O método básico de funcionamento dos algoritmos é inferir a banda que o receptor pode utilizar, e se inscrever no número de camadas (grupos multicast) equivalentes às possibilidades do receptor. Os dois algoritmos utilizam a técnica de pares de pacotes (descrita no Anexo A), a fim de evitar que o receptor tente utilizar uma banda acima das possibilidades físicas da rede.

O algoritmo ALMP, descrito no capítulo 5, tem como principal objetivo a transmissão de fluxos multimídia em multicast através de uma rede virtual privativa, que poderia ser criada na Internet através de um mecanismo de QoS, que separaria a rede tradicional (TCP) da rede multimídia.

Através das comparações entre os algoritmos ALMP, ALMTF, RLM, RLC e TFMCC, resumidas na tabela 7.6, pode-se concluir que o ALMP possui uma série de vantagens em relação aos outros algoritmos, como sua estabilidade, transmissão de forma eqüitativa com suas próprias sessões e escalabilidade para um grande número de receptores, já que não existe necessidade de comunicação do receptor para o transmissor. Os índices de VCM (Variações de Camada por Minuto) ficaram bastante baixos, e o índice de justiça (*Fairness Index*) ficou próximo a um nas situações em que

o algoritmo foi proposto, ou seja, sem tráfego TCP concorrente. Na verdade, índices dessa ordem só puderam ser obtidos porque o algoritmo foi projetado para concorrer com seu próprio tráfego, sem a existência de fluxos TCP concorrentes. Quando entra um fluxo TCP, ele elimina o fluxo ALMP, pois o TCP é bem mais agressivo.

O algoritmo implementado do ALMP mostrou semelhança com os resultados obtidos nas simulações, e verificou-se na prática o fato de uma máquina sobrecarregada não conseguir ler todos os pacotes, adaptando-se automaticamente às suas condições.

O principal problema do ALMP é quando se pretende utilizar o sistema SAM na Internet, com tráfego TCP concorrente. Para isso, foi criado o algoritmo ALMTF, descrito no capítulo 6.

Para transmitir de forma eqüitativa com fluxos TCP concorrentes, foi necessário seguir o modelo de incremento e decremento de banda semelhante ao utilizado pelo TCP, com suavizações onde possível a fim de melhorar a estabilidade do algoritmo. Para tanto, o ALMTF utilizou dois mecanismos de controle de fluxo: a) baseado em "janela de congestionamento"; b) baseado na equação do TCP. Juntos, esses mecanismos permitiram ao algoritmo inferir a sua banda eqüitativa na rede, mesmo na presença de tráfego concorrente.

Através das simulações efetuadas, conclui-se que o algoritmo ALMTF se adapta em ambientes heterogêneos, permite escalabilidade de tráfego, transmite de forma eqüitativa com seu próprio tráfego e também com tráfego TCP concorrente. Na verdade, o ALMTF foi o único algoritmo que adaptou-se em todas as condições testadas nas comparações efetuadas no capítulo 7, como pode ser constatado pela tabela 7.6. Os índices de eqüidade de tráfego (*Fairness Index*) do ALMTF ficaram muito próximos do ideal em todos os casos analisados. Entretanto, para obter essa eqüidade com o TCP, ele ficou pior que o ALMP nos quesitos escalabilidade e estabilidade.

Em linhas gerais, pode-se concluir que o ALMP seria o algoritmo mais indicado para utilização em redes privativas (sem a presença do TCP concorrendo), devido à sua característica de adaptação em ambientes heterogêneos, aliada à sua alta estabilidade, escalabilidade e eqüidade com tráfego concorrente do seu próprio tipo, independente do mesmo iniciar em momentos diferentes.

Caso seja necessário utilizar a transmissão em ambientes que necessitem concorrer com tráfego TCP, o mais indicado seria o ALMTF, principalmente devido à sua capacidade de dividir banda de forma eqüitativa com tráfego TCP, não importando se o mesmo inicia antes ou depois. Além disso, o ALMTF mostrou possibilidade de adaptação em ambientes heterogêneos, tempo de adaptação rápido, boa escalabilidade e alto grau de eqüidade com seu próprio tráfego.

Conclusões mais específicas sobre cada algoritmo e sobre as comparações podem ser encontradas nos próprios capítulos onde tais assuntos foram detalhados, ou seja, nos itens 5.4 (Conclusões sobre o ALMP), 6.3 (Conclusões sobre o ALMTF) e 7.7 (Conclusões sobre as comparações efetuadas).

A utilização de pares de pacotes mostrou aumentar a estabilidade dos algoritmos, entretanto, seu uso é particularmente útil quando a inscrição na próxima camada gera uma banda superior à do enlace. Nos casos onde há concorrência entre vários fluxos ALM diferentes, percebe-se pouca diferença em relação à não utilização

dos pares de pacotes. Outra conclusão é que o uso de pares de pacotes gera um congestionamento levemente superior nas filas dos roteadores intermediários, devido à rajada de dois pacotes. Isso pode ser observado, por exemplo, nas figuras 5.12, 5.26 e 5.27.

Pode-se inferir que, num ambiente real, é melhor a utilização de poucas camadas, caso seja possível, pois isso diminui o número de fluxos nas filas dos roteadores intermediários, diminuindo também o tráfego de controle e o número de grupos multicast necessários ao algoritmo.

Outra conclusão obtida neste trabalho é que seria bastante útil a utilização de descarte por prioridades, onde as camadas superiores (menos prioritárias) teriam uma probabilidade maior de descarte. Na ocorrência de congestionamento, o roteador iria descartar primeiramente os pacotes das camadas superiores, diminuindo a perda das camadas mais baixas, que são mais importantes. Além disso, os algoritmos muitas vezes ingressam na camada superior apenas para verificar se a rede tem condições de suportar o acréscimo de banda, e tais pacotes podem ser descartados sem problemas.

Através das implementações e de simulações, chegou-se à conclusão que, a fim de que o sistema seja eqüitativo, os receptores inscritos em um maior número de camadas devem ter uma sensibilidade maior de congestionamento em relação aos receptores inscritos num menor número de camadas. Isso faz com que, na ocorrência de congestionamento, os receptores que estão utilizando mais banda diminuam sua taxa, permitindo aos receptores que estão entrando um acréscimo de taxa, até a igualdade. Cada um dos algoritmos faz isso de uma forma diferente. O TCP utiliza um mecanismo conhecido como AIMD (*Additive Increase Multiplicative Decrease*), o ALMP utiliza um método logarítmico, e o ALMTF utiliza um mecanismo dez vezes menos agressivo que o TCP, e ainda oferece maior número de tentativas de subir camada aos receptores inscritos em menos camadas. Desses três métodos, o utilizado pelo TCP é o mais agressivo, gerando uma maior variação nos fluxos.

9.1 Contribuições desta Tese

As principais contribuições desta Tese podem ser resumidas nos seguintes itens:

- Criação de um modelo e implementação de um sistema para transmissão de vídeo de forma adaptativa ao usuário, denominado SAM (Sistema Adaptativo para Multimídia);
- Criação de um algoritmo de controle de congestionamento, denominado ALMP, destinado ao uso em redes privativas, ou seja, sem concorrência com o TCP. Esse algoritmo se mostrou bastante estável, se adapta bem em ambientes heterogêneos e transmite de forma eqüitativa com seu próprio tráfego, como visto nos capítulos 5 e 7;
- Criação de um algoritmo de controle de congestionamento, denominado ALMTF, destinado à Internet atual, que transmite de forma eqüitativa com tráfego TCP concorrente. Esse algoritmo foi simulado em diversas condições de rede, e adaptou-se adequadamente em todos os casos testados, conforme visto nos capítulos 6 e 7;
- Criação de um novo algoritmo, que transmite de forma estável, eqüitativa e com

rápido tempo de adaptação, porém, necessita apoio dos roteadores intermediários. Esse algoritmo está descrito em [ROE 2003j], e foi denominado TCP-Stable;
- Análise aprofundada do método de pares de pacotes, através de simulações e implementação prática, mostrando que sua utilização realmente permite a descoberta da banda máxima na rede. Além disso, adaptação do método para transmissão multicast e criação de um filtro para determinação de banda máxima na rede. Essa análise está descrita no Anexo A;
- Constatação de que, se um fluxo for dez vezes mais lento que o TCP, tanto para incremento de taxa como para decremento de taxa, o mesmo mantém as mesmas características do TCP, porém de forma mais estável. Observou-se também um aumento no número de perdas.

9.2 Trabalhos futuros

Esta Tese trouxe uma série de contribuições originais, conforme conclusões apresentadas no item anterior, tendo um papel importante na área de transmissão de vídeo com adaptabilidade. Como extensões ao que foi desenvolvido até o momento, existem algumas sugestões, que são descritas a seguir.

- **Utilização de ECN no SAM**: ao invés de descartar os pacotes, o roteador utiliza fila RED e simplesmente marca os pacotes quando a fila começa a atingir níveis críticos. Dessa forma, o número de descartes é menor e a adaptação acontece da mesma forma;
- **Comparação dos resultados das simulações com a implementação**: Através das simulações obteve-se um conjunto de resultados, que devem ser comparados com os resultados obtidos através da implementação, a fim de verificar a exatidão conseguida com as simulações.
- **Atualização do ALMP com evoluções do ALMTF**: o filtro dos pares de pacotes e o método de sincronização entre receptores foram criados no ALMTF, mas poderiam ser portados para o ALMP. Isso não foi feito pois haveria necessidade de repetir todas as simulações.
- **Utilização de QoS com prioridade por camada**: nesse caso, as camadas mais baixas teriam maior prioridade, assim, o sistema descartaria antes as camadas mais altas, não interferindo na qualidade de recepção. O receptor deveria esperar certo tempo após subir camada para mostrar a qualidade ao usuário, a fim de evitar variações na qualidade da imagem. O transmissor deve escolher as camadas de mais alta prioridade (camadas opcionais ou as mais básicas num vídeo ou áudio dividido em camadas);
- **Uso de *Proxy***: criar uma hierarquia na transmissão onde um *Proxy* intermediário mantém o vídeo em sua memória durante certo tempo. Caso algum receptor entre na rede atrasado em relação à transmissão multicast, este *Proxy* envia o vídeo diretamente a este receptor;
- ***Awareness***: análise semântica da interatividade obtida na transmissão a fim de automatizar o sistema e responder as perguntas automaticamente. Assim, caso exista uma aula com 1 professor e 1000 alunos simultâneos fazendo perguntas, o sistema deve analisar as perguntas e respondê-las automaticamente caso já tenha sido

perguntada. Caso contrário, deve combinar perguntas semelhantes antes de passar ao professor;

- **Adaptação às características da máquina**: a adaptação em relação à capacidade da máquina (CPU) foi modelada, mas não implementada no sistema;
- **Implementação do aplicativo para IPv6**: a implementação do sistema pode ser feita também com IPv6 a fim de disponibilizar o aplicativo para essa tecnologia;
- **Interatividade com o transmissor no nível de aplicação**: é possível criar um canal multicast ou mesmo unicast (de texto) com o transmissor, onde os clientes podem fazer perguntas ou efetuarem comentários. Isso é útil quando um receptor quer interagir com o transmissor, seja em uma aula a distância, uma palestra assistida remota ou mesmo para participar num programa de TV interativa. As alternativas propostas para interatividade são: a) Uso de transmissão multicast: nesse caso, um determinado grupo multicast é destinado ao retorno de perguntas pelos receptores. Estes podem fazer as perguntas via texto, voz ou videoconferência. A vantagem é que todos receptores cadastrados nesse grupo vão receber a informação, facilitando a interação e o debate. O problema é a escalabilidade do sistema, principalmente para videoconferência. Tem que ser definido um sistema de limitação de banda para esse grupo multicast, a fim de que esse tráfego não perturbe o recebimento da transmissão; b) Uso de transmissão unicast: nesse caso, o sistema tem sérias limitações de escalabilidade, mas pode ser muito mais simples e tão eficaz quanto o uso de multicast. A proposta aqui é a existência de uma sala de *chat* onde os participantes fariam as perguntas e os debates. Como a transmissão seria limitada a texto, não afetaria significativamente o tráfego da rede. Outra alternativa é a transmissão da pergunta ao palestrante diretamente, através de um *script* em uma página *web*. O palestrante (ou seu mediador) receberia a pergunta e daria a resposta através da transmissão de áudio. Como a pergunta iria somente ao palestrante, um outro *script* deve ser criado a fim de que os interessados em perguntar vejam (ou ouçam) as perguntas já feitas para não repetirem questões.

Anexo A Análise do mecanismo de pares de pacotes

Este anexo tem como objetivo analisar em detalhes o funcionamento do mecanismo de pares de pacotes e trens de pacotes, utilizado para descoberta de banda da rede, permitindo a inferência de sua velocidade máxima. Uma versão resumida deste estudo foi publicada em [ROE 2003b]. Os resultados podem ser úteis para qualquer algoritmo que necessite inferência de banda entre dois pontos da rede, porém, pretende-se analisar a viabilidade da utilização deste mecanismo no sistema SAM, proposto nesta Tese e descrito no capítulo 4 (O SAM: Sistema Adaptativo para Multimídia).

Para que novos algoritmos baseados em UDP sejam utilizados sem prejudicar as aplicações existentes, os mesmos devem inferir sua fatia de banda e efetuar controle de congestionamento de forma amigável com o TCP, conforme descrito no item 2.5 (Eqüidade de tráfego). Uma informação muitas vezes útil nesse caso é a banda máxima entre transmissor e receptor, bem como a banda eqüitativa da rede. É nesse ponto que o mecanismo de pares de pacotes pode auxiliar, pois através dessa técnica é possível inferir a largura de banda máxima, tanto em unicast como em multicast.

No caso do unicast, o próprio transmissor pode inferir a banda, esperando o ACK dos pacotes do par e se baseando no espaçamento entre eles para calcular a velocidade máxima da rede. Neste caso, deve-se ter cuidado para o receptor não retardar o envio do ACK (técnica de *delayed* ACK, definida na RFC 1122 [BRA 89]). O uso dos ACKs possui duas características negativas [LEG 2000], [LAI 2000]: a) não há como saber se o congestionamento é no sentido transmissor-receptor ou vice-versa, pois o espaçamento pode ter sido causado no canal reverso; b) a reação ao congestionamento é mais lenta, já que o transmissor deve esperar os ACKs para poder reduzir sua taxa de transmissão.

No caso do multicast, normalmente é o receptor que infere a largura de banda da rede, baseado no espaçamento do par de pacotes. Nesse caso, a banda máxima medida é no sentido transmissor-receptor, e como o receptor não utiliza ACKs, diminui a necessidade de geração de tráfego, entretanto, exige um receptor habilitado a receber e calcular a banda máxima a partir dos pares de pacotes, ou seja, o teste não pode ser efetuado a partir de mensagens padrão de *echo*.

A maior parte dos trabalhos relacionados busca a implementação de uma ferramenta que obtenha rapidamente uma resposta para a banda máxima entre dois pontos, gerando tráfego na rede. Esse não é o caso para a utilização que se pretende.

A principal contribuição deste anexo é efetuar uma análise da viabilidade do mecanismo de pares de pacotes orientado a receptor, via UDP, sem a utilização de ACKs e com geração espaçada de pares de pacotes, ou seja, adequado para uso em multicast e sem gerar rajadas de tráfego na rede.

A fim de validar o mecanismo e verificar a possibilidade de utilizar pares de pacotes no algoritmo de controle de congestionamento sendo proposto nesta Tese, diversas simulações foram efetuadas, visando obter a precisão das medições de inferência de banda na Internet atual. Posteriormente, efetuou-se a implementação do sistema e diversos testes foram realizados, tanto em um ambiente controlado de

laboratório como na Internet, com medições via modem de 56 kbit/s, ADSL (*Asymmetric Digital Subscriber Line*), Intranet e Internet2. Várias situações foram analisadas, como o seu uso em enlaces de diferentes velocidades, com vários tamanhos de pacotes e com diferentes RTTs (*Round Trip Time*).

O restante deste anexo é organizado da seguinte forma: o item A.1 (Trabalhos Relacionados) mostra alguns dos trabalhos já desenvolvidos sobre o tema de pares de pacotes ou trem de pacotes. O item A.2 (Descrição do método de pares de pacotes) detalha o funcionamento do método. No item A.3 (Simulações efetuadas), o ambiente de simulação é descrito, bem como os resultados atingidos. O mesmo acontece no item A.4 (Implementação do sistema), porém, numa implementação testada em um ambiente controlado e no ambiente da Internet. No item A.5 (Filtro de amostras), um filtro é proposto a fim de obter um resultado mais preciso. O item A.6 (Conclusões) apresenta as conclusões obtidas com os resultados.

A.1 Trabalhos Relacionados

Um dos primeiros mecanismos de controle de fluxo através de pares de pacotes foi proposto em Keshav [KES 91]. Nele o autor assume que todos os roteadores utilizam filas do tipo "*round-robin*", e todos os fluxos são servidos igualmente. Essa disciplina foi definida em detalhes e denominada "Rate Allocating Servers". Utilizando essa infra-estrutura, o nível de transporte do transmissor envia os pacotes em pares, medindo o espaçamento dos ACKs na chegada e calculando a banda eqüitativa na rede. Essa informação é utilizada em seu esquema de controle de fluxo baseado em taxa.

Outro trabalho relacionado é o proposto por Carter [CAR 96], onde o método de pares de pacotes é utilizado com o objetivo de descobrir a maior largura de banda entre um conjunto de servidores alternativos, permitindo a escolha do servidor com a maior taxa de transmissão. Para validar os argumentos, Carter desenvolveu uma ferramenta chamada *bprobe*, que tem como objetivo estimar a largura de banda máxima no gargalo da rede, e outra chamada *cprobe*, cujo objetivo é estimar a largura de banda eqüitativa entre dois pontos. Na ferramenta *bprobe*, são transmitidos aproximadamente 50 pares de pacotes ICMP do tipo *echo*, com vários tamanhos. O espaçamento entre os ACKs de retorno é medido, e o resultado final da banda é a União dos resultados próximos. Na ferramenta *cprobe*, são transmitidos 4 rajadas de 8 pacotes, e a banda é calculada para cada rajada, prevenindo perda de pacotes em alguma medição. O resultado é a média da banda obtida, ou a banda eqüitativa da transmissão.

Kevin Lai [LAI 2000] utilizou um mecanismo denominado "*packet tailgating*", onde o primeiro pacote do par é ICMP com TTL reduzido, sendo encaminhado juntamente com outro de tamanho pequeno, fazendo com que o segundo pacote seja sempre enfileirado junto com o primeiro, até o descarte do primeiro pelo roteador (quando seu TTL expira). Com base nas mensagens de retorno dos roteadores e ACKs do pacote de *tailgating*, Lai propôs a ferramenta *nettimer*.

Vern Paxson [PAX 97] definiu uma ferramenta denominada PBM (*Packet Bunch Modes*), onde a estimativa de banda é efetuada através de várias rajadas de tamanhos diferentes (trens de pacotes) a fim de aumentar a precisão na medição da banda máxima da rede.

No trabalho de Sylvia Ratnasamy e Steven McCanne definido em [RAT 99], o objetivo foi inferir a árvore de roteamento multicast e a máxima largura de banda entre transmissor e receptores. Utilizou-se pares de pacotes para inferência de máxima largura de banda. O transmissor envia um conjunto de pacotes num determinado grupo multicast, sendo que todos pacotes contêm o *timestamp* do transmissor. O receptor mede o espaçamento entre os pacotes na chegada e aplica um filtro a fim de obter um resultado mais preciso. O filtro funciona no receptor da seguinte forma:

- Descarta os pacotes que não se espaçaram na rede, ou seja, quando o espaçamento dos pacotes no transmissor (visto pelo *timestamp*) é maior que o espaçamento medido na chegada;
- Descarta pares de pacotes com chegadas fora de ordem ou com pacotes perdidos;
- Descarta medições com espaçamento maior que duas vezes um espaçamento considerado válido.

No mínimo 70% dos pares de pacotes devem chegar com informações válidas (sem precisar descartar), caso contrário, é considerada inválida toda a medição. Das amostras válidas, calcula *bwmax*, que corresponde ao valor que aconteceu com maior freqüência (com cada amostra assumindo uma faixa de ±5% do seu valor). Se o número de vezes que ocorreu *bwmax* for maior do que 60% de todas as outras amostras, então assume que o resultado é válido, caso contrário, invalida toda a medição. Simulações mostraram bons resultados.

Outro trabalho relacionado é o protocolo TCP Westwood (TCPW) [GER 2001], que utilizou a taxa de retorno dos ACKs e o espaçamento entre eles para determinar a banda eqüitativa da transmissão. Com base nessa taxa, o TCPW utiliza um mecanismo modificado, o AIAD (*Additive Increase Adaptive Decrease*) evitando diminuir pela metade a taxa de transmissão quando ocorre uma perda, já que nem sempre uma perda significa congestionamento, não precisando então diminuir a taxa de envio de dados. Simulações mostram uma melhora de vazão de até 578% em enlaces wireless sem prejuízo das conexões TCP Reno em andamento. Em testes reais, o protocolo mostrou um aumento de vazão entre 31% e 185% numa rede de alta velocidade da NASA, e de 47% na Internet quando a conexão TCP inclui um enlace de satélite.

A.2 Descrição do método de pares de pacotes

Um efeito bem conhecido nas redes de computadores é o espaçamento em pacotes adjacentes causado pelo limite físico de banda no enlace de menor velocidade entre origem e destino, ou seja, no gargalo da rede [KES 91], [JAC 88]. A essência do mecanismo pode ser ilustrada através da figura A.1, onde se pode ver que, quando passam por um gargalo na ausência de tráfego concorrente, os pacotes sofrem um espaçamento que é preservado nos enlaces de maior velocidade. Através desse espaçamento e do tamanho dos pacotes, é possível inferir a largura de banda entre origem e destino.

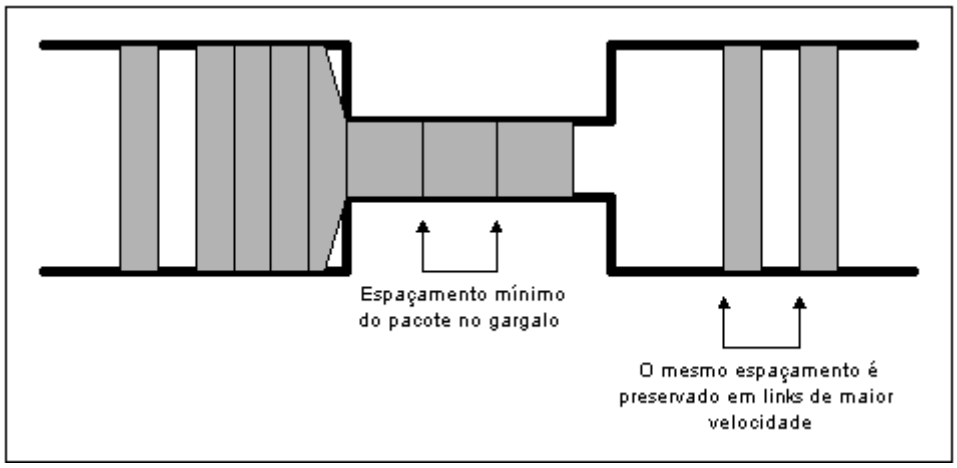


FIGURA A.1 – Funcionamento básico do mecanismo de pares de pacotes

Para comprovar o funcionamento numa topologia multicast simples, foi modificado o código fonte do simulador NS2 a fim de que ele pudesse transmitir em pares de pacotes, e foi efetuada uma simulação, conforme topologia mostrada na figura A.2. Como pode-se ver, os pacotes são transmitidos em pares através de um enlace de 2 Mbit/s, e sofrem um espaçamento no gargalo da rede, representado pelo roteador 1, que possui uma largura de banda de 1 Mbit/s. Esse espaçamento permanece até o receptor R1, e pode ser visto claramente no enlace de 10 Mbit/s.

O tamanho dos pacotes transmitidos foi de 500 bytes, e o espaçamento medido no receptor R1, do início da chegada do primeiro pacote até o início da chegada do segundo pacote foi de 4 ms (visto através do simulador, tempos de 2,738691 s - 2,734691 s = 4 ms). Com esses dados, o receptor infere que o máximo que a rede consegue transmitir é 500 bytes (4.000 bits) em 4 ms, ou seja, 1 Mbit/s, que é equivalente ao enlace de menor velocidade, ou o gargalo da rede.

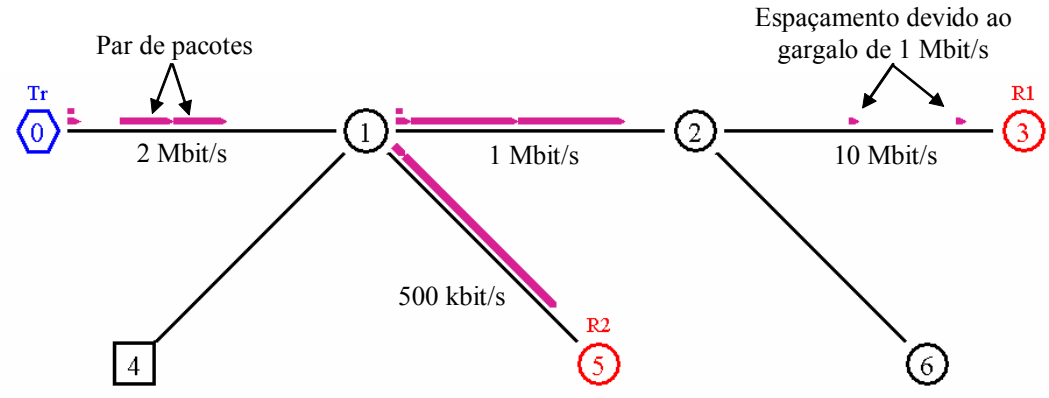


FIGURA A.2 – Exemplo de inferência de banda por pares de pacotes

Entretanto, numa rede real, o tráfego concorrente dificulta a utilização de pares de pacotes de forma precisa, especialmente quando se utilizam filas do tipo FIFO, que é a mais utilizada na Internet [CRO 2000]. Alguns erros de inferência no receptor são descritos a seguir:

- **Inferência de banda maior que a real**: o par de pacotes sofre um espaçamento no gargalo da rede, conforme mostra a figura A.3 (no item (a) e enlace de 1 Mbit/s). Entretanto, caso exista tráfego concorrente (representado pelos pacotes brancos) após o gargalo da rede, o espaçamento pode ser eliminado na fila do roteador 2. Isso faz com que a transmissão do par de pacotes no enlace de 10 Mbit/s aconteça sem espaçamento algum, fazendo o receptor inferir uma largura de banda de 10 Mbit/s (no pior caso) ao invés de 1 Mbit/s.
- **Inferência de banda menor que a real**: os roteadores recebem tráfego de várias interfaces, podendo acontecer de inserir um pacote entre o par de pacotes transmitido, provocando um "escorregamento" no segundo pacote do par, levando a um espaçamento maior do que aconteceria devido simplesmente ao tráfego concorrente. Caso isso aconteça no gargalo, como, por exemplo, no roteador 1 da figura A.3 (item b), o receptor vai inferir uma largura de banda menor do que o real, pois o espaçamento é maior do que deveria. De acordo com a figura, com pacotes de 500 bytes, o espaçamento para inferir uma banda eqüitativa deveria ser 4 ms, gerando uma inferência de banda de 1 Mbit/s, entretanto, o espaçamento é de 12 ms, e a banda inferida é de 333 kbit/s, o que é menor do que o real. Outra causa de inferência de banda menor pode ser devido aos atrasos gerados pelo Sistema Operacional (os pares de pacotes podem não ser transmitidos um após o outro, no caso da geração de uma chamada de sistema entre eles) ou pela concorrência da rede local (a taxa de transferência do Ethernet compartilhado não é determinística).

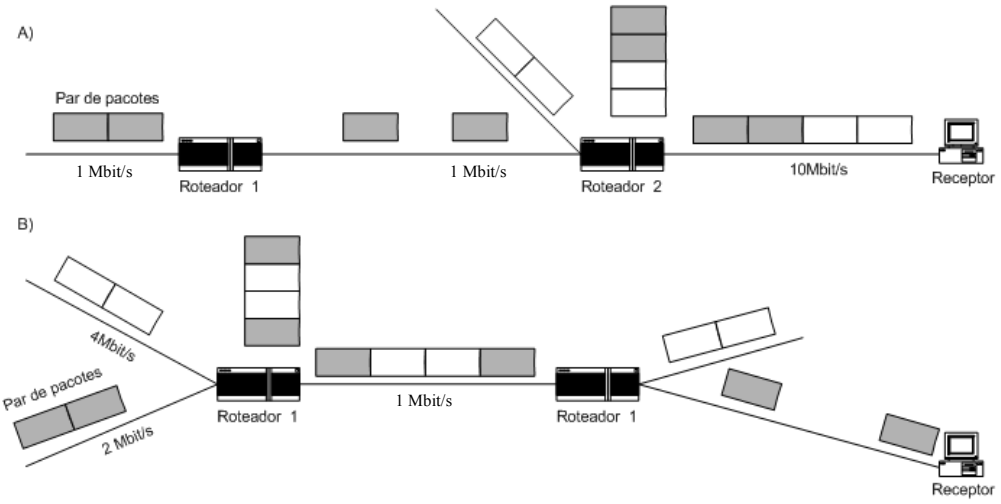


FIGURA A.3 – Problemas no uso de pares de pacotes em redes reais

Caso o roteador utilize uma política de filas que aproxime o conceito de GPS (*Generic Processor Sharing* – [ROE 2003h]), a distribuição da transmissão de dados será mais precisa entre os diversos fluxos que passam pelo roteador, permitindo maior precisão na inferência de banda através de pares de pacotes. O ideal seria a utilização de uma política de filas do tipo WF^2Q (*Worst Case Fair Weighted Fair Queuing* – [ROE 2003h), conforme proposto por Legout no desenvolvimento do protocolo PLM (*Packet Pair Receiver Driven Layered Multicast* - item 3.7). Entretanto, essa não é a realidade da Internet atual, portanto, tal alternativa não foi analisada nesta Tese.

Outras dificuldades relativas à utilização de pares de pacotes estão descritas em

[PAX 97] e resumidas a seguir:

- **Reordenamento de pacotes na rede:** devido a modificações de roteamento, algumas vezes os pacotes chegam ao receptor em uma ordem diferente do que foram transmitidos, e esta amostra reordenada deve ser descartada, pois não serve para cálculo de banda. Em um conjunto de testes realizado em 1995, 2% de todos pacotes de dados chegaram fora de ordem, o mesmo acontecendo com 0,6% dos ACKs. Os mesmos testes foram repetidos em 1996, e os resultados foram 0,3% e 0,1%, respectivamente;
- **Limitações na resolução de relógio:** caso o receptor não consiga uma precisão de leitura de tempo menor do que 10ms, então, para um tamanho de pacote de 500 bytes, o receptor pode não saber distinguir entre uma banda de 50 kbytes/s (quando lê 10ms) e uma banda infinita (quando lê zero);
- **Mudanças dinâmicas na velocidade máxima da rede:** caso a velocidade máxima da rede mude instantaneamente no meio do teste, o método vai calcular resultados errôneos. Isso pode acontecer quando um novo enlace é ligado, por exemplo;
- **Enlaces multi-canais:** alguns enlaces são compostos de vários enlaces físicos compondo um de maior velocidade (por exemplo, dois E1 operando em paralelo). Pode acontecer de um par de pacotes chegar ao roteador e o mesmo encaminhar cada pacote através de um enlace físico, comprometendo a medição no receptor.

Apesar das dificuldades listadas, Paxson chegou à conclusão que o método funciona bem quando orientado a receptor (o espaçamento entre os pares é medido somente numa direção), entretanto, quando o método é orientado a transmissor (baseado no espaçamento entre os ACKs de retorno), o método gera estimativas corretas somente 60% das vezes.

A.3 Simulações efetuadas

Como a presente Tese lida com codificação em camadas e adaptação no lado do receptor, as simulações efetuadas para verificar o funcionamento desta técnica foram efetuadas em multicast, e a inferência de banda efetuada pelo receptor, o que é mais rápido e mais preciso [PAX 97].

O simulador de redes utilizado foi o NS2, e os objetivos das simulações em relação ao método de pares de pacotes foram os seguintes:

- Analisar o funcionamento do método através de simulação;
- Analisar o impacto da variação no tamanho do pacote;
- Verificar se o método funciona para redes de baixa e alta velocidade;
- Analisar as conseqüências da variação do RTT.

As simulações foram efetuadas via orientação de um trabalho de conclusão da UNISINOS [FIN 2003], com o apoio do método já implementado no simulador NS2.

A fim de atingir os objetivos propostos para as simulações, utilizou-se a topologia mostrada na figura A.4. Na figura, estão representados os transmissores e receptores de pares de pacotes (PP) e de tráfego concorrente (TCP).

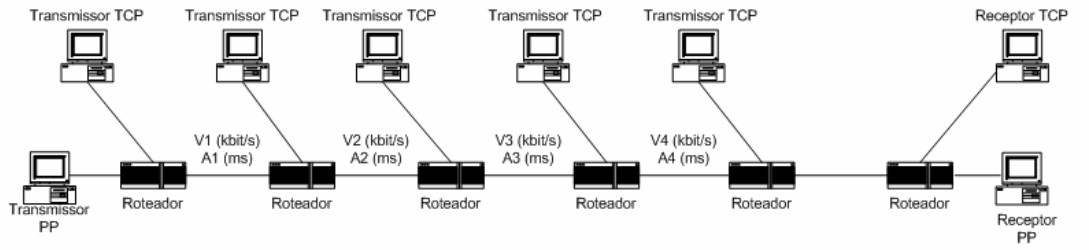


FIGURA A.4 – Topologia da rede simulada

Os enlaces entre os roteadores possuem velocidades e atrasos variáveis, sendo representado por V (em kbit/s) e A (em ms). Todos outros enlaces possuem uma largura de banda de 10 Mbit/s e atraso de 1 ms.

O código do simulador foi modificado de forma que o transmissor enviasse um par de pacotes aproximadamente a cada 500 ms. Cada par de pacotes contém um número de seqüência, que avisa ao receptor se o pacote é o primeiro ou o segundo do par. Quando chega o primeiro pacote, o receptor armazena o tempo de chegada. Quando chega o segundo, ele calcula a diferença de tempo e faz a inferência da banda, gravando o resultado num arquivo de *log*. Caso chegue um pacote com o número de seqüência errado, o receptor assume que houve erro na transmissão, ignorando a medida para esse par de pacotes.

Diversas simulações foram efetuadas a fim de responder cada um dos objetivos listados acima. Cada item a seguir trata da metodologia e resultados obtidos para cada um dos objetivos.

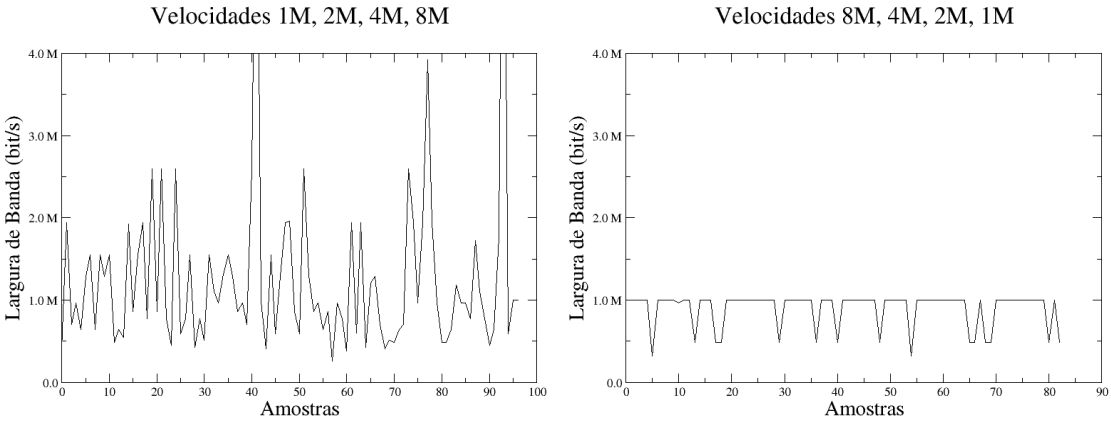
A.3.1 Análise do funcionamento do método através de simulação

A simulação a seguir visa simplesmente verificar a funcionalidade do método de pares de pacotes. Para isso, foram iniciadas 20 sessões FTP (quatro em cada transmissor TCP) concorrentemente com o transmissor de pares de pacotes, numa simulação de 50 segundos. Nesse conjunto de simulações, a velocidade dos enlaces intermediários (V1 a V4) foi de, respectivamente, 1Mbit/s, 2Mbit/s, 4Mbit/s e 8Mbit/s. Os valores foram escolhidos de forma a gerar os problemas descritos através da figura A.3. Posteriormente, as velocidades foram invertidas (8M, 4M, 2M, 1M). O atraso (A1 a A4) foi fixo em 1 ms, e o tamanho de todos pacotes foi de 1000 bytes.

Pode-se observar no gráfico “a” da figura A.5 os dois tipos de problema que fazem com que o receptor infira banda de forma errônea, conforme descrito anteriormente. O receptor inferiu banda erroneamente, chegando a ter picos de 8Mbit/s, quando o espaçamento entre os pares de pacotes desaparecia na fila do roteador 4 (anterior ao enlace de 8Mbit/s). Entretanto, pode-se ver que a média ficou próxima ao resultado correto, que é de 1 Mbit/s, levando a crer que seja possível criar um algoritmo para desprezar os picos e inferir a banda de forma aproximadamente correta.

O gráfico “b” teve as velocidades invertidas, portanto, o espaçamento dos pares de pacotes nunca diminuía. Esse é o motivo pelo qual nenhuma inferência deu acima de 1Mbit/s. Entretanto, quando pacotes TCP entravam no meio do par, causavam atraso e provocavam uma inferência de banda menor do que o gargalo da rede. Da mesma forma

que no primeiro gráfico, a média ficou próxima da realidade.



a) Velocidade crescente no enlace b) Velocidade decrescente no enlace

FIGURA A.5 – Inferência da banda do receptor de pares de pacotes

A.3.2 Análise do impacto da variação no tamanho do pacote

Para analisar o impacto da variação do tamanho do pacote na precisão das medições, utilizou-se pacotes de 64 bytes, 500 bytes, 1000 bytes e 1500 bytes, com e sem tráfego concorrente. O tráfego concorrente foi composto de 5, 10, 20 e 40 sessões FTP divididas igualmente nos transmissores TCP da figura A.4, com tamanho de pacote de 1000 bytes. As velocidades V1 a V4 foram, respectivamente, 1M, 2M, 4M e 8Mbit/s, pois geram os dois problemas descritos na figura A.3. O atraso A dos enlaces foi mantido em 1ms. Sem tráfego concorrente a inferência de banda é perfeita, e todas as medidas são de 1Mbit/s.

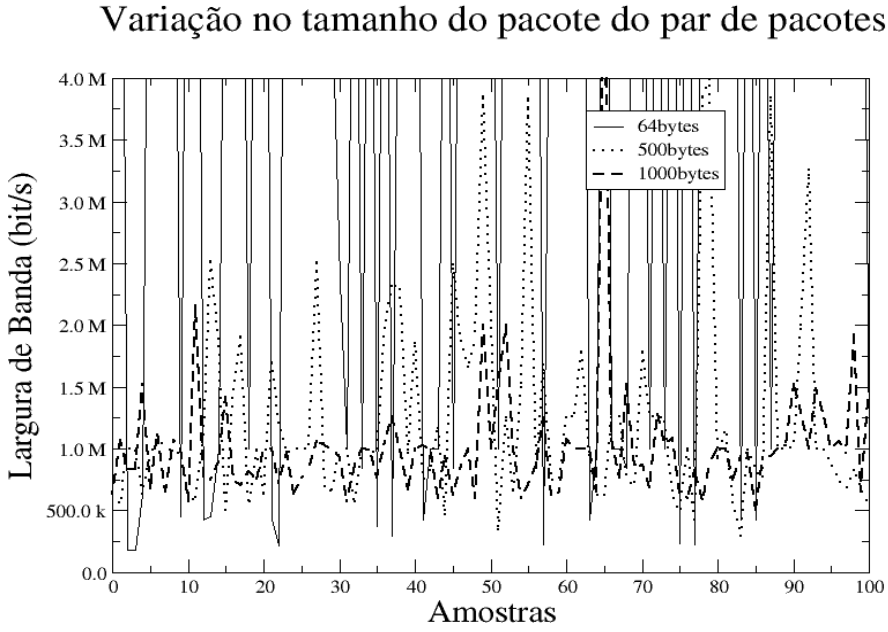


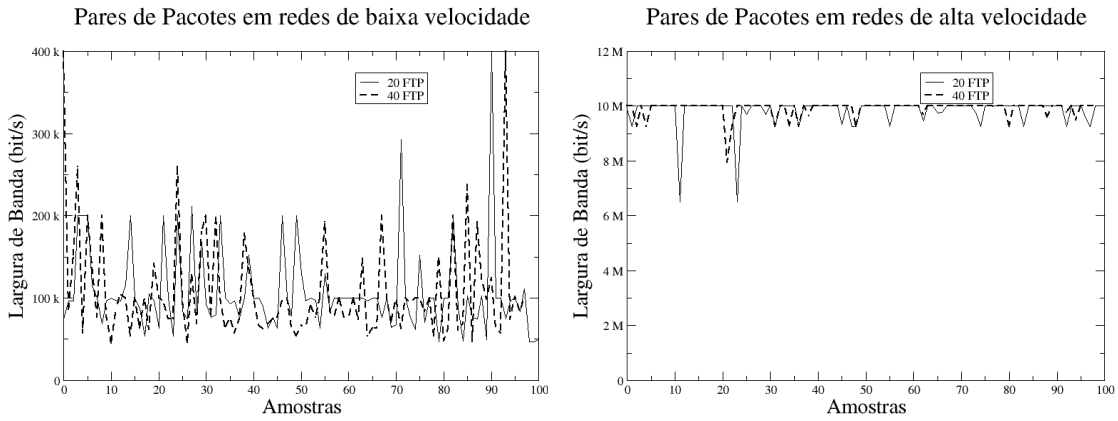
FIGURA A.6 – Pares de pacote com 64 bytes, 500 e 1000 bytes

Com tráfego concorrente, as variações acontecem. A figura A.6 mostra 20 sessões TCP concorrendo com o par de pacotes. Pode-se ver que o tamanho do pacote influencia na medição, e pacotes pequenos são menos estáveis. Com pacotes de 64 bytes, houve vários picos de 8Mbit/s (o pacote era muito pequeno frente ao do TCP, e perdia o espaçamento na fila do roteador). Com pacotes de 500 bytes o pico máximo foi de 4Mbit/s, porém, a média das inferências ficou próxima do gargalo de 1Mbit/s. Com pacotes de 1000 bytes, ou seja, de tamanho igual aos pacotes TCP, a inferência foi a mais estável e correta de todas. Essa característica se manteve em inúmeros outros testes realizados, mostrando que pacotes de tamanho maior geram inferência de banda mais precisa.

A.3.3 Verificação se o método funciona para redes de baixa e alta velocidade

Para analisar se existe alguma variação na precisão do método para redes de alta e baixa velocidade, variou-se a velocidade V1 a V4 dos enlaces da seguinte forma: a) 100kbit/s, 200kbit/s, 400kbit/s e 800kbit/s; b) 1Mbit/s, 2Mbit/s, 4Mbit/s e 8Mbit/s; c) 10Mbit/s, 20Mbit/s, 40Mbit/s e 80Mbit/s (nesse caso se aumentou a velocidades dos enlaces de 10Mbit/s para 100Mbit/s). O tráfego concorrente variou de 5, 10, 20 e 40 sessões FTP, com tamanhos de pacote de 1000bytes. O tamanho dos pacotes do par também foi de 1000 bytes. O atraso A foi mantido em 1ms.

A figura A.7 mostra os resultados para os gargalos de 100kbit/s (gráfico "a") e 10Mbit/s (gráfico "b"), pois o de 1Mbit/s já foi visto anteriormente. Pode-se ver nos dois casos que, tanto para 20 como 40 sessões FTP concorrentes, o sistema inferiu aproximadamente a banda correta, na média.



a) Enlace de 100 kbit/s b) Enlace de 10 Mbit/s

FIGURA A.7 – Gargalo de 100 kbit/s e 10Mbit/s

A.3.4 Análise das conseqüências da variação do RTT

Nesse conjunto de simulações, o atraso A de cada enlace foi variado a fim de analisar seu impacto sobre a precisão da inferência de banda. Assim, as variáveis A1 a A4 variaram de 1ms, 10ms e 100ms em todos enlaces simultaneamente. Com atraso de 1ms, o RTT do primeiro transmissor TCP é de 10ms, e com atraso de 100ms, o RTT passa para mais de 800ms. As velocidades dos enlaces foram mantidas em 1Mbit/s,

2Mbit/s, 4Mbit/s e 8Mbit/s, e os testes foram efetuados com 5, 10, 20 e 40 sessões FTP concorrentes. O tamanho dos pacotes TCP e dos pares de pacotes foi fixo em 1000 bytes.

A figura A.8 mostra o resultado para 20 sessões FTP concorrentes. Pode-se ver que, com RTT grande, a precisão do método é maior. Isso se deve à demora de ajustes do tráfego TCP, que vai ter um menor número de pacotes circulando na rede a cada momento.

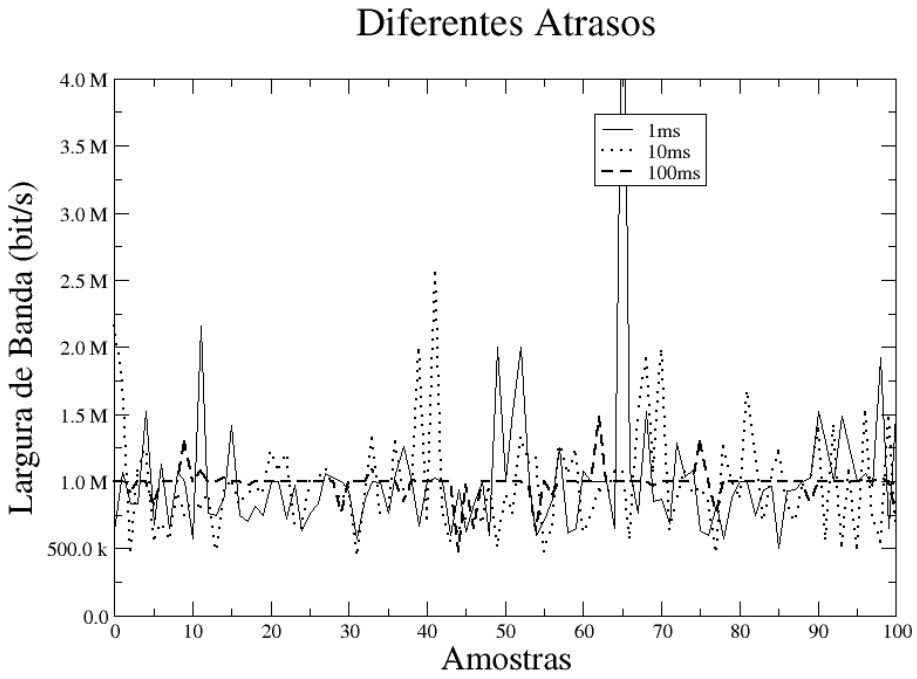


FIGURA A.8 – Impacto da variação do RTT na precisão de inferência de banda

A.4 Implementação do sistema

O objetivo da implementação foi comprovar os resultados das simulações numa rede real, com tráfego concorrente e diferentes velocidades de enlace. Na implementação real, outras variáveis afetam o método de pares de pacotes, como o atraso no sistema operacional, a concorrência no *hub* de saída de rede, e a variabilidade de tráfego na Internet. Assim, os objetivos totais da implementação foram os mesmos das simulações acrescidos dessas outras variáveis mencionadas, e estão descritos a seguir:

- Analisar o erro gerado pelo sistema operacional e concorrência na rede local;
- Analisar o impacto da variação no tamanho do pacote;
- Verificar se o método funciona para redes de baixa e alta velocidade.

O sistema é composto de um módulo transmissor e um módulo receptor. A implementação do receptor foi efetuada em Unix, pois esse sistema permite uma maior precisão na leitura de tempo. Já o transmissor foi implementado para Unix e Windows,

e a análise do espaçamento entre os pacotes mostrou que ambos são precisos, conforme descrição adiante.

O módulo transmissor envia, de tempos em tempos, uma rajada de pacotes com determinado tamanho. Essa rajada pode ser configurada para dois pacotes, caracterizando o par de pacotes (*packet-pair*), ou para mais pacotes, sendo chamada de "trem de pacotes", ou *packet-train*. O protocolo utilizado é o UDP e a linha de comando do transmissor tem as seguintes opções:

```
pptrans –h IPdest –p porta –s tampp –d atraso –n tamtrem

-h IPdest: IP do destino;
-p porta: porta a ser utilizada no destino;
-s tampp: tamanho dos pacotes a serem transmitidos;
-d atraso: atraso entre cada trem de pacotes, em ms;
-n tamtrem: número de pacotes na rajada.
```

O parâmetro "*-n*" foi criado para verificar se a precisão do método aumenta caso forem transmitidos mais de dois pacotes em seqüência, ou seja, se em vez de um par de pacotes, for transmitido um trem de pacotes.

O módulo receptor recebe o trem de pacotes e calcula a largura de banda com o transmissor baseando-se no espaçamento entre eles. Os resultados são vistos na tela do micro e também gravados em um arquivo de *log*, a fim de que se faça uma análise gráfica. A linha de comando do receptor tem as seguintes opções:

```
pprec –p porta –n tamtrem –o arquivo

-p porta: porta a ser utilizada no receptor;
-n tamtrem: número de pacotes da rajada;
-o arquivo: nome do arquivo de log de saída.
```

Todo pacote que circula na rede é composto de cabeçalho e dados. Nos testes efetuados, havia um cabeçalho Ethernet produzindo um *overhead* de 18 bytes, um cabeçalho IP de 20 bytes e um cabeçalho UDP de 8 bytes. Em função desses cabeçalhos, o parâmetro de tamanho de pacote escolhido pelo usuário no transmissor (opção –s) era reduzido em 46 bytes a fim de deixar o tamanho final do pacote conforme solicitado. Entretanto, ainda foram desprezados o preâmbulo do pacote Ethernet (8 bytes) e o "*Inter Frame Gap*" (IFG), equivalente a 12 bytes ou 96 bits, que é utilizado para permitir compartilhamento nas redes locais [SPU 2000]. Em linhas gerais, o algoritmo do transmissor efetua as operações mostradas no código a seguir, onde os pontos principais foram numerados a fim de facilitar a explicação posterior:

```
Lê parâmetros de entrada();              (1)
Seqno = 0;
while ()
      for(i=0; i<tamtrem; i++)
            buffpp[0]=seqno++;           (2)
            sendto(...)
      usleep (ppdelay*1000);             (3)
```

Inicialmente, os parâmetros (*-h*, *-p*, *-s*, *-d* e *-n*) são lidos e armazenados (número "1" do código). Em seguida, o programa entra no laço principal, ficando nele até o término do programa. A posição *buffpp[0]* contém o indicativo do número de seqüência

do pacote no trem de pacotes transmitido (número “2” no código). Para que o algoritmo funcione, o tamanho do trem de pacotes deve ser sempre 2^n, onde *n* varia de 1 (par de pacotes) até 8 (rajadas de 256 pacotes). Após transmitir o trem de pacotes, o programa entra em espera pelo tempo (em milisegundos) solicitado pelo usuário (número “3” no código).

O código que é executado no receptor deve descobrir se houve perda de pacotes na rede, incrementando um indicador de perdas caso positivo. A seguir pode ser visto resumidamente o algoritmo executado no receptor, que é explicado em seguida.

```
while(1)
      numrec = recvfrom(...);
      gettimeofday(...);                                          (1)

      SE (PRIMEIRO_PACOTE_DO_TREM)
            Armazena_Tempo(); flag = 1;                           (2)
      ELSE // outros pacote da mesma rajada
            SE (ERRO_SEQÜÊNCIA)
                  IncPerdas(); Continua_laço;
            SE (ULTIMO_PACOTE_DO_TREM)
                  Flag = 0;
                  diftempo = tempoultimo - tempoprimeiro;
                  // Cálculo da banda                             (3)
                  BW=((tamtrem-1)*((numrec+46)*8)*1000/diftempo);
                  fprintf(flog...); // grava no arquivo de log
```

No laço principal, o receptor lê o pacote da placa de rede, armazenando imediatamente o número de bytes que vieram (*numrec*) e o tempo que o pacote chegou (número “1” no código). Em seguida, o receptor descobre se o pacote é o primeiro do trem de pacotes, lendo a posição inicial do *buffer* de recepção (se múltiplo de *tamtrem*, é o primeiro). Caso seja o primeiro pacote do trem, o receptor simplesmente armazena o tempo e liga uma flag indicando que o primeiro pacote chegou (número “2” no código). Caso seja algum outro pacote da rajada, o receptor verifica se houve perda de pacotes (número de seqüência inválido ou *flag* em zero), incrementando um contador de perdas. Caso não haja perdas e o pacote seja o último do trem de pacotes (segundo no caso do par de pacotes), o receptor calcula a diferença de tempo na chegada dos pacotes e efetua o cálculo da banda, em kbit/s (número “3” no código). O cálculo da banda deve levar em consideração os 46 bytes de cabeçalho, conforme descrito anteriormente. Após obter o resultado, atualiza um *log* para posterior análise e geração dos gráficos.

Para efetuar os testes, se utilizou um ambiente controlado e também a Internet, com acesso via modem de 56 kbit/s, ADSL, Intranet, Internet tradicional e Internet2.

O ambiente controlado de teste pode ser visto na figura A.9. O transmissor envia pares de pacotes configurados com as opções de acordo com cada objetivo. O *hub* é utilizado para verificar o erro devido ao sistema operacional e concorrência na rede. Para gerar tráfego concorrente, utilizou-se conexões FTP em paralelo. O roteador é uma máquina FreeBSD, e é possível configurar a velocidade do enlace via comando *ipfw*. Em alguns testes o transmissor e o receptor PP ficavam na mesma sub-rede.

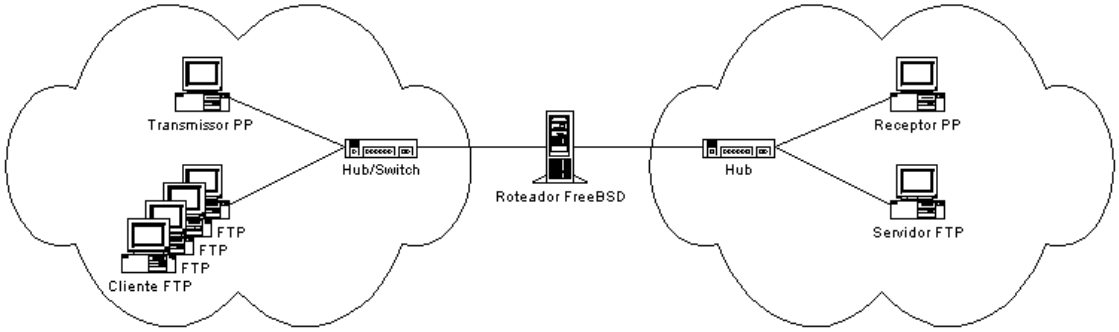


FIGURA A.9 – Ambiente controlado de teste

A.4.1 Atraso devido ao Sistema Operacional e concorrência na rede

Para verificar o atraso de saída dos pares de pacotes causados pelo sistema operacional e concorrência na rede, os pacotes foram capturados através do software *tcpdump*, e os resultados foram analisados com a interface do Ethereal[1], onde o espaçamento entre os pares foi medido. Caso o atraso fosse superior ao esperado, seria considerado uma indicativa de erro na geração dos pares de pacotes.

A figura A.10 mostra a tela do analisador dos pacotes, que foram capturados sem tráfego concorrente. Como cada pacote foi configurado com um tamanho de 1000 bytes, ou 8 kbits, e a rede é de 10 Mbit/s, pode-se calcular que o intervalo *i* de saída entre os pacotes deve ser, teoricamente: *i = 8 kbits / 10Mbit/s*, ou seja, *i = 0,8ms*. O espaçamento medido entre os primeiros quatro pares de pacotes foi de 0,793ms, 0,813ms, 0,806ms e 0,808ms, gerando um erro de aproximadamente 8µs, ou 1%. Foi considerado que o resultado medido está coerente com a teoria. O mesmo se repete entre outros tamanhos de pares de pacotes, bem como no sistema operacional tipo Unix, comprovando que o atraso no sistema operacional é desprezível para pacotes de 1000 bytes. Entretanto, vale lembrar que, para pacotes de 100 bytes, o atraso normal é de 0,08ms, ou 80µs, e um erro de 8µs corresponde a 10%, que é inserido já na saída dos pacotes.

No.	Time	Source	Destination	Protocol
1	0.000000	192.168.2.2	192.168.2.3	UDP
2	0.000793	192.168.2.2	192.168.2.3	UDP
3	0.504387	192.168.2.2	192.168.2.3	UDP
4	0.505200	192.168.2.2	192.168.2.3	UDP
5	1.009245	192.168.2.2	192.168.2.3	UDP
6	1.010051	192.168.2.2	192.168.2.3	UDP
7	1.509196	192.168.2.2	192.168.2.3	UDP
8	1.510004	192.168.2.2	192.168.2.3	UDP

FIGURA A.10 – Espaçamento devido ao sistema operacional

As medições foram realizadas com e sem processos rodando em paralelo. Sem processos em paralelo, as medições ficaram bastante estáveis. Com processos em paralelo (abertos aleatoriamente durante todo o teste), alguns pares de pacotes ficaram mais espaçados que o normal. Por exemplo, o par de pacotes número 46 sofreu um espaçamento de 2,554ms, e o correto seria de 1,2ms (pois o teste foi feito com pacotes de 1500 bytes). A conseqüência é que a banda medida ficou em 4,6 Mbit/s, e não 10

[1] www.ethereal.com

Mbit/s. Entretanto, foram erros esporádicos e não chegaram a inviabilizar as medições.

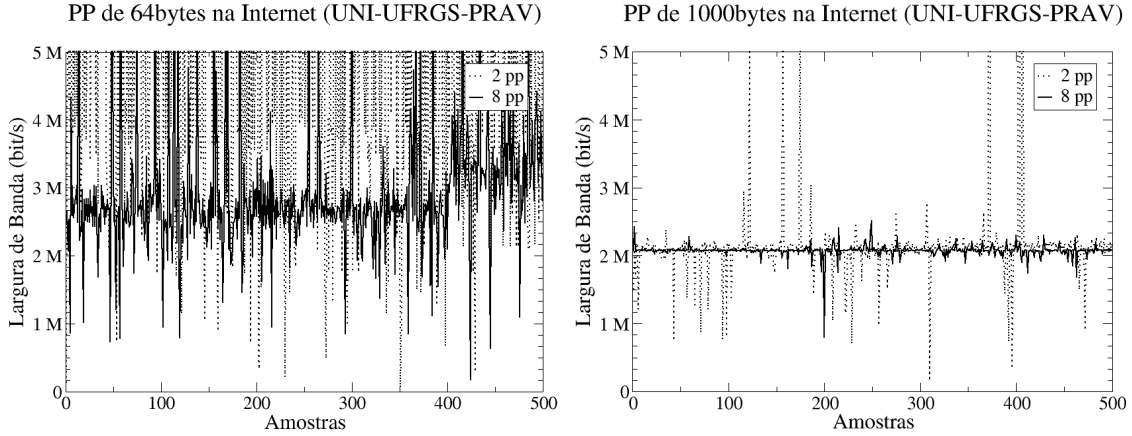
Com tráfego concorrente no mesmo *hub*, algumas vezes entra um pacote TCP entre os dois pacotes do par, gerando um erro grande já na saída da rede, como era de se esperar.

A.4.2 Variação no tamanho do pacote

Para analisar se o método é afetado pela variação no tamanho do pacote, o programa foi rodado repetidamente em ambiente controlado e na Internet, e o tamanho dos pacotes foi modificado, assumindo os valores de 64 bytes, 500 bytes, 1000 bytes e 1518 bytes. Foram efetuadas 500 medições para cada teste, e o espaçamento entre cada par de pacotes ficou em 0,5s. O teste foi repetido para um par de pacotes (*tamtrem* = 2) e uma rajada de 8 pacotes (*tamtrem* = 8), com o objetivo de ver se a precisão do método é afetada pelo tamanho da rajada.

A figura A.11 mostra uma medição realizada a partir da sala do PRAV (Pesquisa em Redes de Alta Velocidade), passando pela Intranet da Unisinos e por um enlace de 2Mbit/s com o provedor Terra. A partir daí o sinal entra na Universidade Federal do Rio Grande do Sul e volta à sala do PRAV através do enlace de Internet2 (155 Mbit/s). A topologia é conhecida, e apesar da Unisinos possuir 4 Mbit/s com a Internet comercial, o enlace é composto por dois E1, de 2Mbit/s cada.

O gráfico da figura A.11a mostra o resultado para pacotes de 64bytes, com rajadas de 2 e 8 pacotes. Pode-se ver que a instabilidade é muito grande com pares de pacotes (2pp), porém, melhora um pouco com rajadas de 8 pacotes. No gráfico da figura A.11b, o tamanho do pacote foi de 1000 bytes, sendo bem mais estável que o primeiro. Com pares de pacotes, o sistema apresentou alguns picos, mas a média se mostrou bem próxima do valor real da rede. Com rajadas de 8 pacotes o sistema ficou bastante estável, inferindo a banda correta da rede na grande maioria das amostras. Outros tamanhos de pacotes foram utilizados, e chegou-se à conclusão que pacotes de 500 bytes já resultam numa boa precisão de medida.



a) Pacotes de 64 bytes b) Pacotes de 1000 bytes

FIGURA A.11 – Pacotes de 64 bytes e 1000 bytes, rajadas de 2 e 8 pacotes

Os mesmos testes foram repetidos em ambiente controlado, com e sem tráfego concorrente. Os resultados foram similares, porém, com uma dificuldade adicional: a

limitação de tráfego utilizada no roteador (comando *ipfw*) sofria de certas instabilidades, provavelmente causado pelo algoritmo conformador de tráfego. A conseqüência é que cerca de 10% das vezes o roteador enviava dois pacotes com o máximo de sua velocidade (10 Mbit/s), causando erro naquela medição. Apesar disso, o restante das medições dava uma boa idéia do funcionamento do algoritmo, e os resultados no ambiente controlado foram similares ao ambiente real, ou seja, mostraram uma maior instabilidade em pacotes com tamanho pequeno.

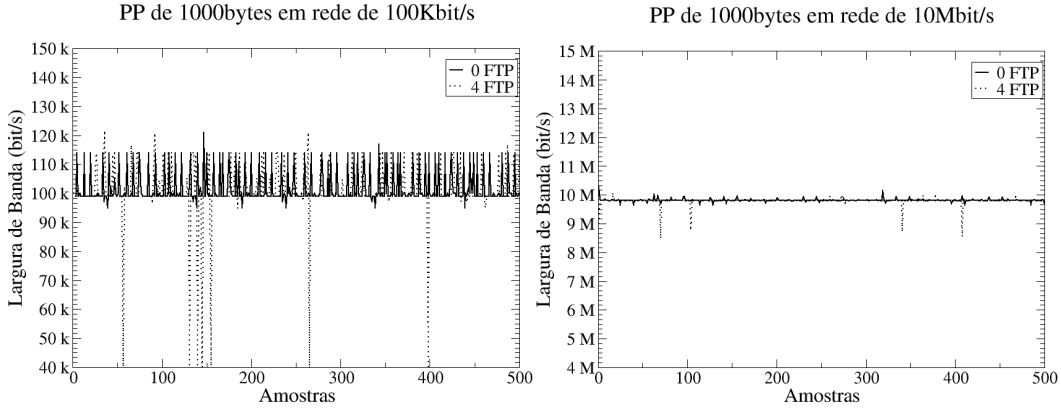
A.4.3 Variação na velocidade dos enlaces

Para analisar se o método funciona e detecta a banda em enlaces de várias velocidades, o programa foi rodado repetidamente em ambiente controlado e na Internet. O tamanho dos pacotes foi mantido em 1000 bytes, pois dá uma boa precisão na medida, conforme visto no item anterior. Foram efetuadas 500 medições para cada teste, e o espaçamento entre cada par de pacotes ficou em 0,5s.

Em ambiente controlado, a velocidade do roteador foi configurada para 100 kbit/s, 500 kbit/s e 1 Mbit/s. Além disso, utilizou-se diretamente duas estações se comunicando na mesma sub-rede através de um *hub* de 10 Mbit/s, e posteriormente através de um *switch* de 100 Mbit/s. Colocou-se uma, duas e quatro sessões FTP concorrentes a fim de verificar as diferenças nos resultados.

O gráfico da figura A.12a mostra o resultado para uma banda de 100 kbit/s, com e sem tráfego concorrente. Pode-se ver que o sistema detectou o resultado correto com uma boa precisão na maior parte das amostras, havendo certos erros devido ao software de roteamento (*ipfw*) e ao tráfego concorrente. Os testes com banda de 500kbit/s e 1Mbit/s produziram resultados semelhantes, e os gráficos referentes aos mesmos podem ser vistos em http://www.inf.unisinos.br/~roesler/tese e no CD anexo a esta Tese.

O segundo da figura A.12b mostra o resultado para uma banda de 10 Mbit/s, com transmissor, receptor e sessões FTP no mesmo *hub*. A qualidade das medições aumentou, principalmente devido à eliminação do software de roteamento *ipfw*.



a) Banda de 100 kbit/s b) Banda de 10 Mbit/s

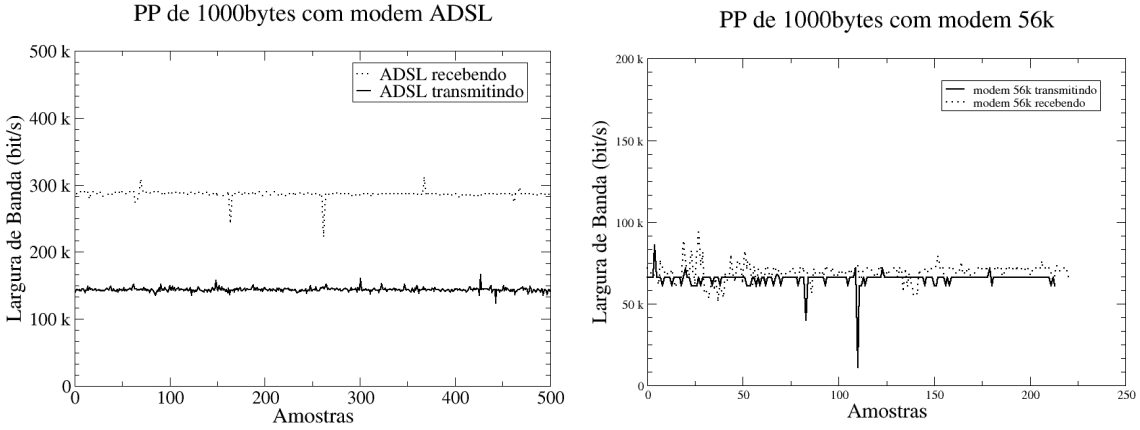
FIGURA A.12 – Ambiente controlado, 100 kbit/s e 10 Mbit/s

O mesmo teste foi repetido para um *switch* de 100Mbit/s, com vários tamanhos de pacote. Para pacotes de 1000 bytes a média ficou em 101 Mbit/s, com um máximo de 160 e mínimo de 72 Mbit/s. O resultado foi muito similar com tráfego FTP concorrente.

Na Internet, foram realizados diversos testes, como, por exemplo, o mostrado na figura A.11, onde o sinal passava pela intranet da Unisinos, provedor comercial, UFRGS e voltava, via Internet2, para o PRAV, na Unisinos. Além disso, testou-se entre duas máquinas na Intranet da Unisinos (gargalo de 10 Mbit/s), obtendo-se um resultado bastante estável, com média de 10,8 Mbit/s.

Nos testes de baixa velocidade, utilizou-se ADSL e modem 56kbit/s, e os resultados estão nos gráficos da figura A.13. O primeiro foi realizado entre a Unisinos e uma residência com ADSL (banda de saída de 160 kbit/s e entrada de 320kbit/s). O gráfico "a" da figura A.13 mostra o resultado obtido para pacotes de 1000 bytes. Como pode ser visto, o tráfego medido ficou um pouco abaixo do valor oficial configurado no modem, entretanto, bastante estável.

O gráfico "b" da figura A.13 mostra o resultado obtido entre uma residência com modem de 56 kbit/s e a Unisinos, e pacotes de 1000 bytes. O resultado mostrou uma banda para transmissão ou recepção de aproximadamente 64 kbit/s, entretanto, os modens possuem uma taxa de 56 kbit/s para recepção, mas somente 33,6 kbit/s para transmissão. Uma diferença observada é devido ao algoritmo de compressão V.42 bis existente no modem, mas isso não foi investigado.



a) ADSL 320 / 160 kbit/s b) Modem 56 / 33,6 kbit/s

FIGURA A.13 – Conexão Internet via ADSL e modem de 56 kbit/s

A.5 Filtro de amostras

Numa recepção onde os pares de pacotes vão determinar a banda máxima do sistema, como por exemplo nos algoritmos ALMP e ALMTF, vistos nos capítulos 5 e 6, é importante a existência de um filtro que suavize os resultados, pois os mesmos podem variar bastante.

O filtro utilizado no ALMTF (detalhado no item 6.1.7) é bastante simples, sendo replicado na figura A.14. As primeiras dez medidas são simplesmente inseridas numa fila. O primeiro valor de banda máxima só é obtido após as dez primeiras medições, sendo a média entre os dez primeiros valores (item "a" na figura). Após esse instante, cada medida nova que chega deve estar entre ±30% da média atual, caso contrário é descartada. Estando dentro desta faixa, a nova medida substitui a mais antiga da fila

(medida 11 substitui medida um, medida 12 substitui medida dois, e assim por diante). Com o novo valor, a média é recalculada, obtendo-se o novo valor de *bwmax* (item "b" na figura).

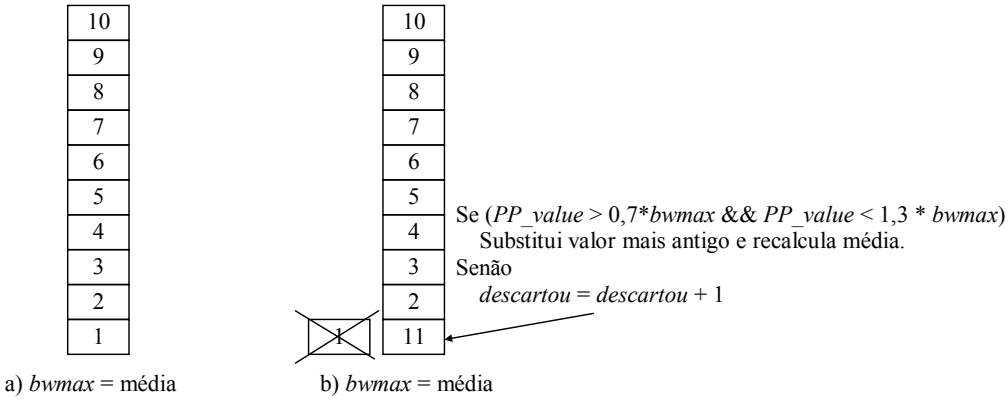


FIGURA A.14 – Filtro de pares de pacotes no ALMTF

Caso aconteçam dez descartes em seqüência (durante dez medições o valor obtido não estava entre ±30% da média atual), considera-se que a topologia da rede mudou por algum motivo e o processo é reiniciado.

Esse filtro foi implementado em linguagem C num programa denominado "*filtropp.c*", e encontra-se disponível em http://www.inf.unisinos.br/~roesler/tese e no CD anexo a esta Tese. Algumas medições obtidas nos itens anteriores serão utilizadas no filtro e descritas a seguir, com o objetivo de analisar sua eficiência.

Nas primeiras simulações mostradas neste anexo, foi variado a banda no enlace central para refletir 1Mbit/s, 2Mbit/s, 4Mbit/s e 8Mbit/s. O resultado com e sem o filtro é mostrado na figura A.15. Como pode ser visto, a utilização do filtro eliminou todos os resultados indesejáveis, deixando o valor da banda máxima bastante próximo da realidade, que é de 1 Mbit/s.

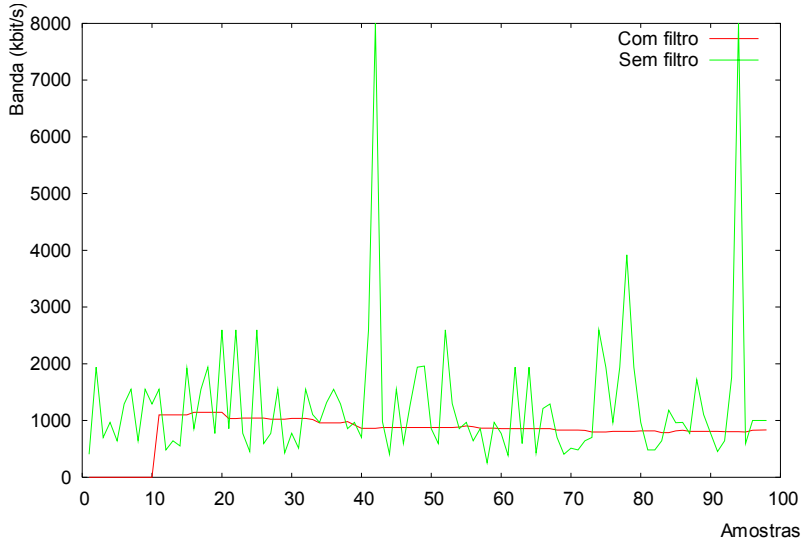


FIGURA A.15 – Simulação 1M2M4M8M com e sem o filtro de pares de pacotes

Para pacotes de 64 bytes, o filtro não funcionou, pois as medições variaram

excessivamente, entretanto, o filtro funcionou bem para pacotes de 500 bytes ou 1000 bytes.

Para redes de baixa e alta velocidade, o filtro também funcionou bem. A figura A.16 ilustra o funcionamento do mesmo para uma rede de 100 kbit/s com 40 FTPs concorrentes. Como pode ser visto, inicialmente a média ficou em aproximadamente 150 kbit/s, ou seja, acima do valor real, entretanto, ao longo das amostras, o resultado foi para bem próximo da banda máxima do enlace.

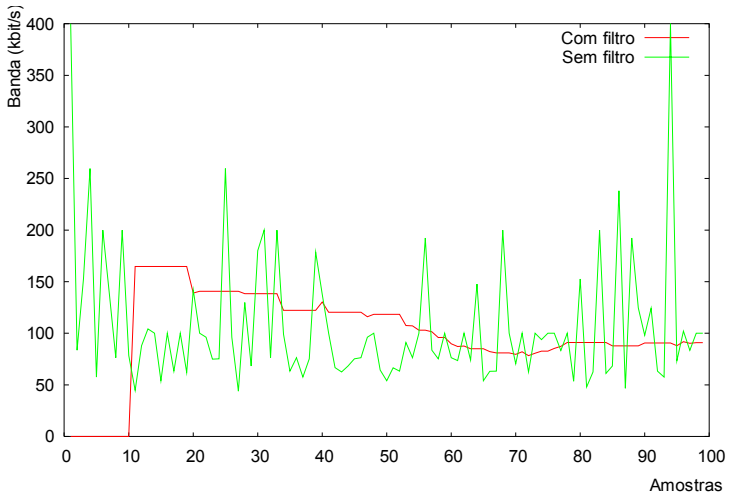


FIGURA A.16 – Análise do filtro para redes de 100 kbit/s com 40 FTPs concorrentes

Em redes com RTTs variados o filtro também funciona adequadamente. A figura A.17 mostra o comportamento do filtro para uma rede com RTT de 1ms, que é o pior caso visto nas simulações anteriores. Como pode ser observado, o filtro mostrou o resultado correto, de 1Mbit/s.

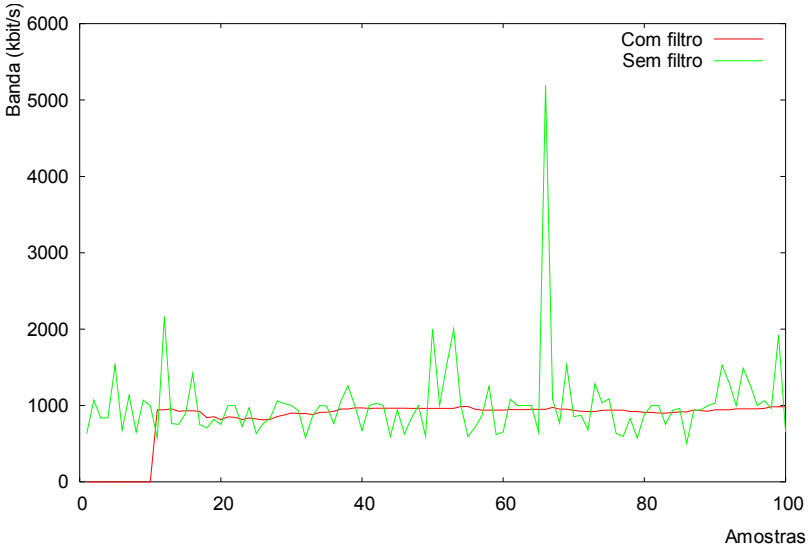


FIGURA A.17 – Análise do filtro para RTT de 1ms

Foram analisadas todas as simulações e medições vistas anteriormente neste anexo, e o filtro funcionou adequadamente em todas elas, com exceção da simulação com pacotes de 64 bytes, que não deve ser utilizado na prática, pois gera resultados

completamente irreais.

A.6 Conclusões

Este anexo apresentou uma análise detalhada do método de pares de pacotes sobre UDP, que permite a inferência de banda no receptor sem necessidade de enviar ACKs ao transmissor. Além disso, os pares de pacotes eram enviados aproximadamente a cada 0,5s, não gerando rajadas nem excessivo tráfego na rede. Um dos objetivos foi a análise da viabilidade de integração no SAM, que utiliza o próprio conteúdo multimídia em pares de pacotes, e os receptores não enviam ACKs, não gerando tráfego adicional. Através das simulações e implementação, concluiu-se que é viável a utilização do método para que o receptor descubra a banda máxima, mas não a banda eqüitativa. Além disso, o método pode ser útil para inferir a banda máxima fim-a-fim de quaisquer novos mecanismos de controle de congestionamento, principalmente os baseados em multicast.

Para analisar o método, utilizou-se simulação e também uma implementação real em linguagem 'C'. Através do ambiente de simulação, verificou-se que o método de pares de pacotes funciona, porém, algumas vezes infere banda acima da verdadeira, e algumas vezes abaixo. O motivo dessas inferências errôneas foi verificado através das simulações, e acontecem devido à diminuição do espaçamento dos pacotes nos roteadores de maior velocidade localizados após o gargalo (infere banda maior), ou entrada de tráfego concorrente no meio do par de pacotes (infere banda menor). Entretanto, mesmo com esses problemas, na média, o método funciona adequadamente.

Tanto as simulações como os resultados da implementação mostraram que o método é menos estável com pacotes pequenos, gerando variações bruscas de inferência. Esse fato poderia ser utilizado por outros algoritmos de inferência de banda baseados em pares de pacotes existentes na literatura, que enviam rajadas de pacotes pequenos e grandes. De acordo com as medidas efetuadas, não vale a pena utilizar pacotes de tamanho pequeno. O motivo é que o pacote pequeno ocupa menos tempo na rede que o pacote grande, e uma pequena variação no espaçamento gera um grande impacto.

Uma alternativa para melhorar a estabilidade e a precisão da medida, com qualquer tamanho de pacote, é utilizar uma rajada maior (trem de pacotes). Nos testes, foram testadas rajadas de 8, 16 e 32 pacotes. Com 8 pacotes já se percebe uma diferença grande nos resultados, como pode ser visto nos gráficos apresentados.

Alguns trabalhos da literatura afirmam que a utilização de trem de pacotes permite a inferência da banda eqüitativa da rede, e não a banda máxima, porém, os resultados obtidos mostraram que a banda obtida é a banda máxima.

Finalmente, foi criado um filtro para análise dos resultados em tempo real. O filtro funcionou adequadamente, conforme visto nos gráficos comparando o resultado com e sem filtro. O mesmo está sendo utilizado pelo algoritmo ALMTF para descoberta de banda máxima na rede.

Anexo B Descrição do conteúdo do CD

Este anexo tem como objetivo descrever o CD que está sendo fornecido juntamente com esta Tese. Em http://www.inf.unisinos.br/~roesler/tese pode ser obtido o mesmo conteúdo. Neste CD podem ser encontrados todos os programas de apoio desenvolvidos para esta Tese, bem como a maior parte da bibliografia utilizada, o resultado das simulações, o código dos algoritmos no simulador e o código do algoritmo na implementação real.

O diretório raiz do CD é composto dos seguintes diretórios, que serão detalhados nos próximos itens:

- **Index.html**: contém a descrição geral do CD para ser vista através de um navegador da Internet;
- **Tese.pdf**: o documento da Tese;
- **Artigos Tese**: artigos utilizados na Tese, conforme referências;
- **Netsim**: é o código-fonte e o executável do simulador NS2, com todos os algoritmos já instalados (ALMP, ALMTF, RLM e TFMCC);
- **Codigo NS – Algoritmos**: código-fonte dos algoritmos implementados no NS2, em linguagens *tcl* e C++. Além disso, consta o manual de instalação dos mesmos;
- **Simulações**: contém o resultado das simulações efetuadas para o ALMP, ALMTF, RLM, TFMCC, ALMJ e TCP_STABLE;
- **Programas Apoio**: contém o código-fonte dos programas de apoio utilizados na Tese;
- **Implementação**: o algoritmo ALMP foi implementado em linguagem C. Nesse diretório encontra-se o código-fonte e os executáveis;
- **Bin**: programas executáveis utilizados na Tese;
- **Pares Pacotes**: simulações e implementação do mecanismo de pares de pacotes;
- **VEBIT**: código-fonte do codificador em camadas VEBIT.

B.1 Diretório “ArtigosTese”

O diretório “ArtigosTese” foi criado com o objetivo de facilitar ao leitor a análise mais aprofundada de alguma referência utilizada, sem que o mesmo tenha o trabalho de procurar na Internet. Os artigos estão ordenados por ano e normalmente utilizando a seguinte notação:

```
AAAA_Nnnn_TTTT…TTT.pdf
```

AAAA é o ano de publicação, Nnnn representa o nome do autor, e TTT significa o título simplificado do artigo. Os seguintes exemplos ilustram algumas referências utilizadas, que podem ser vistas na bibliografia como [FLO 99], [LAI 2000], [KIM 2001], [WID 2001] e [ACH 2003].

- 1999_Floyd_PromotingEndtoEndCongestionControl.pdf;
- 2000_Lai_MeasuringLinkBwUsingDeterministicModelPacketDelay_nettimer.pdf;
- 2001_Kim_ComparisonLayeringStreamReplicationVideoMulticast.pdf;

- 2001_Widmer_SurveyTCPFriendlyCongestionControlExtendedVersion.pdf;
- 2003_Achir_NetworkBasedAdaptationMPEG4MulticastVideoDelivery.pdf.

Além das referências na raiz desse diretório, existem dois outros diretórios com bibliografia específica que preferiu-se deixar separado. Um deles é o diretório "RFCs", que contém as RFCs utilizadas na Tese. Os nomes das RFCs também refletem o seu título, porém, o nome do arquivo inicia com o número da RFC. Outro diretório separado é o "Roesler", que contém os relatórios técnicos e artigos publicados pelo autor desta Tese.

B.2 Diretório "Netsim"

O diretório "Netsim" contém o código fonte do NS2, já com os protocolos implementados. Os protocolos adicionados ao código do simulador foram: ALMP, ALMTF, ALMJ, TCP_STABLE e TFMCC.

B.3 Diretório "CodigoNS_Algoritmos"

O diretório "CodigoNS_Algoritmos" contém o código-fonte dos algoritmos ALMP (capítulo 5) e ALMTF (capítulo 6), implementados para o NS2. Os algoritmos foram desenvolvidos em linguagem *C* e *tcl* onde aplicável. Para inclusão do algoritmo no NS2, é necessário modificar vários diretórios, e para facilitar, foi incluído um arquivo chamado "instalaALMP.html" e "instalaALMTF.html" com a descrição de todas as modificações que devem ser efetuadas a fim de incluir o algoritmo no NS2.

Outras implementações desenvolvidas para o NS2 também constam nesse diretório, como o ALMJ (comentado no capítulo 4) e o TCPStable (descrito em [ROE 2003j]).

Neste diretório também consta o código de instalação do TFMCC. O código do RLM, que também foi utilizado na Tese, já está incluído no NS2 versão 2.1b7.

B.4 Diretório "Simulações"

O diretório "Simulações" contém todas as simulações efetuadas, para todos os algoritmos. Ele está organizado em subdiretórios, cada um com as simulações referentes a um determinado algoritmo. Para auxiliar no entendimento, foi criado um arquivo chamado "index.html" que organiza os conteúdos.

Em cada subdiretório específico de um algoritmo, existem três subdiretórios: "adaptabilidade", "eqüidade" e "escalabilidade", cada um deles com as respectivas simulações. Os subdiretórios internos também possuem uma notação padrão. Por exemplo, o subdiretório "e04_Vebit_5alm" contém a simulação de eqüidade de tráfego, com camadas iguais à do codificador VEBIT e com cinco fluxos ALM concorrentes.

B.5 Diretório “ProgramasApoio”

O diretório “ProgramasApoio” contém diversos programas desenvolvidos para apoiar e automatizar os processos. Alguns deles foram descritos no capítulo 4. Em linhas gerais, os programas de apoio são os seguintes:

- **Almfluxo**: Descobre a banda de um determinado fluxo ALM num nodo específico. Pega todos fluxos de 100 a 1050, ou seja, está preparado para 10 transmissores ALM, cada um podendo ter até 50 camadas. A saída é a banda (em kbit/s) a cada segundo, para ser usada no *gnuplot*;
- **Almfluxototal**: igual ao *Almfluxo*, porém a saída é a vazão total do fluxo;
- **BandaALM**: calcula atraso e perdas para todos os fluxos (ALM, TCP e UDP). Este programa que deve ser melhorado para ficar plenamente funcional;
- **BandaFluxo**: calcula a banda (em Mbit/s) a cada segundo para determinado número de fluxo e receptor localizado em determinado nodo;
- **CargaCPU**: programa que calcula o valor instantâneo da CPU no sistema;
- **Convtrace**: retira o espaço extra que fica no arquivo “out.tr”. Gera arquivo “out2.tr”;
- **FiltroPP**: implementação do filtro de pares de pacotes utilizado no ALMTF, a fim de verificar a eficiência do filtro sugerido;
- **OffsetGraf**: usado para gerar vários gráficos simultaneamente no *gnuplot*, a fim de evitar sobreposição. Alguns deles devem ter um *offset* para que o gráfico fique mais claro.
- **RlmFluxo**: Descobre a banda de um determinado fluxo RLM num nodo específico. Pega todos fluxos de 1 a 50, ou seja, pode ter até 50 camadas. A saida é a banda (em kbit/s) a cada segundo, para ser usada no *gnuplot*;
- **Vcmtcp**: calcula VCM (Variações de Camada por Minuto) de um fluxo TCP equivalente para camadas estilo "tipocamada" (que pode ser exponencial, igual ao VEBIT, de 50k ou de 100k).

B.6 Diretório “Implementação”

O diretório “Implementação” contém os fontes em linguagem C do algoritmo ALMP que foi implementado para Windows e para Unix, conforme descrição no item 5.3.

B.7 Diretório “bin”

O diretório “bin” contém os executáveis dos programas de apoio utilizados durante a Tese. Alguns foram desenvolvidos pelo autor da Tese, e outros são programas auxiliares. Os programas desenvolvidos foram explicados no item B.5, e os auxiliares são os seguintes:

- **Gnuplot**: utilizado para traçar gráficos de simulações;
- **Textpad**: editor de texto;
- **Sed**: programa para substituir linhas de arquivos texto;

- **NS2**: ns.exe compilado conforme o diretório “netsim”;
- **NAM**: nam.exe compilado conforme o diretório “netsim”.

B.8 ParesPacotes

No diretório “ParesPacotes” estão todas as simulações e a implementação do mecanismo de pares de pacotes para descobrir a banda máxima na rede, conforme descrição no Anexo A.

B.9 VEBIT

O diretório “VEBIT” contém o codificador em camadas utilizado no SAM, tanto o código-fonte da dissertação de Bruno [BRU 2003] como o da monografia de conclusão da Especialização em Redes e Internet de Ricardo Herbert [HER 2003]. O VEBIT foi descrito em detalhes no capítulo 8.

B.10 Outros

No diretório “Outros” constam algumas simulações gerais, como filas RED, comparações entre CBR e VBR, e outros que foram utilizados principalmente no capítulo 2.

Referências

[ACH 2003] ACHIR, N.; PUJOLLE, G. Network-based adaptation for MPEG-4 multicast video delivery. In: INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS, ICT, 10., 2003, Taiti, Polinésia Francesa. **Proceedings…** Colmar, France: IEEE, 2003.

[AKY 2001] AKYILDIZ, I. F.; MORABITO, G.; PALAZZO, S. TCP-Peach: a new congestion control scheme for satellite IP networks. **IEEE/ACM Transactions on Networking**, New York, v.9, n.3, p.307-321, June 2001.

[ALL 99] ALLMAN, M.; PAXSON, V.; STEVENS, W. **TCP Congestion Control**: RFC 2581. California: IETF, 1999.

[BAR 2000] BARCELLOS, M.; ROESLER, V. M&M: Multicasting e Multimídia. JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, JAI, 19., 2000, Curitiba. **Anais...** Curitiba: PUC-PR, 2000.

[BEN 96] BENNET, J.; ZHANG, H. WF^2Q: Worst-case fair weighted fair queuing. In: IEEE INFOCOM, 1996, São Francisco, California, EUA. **Proceedings…** New York: IEEE, 1996.

[BLA 98] BLAKE, S. et al. **An Architecture for Differentiated Services**: RFC 2475. California: IETF. 1998.

[BRA 89] BRADEN, R. **Requirements for Internet Hosts**: Communication Layers: RFC 1122. California: IETF, 1989.

[BRA 94] BRAKMO, L. S; O'MALLEY, S. W; PETERSON, L. L. TCP Vegas: new techniques for congestion detection and avoidance. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 1994, London, UK. **Proceedings…** New York: ACM, 1994. p.24-35.

[BRA 97] BRADEN, R. et al. **Resource Reservation Protocol – RSVP**: RFC 2205. California: IETF, 1997.

[BRA 98] BRADEN, B. et al. **Recommendations on Queue Management and Congestion Avoidance in the Internet**: RFC 2309. California: IETF, 1998.

[BRU 2000] BRUNO, G. **Transmissão Multimídia em Ambientes Heterogêneos**. 2000. 61p. Trabalho de conclusão (Graduação em Ciências da Computação) – Centro de Ciências Exatas e Tecnológicas, UNISINOS, São Leopoldo.

[BRU 2003] BRUNO, G. **VEBIT:** um novo algoritmo para codificação de vídeo com escalabilidade. 2003. 95p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[BRU 2003b] BRUNO, G.; ROESLER, V.; LIMA, V. VEBIT**:** um algoritmo para

codificação de vídeo com escalabilidade. In: SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB, WEBMIDIA, 9., 2003, Salvador, BA. **Anais…** Salvador: UNIFACS, 2003.

[BUR 83] BURT, P. J.; ADELSON, E. H. The laplacian pyramid as a compact image code. **IEEE Transactions on Communications**, New York, v.31, n.4, p.532-540, Apr. 1983.

[BYE 98] BYERS, John et al. A digital fountain approach to reliable distribution of bulk data. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 1998, Vancouver, Canada. **Proceedings…** New York: ACM, 1998.

[BYE 2000] BYERS, John et al. FLID-DL: Congestion control for layered multicast. In: INTERNATIONAL WORKSHOP ON NETWORKED GROUP COMMUNICATION, NGC, 2., 2000, Palo Alto, USA. **Proceedings…** New York: ACM, 2000.

[CAR 96] CARTER, R.; CROVELLA, M. **Measuring Bottleneck link speed in packet *switch*ed networks**. Boston: Boston University, Computer Science Department, 1996. 24p. (Technical Report TR-96-006).

[CLA 82] CLARK, D. **Window and acknowledgement strategy in TCP**: RFC 813. California: IETF, July 1982.

[CRO 2000] CROLL, A.; PACKMAN, E. **Managing Bandwidth – Deploying QoS in Enterprise Networks**. New Jersey: Prentice Hall, 2000. 426p.

[DEE 98] DEERING, S.; HINDEN, R. **Internet Protocol, Version 6 (IPv6) Specification**: RFC 2460. California: IETF, 1998.

[FAN 98] FANKHAUSER et al. **The Wavevideo system and network architecture: design and implementation**. 1998. 28p. Relatório Técnico n. 44 (Computer Engineering and Networks Laboratory) – TIK, Zurique, Suíça,. Disponível em: <http://www.tik.ee.ethz.ch/~gfa/papers/TR-44-98.pdf>. Acesso em: abr. 2003.

[FEN 97] FENNER, W. **Internet Group Management Protocol, Version 2**: RFC 2236. California: IETF, 1997.

[FIN 2003] FINZSCH, Peter. **Análise do mecanismo de pares de pacotes para inferência de banda máxima em redes de computadores**. 2003. 86p. Trabalho de conclusão (Graduação em Análise de Sistemas) – Centro de Ciências Exatas e Tecnológicas, UNISINOS, São Leopoldo.

[FLO 99] FLOYD, S.; FALL, K. Promoting the use of end-to-end congestion control in the Internet. **IEEE/ACM Transactions on Networking**, New York, v.7, n.4, p.458–472, Aug. 1999.

[FLO 2000] FLOYD, S. et al. Equation-based congestion control for unicast applications. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 2000, Stockholm, Sweden. **Proceedings…** New York: ACM, 2000. p.43-56.

[FOG 2003] FOGG, C. **The MPEG-FAQ Version 3.2:** MPEG-2. Disponível em:

<http://bmrc.berkeley.edu/projects/mpeg/faq/mpeg2-v38>. Acesso em: abr. 2003.

[GER 2001] GERLA, M.; SANADIDI, M. Y.; WANG, R.; ZANELLA, A. TCP Westwood: congestion window control using bandwidth estimation. In: GLOBECOM, 2001, San Antonio, Texas, USA. **Proceedings…** New York: IEEE, 2001. p.1698-1702.

[GEV 2001] GEVROS, P.; CROWCROFT, J.; KIRSTEIN, P.; BHATTI, S. Congestion control mechanisms and the best effort service model. **IEEE Network**, New York, v.15, n.3, May/June 2001.

[GHA 99] GHANBARI, M. Wavelet based rate scalable video compression. **IEEE Transactions on Circuits System and Video Technology**, New York, v.9, n.2, p.109-122, 1999.

[GON 2000] GONÇALVES, P.; REZENDE, J.; DUARTE, O. Um serviço ativo de distribuição de vídeo multiponto. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 18., 2000, Belo Horizonte. **Anais...** Belo Horizonte: UFMG, 2000, p.211-226.

[GOP 2000] GOPALAKRISHNAN, R. et al. A simple loss differentiation approach to layered multicast. In: IEEE INFOCOM, 2000, Tel Aviv, Israel. **Proceedings…** New York: IEEE, 2000. p.461-469.

[HAN 2000] HANDLEY, M. et al. **More thoughts on reference simulation for reliable multicast congestion control schemes**. 2000. Notas do encontro. Digital Fountain Inc.

[HAN 2003] HANDLEY, M.; FLOYD, S.; PADHYE, J.; WIDMER, J. **TCP Friendly Rate Control (TFRC)**: Protocol Specification: RFC 3448. California: IETF, 2003.

[HEN 2000] HENGARTNER, U.; BOLLIGER, J.; GROSS, T. TCP Vegas Revisited. In: IEEE INFOCOM, 2000, Tel Aviv, Israel. **Proceedings...** New York: IEEE, 2000.

[HER 2003] HERBERT, Ricardo. **Vídeo Escalável por Bitplanes com suporte à transmissão ao vivo**. 2003. 171p. Monografia de especialização (Especialização em Redes de Computadores e Internet) – Centro de Ciências Exatas e Tecnológicas, UNISINOS, São Leopoldo.

[HUI 2000] HUITEMA, C. **Routing in the Internet**. 2nd ed. New Jersey: Prentice Hall, 2000.

[HUS 2000] HUSTON, G. **Internet performance survival guide**. New York: John Wiley & Sons Inc, 2000.

[HUS 2000b] HUSTON, G. TCP Performance. **The Internet Protocol Journal**, California, v.3, n.2, June 2000.

[JAC 88] JACOBSON, V. Congestion avoidance and control. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 1988, Stanford, California, EUA. **Proceedings…** New York: ACM, 1988.

[JAI 90] JAIN, R. Congestion control in computer networks: issues and trends. **IEEE Network Magazine**, New York, v.4, n.3, p.24-30, 1990.

[JAI 91] JAIN, R. **The art of computer systems performance analysis**. New York: John Wiley & Sons Inc, 1991. 681p.

[KES 91] KESHAV, S. A control theoretic approach to flow control. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 1991, Zurique, Suíça. **Proceedings…** New York: ACM, 1991.

[KIM 2001] KIM, T.; AMMAR, M. A comparison of layering and stream replication video multicast schemes. In: NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, NOSSDAV, 2001, Port Jefferson, New York. **Proceedings…** New York: ACM, 2001.

[KOU 98] KOUVELAS, I.; HARDMAN, V.; CROWCROFT, J. Network adaptive continuous-media applications through self organized transcoding. In: NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, NOSSDAV, 1998, Cambridge, Massachussets, EUA. **Proceedings…** New York: ACM, 1998.

[LAI 2000] LAI, K.; BAKER, M. Measuring link bandwidths using a deterministic model of packet delay. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 2000, Estocolmo, Suécia. **Proceedings…** New York: ACM, 2000.

[LEG 2000] LEGOUT, A.; BIERSACK, W. Pathological Behaviors for RLM and RLC. In: NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, NOSSDAV, 2000, Chapel Hill, Carolina do Norte, EUA. **Proceedings…** New York: ACM, 2000. p. 164-172.

[LEG 2000b] LEGOUT, A.; BIERSACK, E. PLM: Fast Convergence for Cumulative Layered Multicast Transmission Schemes. In: ACM SPECIAL INTEREST GROUP OF COMPUTER / COMMUNICATION SYSTEM PERFORMANCE, ACM SIGMETRICS, 2000, Santa Clara, California, EUA. **Proceedings…** New York: ACM, 2000.

[LI 98] LI, X.; PAUL, S.; AMMAR, M. Layered video multicast with retransmission (LVMR): evaluation of hierarchical rate control. In: IEEE INFOCOM, 1998, San Francisco, California, EUA. **Proceedings…** New York: IEEE, 1998.

[LI 99] LI, X.; PAUL, S.; AMMAR, M. Video Multicast Over the Internet. **IEEE Network**, New York, v.13, n.2, p.46-60, Mar./Apr. 1999.

[LI 2001] LI, W. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v.11, n.3, p.301-317, Mar. 2001.

[LI 2003] LI, B.; LIU, J. Multirate video multicast over the Internet: an overview.

IEEE Network, New York, v.17, n.1, p.24-29, Jan./Feb. 2003.

[LIU 2002] LIU, J.; BO, L.; ZHANG, Y. A hybrid adaptation protocol for TCP-friendly layered multicast and its optimal rate allocation. In: IEEE INFOCOM, 2002, New York, EUA. **Proceedings…** New York: IEEE, 2002.

[LOE 89] LOEFFLER, C.; LIGTENBERG, A.; MOSCHYTZ, C. S. Practical fast 1ddct algorithm with eleven multiplications. In: INTERNATIONAL CONFERENCE OF ACOUSTICS SPEECH AND SIGNAL PROCESSING, 1989, Glasgow, Scotland. **Proceedings…** New York: IEEE, 1989. p.988-991.

[LOM 2003] LOMBARDO, A.; SCHEMBRA, G. Performance evaluation of an adaptive-rate MPEG encoder matching Intserv traffic constraints. **IEEE/ACM Transactions on Networking**, New York, v. 11, n.1, p.47-65, Feb. 2003.

[LUB 2002] LUBY, M.; GOYAL, V. Wave and equation based rate control building block. **Internet draft:** draft-ietf-rmt-bb-webrc-04.txt. California: IETF, 2002.

[LYO 2000] LYONNET, F. An application-level CPU scheduler for videoconferencing systems. In: PACKET VIDEO, 2000, Cagliari, Italy. **Proceedings…** New York: IEEE, 2000.

[MAH 97] MAHDAVI, J.; FLOYD, S. **TCP friendly unicast rate-based flow control**. Technical note sent to the end2end-interest mailing list. 1997. Disponível em: <http://www.psc.edu/networking/papers/tcp_friendly.html>. Acesso em: set. 2003.

[MCC 96] McCANNE, S.; JACOBSON, V.; VETTERLI, M. Receiver driven layered multicast. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 1996, Stanford, California, USA. **Proceedings…** New York: ACM, 1996. p.117-130.

[MCC 2003] McCANNE, S.; FLOYD, S. **NS Network Simulator**. Disponível em: <http://www.isi.edu/nsnam/ns/>. Acesso em: out. 2003.

[MOG 90] MOGUL, J.; DEERING, S. **Path MTU Discovery**: RFC 1191. California: IETF, 1990.

[NIC 98] NICHOLS, K. et al. **Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers**: RFC 2474. California: IETF, 1998.

[PAD 98] PADHYE, J.; FIROIU, V.; TOWSLEY, D.; KUROSE, J. Modeling TCP throughput: a simple model and its empirical validation. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 1998, Vancouver, Canada. **Proceedings…** New York: ACM, 1998.

[PAD 99] PADHYE, J.; KUROSE, J.; TOWSLEY, D.; KOODLI, R. A model based TCP-friendly rate control protocol. In: NETWORK AND

OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, NOSSDAV, 1999, New Jersey, USA. **Proceedings…** New York: ACM, 1999.

[PAD 2000] PADHYE, J.; FIROIU, V.; TOWSLEY, D.; KUROSE, J. Modeling TCP Reno performance: a simple model and its empirical validation. **IEEE/ACM Transactions on Networking**, New York, v.8, n.2, Apr. 2000.

[PAS 97] PASSMORE, D. Delayed Voice-over-IP. **Business Communications Review**, Illinois, n.12, 1997.

[PAU 98] PAUL, Sanjoy. **Multicasting on the Internet and its applications**. Massachussets: Kluwer Academic Publishers, 1998. 421p.

[PAX 97] PAXSON, V. End-to-end Internet packet dynamics. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 1997, Cannes, France. **Proceedings…** New York: ACM, 1997. p. 139-152.

[PEA 96] PEARSON, M. et al. **Current techniques for measuring and modeling ATM traffic**. Hamilton, New Zeland: The University of Waikato, Department of Computer Science, 1996. (Technical Report Working Paper 96/12).

[POS 80] POSTEL, J. **User Datagram Protocol**: RFC 768. California: IETF, 1980.

[POS 81a] POSTEL, J. **Internet Protocol**: RFC 791. California: IETF, 1981.

[POS 81b] POSTEL, J. **Internet Control Message Protocol**: RFC 792. California: IETF, 1981.

[POS 81c] POSTEL, J. **Transmission Control Protocol**: RFC 793. California: IETF, 1981.

[PUR 93] PURI, A.; WONG, A. Spatial Domain Resolution Scalable Video Coding. In: CONFERENCE ON VISUAL COMMUNICATIONS AND IMAGE PROCESSING, 1993, Boston, USA. **Proceedings…** Boston: SPIE, 1993. p.718-729.

[RAM 99] RAMAKRISHNAN, K.; FLOYD, S. **A Proposal to add Explicit Congestion Notification (ECN) to IP**: RFC 2481. California: IETF, 1999.

[RAT 99] RATNASAMY, S.; McCANNE, S. Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements. In: IEEE INFOCOM, 1999, New York, USA. **Proceedings…** New York: IEEE, 1999. p.353-360.

[REJ 99] REJAIE, R.; HANDLEY, M.; ESTRIN, D. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In: IEEE INFOCOM, 1999, New York, EUA. **Proceedings…** New York: IEEE, 1999.

[REJ 99b] REJAIE, R.; HANDLEY, M.; ESTRIN, D. Quality adaptation for congestion controlled video playback over the Internet. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 1999, Cambridge, Massachussets, EUA. **Proceedings…** New York: ACM, 1999.

[RHE 2000] RHEE, I.; OZDEMIR, V.; YI, Y. **TEAR**: TCP emulation at receivers – flow control for multimedia streaming. North Carolina: NCSU, Department of Computer Science, 2000. (Technical Report).

[RIZ 98] RIZZO, L. Fast group management in IGMP. In: HIGH PERFORMANCE PROTOCOL ARCHITECTURES (Hipparch Workshop), 1998, London, UK. **Proceedings…** London: UCL, 1998.

[ROE 2000a] ROESLER, V. Transmissão multimídia em Redes de Computadores: um relato para redes locais e Internet2. **NewsGeneration**, São Paulo, v.4, n.4, 2000.

[ROE 2000b] ROESLER, V.; BRUNO, G.; LIMA, V. Adaptability in multimedia transmissions using layering multicast and QoS guarantees. In: PROTOCOLS FOR MULTIMIDIA SYSTEMS, PROMS, 2000, Cracóvia, Polônia. **Proceedings...** Cracow: University of Mining and Metallurgy, 2000.

[ROE 2000c] ROESLER, V.; BIRCK, D.; KIELING, A.; LIMA, V. Especificação de um protótipo para convergência entre TV e Web. In: SEMINÁRIO REGIONAL DE INFORMÁTICA, 10., 2000, Santo Angelo, RS. **Anais...** Santo Angelo: URI - Universidade Regional Integrada, 2000.

[ROE 2001] ROESLER, V.; BRUNO, G.; LIMA, V. ALM: Adaptive Layered Multicast. In: SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E HIPERMÍDIA, SBMIDIA, 7., 2001, Florianópolis, SC. **Anais…** Florianópolis: UFSC, 2001.

[ROE 2002a] ROESLER, V.; BRUNO, G.; LIMA, V. Análise de estabilidade e imparcialidade em um novo algoritmo para transmissão multicast em camadas. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 2002, Búzios, RJ. **Anais...** Rio de Janeiro: NCE/UFRJ, 2002. v.2, p.784-799.

[ROE 2002b] ROESLER, V.; BRUNO, G.; BRAGA, H.; BALBINOT, L.; ANDRADE, M.; LIMA, V. Uma ferramenta adaptativa para transmissão e recepção de sinais multimídia ao vivo. In: SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E HIPERMÍDIA, SBMIDIA, 7., 2002, Fortaleza, CE. **Anais...** Fortaleza: Universidade Federal do Ceará, 2002.

[ROE 2002c] ROESLER, V.; BRUNO, G.; LIMA, V. Análise de estabilidade em um algoritmo para controle de congestionamento de transmissões multimídia em camadas. In: SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E HIPERMÍDIA, SBMIDIA, 7., 2002, Fortaleza, Ceará. **Anais...** Fortaleza: Universidade Federal do Ceará, 2002.

[ROE 2003a] ROESLER, V.; BRUNO, G.; LIMA, V. A new receiver adaptation

method for congestion control in layered multicast transmissions. In: INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS, ICT, 10., 2003, Taiti, Polinésia Francesa. **Proceedings…** Colmar, France: IEEE, 2003.

[ROE 2003b] ROESLER, V.; FINZSCH, P.; ANDRADE, M.; LIMA, V. Análise do mecanismo de pares de pacotes visando estimar a banda da rede via UDP. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 21., 2003, Natal, RN. **Anais…** Natal: UFRN, 2003.

[ROE 2003c] ROESLER, V.; ANDRADE, M. On-line remote class with live video transmission: a study case. In: COMPUTERS AND ADVANCED TECHNOLOGY IN EDUCATION / WEB BASED EDUCATION, CATE/WBE, 2003, Rhodes, Greece. **Proceedings…** Greece: IASTED, 2003.

[ROE 2003d] ROESLER, V. **Transmissão multicast**. São Leopoldo: UNISINOS, Centro de Ciências Exatas e Tecnológicas, PRAV: Pesquisa em Redes de Alta Velocidade, 2003. 26p. (Relatório Técnico 01/2003).

[ROE 2003e] ROESLER, V. **Desempenho em redes TCP**. São Leopoldo: UNISINOS, Centro de Ciências Exatas e Tecnológicas, PRAV: Pesquisa em Redes de Alta Velocidade, 2003. 23p. (Relatório Técnico 02/2003).

[ROE 2003f] ROESLER, V. **QoS em redes de computadores**. São Leopoldo: UNISINOS, Centro de Ciências Exatas e Tecnológicas, PRAV: Pesquisa em Redes de Alta Velocidade, 2003. 24p. (Relatório Técnico 03/2003).

[ROE 2003g] ROESLER, V. **Transmissão multimídia em redes de computadores**. São Leopoldo: UNISINOS, Centro de Ciências Exatas e Tecnológicas, PRAV: Pesquisa em Redes de Alta Velocidade, 2003. 35p. (Relatório Técnico 04/2003).

[ROE 2003h] ROESLER, V. **Controle de congestionamento em redes de computadores**. São Leopoldo: UNISINOS, Centro de Ciências Exatas e Tecnológicas, PRAV: Pesquisa em Redes de Alta Velocidade, 2003. 20p. (Relatório Técnico 05/2003).

[ROE 2003i] ROESLER, V.; CERON, J.; ANDRADE, M. Aulas remotas on-line utilizando transmissão de vídeo: estudo de caso na Informática da Unisinos. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, SBIE, 14., 2003, Rio de Janeiro, RJ. **Anais…** Rio de Janeiro: UFRJ, 2003.

[ROE 2003j] ROESLER, V. **TCP-Stable: um novo protocolo TCP-Friendly de nível 4**. São Leopoldo: UNISINOS, Centro de Ciências Exatas e Tecnológicas, PRAV: Pesquisa em Redes de Alta Velocidade, 2003. 17p. (Relatório Técnico 06/2003).

[SAN 2002] SANADIDI, M. Congestion control for hybrid high-speed global networks. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 20., SBRC 2002, Búzios, RJ. **Tutorial...** Rio de

Janeiro: NCE/UFRJ, 2002.

[SCH 89] SCHULZRINNE, H. et al. **RTP**: A Transport Protocol for Real-Time Applications: RFC 1889. California: IETF, 1996.

[SER 97] SERVAIS, M. P.; JAGER, G. Video Compression Using the Three Dimensional Discrete Cosine Transform (3D-DCT). In: SOUTH AFRICAN SYMPOSIUM ON COMMUNICATIONS AND SIGNAL, COMSIG, 1997, Grahamstown, South Africa. **Proceedings…** New York: IEEE, 1997. p.27-32.

[SHA 2002] SHAPIRO, J.; TOWSLEY, D.; KUROSE, J. Optimization-based congestion control for multicast communications. **IEEE Communications**, New York, v.40, n.9, p.90-95, Sept. 2002.

[SIS 2000] SISALEM, D.; WOLISZ, A. MLDA: A TCP-friendly congestion control framework for heterogeneous multicast environments. In: INTERNATIONAL WORKSHOP ON QUALITY OF SERVICE, IWQoS, 8., 2000, Pittsburgh, PA. **Proceedings...** New York: IEEE, 2000.

[SIS 2000b] SISALEM, D.; WOLISZ, A. LDA+ TCP-friendly adaptation: a measurement and comparison study. In: NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, NOSSDAV, 2000, Chapel Hill, North Caroline, USA. **Proceedings…** New York: ACM, 2000.

[SPU 2000] SPURGEON, C. E. **Ethernet, the definitive guide**. Sebastopol: O'Reilly, 2000. 500p.

[STE 97] STEVENS, W. **TCP *slow-start*, Congestion Avoidance, Fast Retransmit, and Fast Recovery algorithms**: RFC 2001. California: IETF, 1997.

[TAU 94] TAUBMAN, D.; ZAKHOR, A. Multirate 3-D subband coding of vídeo. **IEEE Transactions on Image Processing**, New York, v.3. p.573-588, Sept. 1994.

[TEN 2003] TENG, Shu-wen; CHANG, Jin-fu. A robust wireless transmission control protocol to cope with channel errors in a long round-trip delay environment. In: INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS, ICT, 10., 2003, Taiti, Polinésia Francesa. **Proceedings…** Colmar, France: IEEE, 2003.

[THO 97] THOMPSON, K.; MILLER, G.; WILDER, R. Wide-area Internet traffic patterns and characteristics. **IEEE Network**, New York, v.11. n.6, p.10-23, Nov. 1997.

[TUR 97] TURLETTI, T.; PARISIS, S.; BOLOT, J. **Experiments with a layered transmission scheme over the Internet**. France: INRIA, 1997. 26p. (Technical report RR-3296).

[VIC 98] VICISANO, L.; RIZZO, L.; CROWCROFT, J. TCP-like congestion control for layered multicast data transfer. In: IEEE INFOCOM, 1998, San Francisco, California, USA. **Proceedings…** New York: IEEE, 1998.

p.996-1003.

[WAN 2002] WANG, Y.; SHUNAN, L. Error resilient video coding using multiple de*script*ion motion compensation. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v.12, n.6, June 2002.

[WET 99] WETHERALL, D. Active network vision and reality: lessons from a capsule-based system. **ACM Operating Systems Review**, New York, v.33, n.5, p.64-79, Dec. 1999.

[WID 2001] WIDMER, J.; DENDA, R.; MAUVE, M. A survey on TCP-friendly congestion control. **IEEE Network**, New York, v.15, n.3, 2001.

[WID 2001b] WIDMER, J.; HANDLEY, M. Extending equation-based congestion control to multicast applications. In: ACM SPECIAL INTEREST GROUP ON COMMUNICATIONS, ACM SIGCOMM, 2001, San Diego, California, USA. **Proceedings…** New York: IEEE, 2001.

[WIL 97] WILSON, D.; GHANBARI, M. Optimization of two-layer SNR scalability for MPEG-2 video. In: INTERNATIONAL CONFERENCE OF ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 1997, Monique, Germany. **Proceedings...** New York: IEEE, 1997. p.2637-2640.

[WU 97] WU, L.; SHARMA, R.; SMITH, B. Thin Streams: an architecture for multicasting layered video. In: WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, 1997, Washington, USA. **Proceedings...** New York: IEEE, 1997.