UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Carlos Eduardo Lenz

Algoritmos para Retransmissão de Vídeo H.264 em Redes Sobrepostas

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Prof. Lau Cheuk Lung, Dr.
(Orientador)

Florianópolis, Abril de 2010

Catalogação na fonte pela Biblioteca Universitária
da
Universidade Federal de Santa Catarina

L575a Lenz, Carlos Eduardo
Algoritmos para retransmissão de vídeo H.264 em redes sobrepostas [dissertação] / Carlos Eduardo Lenz ; orientador, Lau Cheuk Lung. - Florianópolis, SC, 2010.
125 p.: il., grafs., tabs., mapas

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Informática. 2. Ciência da computação. 3. Multicast. 4. Transmissão de imagem. 5. Redes sobrepostas. I. Lung, Lau Cheuk. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 681

Algoritmos para Retransmissão de Vídeo H.264 em Redes Sobrepostas

Carlos Eduardo Lenz

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, área de concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Mário Antônio Ribeiro Dantas, Dr. (Coordenador)

Banca Examinadora

Prof. Lau Cheuk Lung, Dr.
(Orientador)

Prof. Frank Augusto Siqueira, Dr.

Prof. Antonio Marinho Pilla Barcellos, Dr.

Prof. Carlos Barros Montez, Dr.

Prof. Edson Nascimento Silva Junior, Dr.

Agradecimentos

Uma jornada acadêmica acabou e o objetivo foi alcançado. Neste momento olho para trás a fim dar o devido reconhecimento aos parceiros e para lembrar dos amigos presentes nestes dois anos.

No dia-a-dia no LaPeSD[1] foram muitos os companheiros, o que permitiu tantas conversas instigantes e troca de informações importantes. Fernando, Pedro, Jefferson, Denise, Aquiles, Tiago, Alex e Valdir: valeu o companheirismo e desejo sucesso nas suas futuras empreitadas!

Quero agradecer ao meu orientador, prof. Lau, pelo desafio dessa linha de pesquisa, suas direções e apoio em todas as etapas da produção desse trabalho. Foi uma excelente oportunidade para crescer tanto nos conhecimentos teóricos quanto ao colocá-los em prática. Também faço menção ao prof. Frank, que com suas opiniões e correções nas revisões dos artigos submetidos contribuiu para aperfeiçoar a exposição das idéias presentes neste trabalho. Aos membros da banca avaliadora: suas sugestões e elogios foram muito apreciados na revisão e melhoria deste texto.

Mas não se faz pesquisa sem estrutura, por isso o PPGCC[2] da UFSC[3] foi muito importante para permitir a conclusão deste trabalho. Aos professores e funcionários do programa e da UFSC, parabéns pelo trabalho em apoio à pesquisa, sua função é muito importante. Inclusive os que já deixaram o serviço, como a Verinha, ex-secretária do programa que curte sua merecida aposentadoria.

Mas nem só de pesquisa vive o mestrando, e nesse quesito começo lembrando dos meus pais Carlos e Alzira, que tanto me incentivaram e apoiaram. Beijo mãe, beijo pai. E Paulo, meu irmão, uma abraço forte e estou sempre torcendo por você! Meus avós também são muito importantes para mim, e merecem serem reconhecidos pelo seu caráter e pelo que buscaram para suas famílias: Waldemar e Realcy. O mesmo se aplica aos que já partiram deixando saudades, Affonso e Catharina. O afeto é muito grande por todos! Aos amigos da Igreja Metodista Wesleyana de

[1] Laboratório de Pesquisas em Sistemas Distribuídos
[2] Programa de Pós-Graduação em Ciência da Computação
[3] Universidade Federal de Santa Catarina

Florianópolis: um abraço carinhoso, vocês são todos muito especiais. Finalmente, aquele que proporcionou todas essas oportunidades e pessoas importantes, e me abençoou inúmeras vezes nessa caminhada: te dou graças e louvor, meu Pai do céu!

"Não sabes, não ouviste que o eterno Deus, o Senhor, o Criador dos confins da terra, não se cansa nem se fatiga? E inescrutável o seu entendimento. ... os que esperam no Senhor renovarão as suas forças; subirão com asas como águias; correrão, e não se cansarão; andarão, e não se fatigarão."
Isaías 40:28,31

"Se o grão de trigo caindo na terra não morrer, fica ele só; mas se morrer, dá muito fruto."
João 12:24

Resumo

Com a popularização da banda larga os usuários domésticos passaram a consumir diversos serviços multimídia pela Internet: Telefonia IP, rádio, vídeo sob demanda. Por enquanto, em se tratando de vídeos, os principais serviços disponíveis na rede não oferecem as características presentes na TV, ou seja, não são ao vivo e necessitam de *buferização*, gerando portanto atraso. Eles também requerem uma grande largura de banda nos servidores, linearmente crescente com número de clientes. Através das redes sobrepostas (orientadas a dados) pode-se remediar a falta de suporte ao *IP Multicast*, que seria uma solução para essa classe de aplicação, sem os problemas com a largura de banda. Entretanto tais redes não especificam o tratamento das perdas de pacotes, de modo que ou estas são ignoradas, ou são recuperadas em qualquer situação.

Este trabalho faz uma análise dos recursos do padrão de compressão de vídeo digital H.264 e apresenta critérios claros para retransmissão a partir dos diferentes tipos de codificação presentes nesse formato. Estabelece prioridades a partir desses tipos e define três algoritmos para a escolha de partes perdidas para a recuperação. Um deles (*SeRViSO*[4]) observa uma meta fixa levando em conta as prioridades, em termos de quanto dos dados deve retransmitir. O segundo (Adaptativo) usa uma meta variável, buscando inicialmente a retransmissão de todas as partes perdidas de um segmento, mas a cada vez que a retransmissão de partes do mesmo segmento for solicitada a meta diminui. O último algoritmo (*(m,k)-Firm*) também tem metas fixas, mas uma para cada tipo de codificação, e não tolera perdas de partes com o mesmo tipo de codificação muito próximas. Todos eles são definidos dentro do escopo da proposta da rede sobreposta *SeRViSO*. As vantagens de um tratamento claro das perdas são óbvias: maior adaptação às condições da rede e perdas menos importantes, ou seja, que não redundem em propagação de erro por muitos quadros. Os resultados dos testes com um protótipo confirmam isso, de forma que a perda de partes do tipo mais importante foi consistentemente menor que a perda média.

[4] *Selective Retransmission Video Streaming Overlay*

Palavras chave: P2P, Multicast, Transmissão de Video, Redes Sobrepostas.

Abstract

With the wide spread adoption of broadband Internet access, home users have started to consume several multimedia services: IP Telephony, radio, video on demand. Meanwhile, available video services do not offer the characteristics present in TV, because they are not live and need buffering, having delay. They also require a great server bandwidth, linearly associated to the number of clientes. Through (data-oriented) overlay networks it is possible to overcome the lack of *IP Multicast* support, which would be a solution to this application class, without the bandwidth problem. Nonetheless, these networks do not specify packet loss handling, in a way that either it is ignored, or recovery always happen.

This work does an analysis on the properties of the H.264 standard for digital video compression, and presents clear criteria for retransmission based on the several different types of coding found in this format. It establishes priorities from these types and defines three algoritms for the selection of lost parts for recovery. On of the them (*SeRViSO*) there is a firm target regarding the priorities, in the matter of how much data it should retransmit. The second (Adaptive) uses a movable target, initially seeking every lost part of a segment, but lowering the target each time the segment has parts selected for retransmission. The last algoritm (*(m,k)-Firm*) also has firm targets, but one for each type of coding, and does not tolerate nearby losses of parts of the same type. All of them are defined in the scope of the proposed *SeRViSO* overlay network. The advantages of a clear loss handling are obvious: greater adaptation to the network conditions and less important losses, which means they do not propagate error for many frames. The results of tests performed with a prototype confirm that the loss of parts of the most important type was consistently lower than the average loss.

Keywords: P2P, Multicast, Video Streaming, Overlay Networks.

Lista de Figuras

[5] *Network Abstraction Layer Unit*
[6] *Real-time Transport Protocol*
[7] *Content Distribution Network*
[8] *Reliable Multicast Transport Protocol*

Lista de Tabelas

Lista de Algoritmos

[9] *Negative ACK*

Lista de Acrônimos

ACK *Acknowledgement*

ALM *Application Layer Multicast*

ARC *Analytical Rate Control*

AVC *Advanced Video Coding*

CDN *Content Distribution Network*

CPU *Central Processing Unit*

DCCP *Datagram Congestion Control Protocol*

FIFO *First In First Out*

GOP *Group of Pictures*

GOP *Group of Pictures*

HTTP *Hypertext Transfer Protocol*

IGMP *Internet Group Management Protocol*

IMAP *Internet Message Access Protocol*

IP *Internet Protocol*

ISDN *Integrated Services Digital Network*

ISO *International Organization for Standardization*

ISP *Internet Service Provider*

ITU-T *International Telecommunication Union - Telecomuncation Standardization Sektor*

JVT *Joint Video Team*

LaPeSD Laboratório de Pesquisas em Sistemas Distribuídos

MPEG *Motion Picture Experts Group*

MTU *Maximum transmission unit*

NACK *Negative ACK*

NALU *Network Abstraction Layer Unit*

NAT *Network Address Translation*

PPGCC Programa de Pós-Graduação em Ciência da Computação

PPMAP *Pushing Packets Map*

P2P *Peer-to-Peer*

RGB *Red, Green, Blue*

RMTP *Reliable Multicast Transport Protocol*

RP *Rendezvous Point*

RTCP *RTP Control Protocol*

RTP *Real-time Transport Protocol*

RTSP *Real Time Streaming Protocol*

SCAMP *Scalable Membership Protocol*

SeRViSO *Selective Retransmission Video Streaming Overlay*

SHA *Secure Hash Algorithm*

SIP *Session Initiation Protocol*

SMTP *Simple Mail Transfer Protocol*

TCP *Transmission Control Protocol*

TFRC *TCP Friendly Rate Control*

UDP *User Datagram Protocol*

UFSC Universidade Federal de Santa Catarina

URL *Unified Resource Location*

VCEG *Video Coding Experts Group*

VoIP *Voice over IP*

WLED *Wireless Loss Estimation using DiffServ*

Sumário

[10] *TCP Friendly Rate Control*
[11] *Peer-to-Peer*

[12] *Internet Protocol*

Capítulo 1

Introdução

1.1 Motivação

A oferta de banda larga aos usuários residenciais tem aumentado em todo o mundo, ao ponto de atender aos requisitos da transmissão de multimídia em tempo real. Preços mais acessíveis e maior velocidade também ajudam a popularizar esse serviço. Entre as novas tecnologias oferecidas estão as sem-fio, que oferecem amplas áreas de cobertura, como WiMax e 3G, esta última oferecendo também mobilidade.

Chegou-se a um ponto onde a Telefonia IP e o rádio já estão disponíveis, visto que seus requisitos em termos de largura de banda já são atendidos a contento. Entretanto, a programação multimídia (áudio e vídeo) oferecida apresenta diferenças significativas ao paradigma televisivo. Enquanto a mídia tradicional é ao-vivo, os serviços de vídeo na internet[1][2] consistem de vídeos disponíveis sob-demanda, com transmissão confiável que leva a grande *buferização*.

Um problema que parece afetar o vídeo digital mais que o áudio é a necessidade de largura de banda. Mesmo que os mais recentes padrões ofereçam cada vez maior compressão – as técnicas são mais avançadas e a maior capacidade de processamento dos dispositivos de hoje permite que sejam computacionamente mais complexos – as expectativas dos usuários por imagens com resolução cada vez maior tornam esse problema sem solução definitiva à vista.

Em geral, as arquiteturas tradicionais para distribuir conteúdo enfrentam problemas de escalabilidade porque a largura de banda necessária nos servidores cresce linearmente com o número de clientes. Se o serviço usa poucos dados, o crescimento do seu custo é pago pelo aumento da receita, mas quando é *data-intensive* a conta não fecha tão fácil e dificuldades de crescimento surgem mais cedo e mais frequentemente. Embora o IP *Multicast* pudesse ser usado para isso, os provedores de serviço o

[1] `http://youtube.com`
[2] `http:/vimeo.com`

mantém desabilitado. Resta a criação de protocolos de *multicast* na camada de aplicação, em arquiteturas em que os clientes assumem parte da responsabilidade de distribuir o conteúdo, semelhantes às redes *peer-to-peer*.

1.2 Objetivos, Metodologia, Limitações

O objetivo deste trabalho é propor um mecanismo de distribuição de vídeo digital escalável que realize retransmissão seletiva dos pacotes perdidos. Assim, este trabalho realiza uma análise das perdas e formas de recuperação, problemas que não vinham sendo tratadas em redes sobrepostas anteriores [Tran, Hua e Do 2003, Zhang et al. 2005, Wang, Xiong e Liu 2007, Zhang et al. 2005].

Para isso, adotou-se a seguinte metodologia:

1. Estudar as técnicas para transmissão de multimídia em tempo real, principalmente nas redes sobrepostas.

2. Outro estudo é feito para determinar características do mais recente padrão de vídeo digital – o H.264 – que permitam definir critérios que auxiliem na criação de um mecanismo inteligente de recuperação de perdas. Neste contexto, inteligente significa que as perdas que impactariam mais severamente a qualidade do vídeo teriam prioridade na recuperação.

3. O mecanismo de retransmissão é especificada na forma de algoritmos que controlem a escolha dos pacotes para recuperação. As condições para a execução dos algoritmos também são descritas. São três os algoritmos propostos, com diferentes graus de flexibilidade no tratamento das perdas, a fim de determinar o que se encaixa melhor a cada situação.

4. Um protótipo da proposta foi então implementado para comparar os algoritmos de retransmissão propostos com uma recuperação Tradicional que ignora o conteúdo do vídeo.

Espera-se que a retransmissão seletiva aceite perdas mais altas em relação ao modo tradicional. Mas isso deve ser compensado por dois benefícios: menor tráfego com retransmissões e perdas 'inteligentes', isto é, que afetem o mínimo possível a qualidade.

Este trabalho apenas estuda critérios para retransmissão de vídeo, de modo que os pacotes contendo áudio ou outros dados recebam um tratamento padrão e não são investigados a fundo. Sendo o H.264 o padrão mais recente e avançado de compressão de vídeo, foi o único adotado, de

modo que as prioridades são definidas tendo por base os tipos dos dados declarados nos cabeçalhos das partes do vídeo. Embora facilmente possa-se definir uma função que ofereça suporte a outros formatos, mantendo-se os algoritmos propostos aqui. Também, a rede sobreposta apresentada é orientada a dados pura – isto é, não-híbrida, pois segue a abordagem de *PULL* [Zhang et al. 2005] – deste modo o trabalho foca principalmente nos critérios de retransmissão. Entretanto, a organização da rede é ortogonal à forma de transmissão, de forma que o protocolo pode ser complementado com mecanismos *PUSH*.

1.3 Estrutura do Trabalho

A sequência deste trabalho foi organizada nas seguintes divisões:

- Os conceitos básicos da representação digital de vídeo em geral e o H.264 em particular são apresentados em **Vídeo Digital – Capítulo 2**, junto de alguns requisitos específicos à transmissão de multimídia.
- Em seguida, **Arquiteturas para Sistemas de Vídeo em Rede – Capítulo 3** analisa algumas arquiteturas de sistemas distribuídos, apresentado também os inconvenientes de cada uma delas para o cenário de uso alvo, ou seja: TV pela Internet. A lista inclui: cliente-servidor, CDNs, P2P, *IP Multicast* e outras.
- **Trabalhos Relacionados – Capítulo 4** trata das redes sobrepostas para vídeo ao vivo pela Internet, incluindo as baseadas em árvore de distribuição e as orientadas a dados, e também as híbridas, que suportam *PUSH* e *PULL*. Também são incluídas duas técnicas de retransmissão seletiva, não voltadas para redes sobrepostas.
- A proposta é apresentada a seguir, em **Rede Sobreposta *SeRViSO* – Capítulo 5**, contendo a motivação para superar as técnicas anteriores, a arquitetura, os tipos de nós e seus relacionamentos, a organização e construção da rede, o mecanismo de transmissão (incluindo a retransmissão), algoritmos e os tipos de mensagens empregados.
- A rede proposta é avaliada em **Avaliação – Capítulo 6**, incluindo as características do protótipo e a metodologia dos testes, e terminando por apresentar os resultados – trazendo os gráficos (e tabelas) do tráfego gerado, mensagens de NACK e perdas – junto das interpretações dos mesmos.
- Finalmente, **Conclusão – Capítulo 7** apresenta as conclusões finais deste estudo e o resumo do pontos mais importantes, incluída uma perspectiva sobre trabalhos futuros neste tema e/ou relacionados.

Capítulo 2

Vídeo Digital

Desde os primeiros métodos criados para gravação e reprodução de vídeo, este consistiu de sequências de imagens estáticas que, quando apresentadas uma após a outra com pequeno intervalo, apresentam sensação de movimento ao sistema visual humano. Tais imagens têm muitas semelhanças, isto é, duas imagens consecutivas possuem quase todos os objetos em posições muito próximas na cena, de outra forma os objetos 'saltariam' de posição quando exibido o vídeo. Outras semelhanças ocorrem dentro de cada imagem, num céu azul, em nuvens, numa camisa, etc.

Todo o processo de compressão de vídeo digital pode ser resumido na divisão do mesmo em partes menores de forma que os padrões repetidos, e portanto redundantes, possam ser detectadas. Essa redundância pode ser tanto dentro de uma mesma imagem quanto entre imagens consecutivas. Como uma compressão sem perdas não atingiria os níveis necessários, a compressão com perdas é adotada, mas utilizando mecanismos que minimizem as perdas perceptíveis à visão humana. O estudo apresentado nesse trabalho serve de base para os algoritmos propostos no capítulo 5, pois somente explorando a representação digital do vídeo eles poderão ser seletivos, como é o objetivo deste trabalho.

A divisão do vídeo em partes menores é um processo básico que é semelhante em todos os padrões baseados em blocos, embora detalhes como o tamanho dos blocos possam variar. Na figura 2.1 tem-se uma visão geral das partes nas quais um vídeo é quebrado para codificação no MPEG-2. Os padrões MPEG-1, MPEG-2, H.263 e MPEG-4, entre outros, são resultados dos trabalhos do grupo JVT[1][2], formado pelos especialistas em codificação de vídeo da ISO[3] (MPEG[4]) e da ITU-T[5] (VCEG[6]).

[1] *Joint Video Team*
[2] `http://www.itu.int/ITU-T/studygroups/com16/jvt/`
[3] *International Organization for Standardization*
[4] *Motion Picture Experts Group*
[5] *International Telecommunication Union - Telecomuncation Standardization Sektor*
[6] *Video Coding Experts Group*

Os padrões são publicados pelas duas organizações com nomes diferentes (MPEG-*x* e H.26*x*). Os conceitos apresentados abaixo correspondem a esta família de padrões como um todo [Wiegand et al. 2003]:

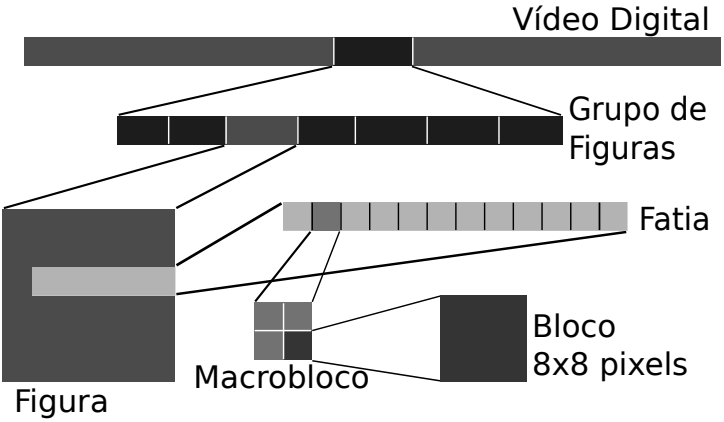


Figura 2.1: MPEG-2 – Visão Geral.

1. **Grupo de Figuras** (GOP[7]): embora explorar as repetições entre diferentes figuras permita aumentar a taxa de compressão, quando ocorrem perdas os erros se acumulam – chamado efeito à deriva (*drift*) – de modo que é importante inserir pontos de sincronização (figuras auto-contidas). Neste ponto há a garantia que as imagens anteriores não serão mais usadas na compressão das posteriores. As figuras do grupo compartilham uma relação de dependência com a figura de sincronização (a primeira) – visto necessitar de seus dados direta ou indiretamente – e entre si;

2. **Figura** (*Picture*): é uma das imagens individuais que compõem o vídeo quando exibidas em sequência. Além de figuras que permitem a sincronização, existem outros tipos com variados graus de compressão.

[7] *Group of Pictures*

Para a criação dos primeiros padrões da televisão analógica, algumas adaptações foram feitas voltadas à tecnologia da época: tubos de raios catóticos que de início não tinham frequência de varredura muito alta. A solução encontrada foi: em vez de capturar e exibir da primeira à última linha a cada quadro – vídeo progressivo – são usadas duas figuras (chamadas campos) em sequência, cada uma contendo uma parte da cena, uma com as linhas pares e outra as impares – vídeo entrelaçado (figura 2.2). Desta forma a quantidade de informações necessária é a metade e ainda que a frequência de varredura seja a mesma, apenas metade das linhas precisam ser atualizadas a cada varredura. Ainda que isso não faça mais sentido para a tecnologia digital presente nos televisores e monitores LCD e plasma, seu legado persiste na opção por vídeo entrelaçado presente nos padrões atuais de vídeo digital. Também existe a possibilidade de separar um imagem em campos capturados no mesmo instante, o que não torna o vídeo entrelaçado;

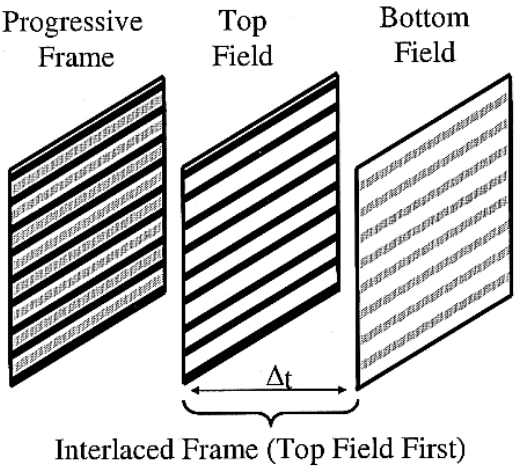


Figura 2.2: Vídeo Progressivo e Entrelaçado [Wiegand et al. 2003].

3. **Opção por codificar uma Figura como um Quadro ou dois Campos**: a relação estatística entre duas linhas adjacentes é menor quando há movimentação, numa situação onde pode ser vantajoso utilizar campos como num vídeo entrelaçado, já com cenas paradas ocorre o inverso. Por isso, o programa codificador deve usar o modo

campo ou modo quadro de acordo com a presença de movimento na cena – recurso chamado *Adaptive Frame/Field Coding*. Mas como é comum as cenas conterem regiões com e sem movimento, existe o *Macroblock-Adaptive Frame/Field*, que partes menores da imagem (um macrobloco no MPEG-2 ou um par deles no H.264) escolham o modo de codificação mais apropriado;

4. **Fatia**: as figuras são quebradas em fatias independentes entre si (mas possivelmente dependentes de fatias de outras figuras), limitando assim a predição espacial. Geralmente os macroblocos que compõem a fatia são tomados em ordem de varredura (figura 2.3). São as fatias que podem ser armazenadas e transmitidas individualmente, não as partes menores (macroblocos e blocos);

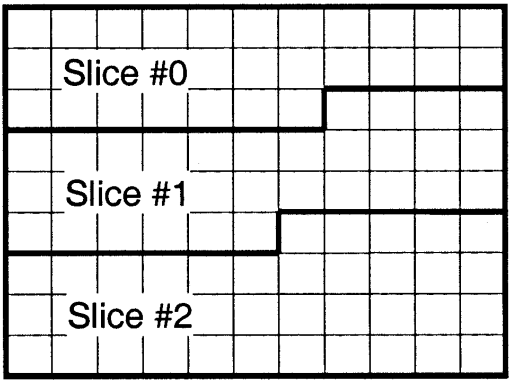


Figura 2.3: Fatias [Wiegand et al. 2003].

5. **Macroblocos**: são as unidades básicas para as operações de codificação. Eles são preditos, ou seja, (de)codificados baseados em outros macroblocos – de outros quadros (predição temporal) ou de macroblocos anteriores na mesma fatia (predição espacial). Esta última também é chamada de compensação de movimento, visto que entre dois instantes do mesmo vídeo um objeto pode ter se "movido" na cena. As amostras de cores não seguem o sistema RGB[8] (com componentes vermelho, verde e azul), mas o $YCbCr$[9]. Origi-

[8] *Red, Green, Blue*

[9] Y é *lumma* (brilho), e Cb e Cr são componentes de *chroma* – o quanto a cor se aproxima do azul e do vermelho e se afasta do tom de cinza original gerado pelo *lumma*

nalmente são 8 bits por amostra de cada um dos componentes Y, Cb e Cr, mas padrões mais recentes podem suportar, por exemplo, 10 bits para uso em aplicações profissionais de altíssima fidelidade visual. Além disso, cada macrobloco (que representa uma parte quadrada da figura) tem mais amostras de *lumma* do que de *chroma*, visto que a visão humana é mais precisa quanto ao brilho. Ou mais precisamente: 16×16 amostras de *lumma* e 8×8 de *chroma*. Essa amostragem é chamada de 4:2:0 e usa um terço do espaço em relação à amostragem 4:4:4, que tem quantidade de amostras igual para todos os componentes. A figura 2.4 traz essas e algumas das outras subamostragens possíveis;

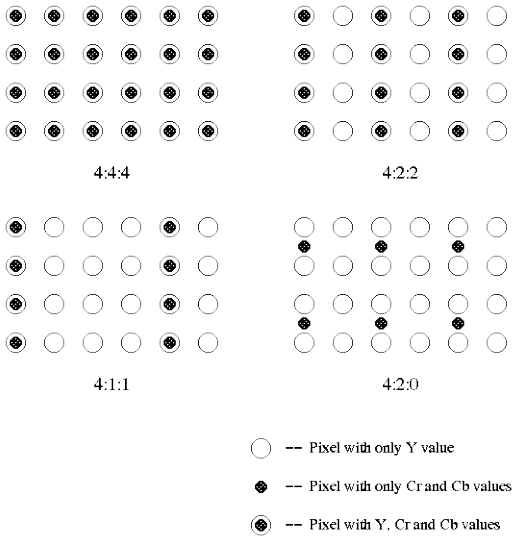


Figura 2.4: Subamostragem de *Chroma* [Basics of Video 2010].

6. **Bloco**: as diferenças entre os valores da predição e as cores originais sofrem uma codificação especial, e para isso são usados os blocos (são menores que os macroblocos). Eles sofrem uma transformada de cosseno (que possui uma transformação inversa para decodificação) e tomados em zigzag para a aplicação de um método de codificação por entropia, aproveitando o fato de que os coeficientes importantes (isto é, diferentes de zero) foram concentrados no início da sequência. A figura 2.5 mostra como funciona a codificação

zigzag em imagens JPEG.

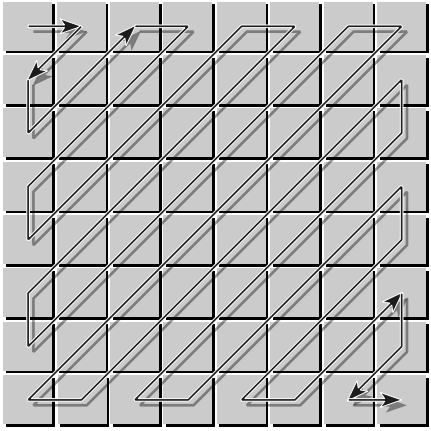


Figura 2.5: Codificação Zigzag em imagens JPEG [JPEG, Entropy Encoding 2010].

Um grupo de figuras inclui quadros codificados com diferentes processos de predição (figura 2.6) para melhorar a compressão:

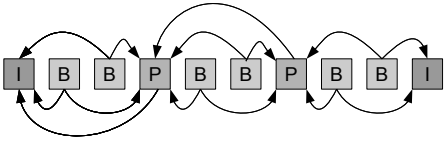


Figura 2.6: Grupo de Figuras

1. ***I**ntra*: esse tipo de quadro é inteiramente autocontido. Permite apenas predição espacial (entre macroblocos da mesma fatia). Atinge a menor compressão mas realiza a sincronização, por isso sua perda afeta a reprodução do grupo inteiro. Todos os macroblocos da fatia devem usar predição **I**;
2. ***P**reditiva*: predito com auxílio de quadro anterior **I** ou **P**. No momento da codificação pode-se determinar quantos deles ocorrem entre dois quadros **I**. Oferece compressão média e tem custo (em termos de uso de *buffers* e complexidade computacional) inferior aos

quadros **B**. Neste tipo de fatia, alguns dos macroblocos também podem usar predição **I**, se for conveniente. Quando é perdida, afeta a reprodução dos quadros **B** vizinhos, do **P** seguinte (diretamente) e possivelmente dos subsequentes (indiretamente);

3. ***B**idirecional*: predito com auxílio de um quadro anterior e outro posterior (ambos **I** ou **P**). Podem existir vários em sequência, dependendo dos parâmetros de codificação. Permite a maior compressão dos dados e sua perda não afeta a decodificação de outras figuras. Os macroblocos da fatia podem usar qualquer tipo de predição.

2.1 H.264

A especificação do MPEG-4 foi concluída em 1999. Ela é dividida em diversas partes, e cada uma delas detalha um aspecto, como: parte 1 – Sincronização e multiplexação Áudio/Vídeo, parte 2 – Vídeo[Ebrahimi e Horne 2000], parte 3 – Áudio, etc. O objetivo desse padrão é permitir a codificação de vídeo de forma mais comprimida que o MPEG-2, mantendo a qualidade. Assim poderia alcançar taxas de *bits* que permitissem a transmissão na Internet. Mas o desenvolvimento não parou e a parte 10 – AVC[10] foi acrescentada em 2003, permitindo maior compressão de vídeo em relação à parte 2. O nome H.264 corresponde à denominação dada pelo ITU-T.

Comparando o AVC em relação à "Parte 2 - Visual" e aos padrões anteriores, muitas das melhorias consistiram em flexibilizar o uso de procedimentos já existentes, de forma que os programas de codificação pudessem alcançar maiores taxas de compressão. Como apenas o processo de decodificação de entradas válidas foi especificado, os codificadores tem liberdade para buscar formas mais eficientes, contanto que emitam conteúdo H.264 válido. Recursos avançados [Wiegand et al. 2003]:

1. **Sequência** (anteriormente Grupo) **de Figuras**: não precisa mais seguir regras tão rígidas como as da figura 2.6, de forma que não apenas a figura anterior pode ser usada para a predição como qualquer figura anterior da sequência, ou mesmo uma lista de figuras (figura 2.7). O codificador pode usar a predição do tipo **P** quantas vezes quiser, deixando o tipo **B** para quando isso realmente aumentar a taxa de compressão. O gerenciamento do *buffer* de figuras anteriores que podem ser usadas para predição ficou mais complexo;

2. **Predição ponderada**: útil para tratar de sombras e objetos cuja iluminação incidente mude durante a cena;

[10] *Advanced Video Coding*

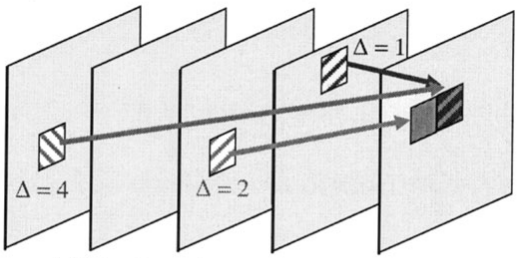


Figura 2.7: H.264 – Múltiplos quadros como referência [Wiegand et al. 2003].

3. **Codificação I**: antes uma figura **I** indicava que as figuras anteriores não eram mais necessárias, mas agora esse papel é das fatias **IDR** (*Instantaneous Decoding Refresh*). As fatias **I** ainda são autocontidas, mas não delimitam as sequências porque uma figura posterior pode ser predita com base em fatias anteriores à ela;

4. **Codificação B**: tais fatias podem ser usadas como referência na predição;

5. **Transformação Inteira**: não gera erro por arredondamento na hora de aplicar a transformação inversa, nem diferença entre implementações (como na transformação de cosseno). Pode ser implementada usando apenas adição e deslocamento de inteiros de 16 bits;

6. **Ordenação de Macroblocos Flexível**: usa grupos de fatias para "mesclar" vídeos de diferentes origens ou mesmo separar macroblocos adjacentes em grupos diferentes, respectivamente os dois exemplos da figura 2.8. Desse modo, caso um seja perdido, técnicas de recuperação pode ser adotadas para cada macrobloco, a partir dos dados dos macroblocos vizinhos no grupo de fatias que não foi perdido;

7. **Codificação de Entropia**: Todos os elementos de sintaxe do padrão (exceto os coeficientes da transformada) são codificados numa única tabela que comprime as sequências em palavras-chave de tamanho variável. Duas codificações que se adaptam ao contexto estão disponíveis para os coeficientes da transformada: CAVLC e CABAC (que não está disponível em todos os perfis);

8. **Filtro Removedor de "Blocos"**: devido ao algoritmo ser baseado

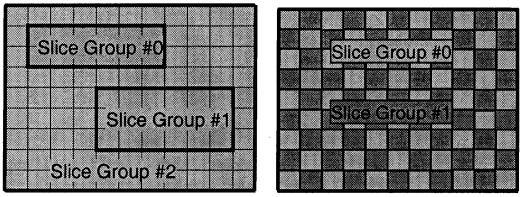


Figura 2.8: H.264 – Ordenação de Macroblocos Flexível [Wiegand et al. 2003].

em blocos, as bordas destes podem ficar destacadas, piorando a qualidade da imagem. Quando o filtro detecta este problema ele suaviza a borda, senão o bloco não é alterado. Outro benefício é que a predição usa os blocos após o filtro, que tem mais qualidade;

9. **Partições**: permitem dividir os dados de uma fatia. Existem três tipos, apresentados aqui em ordem crescente de tamanho e decrescente de importância:

   (a) **A** Contém os cabeçalhos e permite a reprodução parcial;

   (b) **B** Contém as texturas de codificação **I**;

   (c) **C** Contém as texturas de predição **P** ou **B**.

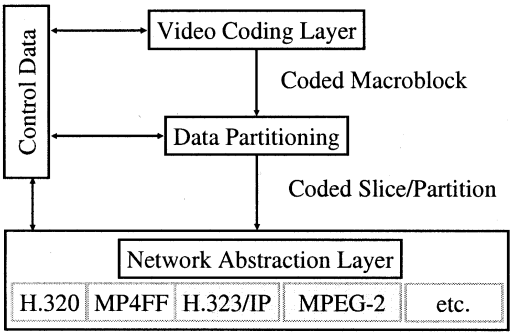


Figura 2.9: H.264 – Camadas [Wiegand et al. 2003].

O H.264 permite o particionamento dos dados que são codificados de forma a acomodar os requisitos da aplicação. Como a figura 2.9

mostra, diferentes formas de encapsulamento podem ser usadas. Esses aspectos permitem atender aos requisitos da transmissão de vídeo e do mero armazenamento, que podem entrar em conflito quanto a:

1. Tamanho dos pacotes;
2. Mecanismos de correção de erro;
3. Controle de fluxo.

A camada de abstração de rede foi introduzida para permitir a codificação segundo padrões anteriores de transmissão de vídeo ao mesmo tempo que uma nova representação em mais baixo nível é usada. São os NALUs, pacotes que facilitam a adaptação ao MTU[11] da rede. Podem conter [Wenger 2003]:

1. **Fatia**;
2. **Partição**;
3. **Conjunto de parâmetros de figura ou de sequência**: são informações pertinentes à decodificação de uma figura ou de uma sequência inteira. É mais importante que uma partição **A** ou uma figura **I**.

Ele possui um cabeçalho (figura 2.10) simples de um *byte* com o seguinte formato:

1. **Tipo de dados**: 5 *bits*;
2. **Importância para referências**: 2 *bits*. O codificador indica o quanto outros NALUs precisarão deste;
3. **Falha**: 1 *bit*. Equipamentos de rede que encontrarem erros na validação de pacotes e entendam o H.264 podem entregar o pacote com o *bit* marcado, de forma que a aplicação pare de esperar por ele e o decodificador saiba que deve descartá-lo.

No padrão também existem variantes das fatias **I** e **P** que permitem usar diferentes figuras como referência. Esses aspectos não são usados neste trabalho; as variantes são tratadas como os tipo originais. Os algoritmos de seleção deste trabalho (seção 5.6) se baseiam no campo tipo do cabeçalho NALU.

[11] *Maximum transmission unit*

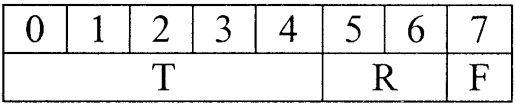


Figura 2.10: H.264 – Cabeçalho NALU [Wenger 2003].

2.2 Transmissão de multimídia

O capítulo 3 apresenta arquiteturas que podem ser adotadas para a construção de sistemas em rede. Esta seção traz apenas características específicas que influenciam na escolha da arquitetura e protocolos comumente usados para a transmissão de multimídia.

A transmissão de vídeo digital exige uma quantidade de dados considerável (maior que outras classes de aplicação) e uma taxa de *bits* aproximadamente constante. Em redes com reserva de recursos (ISDN[12], por exemplo) esses recursos são automaticamente atendidos (arcando com o seu custo), mas quando a garantia de entrega é por melhor-esforço ocorrem variações na largura de banda disponível, devido à competição pelos recursos com outras aplicações. Já o provisionamento de recursos [Braden et al. 1997, Nichols et al. 1998] não é uma solução disponível para todos os clientes de um serviço aberto na Internet. Algumas preocupações relacionadas à qualidade de serviço percebida pelo usuário são derivadas desse problema:

1. **Atraso**: uma rede que não suporta a vazão requerida atrasa a entrega dos pacotes, que passam a se acumular nas filas do roteadores, e uma parte é eventualmente descartada. Se a transmissão for com garantia de entrega (TCP[13]) eles serão re-enviados, mas toda a transmissão é atrasada porque a entrega é por ordem. Em transmissões sem garantia de entrega (UDP[14]) o ideal é usar protocolos com controle de congestionamento (seção 2.2.2). Em qualquer situação a aplicação precisará realizar a *buferização* para manter a continuidade quando da exibição, aumentando a latência (tempo de espera) para a exibição;

2. **Variação de Atraso** (*jitter*): quando os pacotes chegam com atrasos diferentes, essa variação deve ser compensada, seja por um período

[12] *Integrated Services Digital Network*
[13] *Transmission Control Protocol*
[14] *User Datagram Protocol*

de bufferização inicial preventivamente mais longo, ou através de uma parada durante a exibição, ou aceitando a perda de parte dos dados;

3. **Sincronização entre Mídias**: ocorre quando as mídas são transmitidas em pacotes separados. Existe um limite de tolerância da percepção humana quanto a eventos relacionados, como o movimento dos lábios e o som. Dessa forma, existe uma pequena tolerância em que não ocorre perda nem o usuário percebe a falta de sincronia entre as mídias (vídeo, áudio, legenda).

4. **Perdas**: vídeo digital comprimido reage a perdas de igual volume com diferentes graus de intensidade, dependendo do lugar onde ocorrem, sejam cabeçalhos, vetores de movimento, coeficientes das transformadas, etc. Os erros se propagam com maior ou menor impacto dependendo do tipo do quadro, até o próximo ponto de sincronização.

   As perdas precisam ser consideradas inclusive para decidir a taxa de *bits* usada na compressão do vídeo. Foi demonstrado que o aumento da taxa não necessariamente aumenta a qualidade percebida pelo usuário, devido às perdas geradas quando o limite do canal é excedido [Verscheure, Frossard e Hamdi 1999].

Esses fatores estão inter-relacionados. Por exemplo, as técnicas de recuperação de perdas afetam um canal já sobrecarregado aumentando a variação do atraso momentaneamente. Assim, elas são melhor aplicadas quando a degradação da qualidade devida à perda for maior que a devida ao atraso. Isso porque a percepção de perda de qualidade é semelhante nos dois casos [Claypool e Tanner 1999].

2.2.1 Protocolos

Vídeo sob Demanda geralmente é feito sobre o TCP, com *bufferização* para buscar maior continuidade, como faz o *YouTube* e outros serviços similares. Entretanto, para limitar o atraso, o *streaming* de áudio ou multimídia ao vivo é geralmente feito sobre UDP.

Como as aplicações com transmissão de multimídia têm requisitos em comum, o RTP foi definido para fornecer esses mecanismos básicos em redes IP. Esse protocolo é adaptado por especificações e aplicações mais específicas dependendo do tipo de mídia (áudio e/ou vídeo), *codec* (MPEG-1 Layer 3, MPEG-2, H.264, etc), e aplicação (*streaming*, audio/-vídeo conferência, VoIP[15], etc). Características do RTP [Schulzrinne et al.

[15] *Voice over IP*

2003, Liu 1999]:

1. Baseado em sessão, identificada pela porta UDP e endereço IP onde os dados são recebidos. Cada mídia deve ser transmitida em uma sessão própria;

2. Pode funcionar com *multicast* para conferências, por exemplo;

3. Algumas aplicações como VoIP usam *codecs* de taxa de *bits* fixa (*constant bit rate*) e intervalo entre pacotes constante. Já vídeo é mais eficiente quanto codificado a taxa variável (*variable bit rate*). O RTP é indiferente ao tamanho dos pacotes e não especifica formas para garantir do canal de comunicação a taxa necessária;

4. Cabeçalho com 12 *bytes*, além dos 8 e 20 *bytes* dos cabeçalhos UDP e IP (figura 2.11). Entre os campos estão o tipo do *payload*, o número de sequência e o *timestamp*, de forma que áudio e vídeo possam ser sincronizados e a perda de pacotes detectada no cliente;

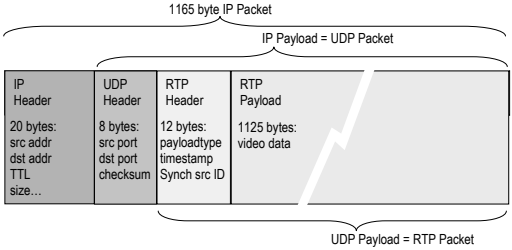


Figura 2.11: Exemplo de pacote RTP [MacAulay, Felts e Fisher 2005].

5. Requer um protocolo de controle associado (RTCP[16]), operando numa sessão separada. Ele é usado para fornecer *feedback* periodicamente a respeito da qualidade da transmissão. Todos os participantes são avisados em conferências, de modo que cada um possa descobrir se os problemas são locais ou globais. A partir dessas informações os emissores podem determinar a troca do *codec* ou de seus parâmetros, a fim de reduzir as perdas;

6. A configuração e controle de chamada deve ser definida por outro protocolo de acordo com as necessidades da aplicação, como

[16] *RTP Control Protocol*

o RTSP[17] [Schulzrinne, Rao e Lanphier 1998] para Vídeo sob Demanda, e o SIP[18] [Rosenberg et al. 2002] ou o H.323 para telefonia [Schulzrinne e Rosenberg 1998].

Uma especificação [Wenger et al. 2005] define o formato do *payload* para vídeo H.264. De fato, esse padrão foi concebido também levando em conta a transmissão por RTP [Wenger 2003]:

1. Os NALUs podem ser transmitidos cada um em um pacote RTP, ou ser quebrados (quando maiores que o MTU) ou unidos (quando pequenos), mesmo se tiverem *timestamps* diferentes;

2. Os "conjuntos de parâmetros" podem ser transmitidos previamente através de algum protocolo de controle com garantia de entrega, dada a sua importância para vários pacotes.

2.2.2 Controle de Congestionamento e TFRC

Os protocolos mais importantes para os usuários da Internet são baseados no TCP [Postel 1981]: HTTP[19] [Fielding et al. 1997], SMTP[20] [Postel 1982], IMAP[21] [Crispin 2003], etc. Seguem as regras do TCP para uso dos recursos da rede, adaptando, portanto, o volume de dados que pode ser transferido pela aplicação à taxa de erros do canal de comunicação, valores que variam. A taxa de erros é influenciada pelo volume de dados transferido, de modo que quando este se aproxima do limite do canal, mensagens passam a ser perdidas e a taxa sobe. O TCP dispõe de um mecanismo que ajusta a vazão em função da taxa de erro e, portanto, *bufferiza* mensagens. Como cada canal de comunicação possui seu limite físico (ou contratado), permitir vazão além do limite piora a comunicação exponencialmente.

A especificação do TFRC[22] [Handley et al. 2003] oferece um mecanismo de controle de congestionamento, para uso em comunicação *unicast*. Mas não há um protocolo TFRC definido, de forma que duas implementações sejam interoperáveis. Por exemplo, os formatos das mensagens transferidas são definidos apenas conceitualmente, isto é, que informações são necessárias, mas não a representação das mesmas.

Visto que o fluxo TCP é tão importante para a Internet, quando aplicado à transmissões em tempo real, o mecanismo do TFRC oferece um

[17] *Real Time Streaming Protocol*
[18] *Session Initiation Protocol*
[19] *Hypertext Transfer Protocol*
[20] *Simple Mail Transfer Protocol*
[21] *Internet Message Access Protocol*
[22] *TCP Friendly Rate Control*

controle de congestionamento próximo do funcionamento do TCP. Mas, como seu objetivo é controlar transmissões de tempo real como as de telefonia e distribuição de mídia (seção 2.2), ele não responde perdas de pacotes individuais, mas sim a eventos de perda: qualquer quantidade de perdas dentro de um intervalo de tempo igual ao atraso fim-a-fim conta um evento de perda. Desta forma, o TFRC pode responder mais suavemente às perdas. Seu mecanismo foi projetado para aplicações que utilizem pacotes de tamanho fixo (ou aproximadamente fixo) e desejem responder ao congestinamento através da diminuição da taxa de transmissão.

O controle mais complexo ocorre no lado receptor: as últimas mensagens devem ser guardadas e os eventos de erro calculados, a fim de calcular a taxa de perda que é informada numa mensagem de *feedback*. Essa característica determina a utilização de mais ciclos de CPU[23] no nó cliente, economizando o servidor, que normalmente já controla múltiplas conexões. Além da taxa de erro, o TFRC mede e usa o atraso fim-a-fim para determinar a taxa de emissão permitida.

A fórmula simplificada da vazão (*throughput*) que o TFRC aplica para limitar uma transmissão é a seguinte:

$$X = \frac{s}{R * f(p)}$$

$$f(p) = \sqrt{\frac{2p}{3}} + 12 \sqrt{\frac{3p}{8}}\, p\, (1 + 32p^2)$$

onde:

- s: tamanho (médio) dos pacotes;
- p: taxa de perda;
- R: atraso fim-a-fim.

Os itens do cabeçalho da mensagem de dados do emissor do conteúdo e o conteúdo das mensagens de *feedback* do receptor são detalhados na seção 5.8.

Como o TFRC é baseado em mensagens *unicast*, para usá-lo para *multicast*, cada destino deve ser controlado individualmente, tanto no receptor quanto no emissor.

[23] *Central Processing Unit*

Capítulo 3

Arquiteturas para Sistemas de Vídeo em Rede

Em [Coulouris, Dollimore e Kindberg 2001], os autores definem sistema distribuído como "aquele no qual componentes localizados em computadores em rede se comunicam e coordenam ações apenas pela passagem de mensagens". Eles também apresentam o compartilhamento de recursos como a principal motivação para sua construção. Em relação às dificuldades para sua construção, incluem "heterogeneidade dos componentes, abertura, que permite que componentes sejam adicionados ou substituídos, segurança, escalabilidade – a habilidade de trabalhar quando o número de usuários cresce – tratamento de falhas, concorrência dos componentes e transparência."

Existem diversas formas de se construir um sistema distribuído, cada uma delas mais voltada a um tipo diferente de aplicação. Na sequência deste capítulo serão apresentadas algumas destas formas, de modo que fique claro como as características de cada uma a tornam mais ou menos adequadas ao objetivo deste trabalho, ou seja, transmitir vídeo ao vivo pela Internet visando economia de recursos. Os itens foram ordenados de forma que cada um deles oferece uma alternativa mais próxima do desejado que o anterior.

3.1 Arquitetura Cliente-Servidor

É a forma mais simples de organização de um sistema distribuído, por isso também a mais comum. Existem dois papéis: o servidor fornece e o cliente acessa um serviço. Em aplicações mais complexas um servidor pode ser cliente de outro serviço, numa espécie de composição de serviços, por exemplo: um servidor *web* que obtém os arquivos de um sistema de arquivos em rede e executa chamadas a um servidor de aplicações, que por sua vez acessa um banco de dados.

Visto que essa arquitetura lida com questões de projeto de uma forma que "combina de forma bem próxima o fluxo dos dados com o fluxo de controle" – promovendo modularidade, flexibilidade e extensibilidade – ela favorece a divisão de aplicações em serviços. Mas essa

visão restrita baseada na requisição de serviços individuais por clientes torna-se inconveniente quando são muitos os serviços que precisam ser coordenados [Adler 1995].

Não é o caso de uma aplicação de TV pela Internet, que tem uma interface simples, mas que transfere muitos dados. Um outro problema dessa arquitetura é que o servidor é um ponto único de falha (diminuindo a disponibilidade) e possivelmente o gargalo no desempenho. A solução é a replicação, que oferece "multiplas cópias consistentes dos dados em processos sendo executados em diferentes computadores" [Coulouris, Dollimore e Kindberg 2001]. Isso normalmente é feito através de servidores no mesmo *data center*, sendo que ao cliente é transparente o acesso ao servidor *virtual*, isto é, ele não sabe qual máquina efetivamente responde às suas requisições.

Entretanto, mesmo com maior disponibilidade, a largura de banda permanece um problema, como se vê na figura 3.1. Ela (junto do número de servidores) deve ser provisionada para o volume máximo de dados esperado. Desse modo, se ocorrerem picos de acesso não esperados, parte dos clientes é derrubada ou todos tem seu serviço degradado.

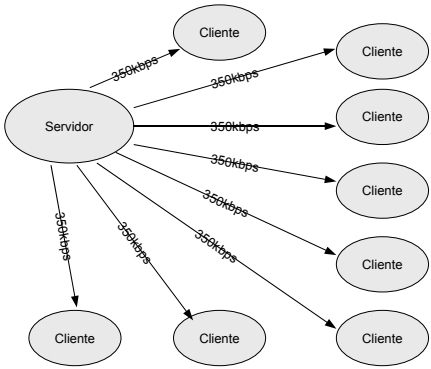


Figura 3.1: Largura de banda – TV pela Internet.

3.2 CDNs

Essas redes oferecem servidores de réplicas (do conteúdo) com a função de entregar o conteúdo em vez dos servidores de origem, podendo estar junto da origem (por exemplo, mesma companhia) ou em outros lu-

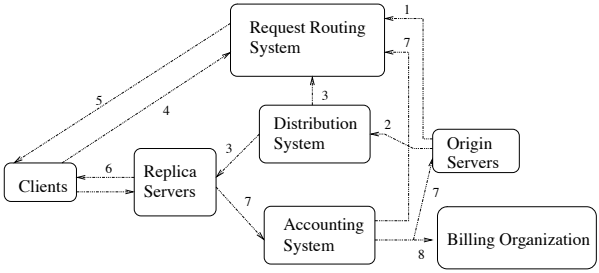


Figura 3.2: Componentes de uma CDN[1] [Peng 2004].

gares, como um serviço contratado [Krishnamurthy, Wills e Zhang 2001]. Elas permitem tratar o problema da latência de acesso aos dados na Internet colocando os servidores de réplicas mais próximos dos clientes. Desta forma, não só o desempenho melhora, mas também diminui a quantidade de tráfego que precisaria atravessar caminhos mais longos na rede. Dentre os componentes das CDNs (apresentados na figura 3.2), os aspectos mais importantes para o seu melhor funcionamento são [Peng 2004]:

1. **Localização dos *Replica Servers***. O Akamai [Kontothanassis et al. 2004] utiliza *data centers* com servidores espalhados pelo globo terrestre. Obviamente essa distribuição é de acordo com o interesse de seus Clientes em atender essas áreas, de modo que clientes distantes das regiões obterão um serviço pior;

2. ***Distribution System***: duas formas de distribuição do conteúdo às réplicas são comuns, através de uma árvore ou rede sobreposta sobre a própria Internet; ou usando um satélite de *broadcast*;

3. ***Request Routing System***: deve levar em conta a distância entre o cliente e os servidores de réplica que possuem o item desejado e a carga dos mesmos para direcionar as requisições para o mais apropriado.

Como as CDN são gerenciadas por uma equipe especializada contratada para isso, o provedor do conteúdo pode abstrair esses problemas de escalabilidade. Mas os custos permanecem, embora uma CDN possa distribuir os recursos não usados por uma aplicação ou empresa cliente para os que necessitam nos picos de acesso, o que permite um crescimento mais "controlável". Isso pode trazer uma pequena redução no custo para o

provedor do conteúdo, dependendo de como a controladora da CDN avalia o valor agregado (*value-added*) por seu serviço. O consumo de largura de banda na ponta (servidor de réplica) não diminui, de modo que nenhum cliente agrega capacidade ao sistema, como nas redes *Peer-to-Peer*.

3.3 Redes P2P

O propósito das redes P2P[2] é permitir o compartilhamento de conteúdo, e considerando que a Internet oferece desafios – p.ex, escalabilidade, nós com conexões intermitentes ou que abandonam a aplicação ou o serviço) – e existem objetivos secundários – roteamento e busca eficiente, anonimicidade, confiança, etc – diversas propostas foram feitas. O *survey* [Lua et al. 2005] oferece detalhes mais específicos que os aqui presentes, que pretendem apenas apresentar os detalhes e a contextualização relevantes à distribuição de vídeo.

Nas redes P2P a transferência de dados também ocorre entre os clientes, de modo que o servidor ou a origem do conteúdo não precisa prover o conteúdo inteiro a cada cliente individualmente. Deste modo, um cliente opera como servidor ao espelhar o conteúdo já recebido para outros clientes. Essas redes podem ser divididas em dois grupos – estruturadas e não-estruturadas – que dizem respeito à topologia e como o conteúdo é distribuído. Também devido à forma de construção de cada grupo, eles oferecem características distintas em termos de roteamento e pesquisa por conteúdo, por exemplo.

3.3.1 Redes P2P Estruturadas

Para tornar as buscas mais eficientes, a topologia desse tipo de rede P2P é controlada e o conteúdo deve ser colocado em lugares determinísticos, de acordo com a Tabela de Espalhamento Distribuída (*Distributed Hash Table*). Seu funcionamento se baseia nos IDs dos nós e nas chaves dos objetos armazenados, ambos únicos. Os IDs devem ser aleatórios e uniformemente distribuídos na faixa de valores válidos. A partir do valor da chave, o mapeamento determina um nó como sendo o responsável pelo armazenamento do objeto correspondente.

A escalabilidade é atingida pelo encaminhamento das mensagens – tanto de armazenamento quanto de consulta – através da topologia (grafo de ligações na rede P2P estruturada) até que cheguem ao nó responsável pela chave. As DHTs garantem que ocorram em média $O(log\ N)$ (sendo N o número de nós na rede) saltos nas buscas, o que não evita uma alta latência nessa operação, devido à geografia. Essas redes não foram aplicadas tão extensamente quanto as não-estruturadas. Outros problemas

[2] *Peer-to-Peer*

intrínsecos ao modelo são:

1. A topologia física é ignorada, podendo aumentar o tráfego de longa distância e o atraso;

2. No mapeamento das chaves em nós as diferenças em capacidades de rede são ignoradas, de modo que os objetos cujas chaves forem armazenadas nos nós mais lentos terão pior tempo de acesso.

3. Pesquisas complexas não são possíveis.

3.3.2 Redes P2P Não-Estruturadas

Surgiram voltadas ao compartilhamento de arquivos, de modo que à medida que os nós se juntam à rede e recebem o conteúdo, aumentam também a capacidade da rede como um todo. As pesquisas devem ser por inundação, uma vez que não há estrutura definida na ligação entre os nós, e cada nó que recebe uma pesquisa encaminhada responde diretamente quando possui o conteúdo desejado.

Diferente das redes estruturadas, isso gera menos sobrecarga para conteúdos populares, ocasionando crescimento no custo das pesquisa linear à sua quantidade e ao tamanho do sistema. Conteúdos pouco replicados também oferecem resultados piores, visto que grande parte dos nós precisa receber a pesquisa para esta produzir resultados. É desta forma que o Gnutella [Ripeanu e Foster 2001] originalmente operava, mas os *Ultra-Peers* [Rasti, Stutzbach e Rejaie 2006] foram adicionados para melhorar o desempenho ao realizar o *cacheamento* das informações dos nós que se associam a ele como folhas (i.e, não são outros *Ultra-Peers*).

O BitTorrent [Pouwelse et al. 2005] segue uma abordagem diferente, na qual não há suporte direto à pesquisa, mas os algoritmos de troca entre os nós são especificados visando a justiça, de modo que cada nó busque contribuir com quem lhe está oferecendo algo, limitando os *free-riders* que prejudicam outras redes P2P [Adar e Huberman].

Os metadados sobre o conteúdo estão presentes no arquivo *.torrent*, que inclui tamanho, nome, URL[3] do *tracker* e *hashes* SHA[4]-1 dos pedaços de tamanho fixo nos quais o arquivo foi dividido. Não há pesquisa por torrents no protocolo, esse serviço é oferecido por servidores como o *The Pirate Bay*[5]. O *tracker* é um elemento centralizado que recebe todos os clientes do *torrent* e retorna uma lista aleatória com outros

[3] *Unified Resource Location*
[4] *Secure Hash Algorithm*
[5] http://thepiratebay.org

clientes do mesmo *torrent*. Nesse ponto os clientes passam a se comunicar diretamente, informando as partes que possuem (e já verificaram o *hash*) e pedindo as que necessitam. Para obter melhor desempenho os dados são enfileirados, parte no *buffer* do TCP e parte na memória. Os dados da memória pode ser descartados devido ao critério de justiça, num processo chamado estrangulamento (*choking*). Assim, para obter reciprocidade, um nó pode se recusar temporariamente a enviar dados se não está recebendo. Mas é importante não fibrilar (estrangular e normalizar muito rápido) e também testar eventualmente novas conexões para verificar se elas são melhores.

Embora seja um bom mecanismo para distribuição de arquivos de vídeo, o BitTorrent não oferece entrega dos pedaços em sequência, de modo que, em geral, um cliente deve esperar o *download* completo do arquivo.

3.4 Comunicação em Grupo e *Multicast*

Num sistema distribuído os processos podem se organizar em grupos para oferecer ou usar um serviço ou recurso [Liang, Chanson e Neufeld 1990]. Desse modo, a comunicação entre os membros do grupo deve ser por mensagens endereçadas ao grupo como um todo. Isto significa que ou os membros se conhecem entre si (menos transparente) ou o endereço do grupo por si só permite a entrega para todos os membros.

Existem grupos abertos e fechados, significando respectivamente que qualquer nó pode enviar mensagens ou apenas os membros do grupo. Os grupos também podem ser simétricos ou assimétricos, quando há um processo coordenador organizando os outros. Existem outros critérios de classificação para os grupos, sendo que a ordenação das mensagens na entrega é um deles: sem ordem, FIFO[6] (para as mensagens de cada nó), causal (FIFO mais ordenação entre o recebimento de uma mensagem e o envio das próximas) e ordenação total – se a ordem for cronológica exige um relógio global. A ordenação total pode ser combinada com a FIFO ou a causal [Défago, Schiper e Urbán 2004]. A transmissão de um vídeo, por exemplo, pode ser realizada num grupo fechado e hierárquico, onde o emissor do vídeo é também o gerente. Neste caso a própria aplicação poderia ordenar as partes do vídeo, sem que as mensagens fossem entregues em ordem.

As mensagens *multicast* são destinadas a múltiplos nós, em oposição às mensagens *unicast* – que tem um único destino – e *broadcast*, onde todos os nós da rede local recebem a mensagem. Já as mensagens *multicast* são recebidas por nós em diferentes sub-redes. Geralmente a

[6] *First In First Out*

comunicação em grupo é construída sobre protocolos *multicast*, embora a falta de um protocolo nativo que alcance todos os nós possa ser remediada através de múltiplas mensagens *unicast*, quando os endereços dos nós são conhecidos.

3.4.1 *IP Multicast*

Multicast já existia em nível de redes locais mas o protocolo IP, que é usado para rotear dados entre redes diferentes, não possuía esse recurso. Dois benefícios principais do *multicast* a nível de rede local justificam a sua inclusão no nível inter-redes [Deering e Cheriton 1990]:

1. É mais eficientes que múltiplas mensagens de *unicast*;
2. É mais simples que gerenciar em cada nó os membros do grupo (que seriam destino de mensagens *unicast*).

O *IP Multicast* foi desenvolvido com as seguintes características [Deering 1989]:

1. Grupos abertos e dinâmicos;
2. Endereço IP de destino pertencente à classe "D" (os primeiros *bits* são 1110) que identifica o grupo;
3. Garantia de entrega: melhor esforço;
4. Encaminhamento entre redes por roteadores *multicast*, que podem ser os mesmos *gateways* inter-redes. Se o *time-to-live* do datagrama for maior que 1, o roteador da rede local o encaminha para as outra redes que tem membros do grupo destino da mensagem;
5. O IGMP[7] é usado para que os nós avisem aos roteadores os grupos dos quais são membros;
6. Os roteadores devem se comunicar usando protocolos específicos para construir as árvores de distribuição dos grupos que ficam manifestas nas suas tabelas.

Para acelerar os testes com o *IP Multicast* enquanto os equipamentos de rede em uso não o suportavam foi criado o MBone [Eriksson 1994]. Para isso, um nó em cada rede deve executar um processo *daemon* que assuma um papel semelhante ao roteador *multicast*. Esse processo encapsula as mensagens *multicast* e encaminha para os processo correspondentes em outras redes, formando túneis entre essas redes por onde passa o tráfego *multicast*.

[7] *Internet Group Management Protocol*

Entretanto o *IP Multicast* não foi adotado em larga escala. Alguns motivos foram apontados em [Diot et al. 2000]:

1. **Migração de Roteadores**: normalmente os ISPs acrescentam os roteadores recém-comprados ao *backbone*, e quando um Cliente contrata um *link* mais robusto, o roteador de borda que o atendia é substituído por um mais potente tirado do *backbone*, que é substituído por um novo, ainda mais potente. A empresa poderia ser obrigada a aposentar um roteador do *backbone* sem suporte para o recurso – não inteiramente depreciado – para oferecer o recurso ao Cliente;

2. **Independência de Domínio**: para aplicações com muitas fontes de baixo tráfego, não justifica o uso de árvores de distribuição individuais pela carga imposta sobre os roteadores. Os problemas surgem quando os protocolos que suportam árvores compartilhadas – baseados em RP[8]s – são usados com fontes em diferentes domínios:

   (a) Tráfego externo pode carecer de controle de taxa ou congestionamento;

   (b) Se o RP estiver em outro domínio o ISP[9] não tem controle sobre o serviço que é oferecido aos seus Clientes;

   (c) Não interessa a um ISP oferecer um RP se não tiver nem fontes nem destinos, seria desperdiçar recursos;

   (d) Publicação escalável e de baixa latência do endereço do ponto de encontro.

3. **Dificuldades de Gerenciamento**: relacionadas com legados, como NAT[10] e *firewalls*, exigindo configuração manual para cada nó quando, por exemplo, não há suporte para endereços IP da classe "D";

4. **Pagando os custos**: como os custos de implantação e gerenciamento são mais altos, a adoção do *IP Multicast* acontece quando a economia no uso da rede justifica esses custos.

3.4.2 *Multicast* Confiável

Esses protocolos buscam garantir a entrega de todas as mensagens a todos os receptores, recuperando as perdas que ocorrem pela entrega dos pacotes na rede ser do tipo "melhor-esforço". Outro problema que eles devem tratar é o descarte de mensagens passadas que não são mais

[8] *Rendezvous Point*
[9] *Internet Service Provider*
[10] *Network Address Translation*

necessárias para retransmissão. As estratégias para a construção desses protocolos podem ser encaixadas nas seguintes categorias, de acordo com o sistema de retransmissão [Levine e Garcia-Luna-Aceves 1998]:

1. **Iniciada pelo Emissor**: baseada na recepção de ACKs (*unicast*) para as mensagens recebidas sem erro de todos os receptores, de forma que mantém o estado de cada um a fim de descartar as mensagens já recebidas por todos, num funcionamento que surgiu para comunicações por satélite [Mase et al. 1983]. As retransmissões também podem ser controladas no próprio emissor associando a espera por cada ACK[11] a um *timeout*. Ou no receptor, que manda um NACK ao emissor quando a espera por um pacote termina num *timeout*. Este último mecanismo acelera as retransmissões. Levando em conta que a mensagem perdida pode ser a última, o responsável pela detecção das perdas deve também realizar *polling*. O principal problema com essa estratégia é a implosão do número de mensagens ACK, limitando a escalabilidade dos protocolos;

2. **Iniciada pelo Receptor**: não utiliza confirmações por ACKs, apenas NACKs são enviados pelos receptores quando ocorre o *timeout* ou perdas são detectados [Floyd et al. 1995]. Não é possível ao emissor liberar memória, visto que nunca terá certeza de que todos os receptores tem algum pacote. Para evitar um implosão de mensagens de NACK, essas também são enviadas por *multicast* e apenas após um *timeout* aleatório, de modo que seu envio seja cancelado se for recebido de outro nó um NACK para a mesma mensagem;

3. **Baseada em Árvore**: divide os receptores em grupos locais organizados numa árvore. Os receptores se comunicam apenas com o líder de seu grupo, e este também com o grupo acima. Os líderes são responsáveis por receber as mensagens e encaminhar para os membros do grupo. Desse modo, os ACKs são controlados dentro do grupo, e apenas quando todos os receptores do grupo enviaram ACKs que o grupo acima é avisado, e assim eventualmente o emissor é avisado. NACKs locais podem ser usados para acelerar a restransmissão, de modo que ela seja independente para cada grupo, permitindo aos caminhos mais lentos atrasar apenas o seu próprio grupo local. A árvore elimina o problema da implosão de ACKs, em relação à primeira categoria;

4. **Baseada em Anel**: esses protocolos foram desenvolvidos para aplicações que necessitem ordenação total das mensagens.

[11] *Acknowledgement*

O RMTP [Lin e Paul 1996] é baseado em árvore, e usa controle de fluxo e congestionamento para evitar sobrecarregar receptores lentos e caminhos com pouca largura de banda. Ele entrega as mensagens em sequência de *um* emissor para um grupo de receptores. Os dados são divididos em pacotes numerados sequencialmente, com tamanho fixo, exceto o último. Os ACKs são periódicos, consistindo em um número L e um *bitmap* V. O *bitmap* especifica os pacotes perdidos com zeros, e a primeira posição corresponde a L, sendo que todos os pacotes anteriores já foram recebidos corretamente. A figura 3.3 ilustra como os Receptores Designados (líderes dos grupos locais) evitam que o emissor seja sobrecarregado com ACKs.

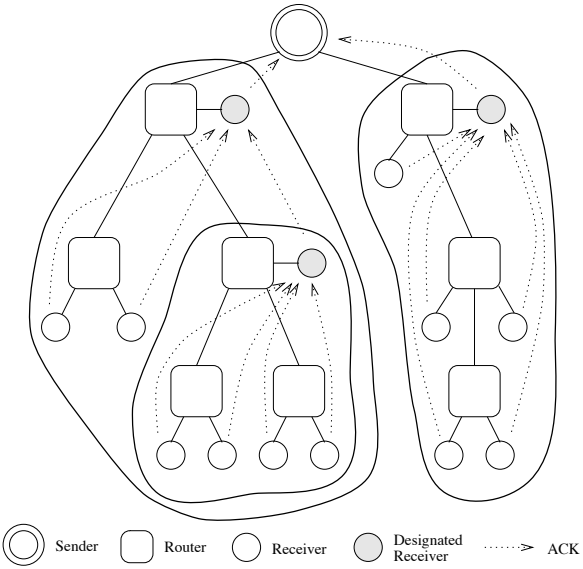


Figura 3.3: RMTP [Lin e Paul 1996].

Nem sempre o *multicast* confiável é necessário. Em aplicações multimídia, por exemplo, o atraso que ele causa pode ser pior que a perda de eventuais pacotes.

3.4.3 *Multicast* Semanticamente Confiável

Enquanto as garantias oferecidas pelos protocolos de *Multicast* Confiável são desejáveis para várias aplicações, os problemas de desempenho

não são. Visto que os nós devem manter as mensagens recebidas até que todos os nós tenham reconhecido seu recebimento, basta um nó lento para que os *buffers* encham e o emissor seja avisado para diminuir a vazão, prejudicando o grupo como um todo. A lentidão também pode acontecer devido à variação temporária de tráfego em um componente de rede entre qualquer par de membros do grupo. A solução é atenuar (*weaken*) a confiabilidade sem que as mensagens sejam perdidas livremente, escolhendo quais mensagens devem ser recuperadas e até que ponto. Isso funciona porque certas mensagens, ao serem enviadas, tornam mensagens anteriores obsoletas, podendo ser descartadas. Esses conceitos foram apresentados em [Pereira, Rodrigues e Oliveira 2003], junto de dois protocolos que suportam a obsolescência de mensagens, de forma que ao ser enviada pela aplicação, uma mensagem *multicast* informa (através de um *bitmap*) quais das mensagens anteriores do mesmo processo tornam-se obsoletas.

Esses protocolos, entretanto, não são voltados à transmissão de vídeo. Em vídeo, embora existam quadros de sincronização que tornam irrelevantes os quadros anteriores (codificação *intra*), não é apenas a informação mais recente que é mais importante, mas a continuidade entre os quadros que oferece sensação de movimento ao sistema visual humano. Outro problema é que os quadros de sincronização são em número reduzido para permitir maior compressão. Também, a obsolencência entre os outros tipos de quadros não é especificada, tornando o *multicast* equivalente ao confiável nestes casos, ou é especificada de forma muito relaxada, permitindo perdas consecutivas. Um protocolo mais apropriado é o apresentado em [Queiroz et al. 2009], que adapta a técnica de escalonamento $(m, k) - firm$ para avaliar as sequências de quadros de vídeo transmitidas. Ele é parametrizável quanto ao nível de confiabilidade desejado, de forma que m mensagens entre cada janela contendo k (originadas do mesmo emissor) podem ser perdidas no máximo, mas nunca duas consecutivas. Quando essa restrição é violada, a retransmissão deve ocorrer. Desta forma as perdas não se acumulam entre os quadros de sincronização, o que levaria a uma maior percepção por parte do usuário.

3.4.4 Protocolos Fofoqueiros

Frente à necessidade de oferecer comunicação de grupo para aplicações na Internet, os limites de escalabilidade dos mecanismos tradicionais (seção 3.4) foram atacados. Os protocolos fofoqueiros, como ferramentas para oferecer *multicast* a grupos massivos e geograficamente dispersos, baseiam-se na teoria matemática das epidemias para disseminar as mensagens probabilisticamente, usada pela primeira vez em [Demers et al. 1987], sendo que a ênfase em escalabilidade veio em [Golding e

Long 1992]. Neles, cada nó envia suas mensagens a um subconjunto aleatório de nós do grupo – não é necessário que cada nó conheça o grupo inteiro – de modo que a medida que o grupo cresce em tamanho, a carga sobre os nós cresce logaritmicamente. Cada mensagem recebida é também repassada a um subconjunto aleatório, mas apenas para a primeira cópia recebida. Protocolos fofoqueiros lidam bem com ambientes onde os nós eventualmente ficam indisponíveis, mas onde o grupo como um todo varia pouco. Para oferecer determinismo, sistemas de recuperação podem ser combinados para oferecer a confiabilidade que falta ao multicast probabilístico. Mas para manter alta a probabilidade de todos os nós receberem cada mensagem disseminada, é necessário que cada nó tenha uma visão parcial do grupo com um tamanho mínimo em função do tamanho do grupo. De outra forma, o multicast não funcionaria a contento, e nós deixariam de receber mensagens mais frequentemente que o desejável.

Além dessas características, o SCAMP[12] [Ganesh, Kermarrec e Massoulié 2003] foi desenvolvido para permitir aos nós criarem as suas visões parciais (locais) do grupo sem depender de conhecimento global sobre todos os nós do mesmo. Desta forma, cada visão parcial obedece à premissa de estar relacionada ao tamanho do grupo – para alcançar alta probabilidade de que todos os nós recebam as mensagens – mesmo que este seja desconhecido. Suas propriedades:

- Escalabilidade: crescimento lento da visão parcial.
- Confiabilidade: alta probabilidade que todos nós recebam mensagens.
- Descentralização: sem servidores nem conhecimento do tamanho do grupo.
- Recuperação de Isolamento: é necessário porque cada nó mantém sua própria visão parcial.

Protocolo básico

1. **Inscrição**. Um nó entra no grupo mandando uma requisição a um contato (por exemplo, de uma lista pública). Este encaminha (*forward*) a mensagem a todos os membros de sua visão parcial. Envia também c cópias a membros escolhidos aleatoriamente, de modo que as cópias ofereçam a tolerância a faltas desejada.

[12] *Scalable Membership Protocol*

Ao receber uma inscrição encaminhada, um nó pode acrescentar à sua visão parcial com uma probabilidade que depende do tamanho de sua visão parcial, ou encaminhá-la mais uma vez, o que acontece até que algum nó finalmente aceite a inscrição. Mas enquanto o grupo é pequeno é possível que a inscrição seja encaminhada infinitamente, por isso quando um nó recebe pela décima-primeira vez a mesma requisição, ela é simplesmente descartada.

A visão parcial (*PartialView*) contém os membros para os quais o nó deve encaminhar as mensagens do grupo, mas também é necessária uma visão de entrada (*InView*), contendo os membros dos quais o nó recebe mensagens. Ao aceitar uma inscrição, o nó informa ao novo membro para que este o adicione em sua visão de entrada.

2. **Cancelamento** (**unsubscription**). Para deixar a rede, um nós informa a todos na sua visão de entrada para substituí-lo, onde cada um recebe um nó da visão parcial. Se a visão de entrada do nó for maior que a visão parcial, os nós que sobrarem são avisados para remover o nó que está deixando a rede.
3. **Detecção de Isolamento**. Com auxílio de marca-passo (*heartbeat*), o nó detecta que está isolado (quando para de receber mensagens *hearbeat*) e precisa se re-inscrever usando um dos nós da sua visão parcial. As mensagens do marca-passo são enviadas aos membros da visão parcial do nó.

Balanceamento do grafo. Visto que não existe um lista global com todos os nós do grupo, geralmente alguns nós podem ser tornados públicos para a inscrição. Isso nem sempre gera visões parciais do tamanho correto, prejudicando o funcionamento do grupo, visto que as novas inscrições não distribuem uniformemente a escolha do nó de contato. Por isso, além do protocolo básico existem mecanismos que permitem rebalancear o grafo.

1. **Indireção**. Muda o processo de encaminhamento de inscrições, de modo que elas sejam encaminhadas enquanto um contador é decrementado até que atinja zero, quando é aceita. A escolha do membro dentro da visão parcial para quem a inscrição é encaminhada é dada de acordo com probabilidades baseadas nos pesos das arestas do grafo (mais detalhes no trabaho [Ganesh, Kermarrec e Massoulié 2003]). Um nó que receba um pedido de inscrição a repassa para um dos nós na visão

parcial, a partir das probabilidades calculadas para cada nó. Esse processo é encerrado por um contador decrescente presente na mensagem.

2. **Arrendamento** (*lease*). As inscrições tem validade, por isso os nós devem se re-inscrever periodicamente com um nó aleatório de sua visão parcial. Diferente da inscrição, aqui a visão parcial não é modificada.

3.5 Comparativo

A tabela 3.1 avalia as técnicas apresentadas neste capítulo sob quatro aspectos. A importância de cada um deles é:

- Se os dados forem entregues em qualquer ordem será necessária a *buferização* da mídia inteira antes que exibição seja possível sem interrupção.
- Transmissão confiável pode gerar atrasos piores que perdas controladas.
- Se os nós não contribuírem a capacidade dos servidores limita o número de clientes.
- Se algum nó depende exclusivamente (direta ou indiretamente) de um nó fixo para receber o conteúdo ou parte pré-definida dele, a sua desconexão provoca a perda de parte do conteúdo até que uma nova associação seja estabelecida.

Entre as observações, destaca-se ver que o *(m,k)-Firm* não faz distinção entre quadros dos tipos I, P e B. Já os protocolos fofoqueiros usam mensagens repetidas, recebidas de diferentes fontes, para garantir a entrega confiável. Dessa forma, somente são apropriados para pequenos volumes de dados.

Técnica	Ordenação aproximada	Atraso retransm.	Nós contribuem	Falha em nó[a]	Observação
Cliente / Servidor	Sim	Depende	Não	Leve	
CDN	Sim	Depende	Não	Leve	
Redes P2P	Não	Sim	Sim	Leve	Apenas arquivos.
Multicast					
IP	Sim	Não	Não	Leve	Indisponível.
Confiável	Sim	Sim	Não[b]	Leve[c]	
(m,k)-Firm[d]	Sim	Não	Não	Leve	$I = P = B$
Protocolos Fofoqueiros	Sim	Não	Sim	Leve	Mensagens repetidas.
ALM[ef]	Sim	Não	Parcial[g]	Grave	

[a] A taxa de abandono (*churn rate*) é alta nesse tipo de aplicação.
[b] Parcial se baseado em árvore.
[c] Grave se baseado em árvore ou iniciado pelo servidor sem NACK.
[d] Semanticamente Confiável.
[e] *Application Layer Multicast*
[f] Veja o item 4.1.1.
[g] Apenas nós internos.

Tabela 3.1: Comparativo das Arquiteturas.

Capítulo 4

Trabalhos Relacionados

O modelo Cliente/Servidor (seção 3.1) simplesmente não é facilmente escalável ao nível necessário a serviços populares na Internet, o crescimento nos custos de rede é linear e gerência torna-se complexa. Já com a adoção de CDNs (seção 3.2) a curva do crescimento dos custos é amenizada (por eles serem compartilhados entre todos os clientes da CDN) e o crescimento pode ser melhor gerenciado (terceirizado), mas ainda assim o provedor do serviço acaba pagando por cada *byte* consumido por um cliente. Já as redes P2P transferem apenas arquivos, cujas partes são entregues fora de ordem, por isso não permitem transmissões ao vivo.

Dentre as alternativas de *multicast* mostradas no capítulo 3, o *IP Multicast* (seção 3.4.1) apresenta custos e riscos para toda a Internet, motivos pelos quais não foi habilitado nos roteadores da rede. No caso do MBone, a solução foi transferir o custo e o risco para as sub-redes que decidissem adotá-lo, mas sem tratar os problemas que os ISPs têm com o *IP Multicast*. Já os protocolos de *multicast* construídos sobre a camada de aplicação (*Application Layer Multicast*) apresentam outros inconvenientes. Visto que vídeo é uma mídia temporizada, protocolos confiáveis (seção 3.4.2) acabam transmitindo informações que já perderam sua necessidade por terem sido substituídos por imagens e áudio mais recentes. Os protocolos semi-confiáveis (seção 3.4.3) tratam este ponto, ainda assim eles estão mais preocupados com a corretude da comunicação, do ponto de vista de um grupo, do que com a capacidade (em termos de rede) de cada nó e da taxa de abandono[1], que refletem na qualidade da experiência do usuário.

[1] *Churn rate*. Quando há o abandono num grupo de *multicast*, após o tempo de detecção (que serve para diferenciar abandono de um problema temporário), exige-se um tempo para que o roteamento das mensagens nas áreas afetadas seja re-direcionado. Como o vídeo requer transmissão constante de dados, mesmo se a taxa for pequena isso prejudica significativamente muitos clientes.

4.1 Redes Sobrepostas

A idéia de redes sobrepostas é que sobre o protocolo IP seriam construídos outros protocolos (de *multicast* na camada de aplicação) para fornecer seu próprio roteamento de mensagens, de modo que os clientes sejam responsáveis por repassar os dados para outros clientes, compartilhando seus recursos de rede. Os primeiros trabalhos na literatura adotam a nomenclatura *Application Layer Multicast* (seção 4.1.1.1), o que já demonstra sua proximidade das técnicas de *multicast* não-confiável. Trabalhos posteriores passaram a adotar *Data-Driven Overlay Network* (seção 4.1.2.1), num momento em que as técnicas deixaram de ter todas as aplicações do *multicast* convencional.

4.1.1 Redes *PUSH*

As primeiras abordagens para *streaming* de vídeo distribuído – também chamadas de ALM – ainda eram muito próximas das técnicas de *multicast*, adaptadas a uma realidade um pouco diferente: apenas o servidor poderia ser origem de mensagens. À medida que essas técnicas eram testadas, os primeiros problemas foram apontados, de forma que novas técnicas foram apresentadas para oferecer maior robustez. Características comuns:

1. Uma árvore é criada contendo o servidor na raiz e todos os clientes nos nós internos e nas folhas.

2. O vídeo flui em mensagens periódicas da raiz até as folhas, de modo que os dados são "empurrados" (*PUSH*) por cada nó logo que são recebidos.

4.1.1.1 ZIGZAG

Visto que árvores de muita profundidade levariam a um atraso excessivo, o ZIGZAG [Tran, Hua e Do 2003] procura manter os clientes em grupos (*clusters*) e construir a árvore a partir deles, de forma que a profundidade final da árvore é limitada.

4.1.1.1.1 Organização Administrativa Numa aplicação onde os nós entram e saem do grupo a todo instante, muitas vezes sem uma aviso prévio, a árvore de multicast sofre modificações constantes. Para acelerar a sua adaptação e delimitar o fluxo de mensagens de controle existe a Organização Administrativa (figura 4.2), à qual todos os clientes devem pertencer. Com o auxílio da figura pode-se observar que:

1. Não se trata de uma árvore de nós da rede, mas de uma árvore de grupos (*clusters*).

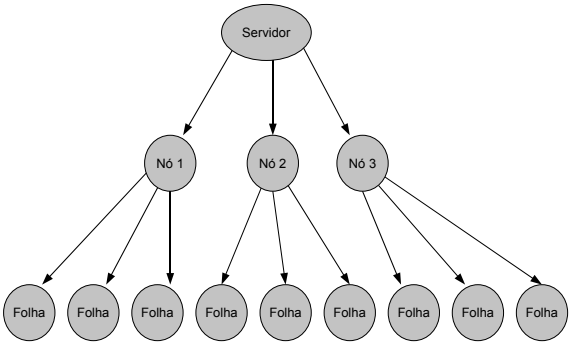


Figura 4.1: Distribuição por árvore.

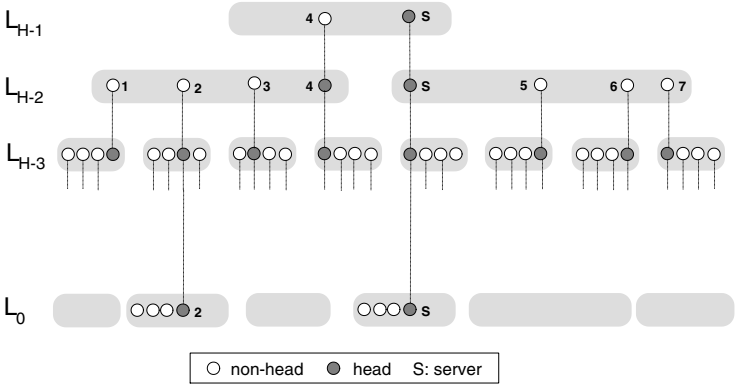


Figura 4.2: ZIGZAG – Exemplo de Organização Administrativa [Tran, Hua e Do 2003].

2. Cada grupo contém entre k e $3k$ nós, exceto na camada do topo $(H-1)$, cujo tamanho mínimo é 2.

3. A camada mais baixa (L_0), contém todos os nós em algum dos seus grupos.

4. Um nó de cada grupo é o chefe (*head*).

5. Cada nó da rede pertence a grupos em várias camadas em sequência. Desse modo, se ele pertence à camada L_x, ele também pertencerá a um grupo em cada uma das camadas $L_{x-1,x-2\ \ldots\ 0}$. A exceção são os nós que estão apenas na camada L_0.

6. Na camada mais alta em que está presente, um nó não é chefe, exceto para o servidor na camada do topo.

7. Nas camadas abaixo o nó é sempre o chefe de seu grupo.

Baseados nas camadas e grupos da organização administrativa e na figura dos chefes, alguns novos papéis são derivados:

1. **Subordinado** (*subordinate*): Os outros nós de um grupo são subordinados ao chefe.

2. **Chefe-Estrangeiro** (*foreign head*): Um nó não-chefe de um nível j assume esse papel com os subordinados de seu chefe no grupo de nível inferior.

3. **Subordinado-Estrangeiro** (*foreign subordinate*): associado ao papel anterior.

4. **Grupo-estrangeiro** (*foreign cluster*): é o grupo dos subordinados estrangeiros do chefe estrangeiro.

4.1.1.1.2 Árvore de Multicast Com a ajuda da árvore definida pela organização administrativa, mesmo não seguindo sua topologia, a árvore de *multicast* (figura 4.3) é construída de acordo com algumas regras:

1. Um nó não possui ligações, nem de entrada nem de saída, exceto na camada mais alta.

2. Suas ligações de saída só podem ser para subordinados-estrangeiros. As ligações entre o servidor e seus subordinados na camada do topo são a única exceção à regra.

3. A ligação de entrada de um nó vem de um de seus chefes-estrangeiros.

As nomenclaturas nós pai e filho permanecem válidas e são usadas a seguir se referindo à árvore de *multicast*, mas não à organização administrativa. Ao receber dados do vídeo de seu pai na árvore, o nó automaticamente encaminha-os aos seus filhos. Para uma exibição sem

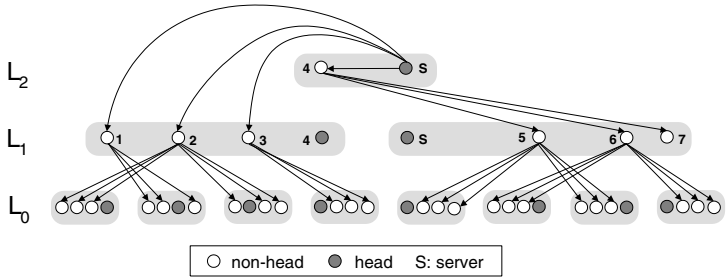


Figura 4.3: ZIGZAG – Exemplo de árvore de *multicast* ($H = 3$, $k = 4$) [Tran, Hua e Do 2003].

interrupções o servidor deve continuar gerando continuamente as novas partes do vídeo.

Visto que os grupos limitam a profundidade da árvore, o atraso e a variação (entre os diferentes nós) do atraso médio são reduzidos, considerando o número de clientes.

4.1.1.1.3 Protocolo de Controle As informações necessárias para construir e manter tanto a organização administrativa quanto a árvore de *multicast* são enviadas periodicamente pelos nós. Dependendo do tipo de informação há um destinatário diferente:

1. Aos vizinhos de grupo: o seu grau (número de arestas de nós filhos na árvore de multicast).

2. Chefe: número de nós atendidos em cada grupo estrangeiro (cada grupo é identificado pelo seu Chefe).

3. Nó-Pai: dois indicadores booleanos. Tomando essas informações de todos os nós filhos, um nó determina o valor a retornar ao seu pai através de uma operação booleana de agregação.

   (a) Alcançável (*reachable*): há um caminho na árvore de *multicast* até algum cliente na camada 0? Agregação: "E".

   (b) Adicionável (*addable*): existe algum caminho até um nó na camada 0 cujo grupo aceite a adição de um novo nó? Agregação: "OU".

4.1.1.1.4 Funcionamento As informações do protocolo de controle são usadas tanto na adição e remoção de clientes como na reconfiguração da organização e da árvore para alcançar uma operação mais otimizada.

Na adição de um nó o servidor é contatado primeiro, podendo aceitar e adicionar o nó ao grupo ou repassar o pedido ao Chefe de um dos subordinados no grupo da camada mais alta. Essa busca continua sendo redirecionada até que o nó seja aceito em um grupo da camada mais baixa, seguindo o algoritmo 4.1:

Algoritmo 4.1 Adição no ZIGZAG.

1: **when** $folha \implies$ acrescenta nó ao grupo, sendo que deve ter o mesmo pai.
2: **when** $adicionável \implies$ encaminha a um filho adicionável.
3: **when** $alcançável \implies$ encaminha a um filho alcançável.
4: *OBS: O redirecionamento seleciona o filho que obtém menor atraso fim-a-fim na propagação dos dados entre o servidor e o novo nó.*

Como esse algoritmo não trata o momento em que os grupos estouram o tamanho máximo estabelecido, um algoritmo de divisão (*split*) é executado (figura 4.4), para dividir o grupo em dois e eleger como Chefe do novo grupo o nó com menor grau (Y na figura). Como este nó precisa ser incluído na camada acima, seus antigos filhos na árvore são repassados ao nó de menor grau no mesmo grupo.

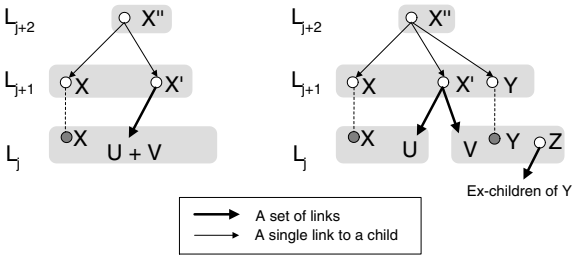


Figura 4.4: ZIGZAG – Antes e depois do *split* [Tran, Hua e Do 2003].

Na saída (ou falha), o Chefe do grupo da camada mais alta à qual o nó pertence seleciona o subordinado de menor grau como pai dos novos órfãos (figura 4.5). Para recompor a organização administrativa, um dos subordinados do nó que falhou na camada mais baixa é escolhido aleato-

riamente e passa a substituí-lo em todos os grupos, sendo Chefe em todos exceto no de camada mais alta. Com esse procedimento e com o de entrada os papéis da organização administrativa e as restrições da árvore de *multicast* ficam preservadas.

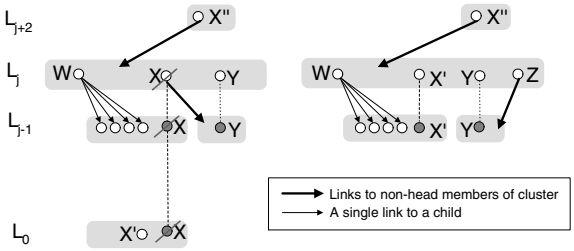


Figura 4.5: ZIGZAG – Saída ou falha [Tran, Hua e Do 2003].

No entando, muitas saídas podem fazer que um grupo acabe com tamanho menor que o especificado, de forma que periodicamente pode ocorrer uma junção (*merge*), semelhante no caso da divisão. Igualmente periódico é o processo de otimização da árvore de *multicast*, de modo que a otimização pode buscar aproximar o grau dos nós de um grupo ou deixar o grau desigual, mas de acordo com a capacidade (especificamente a largura de banda) de cada um.

4.1.1.2 Problemas das Redes *PUSH*

1. É sensível ao abandono ou falha nos nós. Mesmo com as melhorias propostas para amenizar esse problema ainda há um atraso entre uma falha e o funcionamento da árvore ser completamente corrigido.

2. Variações na carga de rede de um nó não passam despercebidas pelos nós descendentes na árvore.

3. As folhas não contribuem, aumentando a carga sobre os nós internos.

4.1.2 Redes *PULL*

Mais recentes que as redes *PUSH* originais, as redes *PULL* tem características diferentes, que visam eliminar os problemas das anteriores:

1. Não empregar árvore ou qualquer outro tipo de topologia para propagar as mensagens, deixando que os nós se associem e dados fluam entre os nós da forma que for mais conveniente. A estrutura resultante foi rotulada como malha (*mesh*) em um trabalho [Wang, Xiong e Liu 2007].

2. Como não é escalável que os nós conheçam todos uns aos outros, um protocolo adicional é usado para informar quais membros fazem parte da rede, de forma que os nós precisem conhecer apenas um subconjunto dos participantes (figura 4.6).

3. Os nós estabelecem parcerias entre si para a transmissão dos dados.

4. Para evitar repetições desnecessárias os dados não podem mais ser transmitidos automaticamente, devendo ser solicitados explicitamente (*PULLed*).

5. Os dados são divididos em partes de tamanho estabelecido nessa comunicação, sendo que cada parte pode ser solicitada de um parceiro diferente.

6. Para permitir as solicitações, os nós devem enviar relatórios periódicos das partes disponíveis.

7. Em relação aos problemas das redes *PUSH*: Como os dados são requisitados explicitamente, a tolerância a falhas é automática. E por não haverem folhas, a largura de banda dos nós é compartilhada com maior igualdade, ou de acordo com a capacidade de cada nó.

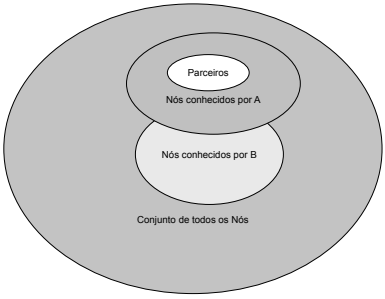


Figura 4.6: Nós conhecidos em redes em malha.

4.1.2.1 *CoolStreaming*/DONet

O DONet [Zhang et al. 2005] lançou vários mecanismos adotados posteriormente. Muito do seu modo de operação e filosofia foi mantido em trabalhos posteriores, enquanto os novos recursos eram adaptados. O diagrama da figura 4.7 ilustra os componentes do modelo e ajuda a entender seu funcionamento como detalhado a seguir.

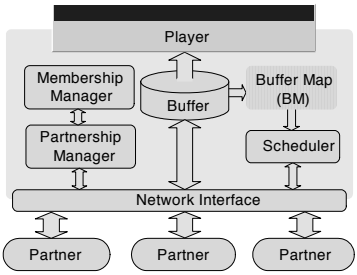


Figura 4.7: DONet – Componentes [Zhang et al. 2005].

4.1.2.1.1 Entrada e Saída Para entrar na rede sobreposta, um nó contata o servidor, que o redireciona a um representante (*deputy*). Este retorna uma lista de nós, que o novo nó insere em sua lista de conhecidos. Para ser mantido na rede o nó difunde periodicamente uma mensagem de membresia (*membership*) através do protocolo fofoqueiro (*gossip protocol*) SCAMP (seção 3.4.4). No recebimento, essas mensagens são usadas para atualizar a lista de nós conhecidos. Para sair o nó difunde uma mensagem por fofoca, e no caso de falha, a mensagem é enviada pelo parceiro que a detectou.

4.1.2.1.2 Parcerias Os nós procuram estabelecer um número mínimo delas. Para tanto, escolhem aleatoriamente alguns nós entre os que conhece e fazem o pedido, que é atendido se não houver esgotado o número máximo de parcerias.

4.1.2.1.3 *BufferMap* As partes nas quais o vídeo é dividido são chamadas segmentos, e devem ter tamanho fixo. É adotado um tamanho equivalente a um 1 segundo de vídeo, de acordo com a taxa de *bits* do mesmo. Mas nem todos os segmentos são importantes para os nós, apenas os que serão exibidos dentro em pouco, por isso o *BufferMap* detalha apenas os segmentos delimitados numa pequena faixa de segmentos ao redor do segmento em exibição no momento, metade antes e metade depois.

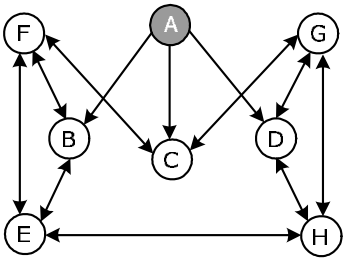


Figura 4.8: DONet – Parcerias [Zhang et al. 2005].

O tamanho sugerido para esta janela deslizante é 120 ou 60. O *BufferMap* é enviado por cada nó aos seus parceiros, sendo que um *bit* por segmento é o suficiente, visto que apenas sua disponibilidade ou não é informada.

4.1.2.1.4 Escalonamento De posse da lista de segmentos disponíveis em cada parceiro, o nó precisa buscar os segmentos que não possui no seu *BufferMap*. Para isso há um algoritmo de escalonamento (figura 4.9) que segue a heurística de selecionar primeiro os segmentos de menor disponibilidade, e frente a alternativas, privilegiar o parceiro com maior largura de banda e tempo disponível.

4.1.2.1.5 Transmissão A requisição usa o mesmo formato do relatório do *BufferMap*, usando um *bit* por segmento. Para o envio dos dados dos segmentos, que deve ser por ordem (de posição na mídia), um protocolo de tempo real deve ser usado, tendo sido adotado o TFRC (seção 2.2.2).

4.1.2.2 Problemas das Redes *PULL*

Comparando a forma de funcionamento das Redes *PUSH* e *PULL*, pode-se verificar o principal problema das últimas: maior atraso, que leva a maior necessidade de *bufferização*.

4.1.3 Redes Híbridas *PULL/PUSH*

Pensando em remediar os problemas das redes *PUSH* e das *PULL*, alguns trabalhos propuseram redes híbridas, que procurem operar por *PUSH* a maior parte do tempo, mas que usem o *PULL* quando necessário. Elas são constituídas de malhas para *PULL*, mas adotam cada uma delas algum mecanismo extra para permitir o *PUSH*.

Input:

$band(k)$:	bandwidth from partner k;
$bm[k]$:	buffer map of partner k;
$deadlin[i]$:	deadline of segment i;
seg_size :	segment size;
$num\_partners$:	number of partners of the node;
$set\_partners$:	set of partners of the node;
$expected\_set$:	set of segments to be fetched.

Scheduling:

for segment $i \in expected\_set$ **do**
  $n \leftarrow 0$
  for j **to** $num\_partners$ **do**
    $T[j,i] \leftarrow deadline[i] - current\_time$;
      //available time for transmitting segments till i;
    $n \leftarrow n + bm[j,i]$;
      //number of potential suppliers for segment i;
  end for j;
  if $n = 1$ **then** //segments with only one potential supplier;
    $k \leftarrow \arg_r\{bm[r,i] = 1\}$;
    $supplier[i] \leftarrow k$;
    for $j \in expect\_set, j > k$ **do**
      $t[k,j] \leftarrow t[k,j] - seg\_size/band[k]$;
    end for j;
  else
    $dup\_set[n] \leftarrow dup\_set[n] \cup \{i\}$;
    $supplier[n] \leftarrow null$;
  end if;
end for i;

for $n = 2$ **to** $num\_partners$ **do**
  for each $i \in dup\_set[n]$ **do**
      //segments with n potential suppliers;
    $k \leftarrow$
      $\arg_r \Big\{ band(r) > band(r') | t[r,i] > seg\_size/band[r],$
        $t[r',i] > seg\_size/band[r'], r, r' \in set\_partners \Big\}$;
    if $k \neq null$ **then**
      $supplier[i] \leftarrow k$;
      for $j \in expected\_set, j > k$ **do**
        $t[k,j] \leftarrow t[k,j] - seg\_size/band[k]$;
      end for j;
    end if;
  end for i;
end for n;

Output:

$supplier[i]$:supplier for unavailable segment $i \in expected\_set$.

Figura 4.9: DONet – Algoritmo [Zhang et al. 2005].

4.1.3.1 mTreebone

O mTreebone [Wang, Xiong e Liu 2007] foi motivado por uma análise dos autores que indicou que, mesmo numa malha, a maior parte dos dados flui por uma ou mais árvores implícitas. Portanto ela acrescenta um árvore para realizar o *push* dos dados. Assim, os nós considerados estáveis pertencem à arvore realizando *push*, enquanto os outros ficam na borda (*outskirts*) e só enviam dados em resposta a requisições *pull* (Figura 4.10(a)). A figura b ilustra a adaptação necessária quando nós da árvore saem da rede. Para manter a malha, o mTreebone adota mecanismo semelhante ao DONet (seção 4.1.2.1), inclusive na escolha do protocolo fofoqueiro, mas o *pull* não é realizaado a menos que seja necessário. O que era chamado parceiro pelo DONet, aqui tornou-se vizinho.

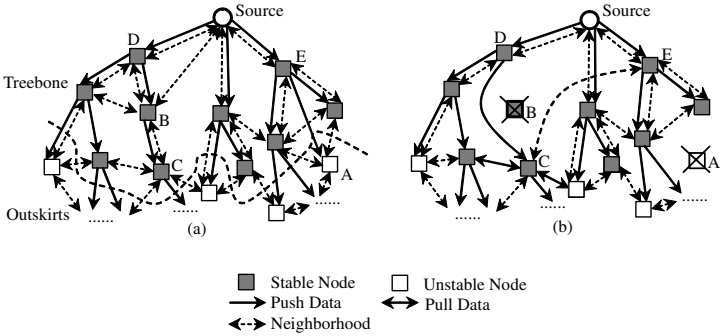


Figura 4.10: mTreebone – Rede Híbrida (a) e Dinâmica dos nós (b) [Wang, Xiong e Liu 2007].

4.1.3.1.1 Árvore *Treebone* Para a eficiência da árvore, a escolha dos nós estáveis deve ser cuidadosa, e o critério usado é escolher nós presentes na rede há mais tempo, o que é melhor do que escolher nós recém-chegados. Um limiar de escolha foi atribuído – idade igual ou superior a 30% do tempo restante da sessão – de forma que não restrinja o crescimento da árvore nem inclua muitos nós instáveis.

A árvore começa contendo apenas o servidor (origem do vídeo). Cada nó entra na rede contatando o servidor e obtendo: uma lista de nós, a duração e o tempo já decorrido da sessão. Dentre os nós estáveis na lista recebida (deve haver pelo menos um), o novo nó se conecta como filho de um deles. Assim o nó permanece como cliente da árvore, e periodicamente verifica se sua idade já permite se considerar estável, momento no

qual o nó se auto-promove (nós com tempo de chegada 10 na figura 4.11).

4.1.3.1.2 Promoção Antecipada Visto que antes de 30% do tempo de sessão nenhum nó poderia se considerar estável, mantendo apenas o servidor como fonte de *push* para todos os nós, há um mecanismo que permite aos nós se tornarem estáveis antes do seu limiar de idade. A cada unidade de tempo, o nó tem probabilidade de $1/(t_{limiar} - nr\_tentativa + 1)$ de promoção antecipada.

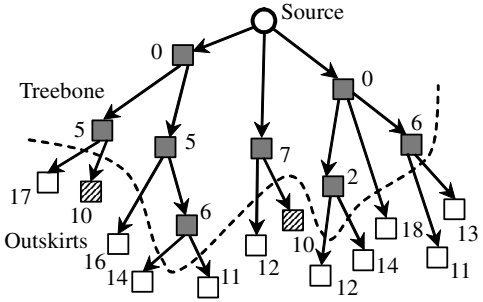


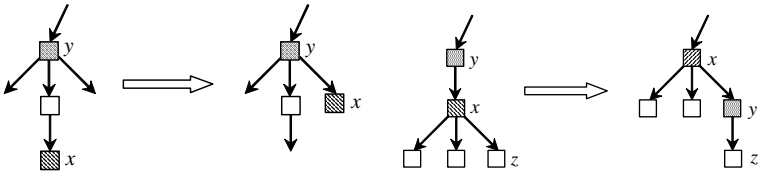
Figura 4.11: mTreebone – Evolução da árvore *Treebone* [Wang, Xiong e Liu 2007].

4.1.3.1.3 Otimizações A figura 4.12 ilustra duas árvores não-otimizadas criadas pelo procedimento descrito anteriormente. Para melhorar tais árvores, os nós checam periodicamente suas condições em comparação aos nós acima na árvore, começando com seu pai.

1. *High-Degree-Preemption* (figura 4.12(a)): se o nó tiver mais filhos que algum nó acima eles trocam de posição.

2. *Low-Jump-Delay* (figura 4.12(b)): se houver um nó acima de seu pai que tiver possibilidade de receber mais um filho, o nó assume esse lugar.

4.1.3.1.4 Detecção de falhas Os dados são esperados continuamente (*Tree-push Pointer* na figura 4.13), e quando não chegam, seja por perdas – temporárias ou pela saída de um nó da árvore – são detectadas pela janela *mesh-pull*, que faz os dados serem buscados pela malha. Os filhos que detectam a saída de seus pais na árvore também precisam encontrar novos pais.



(a) *Low Delay Jump* (b) *High Degree Preemption*

Figura 4.12: mTreebone – Otimizações [Wang, Xiong e Liu 2007].

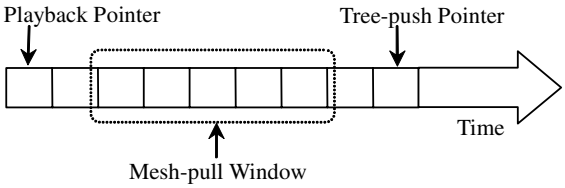


Figura 4.13: mTreebone – Chaveamento *PUSH/PULL* [Wang, Xiong e Liu 2007].

4.1.3.2 GridMedia

Ao contrário do mTreebone, o GridMedia [Zhang et al. 2005] oferece *push* sem o uso de uma árvore, mas apenas com a malha não-estruturada.

O processo de registro (*login*) acontece através de um ponto de encontro (*Rendezvous Point*) – usado apenas para ajudar na construção da rede – que retorna uma lista de nós. Nesse momento o relógio também é sincronizado.

Alguns dos nós obtidos no registro são escolhidos aleatoriamente como vizinhos. Os nós conhecidos e os vizinhos são mantidos numa tabela de membros que contém o tempo decorrido da última mensagem recebida do mesmo. Este valor tem um limite máximo que controla a exclusão do nó. Se um vizinho sair ou falhar, outro nó é escolhido como novo vizinho. Os vizinhos também trocam entre si periodicamente suas tabelas de membros, além de operar como os parceiros do DONet durante o *pull*. Isto é, eles enviam as informações do *BufferMap* e enviam os dados quando solicitados.

Logo que entre na rede, o nó só pode operar por *pull*, tendo que esperar o próximo intervalo – por isso o relógio dos nós foi sincronizado no início – para usar o *push*. A cada intervalo os nós devem se inscrever com os vizinhos para receber os pacotes por *push*. A escolha dos vizinhos é através de uma roleta, cuja probabilidade de escolher um nó é igual ao percentual de tráfego recebido do vizinho no último intervalo. Essa mesma roleta é usada no *pull*. Desta forma as conexões ruins são punidas, mas não completamente eliminadas.

Os pacotes correspondentes a um intervalo são divididos em P partes, de modo que os vizinhos distinguam quais enviar por *push*. A função de *hash* que mapeia pacotes em partes é simples: $mod\ P$, de forma que a cada P pacotes, 1 pertencerá a cada parte. Na inscrição de cada intervalo, os nós enviam aos vizinhos o PPMAP[2], similar ao *BufferMap* por empregar um *bit* para cada parte. Deste modo um nó tem dois PPMAPs para cada vizinho, um de recepção e outro de envio.

4.2 Retransmissão Seletiva

Existem técnicas que ponderam a retransmissão em função do tipo de dados perdidos. Elas se baseiam no fato que os padrões atuais de vídeo digital (seção 2) aplicam técnicas de codificação temporal. Ao mesmo tempo que aumentam a compressão, criam uma hierarquia entre os quadros, isto é, pode-se considerar mais importantes os quadros que mais forem necessários para a decodificação de quadros relacionados. Essas técnicas, entretanto, não foram aplicadas a redes sobrepostas, mas um multicast mais convencional, onde o servidor é o principal responsável por entregar o vídeo a cada cliente.

4.2.1 Técnica Baseada no Tipo do Quadro

O trabalho [Bortoleto 2005] desenvolve um protocolo *multicast* semi-confiável baseado na hierarquia de quadros do MPEG-4 [Ebrahimi e Horne 2000]. Ele também utiliza os receptores vizinhos no grupo para realizar a retransmissão, aliviando a carga do emissor.

Ele assume um *multicast* não-confiável e que os membros do grupo se conheçam. Cada mensagem contém um quadro (I, P ou B) apenas, seu número de sequência e tipo. Contém também os tipos dos oito quadros anteriores, informação que é usada para descobrir o tipo dos quadros perdidos.

Tanto o envio inicial dos dados, como as mensagens de **NACK** e as retransmissões são difundidas no grupo por *multicast*. Dessa forma, os nós que não perderam certo quadro, recebem sua retransmissão mesmo

[2] *Pushing Packets Map*

sem solicitar, o que significa mensagens duplicadas. Também antes de cada mensagem de **NACK** ou retransmissão o nó (seja emissor ou receptor) espera um tempo aleatório. Do contrário haveria uma explosão de mensagens repetidas.

Na chegada de um pacote de dados, o receptor verifica se existem lacunas na sequência de quadros recebidos. Detectada a perda, a retransmissão ocorre de acordo com o tipo do quadro e a taxa de perda medida nos últimos 50 pacotes:

- I (independente da taxa de perda);
- $P \wedge taxa\_de\_perda < 65\%$;
- $B \wedge taxa\_de\_perda < 35\%$.

4.2.2 Técnica para Redes Sem-Fio Sobrecarregadas

Essa proposta [Huszák e Imre 2008] trata do controle da retransmissão apenas. A transmissão é realizada pelo protocolo DCCP[3] [Kohler et al. 2006], que oferece as informações que a técnica necessita: detecção dos pacotes perdidos e a capacidade atual do canal. O DCCP pode utilizar vários protocolos para o transporte, mas nem todos são interessantes para rede sem fio. O TFRC, por exemplo, não distingue a perda da rede, que é naturalmente alta em redes sem fio, da perda por congestionamento. Isso o torna indesejável, por subestimar a capacidade de redes sem fio. Já o ARC[4] [Akan e Akyildiz 2004] e o WLED[5]-ARC [Singh K.D. 2006] não sofrem desse mal. Como a relação dos pacotes perdidos e a capacidade do canal são conhecidos pelo emissor no DCCP, a retransmissão é controlada inteiramente nele e de acordo com sua capacidade. Isto é, o cliente não precisa enviar NACKs, apenas implementar o protocolo DCCP corretamente.

A partir da detecção de perdas, é verificado se a vazão (o *throughput* de um protocolo do transporte com controle de congestionamento indica a capacidade real do canal) é maior que a taxa de *bits* do vídeo somada às retransmissões. Se a retransmissão iria inevitavelmente levar a novas perdas por congestionamento pela exaustão dos recursos do canal, parte das retransmissões não é realizada. Sendo p a taxa de perdas e μ a

[3] *Datagram Congestion Control Protocol*
[4] *Analytical Rate Control*
[5] *Wireless Loss Estimation using DiffServ*

taxa de *bits* do vídeo, as fórmulas:

$$\mu'_I = \rho_I \times p \times \mu$$
$$\rho_I = \frac{\text{tamanho dos quadros I no GOP}}{\text{tamanho do GOP}}$$

estimam a sobrecarga devida para retransmitir os quadros do tipo I, sendo as formulas para quadros P análogas. Se não for possível determinar a frequência dos tipos de quadros pode-se adotar 0.33, por exemplo. Sendo X_{cc} a vazão (*throughput*) do canal de comunicação, seguem as fórmulas para determinar se ocorre a retransmissão:

$$\begin{aligned} \mu < X_{cc} < \mu + \mu'_I &\implies \text{retransmitir I} \\ \mu + \mu'_I < X_{cc} < \mu + \mu'_{I+P} &\implies \text{retransmitir I+P} \\ X_{cc} > \mu + \mu'_{all} &\implies \text{retransmitir todos} \end{aligned}$$

Capítulo 5

Rede Sobreposta *SeRViSO*

Alguns serviços de vídeo sob demanda pela Internet[1] têm restrições folgadas de tempo, visto que o vídeo pode ser obtido inteiramente antes da exibição. Já no caso de transmissões ao vivo, onde uma *bufferização* muito alta é indesejável por atrasar a exibição, a recepção a tempo torna-se mais importante que a completa confiabilidade. Isso acontece porque o olho humano não percebe pequenas falhas num contexto de 24 ou 30 quadros por segundo. Assim a confiabilidade – expressa no procotolo TCP, por exemplo – implica retransmissão e portanto, atraso. Isso significa que dados perdidos ou quase, acabam atrasando pacotes posteriores, possivelmente provocando a perda daqueles que teriam melhores chances de chegar a tempo.

As redes sobrepostas anteriores (seção 4.1), reconhecendo isso, utilizaram o UDP com sua metodologia de não retransmitir pacotes. Por isso não há nenhuma garantia de entrega nesses trabalhos. O assunto retransmissão foi, entretanto, ignorado. Certos trabalhos (seção 4.2) propuseram mecanismos de recuperação seletiva, mas nenhum propôs um sistema visando escalabilidade ao nível da Internet. Desta forma, pode-se considerar as seguintes opções naturais para agregar tratamento de perdas às redes sobrepostas:

1. **Ignorar as perdas.** Isso equivale a negar a existência ou importância das perdas. Na prática, entretanto, a confiabilidade da Internet não pode ser garantida.

2. **Retransmitir cada perda.** Esta opção contraria o princípio exposto acima, que a recepção a tempo é mais importante que a confiabilidade completa.

3. **Escolher partes para a recuperação.** É a linha proposta neste trabalho. Visto que podem existir inúmeras formas de priorizar re-

[1] Por exemplo: http://www.youtube.com

transmissões, este trabalho propõe uma alternativa baseada em características do H.264 (seção 2.1).

À técnica proposta foi dado o nome de *SeRViSO*[2]. Os nós que desejam receber o vídeo se conectam a uma rede sobreposta em malha e começam a trocar mensagens informando segmentos disponíveis e requisitando-os uns dos outros. Quando pedaços forem perdidos, o algoritmo de seleção controla para quais deles a retransmissão é solicitada.

Este capítulo explica o funcionamento do *SeRViSO*. A seção seguinte (5.1) mostra os principais componentes. A forma como a mídia é dividida é assunto da seção 5.2. Os diferentes tipos de nós presentes na rede são categorizados na seção 5.3, e a construção da rede é explicada depois (5.4), seguida do processo de transmissão de dados (5.5). Os diferentes algoritmos usados para a seleção de dados para retransmissão são apresentados na seção 5.6 e o modo "desespero" na 5.7. Finalmente, a seção 5.8 encerra detalhando as informação contidas em cada tipo de mensagem empregada no *SeRViSO*.

5.1 Arquitetura

Os problemas dos modelos em árvore (seção 4.1.1) não combinam com a expectativa de perdas que está no escopo deste trabalho. Por isso, a rede proposta adota um modelo em malha semelhante ao já detalhado (seção 4.1.2) – devido à sua flexibilidade no trato dos nós, que oferece resistência a falhas. Cada nó conhece um subconjunto dos nós presentes em toda a rede sobreposta, sendo que com partes deles são estabelecidas parcerias visando a obtenção do vídeo.

Uma visão geral com os principais componentes que compõem a arquitetura pode ver vista na figura 5.1. A função de cada um deles é:

- **Aplicação**: componente de integração com um tocador de mídia externo. Um novo segmento deve ser enviado para exibição a cada segundo. Após o primeiro segmento ser recebido, o componente espera alguns segundos (quantidade fixa, definida na seção 6.1) para permitir que um *buffer* seja formado, e então inicia a exibição.

- ***Buffer* de Segmentos**: este componente armazena os segmentos que já foram recebidos, total ou parcialmente. É indexado pelo número do segmento e, dentro deste, pelo endereço do elemento.

- **Controle de NACK**: componente que detecta as perdas, avalia a necessidade de retransmissão de acordo com os algoritmos propostos (seção 5.6) e envia mensagens ***NACK*** (seção 5.5.8).

[2] *Selective Retransmission Video Streaming Overlay*

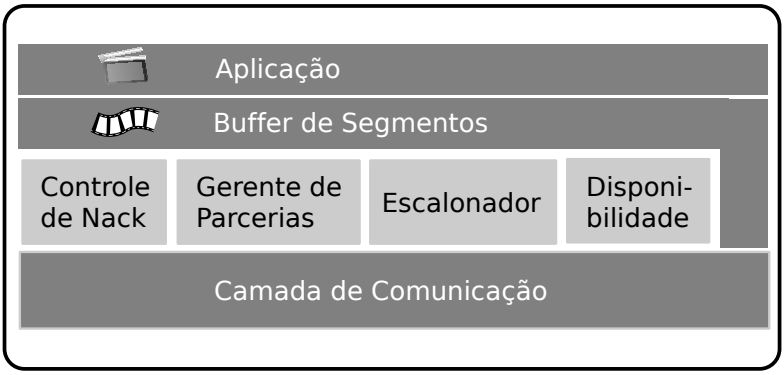


Figura 5.1: Componentes.

- **Disponibilidade**: há uma troca constante de informações sobre a disponibilidade de segmentos entre os parceiros. Este componente deve enviar essas informações frequentemente (seção 5.5.1).
- **Escalonador**: para receber os dados, os parceiros devem ser acionados; por isso, esse componente escolhe entre os que dispõem de cada segmento necessário (seção 5.5.3).
- **Gerente de Parcerias**: controla para que o número de parcerias (e de nós conhecidos) esteja dentro dos limites desejados, solicitando e respondendo solicitações quando necessário (seções 5.4.2 e 5.4.3). Um exemplo de como esse componente enxerga esses conjuntos de nós pode ser visto na figura 5.2.
  - **Nós Conhecidos**: o subconjunto dos nós presentes na rede sobreposta que são conhecidos (seção 5.4.2).
  - **Parceiros**: o subconjunto dos nó conhecidos com os quais foram estabelecidas parcerias (seção 5.4.3).
- **Camada de Comunicação**: cuida do envio e recebimento de mensagens e dados. São suas responsabilidades:
  1. Armazenar as informações de disponibilidade recebidas dos parceiros para uso pelo escalonador.

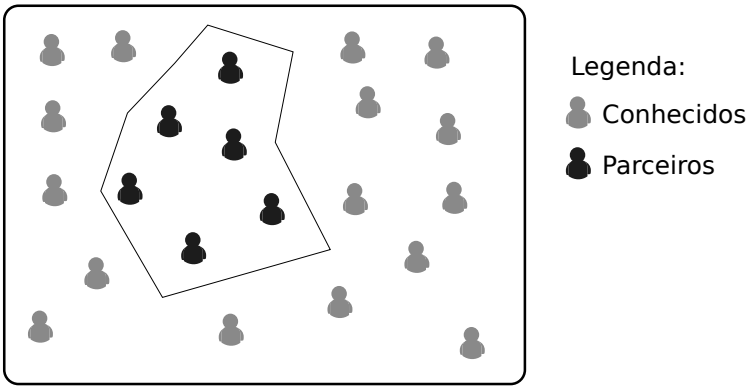


Figura 5.2: Visão do Gerente de Parcerias.

2. A partir das requisições recebidas e mensagens ***NACK***, preparar as mensagens contendo os segmentos solicitados (seção 5.5.4), colocando-os na fila para entrega (seção 5.5.5). Os metadados de cada segmento são enviados em paralelo.
3. Na recepção de dados e metadados, eles são armazenados no *Buffer* (seção 5.5.7).
4. As mensagens de dados precisam ser encapsuladas e controladas pelo controle de congestionamento do protocolo TFRC (seções 5.5.5 e 5.5.10). Já as mensagens de controle (requisições, ***NACK***, etc) são enviadas diretamente.

Existem componentes restritos aos nós especiais (*Rendezvous* e servidores), omitidos na figura 5.1 mas descritos abaixo:

- ***Rendezvous***: nos nós *Rendezvous* há apenas um componente, que responde às mensagens de entrada e saída da rede (seções 5.3.1, 5.4.1 e 5.4.4).
- **Carregador de Mídia**: o nó servidor não necessita de um Escalonador, em vez disso ele precisa de um componente que carregue

a cada segundo um segmento de vídeo. Este segmento é armazenado no *Buffer* e a partir daí seus parceiros são avisados através das mensagens de **Disponibilidade**.

5.2 Segmentação da Mídia

Para transferir o vídeo, ele precisa ser quebrado em partes menores (figura 5.3), da mesma forma que em trabalhos relacionados (seção 4.1.2.1). Esse processo se chama segmentação, que gera segmentos de tamanho equivalente a 1 segundo de vídeo, dada a taxa de bits do mesmo.

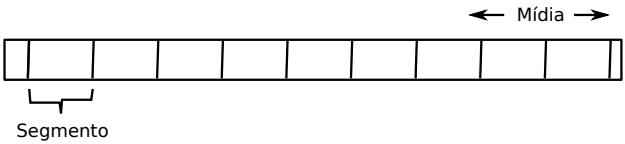


Figura 5.3: Segmentação.

- **Elemento**: como este trabalho se refere a pedaço da mídia que pode ser um H.264 NALU (seção 2.1), um pacote de áudio ou outro desconhecido.

Mas se a segmentação ocorressem igual às técnicas anteriores, fatalmente os elementos seriam quebradas ao meio, o que aumenta a perda no caso de um segmento perdido. Isto porque no começo e no fim do segmento haveriam elementos que começam/terminam nos segmentos anterior e posterior, e seriam imediatamente descartados pelo tocador por estarem incompletos. Por isso, se um segmento terminar no meio de uma unidade, ele é extendido até o fim da mesma. A figura 5.4 ilustra como o tamanho dos segmentos pode variar.

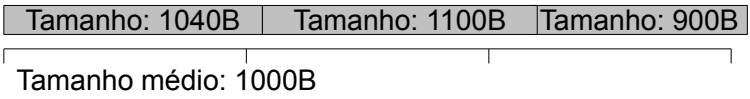


Figura 5.4: Segmentos de tamanho variável.

5.3 Nós

Caracterização dos nós que são necessários para o funcionamento do *SeRViSO*. A figura 5.5 ilustra como os nós descritos abaixo compõe a rede.

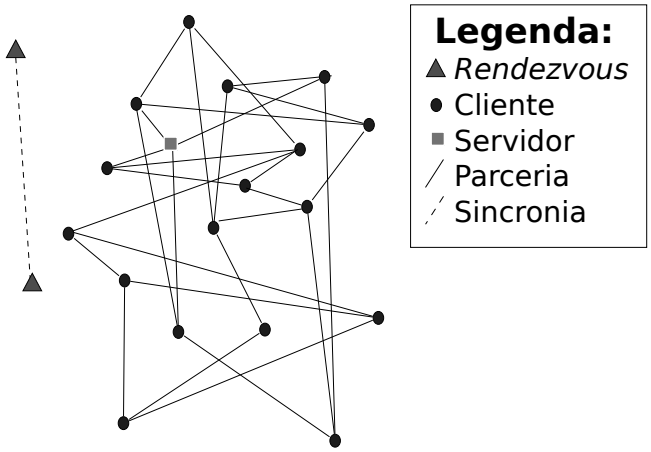


Figura 5.5: Diferentes tipos de nó da rede sobreposta.

5.3.1 *Rendezvous*

Este é um tipo especial de nó que ajuda na construção e manutenção da rede. Para isso, ele mantém uma lista de nós ativos. Ele não é um dos nós interessados no vídeo e, portanto, não participa da sua transmissão. É considerado estar "fora" da rede sobreposta, visto que não estabelece parcerias, etc.

Podem haver um ou vários nós cumprindo este papel, seja devido ao tamanho da rede, seja para oferecer tolerância a faltas. Caso sejam vários eles devem trabalhar de forma replicada, com sincronizações frequentes entre as listas de nós ativos. Pequenas discrepâncias (alguns nós a mais e a menos) eventuais (entre uma sincronização e outra) num conjunto grande de nós não afetam o funcionamento da rede sobreposta, visto serem estatisticamente irrelevantes e cada nó não depender de um (ou poucos) nós para sua conectividade à rede.

5.3.2 Clientes

É um nó que se conecta à rede sobreposta enquanto estiver interessado no recebimento do vídeo. Pode sair da rede a qualquer momento, seja de forma controlada avisando a partida, seja devido a uma falha, o que exige detecção por parte dos outros nós clientes e/ou *Rendezvous*.

5.3.3 Servidor

O nó servidor (que fornece o vídeo) se comporta na rede sobreposta quase da mesma forma que os clientes. Sua comunicação com o *Rendezvous* é igual, de modo que este não pode diferenciá-lo. Quanto à comunicação com os clientes, ele apenas "oferece" o vídeo, nunca "solicitando". Se for desejável "mascarar" o servidor, de modo que clientes "espertos" ou mal intensionados, não possam detectá-lo a fim de priorizar o seu uso, é possível fazê-lo solicitando segmentos eventualmente. Como também pode falhar, pode ser replicado como o *Rendezvous*, onde as cópias devem procurar sincronizar a oferta dos segmentos.

5.4 Construção da Rede

5.4.1 Entrada e Saída da Rede

Para se conectar à rede sobreposta, cada nó envia uma mensagem ***ENTER*** a um *Rendezvous*. Esta mensagem precisa ser repetida, como num marca-passo, para demonstrar que o nó continua ativo.

Numa saída controlada, o próprio nó envia uma mensagem ***LEAVE*** ao *Rendezvous*, e também a cada parceiro (seção 5.4.3). Entretanto, após um *timeout* $t_{partner}$ sem receber mensagens de um nó pode detectar a falha de um parceiro. Neste caso, ele envia uma mensagem ***LEAVE*** no nome do ex-parceiro ao *Rendezvous*. A figura 5.6 demonstra o processo de entrada e saída da rede.

5.4.2 Nós conhecidos

Cada nó precisa conhecer apenas um subconjunto entre todos os nós presentes na rede sobreposta. Para isso, ele deve manter uma lista com os nós por ele conhecidos. Esta lista é limitada em tamanho (seção 6.1), de forma que os nós mais antigos da lista podem ser removidos. Nós podem ser acrescentados nesta lista de diversas formas:

1. Uma mensagem de qualquer tipo é recebida do nó.
2. Através de uma lista de nós presente numa mensagem ***NODES*** recebida numa das seguintes condições:
   (a) Cada mensagem ***ENTER*** enviada por um nó ao *Rendezvous* é respondida com uma pequena parte dos nós ativos de sua lista;

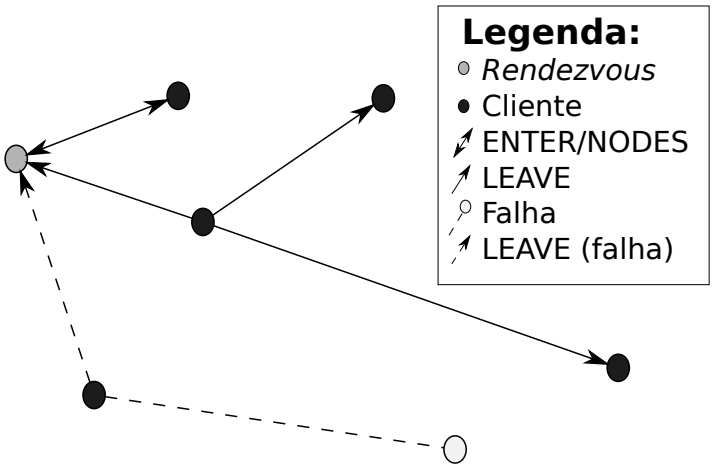


Figura 5.6: Entrada e saída da rede.

(b) Se a lista de nós conhecidos não possui o tamanho necessário (seção 6.1), o nó envia uma mensagem ***NODES*** vazia a algum nó de sua lista. Essa mensagem é interpretada como requisição pelo endereço de alguns nós. O nó que a recebe responde um conjunto de nós como se fosse o *Rendezvous*.

O *Rendezvous* nunca é incluído na lista de nós conhecidos, visto não participar da transferência dos dados. Para o caso de operarem na mesma máquina um processo *Rendezvous* e outro servidor ou cliente, os nós são armazenados como endereço de rede e porta de comunicação (UDP), de forma a diferenciar os diferentes processos.

5.4.3 Parcerias

Nem todos os nós conhecidos são usados para transferência de dados, apenas alguns com os quais são estabelecidas parcerias. Desta forma, cada nó busca estabelecer uma lista de nós parceiros, subconjunto da lista de nós conhecidos, dentro dos limites superior e inferior de tamanho (seção 6.1). Enquanto o limite inferior não for alcançado o nó busca entre os nós conhecidos os que aceitem formar parceria. Como a parceria leva a transferência de dados é apresentado na seção 5.5. Mensagens ***PARTNER***

são usadas para realizar a requisição e confirmação de parcerias, diferenças por um *bit* de marcação. Enquanto a lista de nós ativos de um nó não atingir seu limite superior, este vai confirmar todas as parcerias solicitadas. Quando um nó já tem sua lista cheia e não pode mais aceitar novas parcerias, ele para de responder às mensagens ***PARTNER***. Dessa forma, após o *timeout* da espera pela resposta o nó requisitante procura parceria com outro nó.

5.4.4 Controle da lista de nós ativos pelo *Rendezvous*

Toda vez que recebe uma mensagem ***ENTER*** o *Rendezvous* acrescenta o nó na lista de ativos, se ele ainda não estiver. Um nó sai da lista quando uma das seguintes alternativas acontecer:

1. Uma mensagem ***LEAVE*** for recebida, não importando se a mensagem foi enviada pelo próprio nó ou por um de seus ex-parceiros.

2. Depois que um ***timeout*** $t_{rendezvous}$ sem que o nó tenha enviado nenhuma mensagem ao *Rendezvous* para demonstrar que continua ativo.

5.5 Transmissão

A transmissão dos dados segue através de um diálogo entre parceiros que envolve as fases descritas nos itens desta seção. As etapas são apresentadas na ordem em que ocorrem, tendo em vista a transferência de um segmento. Como múltiplos segmentos são requisitados simultaneamente e o processo de transmissão de cada um deles é independente, as etapas podem ser executadas em paralelo.

5.5.1 Disponibilidade de Segmentos

A partir do estabelecimento das parcerias, os nós começam a trocar mensagens ***BMAP*** entre si. Seu propósito é informar os segmentos da mídia disponíveis, de forma que possam ser solicitados uns dos outros (figura 5.7).

Disponibilidade não significa que o segmento está completo, mas que o algoritmo de seleção para NACK (seção 5.6) parou de solicitar qualquer elemento, o que significa que está satisfeito com as partes do segmento que possui.

5.5.2 Janelas

Dentre todos os segmentos nos quais o vídeo foi dividido, é conveniente definir "janelas" para delimitar sequências de segmentos sobre as quais diferentes componentes do *SeRViSO* atuarão. Tais janelas podem ser

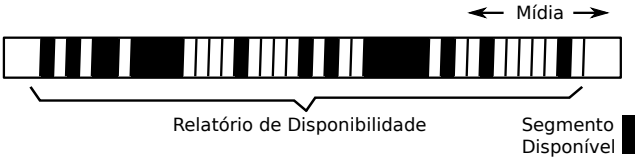


Figura 5.7: Mensagem ***BMAP***.

vistas na figura 5.8, sendo os detalhes de cada uma delas apresentados a seguir. Os tamanhos foram incluídos na seção 6.1.

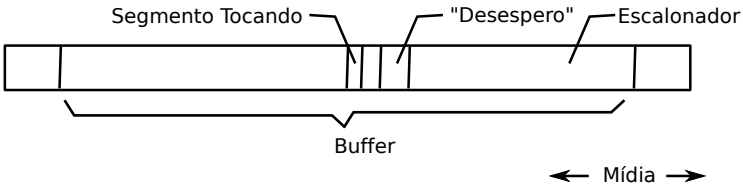


Figura 5.8: Janelas.

1. ***Buffer***: uma sequência ampla de segmentos, usada para delimitar as mensagens ***BMAP*** (seção 5.5.1). É definido em torno do ponteiro do tocador, sendo metade da janela antes e a outra metade depois dele. Normalmente são usados intervalos de um ou dois minutos, como em trabalhos anteriores (seção 4.1.2.1.3);

2. **Escalonador**: uma sequência menor que limita sobre quais segmentos o algoritmo escalonador (seção 5.5.3) vai atuar. Apenas segmentos à frente do tocador fazem parte desta janela;

3. **"Desespero"**: um sequência anterior à janela do escalonador, mas ainda assim alguns segundos à frente do tocador. Usada pelo modo homônimo descrito na seção 5.7.

5.5.3 Escalonamento

Antes de solicitar os segmentos necessários à exibição do vídeo, um nó primeiro deve fazer uma seleção que busque os de maior interesse

entre os que tiveram sua disponibilidade avisada pelos parceiros (seção 5.5.1). Algumas restrições podem ser estabelecidas, que são a razão do estabelecimento de uma janela mais restrita (seção 5.5.2).

1. Segmentos pelos quais o tocador já passou já foram **perdidos ou exibidos** total ou parcialmente, portanto não interessam.
2. Segmentos **muito à frente** do tocador podem esperar que os mais próximos, e portanto mais urgentes, sejam obtidos.
3. Segmentos **muito próximos do momento de exibição** correm grande risco de não ser obtidos a tempo, além de possivelmente atrasar os posteriores que teriam chance de chegar a tempo.

O algoritmo adotado pelo escalonador do *SeRViSO* é o mesmo descrito na seção 4.1.2.1.4, apenas com a imposição de uma janela mais restrita. A escolha do escalonador é ilustrada na figura 5.9. A partir das listas de segmentos que o algoritmo retorna, cada uma destinada a um parceiro, é enviada uma mensagem ***REQUEST*** a cada um deles.

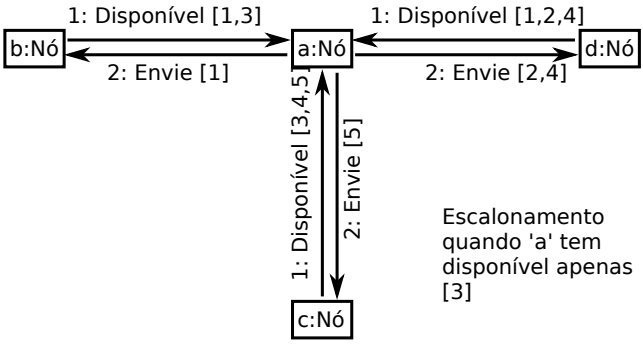


Figura 5.9: Exemplo de escalonamento.

5.5.4 Envio

Quando um nó recebe a requisição por um ou mais segmentos de um parceiro, ele se prepara para atendê-la se possuir os segmentos solicitados (o que acontece se está de acordo com a mensagem ***BMAP*** enviada). Os elementos do segmento são divididos em sequências, e cada uma delas é encapsulada em uma mensagem ***DATA***, que é inserida numa fila de mensagem para entrega (seção 5.5.5). As sequências seguem os limites dos elementos, como no exemplo da figura 5.10.

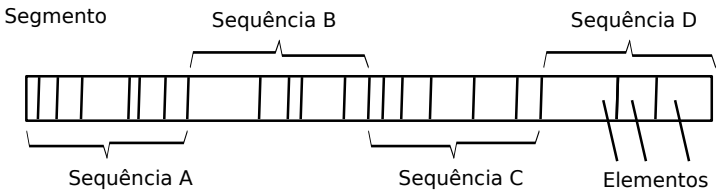


Figura 5.10: Os elementos são agrupados em sequências, cada uma encapsulada numa mensagem ***DATA***.

5.5.5 Filas de mensagens

Para cada parceiro, um nó deve manter um fila de mensagens separada. Essas filas podem ter duas categorias de mensagens:

1. **Controle**: são mensagens pequenas, enviadas diretamente na primeira oportunidade. Somadas, o uso da rede é pequeno, comparado ao próximo tipo (medições na seção 6.3.1.1).

2. **Dados**: são maiores e representam a maior parte dos dados transferidos. Carregam partes do vídeo. Pelo seu potencial para sobrecarregar a rede, são encapsuladas num protocolo de controle de congestionamento que controla o uso da rede (seção 5.5.10). Por isso, se o canal de comunicação entre os dois nós estiver congestionado, a mensagem pode demorar a ser enviada.

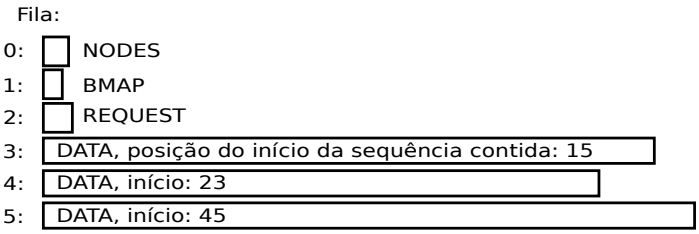


Figura 5.11: Fila contendo mensagens de vários tipos (tamanhos proporcionais às larguras das barras).

Para exemplificar uma fila com mensagens das duas categorias, veja a figura 5.11. As mensagens de dados são ordenadas pelo ende-

reço da sequência de elementos contida, de modo que sejam entregues primeiro as mensagens relativas aos dados mais urgentes, como no caso das retransmissões (seção 5.5.8). As mensagens de controle são colocadas no início da fila porque sempre são enviadas logo que lidas da fila, sem o controle de congestionamento.

Quando um nó envia uma nova mensagem de escalonamento, deixando de solicitar determinado segmento, todos as mensagens relativas a tal segmento são removidas da fila.

5.5.6 Metadados

Em cada mensagem ***DATA*** o nó receptor obtem uma sequência de segmentos, mas apenas os dados e seus limites (início e fim). Então não há como saber onde estão localizados os elementos ali contidos. Como essas informações serão necessárias nos algoritmos de seleção para retransmissão (seção 5.6), faz-se necessária uma mensagem adicional, denominada ***METADATA***.

Assim, toda vez que um segmento é requisitado, em meio às mensagens ***DATA*** é enviada uma mensagem ***METADATA***, contendo detalhes sobre todos os elementos no segmento (mais detalhes na seção 5.8). Essa mensagem também indica os elementos que o parceiro não possui. Tais informações são importantes para o **Controle de NACK**, visto que de outra forma não há como saber a importância dos elementos perdidos. Por isso, se a mensagem ***METADATA*** não for recebida (devido a uma perda, por exemplo), uma mensagem de sinalização é enviada para que ela seja retransmitida.

5.5.7 Recepção

Ao serem recebidos, os segmentos presentes numa mensagem ***DATA*** são armazenados no *Buffer*, de modo que permanecem lacunas no segmento enquanto faltarem mensagens relativas ao mesmo. As informações da mensagem ***METADATA*** também são salvas, de modo que o algoritmo de seleção (seção 5.6) possa avaliar os elementos que faltam (mensagens perdidas) em relação aos que já chegaram. Devem ser armazenados *timestamps* com o tempo de chegada da última mensagem ***DATA*** de cada segmento, a fim de controlar o disparo do processo de restransmissão.

5.5.8 Controle de NACK

De posse dos *timestamps* da recepção dos segmentos, o componente **Controle de NACK** determina para quais segmentos disparar o processo de NACK (algoritmo 5.1). Esse disparo ocorre quando o *timestamp* é muito antigo ou um segmento posterior recebido do mesmo parceiro tem *timestamp* mais recente (linha 2).

Algoritmo 5.1 Gatilho do Controle de NACK para o segmento i.

1: **when** $now - timestamp(i) > TIMEOUT$
2: **when** $\exists$ **segment** $j > i \mid timestamp(j) > timestamp(i)$
$\wedge\ provider\_node(j) = provider\_node(i)$

Quando disparado, o processo de NACK (algoritmo 5.2) chama o algoritmo de seleção (seção 5.6), verifica se o segmento já pode ser considerado disponível (linha 3), senão verifica os elementos que o fornecedor do segmento não possui (linha 5), que devem ser requisitados de outro parceiro, escolhido aleatoriamente (linha 6). Finalmente, a rotina para envio das mensagens é chamada (linhas 7 e 9).

Algoritmo 5.2 Processo de NACK.

1: $selected \leftarrow selectMissingElements(segment(i))$
2: **if** $selected = \emptyset$ **then**
3: $\quad avaliable \leftarrow available \cup \{i\}$
4: **else**
5: $\quad missing \leftarrow \{\forall\ element \in selected \mid marked\_missing(element)\}$
6: $\quad partner \leftarrow choose\_random\{\forall\ p \in partners \mid has\_segment(p, i)\}$
7: $\quad send\_NACK(missing, other\_partner, QNACK)$
8: $\quad selected \leftarrow \{\forall\ element \in selected \mid element \notin missing\}$
9: $\quad send\_NACK(selected, provider\_node(i), NACK)$
10: **end if**

Para mandar uma mensagem de ***NACK*** (algoritmo 5.3), primeiro os segmentos desejados devem ser separados em sequências contínuas, a fim de reduzir a quantidade de dados transferidos (linha 4). Mas é importante distinguir as mensagens dirigidas a outro parceiro que não o que originalmente enviou o resto do segmento. Neste caso, é usada um mensagem ***QNACK***, que pode ser ignorada total ou em parte caso o parceiro escolhido também não possua algum elemento. Quando isso acontece, posteriormente outro parceiro pode ser escolhido aleatoriamente. A única diferença entre mensagens ***NACK*** e ***QNACK*** é o código da mensagem, no seu cabeçalho (seção 5.8).

Algoritmo 5.3 Envio de NACK.

```
1: procedure send_NACK(selected, node, msg_code)
2: if selected ≠ ∅ then
3:     sort(selected)
4:     sequences ← find_contiguous_sequences(selected)
5:     send_to(NACK(msg_code, sequences), node)
6: end if
7: end procedure
```

5.5.9 Retransmissão

Ao receber uma mensagem ***NACK*** o nó prepara uma mensagem ***DATA*** para cada intervalo solicitado. Se for uma mensagem ***QNACK***, serão mensagens ***QDATA***, que tem apenas o código no cabeçalho para fazer diferenciação, como ocorre entre as mensagens de ***NACK***.

Existem casos em que o intervalo não corresponde à posição dos elementos. Nestes casos, o intervalo retornado é ajustado de forma a apenas envolver elementos completos. Como pode ser visto na figura 5.12, o início do intervalo é avançado para coincidir com o início de um elemento, e o fim abrange o último elemento por completo. Esse mecanismo serve de suporte ao modo "desespero" (seção 5.7).

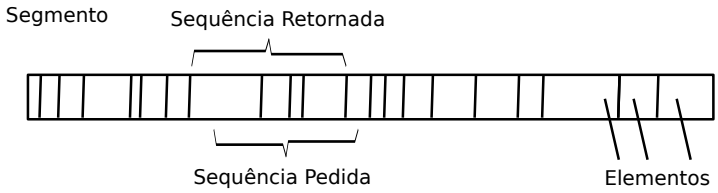


Figura 5.12: Ajuste do intervalo para retransmissão.

5.5.10 Uso do TFRC

As transferências de dados representam a quase totalidade dos dados transferidos, e têm potencial para ocupar totalmente a largura de banda da rede disponível. Desta forma, é importante adotar um mecanismo de controle de congestionamento, de modo que a rede seja usada de acordo com sua disponibilidade. Se o tráfego passa pela Internet isso se torna ainda mais imperativo. Assim, a medida que houver outros tipos

de tráfego competindo pelo uso da rede, as transferências terão sua taxa de transmissão reduzida ao serem detectados erros e atrasos. Na seção 2.2.2 foi descrito um modelo de protocolo para atender esses requisitos.

Este protocolo foi usado em trabalhos relacionados (seção 4.1.2.1) e foi adotado também pelo *SeRViSO*. Este protocolo usa dois tipos de mensagens:

- ***TFRC_SEND***. Encapsulam mensagens de dados.
- ***TFRC_FEEDBACK***. Enviadas periodicamente pelo lado receptor dos dados. Contém as informações que são necessárias para que o emissor ajuste sua taxa de transferência.

Assim, como cada nó (não apenas os parceiros) tem uma fila de mensagens para entrega (seção 5.5.5), cada fila é controlada pelo TFRC individualmente. Esse tratamento individual leva em conta as mensagens ***TFRC_FEEDBACK*** que aquele parceiro específico enviou para determinar a taxa de transferência para ele apenas, visto que a conexão dos outros parceiros tem suas próprias características de largura e utilização de banda, taxa de erro, etc.

Já as mensagens de controle são muito menores e em menor quantidade que as de dados e frequentemente são necessárias sem atraso (por exemplo: ***REQUEST***), motivos pelos quais elas estão dispensadas do encapsulamento pelo TFRC.

5.5.11 Mensagens *MULTI*

Pelo modo de funcionamento do TFRC e devido à equação que ele usa para determinar a taxa de transferência, mensagens pequenas reduzem a taxa de transmissão permitida, podendo impossibilitar a obtenção de vídeo em tempo real. As mensagens ***DATA***, fruto da divisão de um segmento, podem ser ajustadas a fim de atingir um tamanho razoável, tendo em vista essa característica do TFRC. Mas mensagens ***DATA*** geradas em resposta a ***NACK***s podem não ter essa opção, devido às sequências requisitadas serem muito pequenas. Neste caso, um grupo de mensagens ***DATA*** pode ser encapsulado em uma mensagem ***MULTI***, de forma que esta nova mensagem seja posta na fila (seção 5.5.5) e posteriormente enviada através do TFRC sem diminuir a taxa de transferência. Na recepção, as mensagens ***DATA*** são extraídas e tratadas como se tivessem sido recebidas individualmente, exceto pelo TFRC.

5.6 Algoritmos de Controle de Erro

O propósito de um algoritmo de seleção no *SeRViSO* é selecionar elementos dentre os faltantes de um segmento para solicitar a retransmis-

são através de mensagens de ***NACK***. Neste trabalho, são propostos três algoritmos, cada um seguindo determinada heurística. Um aspecto importante é que eles sejam adaptáveis ao momento da rede e aos elementos perdidos, visto que nem todos eles tem a mesma importância para a decodificação, segundo o H.264 (seção 2.1). Para verificar isso os algoritmos foram testados através dos experimentos apresentados no capítulo 6. Todos os algoritmos têm duas características em comum:

- Trabalham e avaliam **cada segmento isoladamente**;
- São precedidos por uma rotina que **varre os elementos** do segmento, **atribuindo um peso a cada** um de acordo com a função *elementWeight()* (algoritmo 5.4), que considera principalmente o tipo de quadro H.264 que o elemento contém.

Algoritmo 5.4 A função $elementWeight(element) : real$ determina o peso de um elemento.

1: **return** $MIN(sizeWeight + kindWeight, 3)$

Algoritmo 5.5 Função $sizeWeight(element) : real$

1: **return** $\frac{MAX(10-log10(size(element)),0)}{10}$

Algoritmo 5.6 Função $kindWeight(element) : real$

1: **function**
2: **when** *not NALU* $\implies 1.5$
3: **when** NON_IDR, IDR $\implies$ *check slice type (I, P, B ...)*
4: **when** I, PARTITION_A $\implies 3$
5: **when** P $\implies 2$
6: **when** B, PARTITION_B, PARTITION_C $\implies 1$
7: **when** PARAMETER_SET $\implies 3$
8: **when** *delimiter* $\implies 0$
9: **otherwise** $\implies 1.5$
10: **end function**

Como se pode ver entre as linhas 4 e 8 do algoritmo 5.6, os pesos classificam os elementos numa escala: dos mais importantes (fatias com

codificação I) aos menos importantes (delimitadores). Aos elementos de áudio e outros é atribuído um peso de valor intermediário (linha 9). Os tipos *PARTITION_x* correspondem às partições descritas na seção 2.1. Já a função *sizeWeight* serve para privilegiar os elementos de prioridade dois de menor tamanho (linha 1). Primeiro porque o tamanho de elementos de peso três (linha 4) não afeta o peso total (linha 1). Segundo porque elementos de baixo peso tem menor chances de serem retransmitidos.

5.6.1 *SeRViSO*

O algoritmo 5.7 parte do somatório dos pesos dos elementos presentes e do somatório dos pesos de todos os elementos (presentes e faltando) do segmento. De posse desses valores, o algoritmo aplica a heurística de buscar satisfazer três metas fixas:

1. Obter **todos** os quadros de peso três (linha 9), principalmente quadros I;

2. Obter **90%** dos elementos considerados **ponderadamente** (linha 1);

3. Garantir **70%** do tamanho total do segmento **em bytes**, por garantia (linha 2).

Enquanto as duas últimas metas não forem atingidas o algoritmo continua executando o *loop* (linha 14) que acrescenta o elemento perdido não-selecionado de maior peso (linha 15) à lista de elementos escolhidos para NACK.

5.6.2 *SeRViSO* Adaptativo

A versão adaptativa do *SeRViSO*(algoritmo 5.8) é uma variação do anterior que busca flexibilizar as metas fixadas de acordo com a situação da rede. Isto significa que ele pode buscar metas superiores às do algoritmo anterior ou tolerar perdas maiores, de acordo com o que a rede de permitir.

Entretanto, avaliar a rede é um assunto à parte. Informações determinadas pelo TFRC poderiam ser usadas, mas introduziriam um acoplamento indesejável, impedindo a substituição posterior desse protocolo. A solução encontrada foi baseada na contagem das vezes que o algoritmo de seleção é executado para cada segmento (linha 2). Desta forma, o número de mensagens ***NACK*** é usado para determinar as metas ponderadas (linha 3) e de tamanho (linha 4), permanecendo o resto do algoritmo igual ao anterior.

Algoritmo 5.7 *SeRViSO*.

1: $MIN\_RATE \leftarrow 90\%$
2: $MIN\_BYTES \leftarrow 0.7 \times SEGMENT\_SIZE$

3: **function** selectMissingElements(segment) : list
4: $sumWeights \leftarrow sum\{\forall\ element \in segment \rightarrow elementWeight(element)\}$
5: $sumBytes \leftarrow sum\{\forall\ element \in segment \mid \neg\ missing(element) \rightarrow size(element)\}$
6: $missing \leftarrow \{\forall\ element \in segment \mid missing(element) \rightarrow element)\}$
7: $sumMiss \leftarrow sum\{\forall\ element \in missing \rightarrow elementWeight(element)\}$
8: $sumAvail \leftarrow sumAll - sumMiss$
9: $always \leftarrow \{\forall\ element \in missing \mid elementWeight(element) = 3 \rightarrow element\}$
10: $missing \leftarrow missing - always$
11: $sumAvail \leftarrow sumAvail\ + sum\{\forall\ element \in always \rightarrow elementWeight(element)\}$
12: $sumBytes \leftarrow sumBytes\ + sum\{\forall\ elementinalways \rightarrow size(element)\}$
13: $sort\_by(missing,\ elementWeight,\ 'descending')$
14: **while** $missing \neq \emptyset\ \wedge (sumAvail/sumWeights < MIN\_RATE \vee sumBytes < MIN\_BYTES)$ **do**
15: $element \leftarrow first(missing)$
16: $selected \leftarrow selected \cup \{element\}$
17: $missing \leftarrow missing - \{element\}$
18: $sumAvail \leftarrow sumAvail + elementWeight(element)$
19: $sumBytes \leftarrow sumBytes + size(element)$
20: **end while**
21: **return** $always + selected$
22: **end function**

Se a contagem de mensagens ***NACK*** fosse feita por parceiro em vez de por segmento, igualmente se poderia estimar a "falhabilidade" da comunicação com ele, e seria possível controlar as metas baseado-se nesse valor. O inconveniente seria que oscilações momentâneas na rede entre os parceiros afetariam as metas dos segmentos daquele nó por muito tempo

Algoritmo 5.8 *SeRViSO* Adaptativo.

1: **function** selectMissingElements(segment) : list
2: $nrNacks \leftarrow numberOfNacks(segment)$
3: $minRate \leftarrow acceptableRate(nrNacks)$
4: $minBytes \leftarrow minimumBytes(nrNacks)$
5: $sumWeights \leftarrow sum\{\forall\, element \in segment \rightarrow elementWeight(element)\}$
6: $sumBytes \leftarrow sum\{\forall\, element \in segment \mid \neg\, missing(element) \rightarrow size(element)\}$
7: $missing \leftarrow \{\forall\, element \in segment \mid missing(element) \rightarrow element)\}$
8: $sumMiss \leftarrow sum\{\forall\, element \in missing \rightarrow elementWeight(element)\}$
9: $sumAvail \leftarrow sumAll - sumMiss$
10: $always \leftarrow \{\forall\, element \in missing \mid elementWeight(element) = 3 \rightarrow element\}$
11: $missing \leftarrow missing - always$
12: $sumAvail \leftarrow sumAvail\ + sum\{\forall\, element \in always \rightarrow elementWeight(element)\}$
13: $sumBytes \leftarrow sumBytes\ + sum\{\forall\, elementinalways \rightarrow size(element)\}$
14: $sort\_by(missing, elementWeight, 'descending')$
15: **while** $missing \neq \emptyset\ \wedge (sumAvail/sumWeights < minRate\ \vee sumBytes < minBytes)$ **do**
16: $element \leftarrow first(missing)$
17: $selected \leftarrow selected \cup \{element\}$
18: $missing \leftarrow missing - \{element\}$
19: $sumAvail \leftarrow sumAvail + elementWeight(element)$
20: $sumBytes \leftarrow sumBytes + size(element)$
21: **end while**
22: **return** $always + selected$
23: **end function**

após a situação ter-se normalizado.

5.6.2.1 Esforço de retransmissão

Para ser adaptativo, o esforço de retransmissão expresso nas metas deve decrescer à medida que mais mensagens ***NACK*** são enviados para o mesmo segmento. Por isso, o algoritmo busca 100% do segmento na meta ponderada para a primeira mensagem ***NACK***, 95% para o seguinte,

depois 90% e assim por diante (algoritmo 5.9). Já na meta de tamanho, os índices são 100%, 90%, 80%, etc (algoritmo 5.10).

Algoritmo 5.9 Ajustando a meta ponderada ao número de mensagens ***NACK***.

1: **function** acceptableRate(numberOfNacks) : real
2: **return** $1 - 0.05 \times numberOfNacks$
3: **end function**

Algoritmo 5.10 Ajustando a meta de tamanho ao número de mensagens ***NACK***.

1: **function** mininumBytes(numberOfNacks) : real
2: **return** $(1 - 0.1 \times numberOfNacks) \times SEGMENT\_SIZE$
3: **end function**

5.6.3 *SeRViSO* $(m, k) - firm$

Os algoritmos anteriores tem o inconveniente de permitirem que elementos consecutivos sejam perdidos. Para evitar é isso, é proposto o algoritmo 5.11, que agrega a premissa $(m, k) - firm$ (seção 3.4.3) ao *SeRViSO*.

Algoritmo 5.11 *SeRViSO (m,k)-Firm.*

1: **function** selectMissingElements(segment) : list
2: $missing \leftarrow \{\forall\ element \in segment \mid missing(element) \rightarrow element)\}$
3: $always \leftarrow \{\forall\ element \in missing \mid elementWeight(element) = 3 \rightarrow element\}$
4: $mid \leftarrow mkfirm(segment, 3, 2, 12)$
5: $low \leftarrow mkfirm(segment, 2, 1, 6)$
6: **return** $always + mid + low$
7: **end function**

Como nos algoritmos anteriores, elementos com peso três são sempre selecionados. Depois os elementos com peso entre dois e três (linha 4) são analisados em grupo pela função $mkfirm()$, seguidos pelos de peso entre um e dois. O último parâmetro indica o nível de exigência,

sendo que para a faixa entre dois e três é tolerada a perda de um a cada doze elementos, e na faixa entre um e dois, um a cada seis. A função $mkfirm()$ (algoritmo 5.12), por sua vez, encontra os elementos da faixa desejada (linha 2) e os separa em grupos do tamanho desejado (linha 3). Se um grupo tiver mais de um elemento perdido (linha 6), os elementos são selecionados a fim de seguir a premissa do *(m,k)-Firm*. A figura 5.13 ilustra esse processo para o grupo de elementos com peso entre 1 e 2.

Algoritmo 5.12 Função *(m,k)-Firm*

```
1:  function mkfirm(segment, top, bottom, size) : list
2:  some ← {∀ element ∈ segment | bottom <=
    elementWeight(element) < top}
3:  groups ← split_into_groups_of(some, size)
4:  for group ∈ groups do
5:     missing ← {∀element ∈ group | missing(element)}
6:     if |missing| > 1 then
7:        remove_last(missing)
8:        selected ← selected + missing
9:     end if
10: end for
11: return selected
12: end function
```

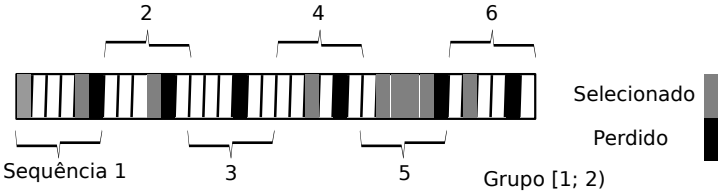


Figura 5.13: Aplicação do *(m,k)-Firm* ao grupo $[1; 2)$.

5.6.4 Modo "Tradicional"

Este não é um algoritmo proposto, mas um sob o qual o *SeRViSO* opera como uma rede sobreposta tradicional que implementasse a opção 2 apresentada no início deste capítulo. Foram contatados dois autores, mas não foi possível obter nenhuma redes sobrepostas anterior (seção

4.1.2). Estas seriam usadas para, uma vez adaptadas, fornecer uma base de comparação nos testes. Por isso o modo tradicional foi implementado no protótipo do *SeRViSO*. Mas por ter um funcionamento mais simples ele permite uma série de otimizações. Elas são necessárias para evitar carregá-lo com características específicas do *SeRViSO*, que tem custo mas seriam irrelevantes neste modo. Otimizações aplicadas:

1. A mídia não é processada à procura de NALUs, visto que esta informação não seria usada;
2. Cada segmento é simplesmente dividido em oito partes de tamanho fixo para transmissão, permitindo a simplificação de algumas mensagens (seção 5.8);
3. Dessa forma, não há necessidades de mensagens ***METADATA***, nem de ***MULTI***;
4. O algoritmo de NACK é mais simples: tenta recuperar todas as partes perdidas.

5.7 Modo "Desespero"

Nos intervalos das janelas do *Buffer* e do Escalonador (seção 5.5.2) pode-se observar que existe um momento a partir do qual um segmento não será mais escalonado, mesmo que ele esteja à frente do tocador. Este cenário, onde um segmento é "descoberto" (isto é, disponibilizado por um parceiro) muito próximo de seu momento de exibição (portanto fora da janela do Escalonador), pode acontecer algumas vezes numa rede sobreposta. Se ocorresse o escalonamento, o parceiro selecionado teria que mandar todos nós em sequência, onde possivelmente nem todos chegariam a tempo, com ou sem NACK. Entretanto, um tratamento diferenciado permite que o segmento seja obtido em paralelo, o que reduz o tempo necessário para tanto, possibilitando que todas as mensagens cheguem a tempo.

Esta técnica é proposta neste trabalho para lidar com esse caso marginal. A razão é que as perdas (objeto desse estudo) influenciam o surgimento destas situações, por dois motivos:

1. Perdas que atrasaram a recepção do segmento pelos parceiros;
2. Perdas de mensagens de disponibilidade (***BMAP***).

Quando um segmento i é detectado nesta situação é aplicado um algoritmo específico (5.13). Ele divide o segmento (linha 11) em intervalos de tamanho fixo, exceto possivelmente o último. Então cada parte

é solicitada de um parceiro diferente que possui o segmento (linha 2), usando uma mensagem ***QNACK*** (a linha 16 chama o algoritmo 5.3). Ele também solicita os metadados a um dos parceiros, de modo que possa usar o algoritmo de seleção (seção 5.6). A partir do momento que os metadados estiverem presentes, as próximas execuções do algoritmo passam a usar o algoritmo de seleção (linha 4) ao invés de dividir o segmento em intervalos de tamanho fixo.

Algoritmo 5.13 Modo "Desespero".

```
1:  INTERVAL_SIZE ← SEGMENT_SIZE / 8
2:  candidates ← {∀ p ∈ partners | has_segment(p, i)}
3:  if has_metadata(i) then
4:      selected ← selectMissingElements(segment(i))
5:      sort(selected)
6:      intervals ← find_contiguous_sequences(selected)
7:  else
8:      partner ← choose_random(candidates)
9:      ask_metadata(partner)
10:     range ← expected_segment_limits(i)
11:     intervals ← split_range(range, SEGMENT_SIZE)
12: end if
13: while intervals ≠ ∅ do
14:     range ← pop(intervals)
15:     partner ← choose_random(candidates)
16:     send_NACK({range}, partner, QNACK)
17: end while
```

5.8 Mensagens

Esta lista contém todos os tipos de mensagens empregadas pelo *SeRViSO* na comunicação entre os nós. Ela detalha entre quais tipos de nó as mensagens trafegam, as informações que cada uma carrega e sugestões de formas de representação eficientes.

1. ***ENTER*** (seção 5.4.1).
   Origem: nó cliente ou servidor.
   Destino: *Rendezvous*.
   Conteúdo: nenhum, já que o endereço do nó (IP e porta UDP) pode ser extraído do quadro UDP.

2. ***LEAVE*** (seção 5.4.1).
   Origem: nó cliente ou servidor.

Destino: *Rendezvous*.
Conteúdo: endereço do nó (se for enviado pelo próprio nó, será igual ao endereço no quadro UDP).

3. ***NODES*** (seção 5.4.2).
   Origem: nó cliente, servidor ou *Rendezvous*.
   Destino: nó cliente ou servidor.
   Conteúdo: lista de endereços de nós, possivelmente vazia.
   Representação: deve incluir IP e porta UDP.

4. ***PARTNER*** (seção 5.4.3).
   Origem e destino: nós cliente ou servidor.
   Conteúdo: indicação se é solicitação ou confirmação.
   Representação: apenas um *bit* é suficiente, sendo '1' o valor da confirmação.

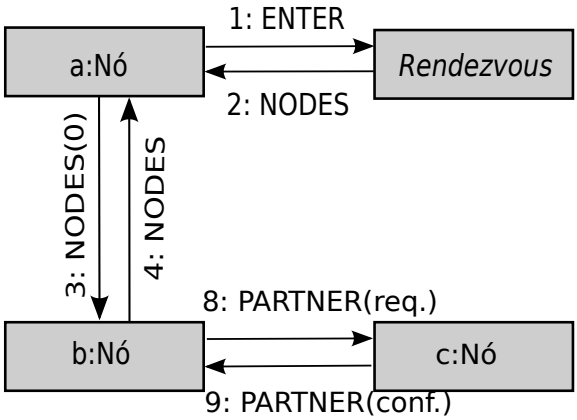


Figura 5.14: Troca de mensagens para conexão na rede sobrepostas, descobrerta de Nós, e estabelecimento de parcerias.

Na figura 5.14 esse primeiro grupo de mensagens é representada:

(a) A comunicação entre o nó 'a' e o *Rendezvous* para entrar na rede (***ENTER***) e o retorno de uma lista de nós (***NODES***);

(b) O pedido que o nó 'a' faz ao nó 'b' por alguns nós da sua lista (***NODES(0)***) e resposta (***NODES***);

(c) O estabelecimento de parcerias através das mensagens de solicitação (***PARTNER(req.)***) e confirmação (***PARTNER(conf.)***).

5. ***BMAP*** (seção 5.5.1).
   Origem e destino: parceiros.
   Conteúdo: indicação dos segmentos que estão disponíveis, limitados à janela atual do nó (seção 5.5.2), e endereço do primeiro segmento representado.
   Representação: um bit por segmento é suficiente, num formato já adotado por trabalhos anteriores (seção 4.1.2.1), onde '1' marca a disponibilidade.

6. ***REQUEST*** (seção 5.5.3).
   Semelhante à mensagem ***BMAP*** em quase todos os aspectos.
   Representação: '1' marca um segmento solicitado.

7. ***METADATA*** (seção 5.5.6).
   Origem e destino: parceiros.
   Conteúdo: posição (em bytes) de início do primeiro elemento contido no segmento e as seguinte informações sobre cada elemento:

   (a) Tamanho;
   (b) Tipo: o tipo do NALU (seção 2.1) ou 'OUTRO' (para áudio, por exemplo);
   (c) Indicação quando o parceiro não possui o elemento. A bit set to indicate it is not available to the supplier.

   Representação: posição, tamanhos e tipos são inteiros, o primeiro deve atender número na casa dos milhões, o segundo nas dezenas de milhares e o terceiro cabe em um byte, com espaço para um *bit* que represente (quando '1') a falta do elemento.

8. ***DATA*** (seção 5.5.4).
   Origem e destino: parceiros.
   Conteúdo: início do segmento dentro do vídeo (em bytes), início do intervalo transferido em relação ao segmento e os dados.
   Representação: o início do segmento deve ser um inteiro de tamanho considerável, já o inicio do interval, nem tanto.

9. ***MULTI*** (seção 5.5.11).
   Origem e destino: parceiros.
   Conteúdo: lista de mensagens ***DATA*** encapsuladas.
   Representação dos itens: as mensagens devem precedidos pelo seu tamanho em bytes (até alguns milhares).

10. ***NACK*** (seção 5.5.8).
    Origem e destino: parceiros.

Conteúdo: posição de início do segmento (em bytes) e lista de intervalos requeridos.
Representação dos intervalos: início em relação ao segmento e tamanhos (inteiros).

11. ***QDATA*** (seções 5.5.8 e 5.7).
    Igual a ***DATA***, exceto pelo código do tipo de mensagem.

12. ***QNACK*** (seções 5.5.8 e 5.7).
    Igual a ***NACK***, exceto pelo código do tipo de mensagem.

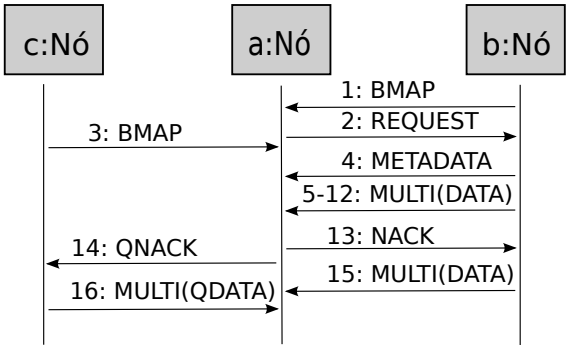


Figura 5.15: Troca de mensagens para transferência de dados.

A figura 5.15 demonstra o uso dessas mensagens para obter os segmentos e realizar a retransmissão dos elementos perdidos:

(a) Primeiro os nós avisam aos parceiros os segmentos que dispõem (***BMAP*** 1 e 3);

(b) A requisição é feita ao parceiro selecionado (***REQUEST***);

(c) A mensagem de metadados do segmento (***METADATA***) e as mensagens contendo os elementos são enviadas em sequência (algumas do tipo ***DATA*** e outras ***MULTI*** que encapsulam mensagens ***DATA*** menores);

(d) De acordo com os metadados, parte dos elementos selecionados para retransmissão é selecionado do parceiro que já havia transferido o segmento (***NACK***) e parte de outro parceiro que também possui o mesmo;

(e) As retransmissões chegam de ambos os parceiros (mensagens ***DATA*** e ***QDATA***, encapsuladas em mensagens ***MULTI*** ou não).

No modo tradicional (seção 5.6.4) algumas das mensagens são diferentes devido à operação simplificada deste:

1. ***METADATA***
   Não existe porque não é necessária.

2. ***DATA*** e ***QDATA***
   Conteúdo: índice (entre '0' e '7') dentro do segmento e os dados.
   Representação: um *byte* para o índice.

3. ***MULTI***
   Não existe porque não é necessária.

4. ***NACK*** e ***QNACK***
   Representação: cada segmento é representado por um *byte*, sendo que cada um dos oito *bits* representa uma das partes do segmento.

Estas são as mensagens emitidas pelo protocolo TFRC (seção 5.5.10) empregada pelo *SeRViSO* para troca de dados com controle de congestionamento:

1. ***TFRC_DATA***
   Origem e destino: parceiros.
   Conteúdo: número do pacote, *timestamp*, *round-trip-time* e dados.
   Representação: o número deve ser inteiro de tamanho suficiente para que não ocorra *overflow* em transmissões mais longas, já o *timestamp* e o *rtt* necessitam de precisão até os milisegundos.

2. ***TFRC_FEEDBACK***
   Origem e destino: parceiros que já iniciaram a transmissão de dados.
   Conteúdo: *timestamp* (do último pacote ***TFRC_DATA*** recebido), atraso interno (desde o recebimento da mensagem), taxa de recepção (medida para o último intervalo de um atraso fim-a-fim) e nível de erro medido.
   Representação: *timestamp* e atraso interno devem ter precisão até os milisegundos, a taxa de recepção necessita de um inteiro que represente quantos bytes por segundo, e para o nível de erro basta um inteiro entre '0' e '100' %.

Capítulo 6

Avaliação

6.1 Protótipo

Foi construído um protótipo na linguagem *Python*[1] que implementa o protocolo descrito, permitindo o teste da rede sobreposta *SeRViSO*. Ele pode operar sob qualquer um dos algoritmos de seleção propostos (seção 5.6). A seguir estão seus parâmetros de funcionamento e suas limitações, a partir da proposta que consta no capítulo 5. Quando for configurável, o valor indicado é o adotado nos testes.

1. Múltiplos **Rendezvous** (seção 5.3.1): o número de nós clientes não chegou a um nível que necessitasse o uso desse recurso, assim ele não foi implementado. Portanto, o protótipo suporta apenas um único **Rendezvous**;

2. **Disponibilidade** de segmentos pelo **Servidor** (seções 5.3.3 e 5.5.1): o servidor avisa a disponibilidade dos segmentos antes que qualquer outro nó o faça, sendo o primeiro alvo para requisições. Para evitar que ele transmita tudo a cada parceiro ele restringe a disponibilidade de cada segmento para apenas 2 dos seus parceiros.

3. **Nós conhecidos** (seção 5.4.2): o limite inferior é 30 e o superior é 60 (configurável).

4. **Parcerias** (seção 5.4.3): o limite inferior é 15 e o superior é 30 (configurável).

5. **Janelas** (seção 5.5.2): os intervalos têm tamanho configurável.

   (a) ***Buffer***: 2 minutos (sendo metade antes e metade depois do segmento tocando a cada instante).

   (b) **Escalonador**: começa no segmento 5 segundos à frente daquele que está tocando.

[1] `http://www.python.org`

(c) **"Desespero"**: começa no segmento 3 segundos à frente daquele que está tocando.

6. Tamanho das **sequências** enviadas (seção 5.5.4): um oitavo da taxa de *bits* do vídeo.

7. *Timeout* de Controle de NACK (algoritmo 5.1): 1800 milisegundos.

8. Tamanho das mensagens ***MULTI*** (seção 5.5.11): o mesmo das sequências (item 6 acima).

9. Cada nó inicia a exibição 10 segundos após a disponibilidade do primeiro segmento, a fim de obter um pequeno *buffer*.

10. A mídia transferida consiste em arquivos pré-gravados MPEG-4 com conteúdo H.264, de modo que o cabeçalho do arquivo precisa ser transferido confiavelmente (modo Tradicional) em vez de usar o algoritmo de seleção em teste;

11. O cabeçalho do arquivo é ignorado quando o teste se dá em máquinas sem interface, também para não influenciar nos resultados;

12. A exibição é feita pelo *ffplay*, utilitário incluído no pacote do conversor de audio e vídeo *ffmpeg*[2]. A integração ocorre através da entrada padrão do processo do tocador, que recebe os dados do vídeo vindos da rede sobreposta *SeRViSO*. Os dados perdidos são substituídos por *bytes* nulos.

6.2 Metodologia

Como ambiente de teste foi adotado o PlanetLab[3]. Vários nós foram adicionados a uma fatia obtida através da Rede Nacional de Ensino e Pesquisa[4]. Cada teste incluiu em média 141.94 nós (desvio padrão de 8.17).

A instalação de programas nos nós do PlanetLab não é trivial, visto tratar-se de um ambiente *linux* bastante limitado – apenas interface por modo caractere, por exemplo – necessitando que bibliotecas e linguagens de programação tenham de ser instaladas na própria fatia do PlanetLab, ao invés de estarem disponíveis automaticamente. Essas dificuldades guiaram a escolha do *Python* como linguagem de implementação do protótipo, visto que já está incluído no ambiente padrão, na versão 2.5.

[2] `http://ffmpeg.org/`
[3] `http://www.planet-lab.org/`
[4] `http://www.rnp.br/pd/planetlab/`

Frequentemente muitos dos nós do PlanetLab não encontravam-se em condições favoráveis, necessitando serem descartados dos testes mesmo pertencendo à fatia. Por exemplo:

1. Sistema de arquivos com permissão somente para leitura;
2. Relógio atrasado ou adiantado;
3. Sem acesso a serviço de DNS;
4. Muita carga na CPU devido a experimentos realizados por outros usuários (pode acontecer de o nó 'congelar' no meio do experimento);
5. Muita carga de rede no nó ou no acesso da instituição que o hospeda à Internet.
6. O sistema operacional dos nós pode ser re-instalado frequentemente, então os programas devem ser recarregados na fatia. Eventualmente a chave pública para acesso *ssh* não é mantida, precisando a antiga ser removida para que a integração seja re-estabelecida.

Dessa forma, os nós executam testes de conexão entre si para determinar quais têm condições de suportar a taxa necessária para a transmissão de vídeo em tempo real. Os nós que não apresentavam conexão de velocidade suficiente com um número significativo de outros nós foram eliminados. Tais restrições provocaram a eliminação de muitos nós, restando para os testes nós da América do Norte e Europa, que possuem acesso rápido entre si (figura 6.1). Mesmo depois do início de um teste as condições de cada nó podem mudar, por isso eles enviam mensagens ***LEAVE*** para os destinos muito lentos. Se isso ocorrer muitas vezes (mais de 45, configurável), o nó assume que o problema é consigo mesmo e não com os outros, abandonando o teste. Foi verificado que o ideal é que o atraso fim-a-fim seja inferior a 100 milisegundos, mas nos testes foi dada uma pequena tolerância de 60 millisegundos (configurável).

Para avaliar a retransmissão foram introduzidos erros aleatórios a uma taxa fixa, sem contar os erros que ocorreram naturalmente na rede. Os testes foram repetidos quatro vezes para cada um dos cenários, caracterizado pelo algoritmo usado pelo nível da taxa de erro (5 ou 20%).

Níveis maiores de perdas não foram usados porque o protocolo de controle de congestionamento usado (TFRC, seção 2.2.2) usa o cálculo das perdas para determinar a taxa máxima de transferência. Desta forma, perdas maiores levam a uma taxa insuficiente para a transmissão de vídeo

Figura 6.1: Nós que atenderam aos requistos para os testes.

em tempo real. Outros protocolos mais agressivos, entretanto, podem oferecer melhores resultados.

Os dados foram registrados nos nós durante a execução de cada teste e transferidos após seu final para a extração das medições apresentadas a seguir. Cada cenário – que envolve um determinado algoritmo e taxa de erro – foi testado quatro vezes e a média das execuções foi usada.

O arquivo usado nos testes possuía taxa de *bits* de $249kb/s$e duração de 30 minutos. Todos os nós observaram um limite de emissão durante o experimento, de $2Mbps$.

6.3 Resultados

Esta seção contém os gráficos e tabelas com os resultados dos experimentos. Cada gráfico contém somente a avaliação dos algoritmos sob um único nível de erro (5 ou 20%) para facilitar a leitura. As tabelas trazem os valores médios entre as quatro execuções de cada cenário na coluna μ, sendo o desvio padrão incluído na coluna σ. Como os valores do desvio padrão foram baixos eles não são discutidos individualmente, mas estão presentes para demonstrar a baixa variabilidade dos resultados.

O texto se refere aos algoritmos *SeRViSO*, Adaptativo e *(m,k)-Firm* como 'algoritmos *SeRViSO*', visto que foram propostos neste trabalho. Ao contrário do modo Tradicional, que é apenas um modo de funcionamento similar ao das redes sobrepostas orientadas a dados anteriores, apenas com a adição de um mecanismo de retransmissão simples.

6.3.1 Tráfego

As medições abaixo avaliam o tipo dos dados transferidos, sejam dados, controle, retransmissões, ou ainda algumas mensagens específicas do protocolo que convêm analisar detalhadamente, como ***QDATA***, ***NACK*** e ***QNACK***.

6.3.1.1 Mensagens de Controle

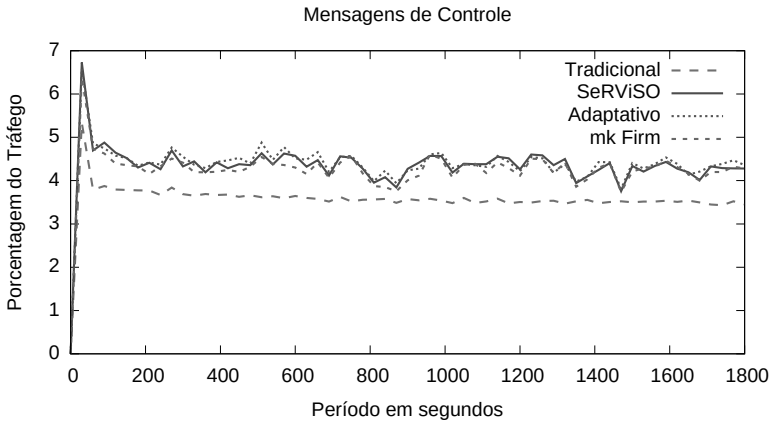
Nesta avaliação, o tráfego de mensagens de controle do protocolo é comparado com o tráfego gerado para a transmissão dos dados (taxa base), sem incluir as retransmissões. O propósito dessa medida é verificar se o nível de tráfego que essas mensagens utilizam atinge níveis prejudiciais ao tráfego das mensagens mais importantes, as de dados. O cálculo é feito pela seguinte fórmula:

$$\frac{mensagens\ exceto\ \{\textbf{\textit{DATA}},\ \textbf{\textit{QDATA}}\}}{mensagens\ \{\textbf{\textit{DATA}},\ \textbf{\textit{QDATA}}\}}$$

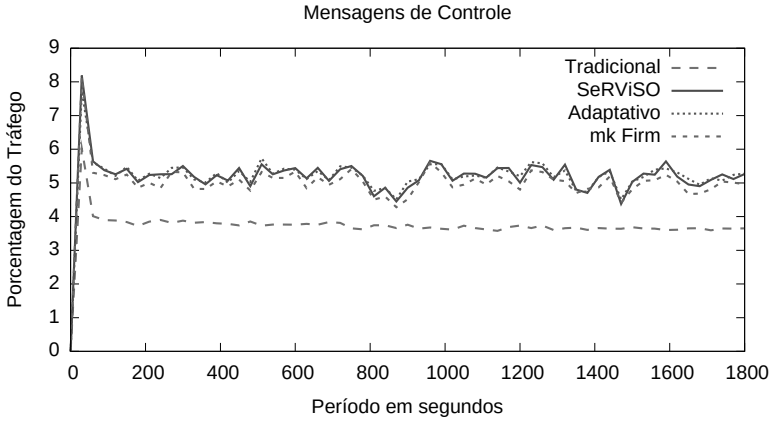
	5% de Perda		20% de Perda	
Algoritmo	μ	σ	μ	σ
Tradicional	3.60%	0.14%	3.74%	0.10%
SeRViSO	4.38%	0.11%	5.23%	0.20%
Adaptativo	4.43%	0.15%	5.24%	0.16%
(m,k)-Firm	4.29%	0.11%	5.03%	0.08%

Tabela 6.1: Mensagens de Controle

Acompanhe a análise na figura 6.2 e na tabela 6.1. Média e desvio padrão são representados pelas letras gregas μ e σ, respectivamente. Em todos os casos essas mensagens elevam o tráfego menos de 6%, mesmo quando as perdas estão em 20%. O pico inicial se refere ao momento em que os nós estão estabelecendo parcerias e nem todos iniciaram a transferência de dados ainda, sendo portanto a taxa base nesse momento menor. Entre os algoritmos do *SeRViSO* não há diferença significativa no volume dessas mensagens, embora levemente seja superior ao modo Tradicional. Há um pequeno crescimento nessas mensagens (menos de 1%), comparando os cenários com 5% e 20% de erro, mas inferior ao aumento nas perdas (15%). As variações presentes nos gráficos e que contrastam com a estabilidade do modo Tradicional são explicadas principalmente pelas mensagens ***METADATA*** e ***NACK***. Como o tamanho e o número dos elementos representados varia a cada segmento, essas mensagens são afe-



(a) 5% de erro.



(b) 20% de erro.

Figura 6.2: Mensagens de controle.

tadas, contrastando com a estabilidade da representação por *bitmap* no modo Tradicional.

6.3.1.2 Retransmissões

Comparando a quantidade de dados retransmitidos com a taxa base pode-se verificar qual o nível de esforço que cada algoritmo emprega para

obter os pacotes perdidos. O propósito dessa medida é quantificar como os mecanismos de cada algoritmo refletem em aumento de tráfego para retransmissões. O cálculo é feito pela seguinte fórmula:

$$\frac{retransmissões}{transmissões \ + \ retransmissões}$$

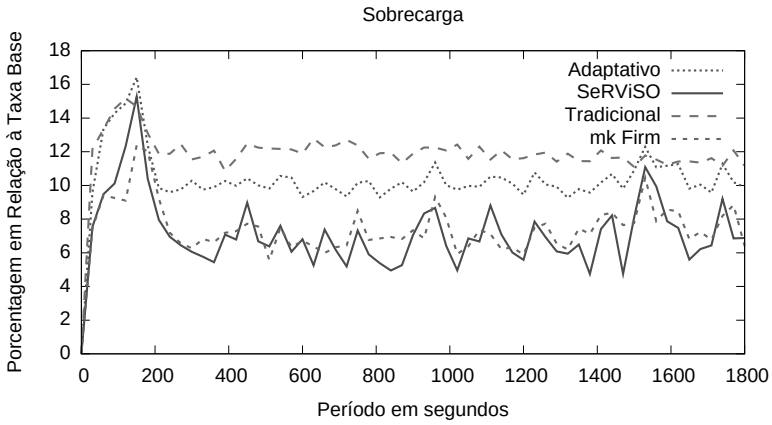
	5% de Perda		20% de Perda	
Algoritmo	μ	σ	μ	σ
Tradicional	12.03%	0.08%	29.27%	0.77%
SeRViSO	7.22%	0.59%	15.28%	0.28%
Adaptativo	10.48%	0.56%	23.03%	0.23%
(m,k)-Firm	7.50%	0.65%	17.56%	0.44%

Tabela 6.2: Sobrecarga por Retransmissões

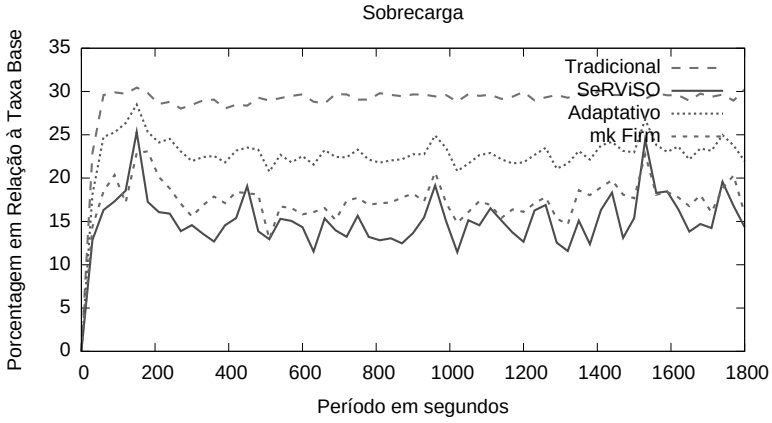
Acompanhe a análise na figura 6.3 e na tabela 6.2. Enquanto o modo Tradicional realiza retransmissões sempre que pode, elevando a sobrecarga, os algoritmos *SeRViSO* realizam esforço menor, em variados graus. *SeRViSO* e *(m,k)-Firm* estão em níveis semelhantes, com o último levemente mais alto devido ao seu critério de evitar perdas próximas. Já o Adaptativo realiza mais retransmissões para os níveis de perda testados, mas em cenários com mais perdas isso poderia se inverter, visto que os outros algoritmos (*SeRViSO* e *(m,k)-Firm* têm metas fixas. A diferença entre as sobrecargas do modo Tradicional e dos algoritmos *SeRViSO* chegou a 14% de economia, em relação à taxa base. Os picos mais acentuados dos algoritmos *SeRViSO* nos gráficos são atribuídos a perdas concentradas (aleatórias) de fatias I, que sempre são escolhidas para retransmissão. Em contraste, quando as perdas recaem em maior número sobre fatias P ou B, os algoritmos são mais tolerantes, gerando mínimos locais nos gráficos. Já o modo Tradicional desconhece o tipo de dados perdidos e recupera indistintamente, apresentando valores mais estáveis.

6.3.1.3 Mensagens *QDATA*

Foi verificado o uso de mensagens ***QDATA*** em lugar das ***DATA***, em comparação à taxa base. Elas ocorrem em dois cenários: no modo "desespero" (seção 5.7) ou quando o parceiro que originalmente enviou o segmento não possui uma parte escolhida para retransmissão e outro parceiro recebeu essa tarefa (seção 5.5.8). O propósito dessa medida é verificar o quão frenquente foi o uso desses modos apresentados neste trabalho. O cálculo é feito pela seguinte fórmula:



(a) 5% de erro.

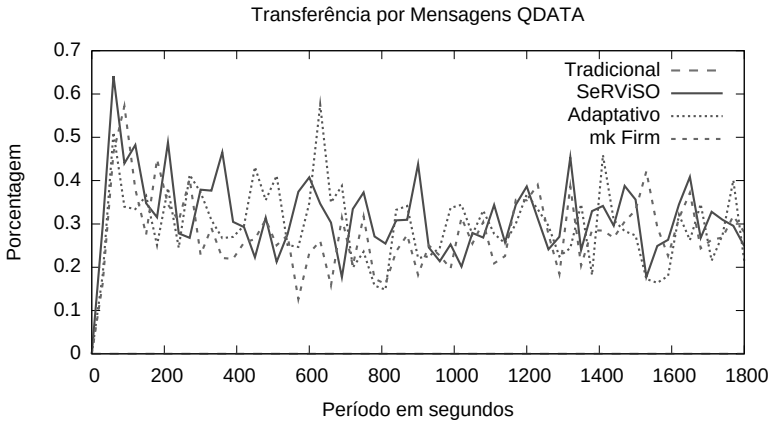


(b) 20% de erro.

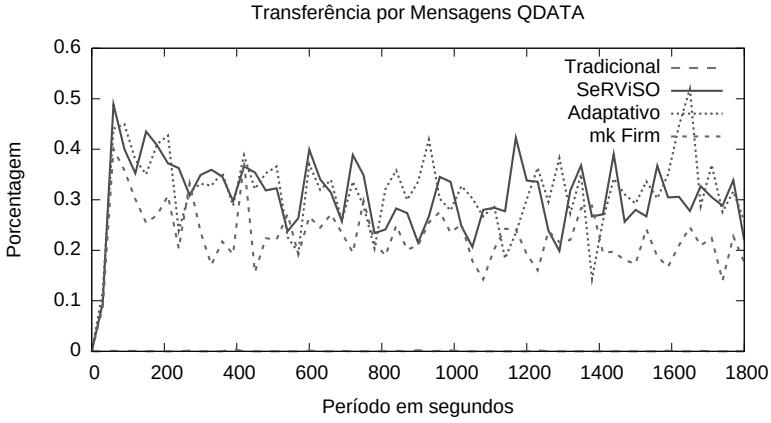
Figura 6.3: Sobrecarga para retransmissões.

$$\frac{mensagens\ \boldsymbol{QDATA}}{mensagens\ \{\boldsymbol{DATA},\ \boldsymbol{QDATA}\}}$$

Acompanhe a análise na figura 6.4 e na tabela 6.3. Não houve diferença significativa no uso de mensagens ***QDATA*** entre os algoritmos



(a) 5% de erro.



(b) 20% de erro.

Figura 6.4: Uso de mensagens ***QDATA***.

SeRViSO, independente da taxa de perda, além do nível ser baixo (pico e média inferiores a 0.65 e 0.35% respectivamente). Isso indica que essa mensagem é necessária em eventos isolados e relativamente raros. O modo Tradicional empregou essa mensagem em níveis tão baixos que não são visíveis nas figuras ou na tabela.

Algoritmo	5% de Perda μ	5% de Perda σ	20% de Perda μ	20% de Perda σ
Tradicional	0.00%	0.00%	0.00%	0.00%
SeRViSO	0.32%	0.04%	0.31%	0.06%
Adaptativo	0.30%	0.01%	0.32%	0.06%
(m,k)-Firm	0.28%	0.15%	0.23%	0.04%

Tabela 6.3: Transferências por Mensagens ***QDATA***

6.3.1.4 Mensagens *(Q)NACK*

A quantidade de NACKs enviados por segundo é um reflexo da forma como cada algoritmo avalia a retransmissão, visto que todos seguem o mesmo processo para controlar a execução do processo de seleção de elementos e envio dessas mensagens. O propósito dessa medida é analisar como os algoritmos propostos fazem uso de mensagens ***(Q)NACK*** para atingir suas metas. O cálculo é feito pela seguinte fórmula:

$$\frac{mensagens\ \{\textbf{\textit{NACK}}, \textbf{\textit{QNACK}}\}\ /\ segundo}{total\ de\ nós}$$

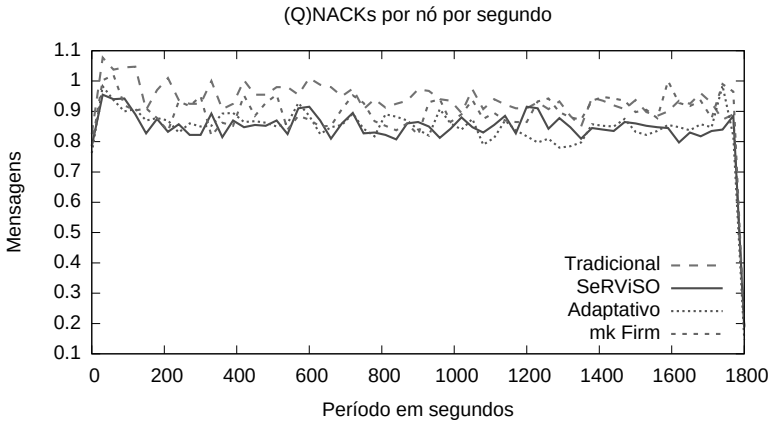
Algoritmo	5% de Perda μ	5% de Perda σ	20% de Perda μ	20% de Perda σ
Tradicional	0.93	0.07	2.08	0.03
SeRViSO	0.85	0.03	1.98	0.07
Adaptativo	0.85	0.03	1.75	0.04
(m,k)-Firm	0.89	0.03	2.05	0.02

Tabela 6.4: Mensagens ***(Q)NACK*** enviadas por nó a cada segundo.

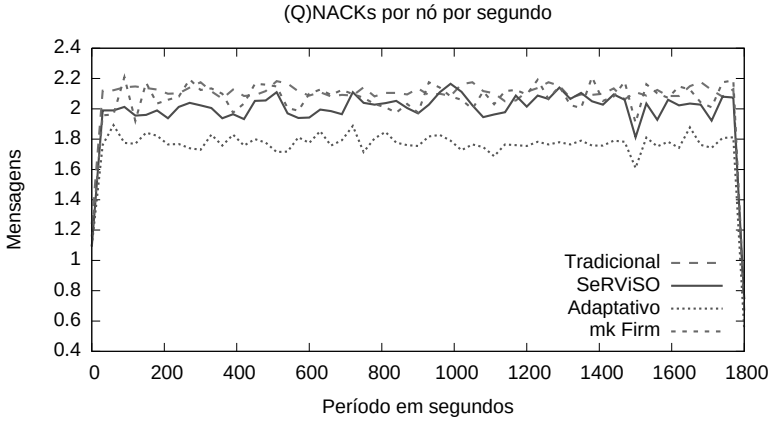
Veja a figura 6.5 e a tabela 6.4. Especialmente a 20% de perda, algumas características ficam mais visíveis:

1. O modo Tradicional envia NACKs (considerando cada segmento) enquanto houver tempo, enquanto os outros param antes, de acordo com suas metas;

2. O *(m,k)-Firm* evita perdas próximas, o que o leva a enviar mais NACKs que o *SeRViSO*;



(a) 5% de erro.



(b) 20% de erro.

Figura 6.5: Uso de mensagens ***(Q)NACK***.

3. O Adaptativo para de enviar NACKs antes dos outros, mesmo realizando mais retransmissões (seção 6.3.1.2). Como ele se adapta às condições da rede, na primeira vez que ele é executado para um determinado segmento, escolhe todas as perdas para retransmissão, sendo que a meta é diminuída a cada execução (seção 5.6.2). Desta

forma, inicialmente ocorrem mais pedidos de retransmissão, mas como eles diminuem o algoritmo para de enviar NACKs mais cedo.

6.3.2 Pacotes Atrasados

O recebimento de pacotes após seu tempo de exibição é duplamente nocivo. Além de já estarem perdidos para o usuário, sua transmissão ocupou um canal de comunicação já saturado. Assim eles podem acabar impossibilitando o recebimento de outros pacotes posteriores que poderiam chegar a tempo. O propósito dessa medida é se os algoritmos permitem que essa situação prejudicial à transmissão em tempo real aconteça. O cálculo é feito pela seguinte fórmula:

$$\frac{dados\ recebidos\ após\ perderem\ seu\ \textbf{deadline}}{dados\ (válidos\ ou\ não)\ enviados\ ao\ tocador}$$

	5% de Perda		20% de Perda	
Algoritmo	μ	σ	μ	σ
Tradicional	0.06%	0.03%	0.13%	0.05%
SeRViSO	0.00%	0.00%	0.00%	0.00%
Adaptativo	0.00%	0.00%	0.00%	0.00%
(m,k)-Firm	0.00%	0.00%	0.00%	0.00%

Tabela 6.5: Pacotes Atrasados

Acompanhe a análise na figura 6.6 e na tabela 6.5. Os algoritmos do *SeRViSO* experimentaram atrasos em níveis tão baixos que não são visíveis na tabela e muito pouco nas figuras. Já o modo Tradicional exibe esse problema, mas muito pouco para ser relevante (0.3% no seu ápice). Tal eficiência deve ser atribuída às janelas do Escalonador e do modo "Desespero", esta última pouco à frente do segmento tocando (seção 5.5.2).

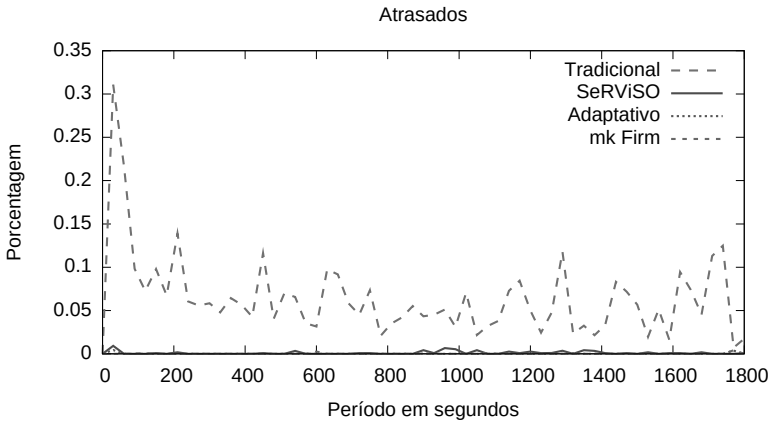
6.3.3 Perdas

Essa medida verifica a quantidade de dados que não chegaram ou que chegaram atrasados. O cálculo é feito através das seguintes fórmulas:

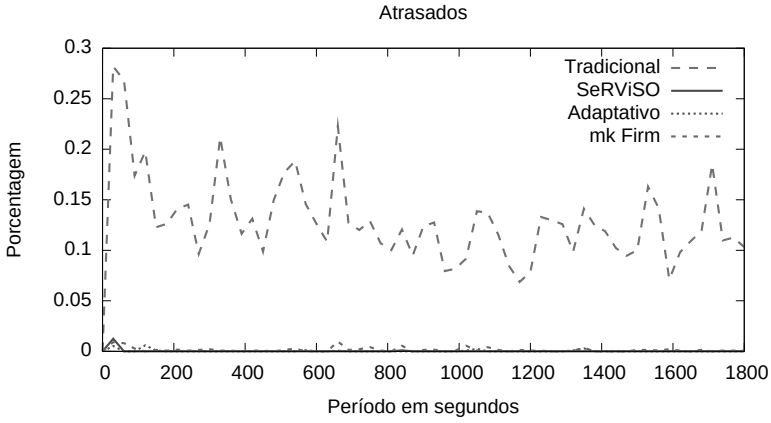
$$Geral: \quad \frac{dados\ perdidos}{dados\ na\ mídia\ original}$$

$$I: \quad \frac{quadros\ I\ perdidos\ /\ quadros\ I\ na\ mídia\ original}{perda\ geral}$$

Acompanhe a análise na figura 6.7 e na tabela 6.6. As perdas medidas mostram a tolerância dos algoritmos na seguinte ordem: $Tradicional$



(a) 5% de erro.

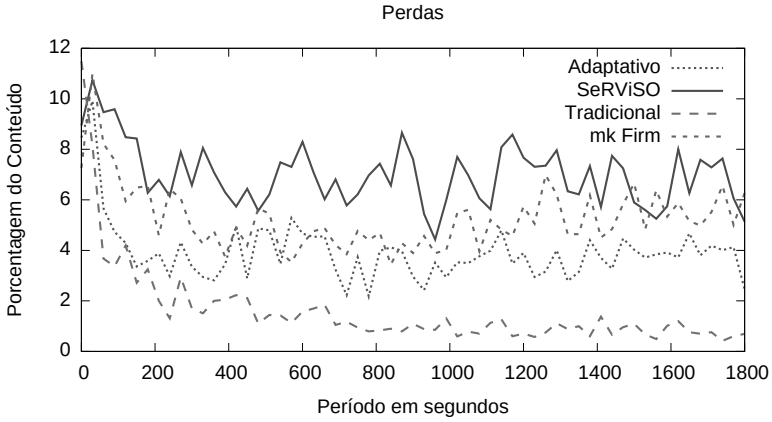


(b) 20% de erro.

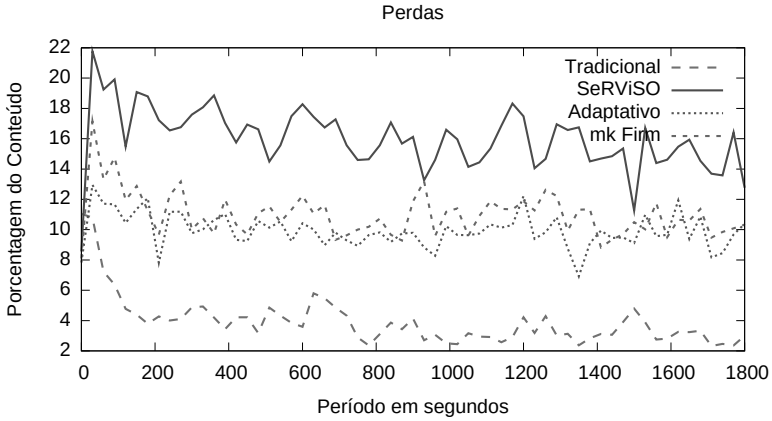
Figura 6.6: Chegada de pacotes atrasados.

$<$ *Adaptativo* $<$ *(m,k)-Firm* $<$ *SeRViSO*. Para as taxas de erro medidas não existe tanta diferença entre Adaptativo e *(m,k)-Firm*, mas o primeiro seria mais tolerante com taxas de perda mais altas. Por isso, as curvas e as médias estão mais próximas no cenário a 20% do que no cenário 5%. Alguns algoritmos superam o valor de 5% de perdas, isso acontece por-

que o valor corresponde às perdas induzidas apenas, não às que ocorreram naturalmente na rede.



(a) 5% de erro.



(b) 20% de erro.

Figura 6.7: Perda média.

Comparando a taxa geral de perda com a medida para cada tipo de fatia pode-se determinar a ‘seletividade’ dos algoritmos, ou seja, o quanto

Algoritmo	*Geral*		*I*	
	μ	σ	μ	σ
5% de Perda				
Tradicional	1.4%	0.5%	101.4%	10.4%
SeRViSO	7.0%	0.5%	57.6%	4.4%
Adaptativo	3.9%	1.0%	81.7%	6.6%
(m,k)-Firm	5.2%	1.5%	64.2%	9.0%
20% de Perda				
Tradicional	3.8%	1.4%	110.3%	11.5%
SeRViSO	16.1%	1.2%	57.1%	7.1%
Adaptativo	10.0%	1.2%	65.3%	3.3%
(m,k)-Firm	11.0%	0.8%	65.7%	7.1%

Tabela 6.6: Perdas Médias.

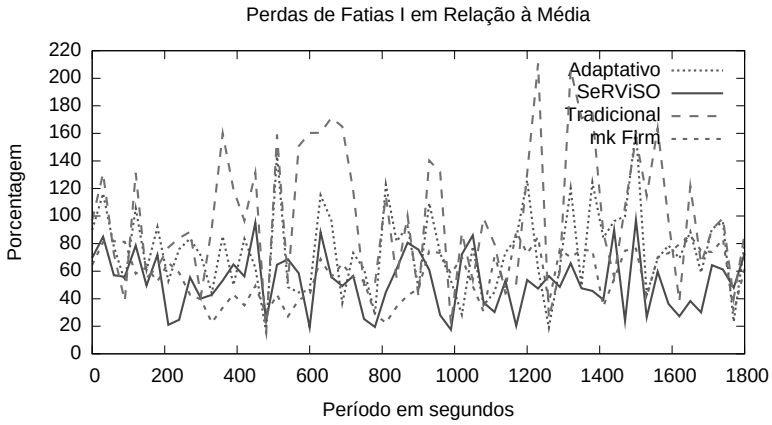
eles se focaram nos quadros mais importantes [5]:

1. As perdas do modo Tradicional são aleatórias;
2. O algoritmo *SeRViSO* básico parece ser mais 'focado' em fatias I. Entretanto deve-se levar em conta que ele apenas está tolerando mais perdas do que os outros tipos de fatias. O Adaptativo, por outro lado, mostra uma profunda mudança nesse aspecto entre os cenários com taxa de perda de 5 e 20%. Se no primeiro as perdas de fatias I eram 81.7% da perda média, no segundo foram de 65.3%, em cenários com maiores perdas a seletividade seria ainda mais intensa;

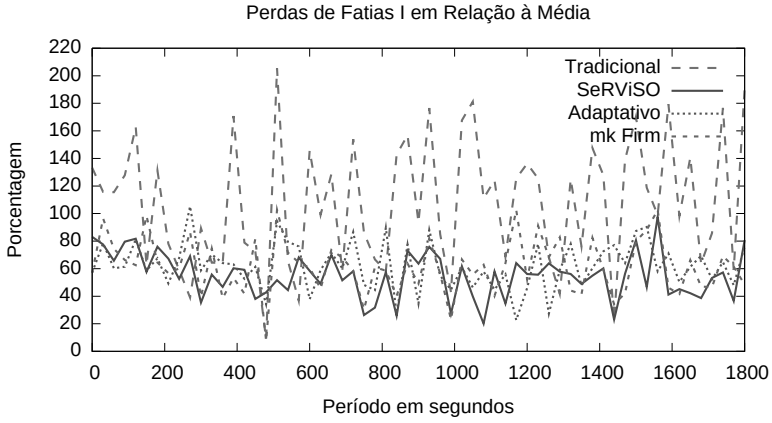
6.3.4 Análise Geral

Como trabalhos relacionados não estavam disponíveis para estudos comparativos, um modo Tradicional foi acrescentado ao protótipo para oferecer uma base de comparação. Avaliando o tráfego extra das retransmissões num cenário com 20% de perdas introduzidas artificialmente, nos modos Tradicional, *SeRViSO*, Adaptativo e *(m,k)-Firm* foram medidos respectivamente: 29.97, 15.28, 23.03 e 17.56% do tráfego base, que consiste apenas nas transmissões de dados. O tráfego de mensagens de controle não foi significativo em nenhum caso: 3.74, 5.23, 5.24 e 5.03%, na mesma sequência. A chegada de pacotes atrasados (após o segmento

[5] Observação: a medição das perdas das fatias com codificação I está incluído numa categoria que contém os outros tipos de elementos avaliados com tendo o mesmo peso pelo algoritmo 5.6.



(a) 5% de erro.



(b) 20% de erro.

Figura 6.8: Perda de fatias I em relação ao total.

já ter sido exibido) foi inferior a 0.3% em todos os casos. Assim, os algoritmos propostos – *SeRViSO*, Adaptativo e *(m,k)-Firm* – economizam 13.45%, 5.44% e 11.12% do tráfego base.

As perdas foram medidas (no mesmo cenário de erros induzidos) em 3.8, 16.1, 10 e 11% do vídeo nos modos Tradicional, *SeRViSO*, Adap-

tativo e *(m,k)-Firm*. Se as perdas das fatias com codificação I (as mais importantes) são comparadas com a perda média, os resultados foram 110.3, 57.1, 65.3 e 65.7%, respectivamente.

Comparando as medidas das retransmissões e perdas citadas nos dois parágrafos acima, e avaliando o funcionamento dos algoritmos sob os dois níveis de perdas induzidas (5% e 20%), tira-se as recomendações da próxima seção.

6.3.5 Qual o melhor algoritmo?

Depende do cenário. Para perdas elevadas ou que variam muito, o Adaptativo consegue se ajustar melhor que os outros. Se a taxa de perda é pequena e a rede tem disponibilidade para retransmissões ele também é a melhor opção. Mas se as taxa de perda for pequena e também estável (sem picos) o modo Tradicional ainda oferece vantagens, devido à sua simplicidade. Quando um determinado nível de qualidade é esperado os modos *SeRViSO* e *(m,k)-Firm* são a melhor escolha, especialmente o último. As metas que foram fixadas nesses algoritmos podem ser alteradas de acordo com a qualidade desejada.

6.3.6 Métrica de Qualidade

Mecanismos simples tentam 'adivinhar' a perda de qualidade da apresentação reconstituindo os grupos de figuras e assinalando a perda de todas as fatias que dependem de outra já perdida [Bortoleto 2005]. Desta forma, conseguem criar uma medida unificada de qualidade. Esse procedimento é simples quando o grupo de figuras tiver padrões de organização claros (por exemplo, IBBPBBPBBPBBI), ainda mais se cada figura consistir de apenas uma fatia.

Mas essa avaliação torna-se mais complexas e sujeita a erros com o H.264, já que múltiplas figuras podem ser usadas como referência e os padrões de organização são extremamente flexíveis, inclusive com a presença de partições. Devido à essas dificuldades neste trabalho obtou-se por comparar o nível geral de perda com o das fatias I. Também outros avanços do H.264 (seção 2.1) permitem reprodução parcial a partir de dados incompletos (p. ex: partição A sem a C correspondente). Assim, os únicos programas que podem realmente atribuir o nível de qualidade são os tocadores de mídia – e não programas externos a eles – sendo que um pode ser 'mais inteligente' que outro na presença de perdas.

Capítulo 7

Conclusão

Em virtude de sua versatilidade, as redes sobrepostas oferecem uma interessante alternativa ao *IP Multicast*. Existem diversos tipos delas, seja em CDNs, redes P2P ou para implementar *multicast* na camada de aplicação. Critérios para retransmissão já haviam sido aplicados em protocolos de *multicast* semanticamente-confiáveis e na retransmissão seletiva de *streaming* cliente-servidor. Agora este trabalho oferece uma nova proposta voltada a redes sobrepostas, baseado nas características do vídeo codificado segundo o padrão H.264, na forma de algoritmos que solicitem a retransmissão de acordo com o tipo de codificação das fatias de vídeo.

Na rede sobrepostas *SeRViSO* os nós clientes descobrem uns aos outros através de nós *Rendezvous*, que provêm somente essa função. Conhecendo uns aos outros eles estabelecem parcerias para trocar partes do vídeo sem saber quem é o servidor (o nó que disponibiliza o vídeo). Cada nó avisa aos parceiros as partes que possui, de modo que possam solicitá-las. Cada parte do vídeo, originalmente disponível apenas no servidor, é distribuídas a todos os nós estáveis da rede, parceria a parceria. As partes perdidas durante a transmissão são avaliadas por um dos algoritmos propostos, que escolhe quais devem ter sua retransmissão requerida.

São propostos três algoritmos, sendo que os dois primeiros estabelecem prioridades para cada parte baseadas no tipo de codificação: I tem prioridade maior que P, cuja prioridade é maior que B. A partir das prioridades o algoritmo *SeRViSO* escolhe todas do tipo I, e também entre os outros, de forma que obtenha 90% da soma das prioridades de todos as partes do segmento, sendo as prioridades baseadas no tipo de codificação e no tamanho da parte. O algoritmo adaptativo desenvolve esse mecanismo, adotando de uma meta que se adapte ao estado da rede em vez de fixa em 90%. Desse modo, na primeira vez que um segmento tem partes escolhidas para retransmissão, todo ele é requisitado, mas nas vezes seguintes a meta é reduzida a cada execução. Já o algoritmo *(m,k)-Firm* opera de forma diferente, separando os pacotes em grupos de acordo com seu tipo (I, P e B). Ele tolera a perda de 1 em cada sequência de 12 do tipo

P, ou 1 em 6 do tipo B, mas nenhum do tipo I – qualquer situação acima disso é selecionada para recuperação.

Pelos resultados da Análise Final (seção 6.3.4), os três algoritmos propostos atingem o objetivo de fornecer uma recuperação seletiva, de forma que para ambientes com taxa de erro imprevisível, ou pequeno, ou grande, o algoritmo Adaptativo oferece benefícios, mas quando a qualidade desejada é fixa o *(m,k)-Firm* (principalmente) é uma boa opção, sempre com a possíbilidade de adaptar as metas à qualidade desejada.

Também verificou-se durantes os testes que o PlanetLab é um instrumento valioso para a realização de testes automatizados, desde que aprenda-se a lidar com seus problemas: instabilidade dos nós individuais e a dificuldade em instalar (e manter) programas nos nós.

Continuando a pesquisa em torno do tema deste trabalho, pode-se enumerar alguns possíveis trabalhos futuros:

1. **Segurança**: testar mecanismos de segurança para redes sobrepostas sobre o *SeRViSO* permite avaliar alguns aspectos mais realisticamente que com simuladores, como por exemplo tráfego, atraso e custo computacional extras;

2. **PUSH**: a partir da integração desse modo de operação ao *SeRViSO*, investigar se os algoritmos continuam adequados ou devem ser adaptados;

3. **Rendezvous X Protocolos Fofoqueiros**: compará-los em diferentes cenários, em termos da quantidade de nós e da taxa de abandono, como o objetivo de determinar quando são necessários vários nós Rendezvous e também qual mecanismo de controle de membresia é mais apropriado em cada cenário;

4. **Localidade e *IP Multicast***: empregando pacotes *IP Multicast* para comunicação dentro de uma sub-rede, nós interessados em determinado conteúdo podem descobrir uns aos outros e cooperar entre si, evitando receber os mesmo dados, o que sobrecarrega a conexão da sub-rede com a Internet sem necessidade. Tal mecanismo pode ser usado tanto em redes sobrepostas como em CDNs ou em qualquer tipo de *multicast*, de modo que o suporte a *IP Multicast* entre diferentes redes não seja necessário, apenas nos nós nas extremidades. Essas técnicas são chamadas de *island multicast* [Jin, Cheng e Chan 2009], e o desafio seria como realizar a retransmissão seletiva de erros nesse contexto de cooperação entre os nós de uma mesma sub-rede.

Durante o desenvolvimento deste trabalho o artigo [Lenz, Lung e Siqueira 2010] foi publicado, correspondendo a um estado intermediário de desenvolvimento da proposta aqui presente (somente um algoritmo é proposto).

Referências Bibliográficas

[Adar e Huberman]ADAR, E.; HUBERMAN, B. *Free riding on gnutella*. Disponível em: <http://www.hpl.hp.com/research/idl/papers/gnutella/index.html>.

[Adler 1995]ADLER, R. M. Distributed coordination models for client/-server computing. *IEEE Computer*, 1995.

[Akan e Akyildiz 2004]AKAN, Ö. B.; AKYILDIZ, I. F. ARC: the analytical rate control scheme for real-time traffic in wireless networks. *IEEE/ACM Transactions on Networking (TON)*, 2004.

[Basics of Video 2010]BASICS of Video. fevereiro 2010. Disponível em: <http://lea.hamradio.si/ s51kq/V-BAS.HTM>.

[Bortoleto 2005]BORTOLETO, C. M. *MULTICAST SEMI-CONFIÁVEL PARA APLICAÇõES MULTIMÍDIA DISTRIBUíDAS*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Paraná, PUC-PR, Brasil, 2005.

[Braden et al. 1997]BRADEN, R. et al. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. [S.l.], set. 1997. 112 p.

[Claypool e Tanner 1999]CLAYPOOL, M.; TANNER, J. The effects of jitter on the perceptual quality of video. In: *Proceedings of the seventh ACM international conference on Multimedia (Part 2)*. [S.l.: s.n.], 1999.

[Coulouris, Dollimore e Kindberg 2001]COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems: concepts and design*. [S.l.]: Addison-Wesley, 2001.

[Crispin 2003]CRISPIN, M. *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. [S.l.], mar. 2003. 108 p.

[Deering 1989]DEERING, S. Host Extensions for IP Multicast (RFC 1112). *Internet Engineering Task Force*, 1989.

[Deering e Cheriton 1990]DEERING, S.; CHERITON, D. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems (TOCS)*, 1990.

[Défago, Schiper e Urbán 2004]DéFAGO, X.; SCHIPER, A.; URBáN, P. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 2004.

[Demers et al. 1987]DEMERS, A. et al. Epidemic algorithms for replicated database maintenance. In: *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*. [S.l.: s.n.], 1987.

[Diot et al. 2000]DIOT, C. et al. Deployment issues for the ip multicast service and architecture. *Network, IEEE*, 2000.

[Ebrahimi e Horne 2000]EBRAHIMI, T.; HORNE, C. MPEG-4 natural video coding–An overview. *Signal Processing: Image Communication*, 2000.

[Eriksson 1994]ERIKSSON, H. Mbone: the multicast backbone. *Communications of the ACM*, 1994.

[Fielding et al. 1997]FIELDING, R. et al. *Hypertext Transfer Protocol – HTTP/1.1*. [S.l.], jan. 1997. 162 p. Disponível em: <http://www.rfc-editor.org/rfc/rfc2068.txt>.

[Floyd et al. 1995]FLOYD, S. et al. A reliable multicast framework for light-weight sessions and application level framing. In: *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication – SIGCOMM*. [S.l.: s.n.], 1995.

[Ganesh, Kermarrec e Massoulié 2003]GANESH, A.; KERMARREC, A.; MASSOULIÉ, L. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 2003.

[Golding e Long 1992]GOLDING, R.; LONG, D. The performance of weak-consistency replication protocols. *University of California at Santa Cruz, Computer Research Laboratory Technical Report UCSCCRL-92-30*, 1992.

[Handley et al. 2003]HANDLEY, M. et al. TCP Friendly Rate Control Protocol (RFC 3448). *Internet Engineering Task Force*, 2003.

[Huszák e Imre 2008]HUSZÁK, Á.; IMRE, S. Content-Aware Selective Retransmission Scheme in Heavy Loaded Wireless Networks. *Wireless and Mobile Networking*, 2008.

[Jin, Cheng e Chan 2009]JIN, X.; CHENG, K.; CHAN, S. Island Multicast: Combining IP Multicast With Overlay Data Distribution. *IEEE TRANSACTIONS ON MULTIMEDIA*, 2009.

[JPEG, Entropy Encoding 2010]JPEG, Entropy Encoding. fevereiro 2010. Disponível em: <http://en.wikipedia.org/wiki/JPEG#Entropy_coding>.

[Kohler et al. 2006]KOHLER, E. et al. Datagram congestion control protocol (DCCP – RFC 4340). *Internet Engineering Task Force*, 2006.

[Kontothanassis et al. 2004]KONTOTHANASSIS, L. et al. A transport layer for live streaming in a content delivery network. *Proceedings of the IEEE*, 2004.

[Krishnamurthy, Wills e Zhang 2001]KRISHNAMURTHY, B.; WILLS, C.; ZHANG, Y. On the use and performance of content distribution networks. In: *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. [S.l.: s.n.], 2001.

[Lenz, Lung e Siqueira 2010]LENZ, C. E.; LUNG, L. C.; SIQUEIRA, F. A. Serviso: A selective retransmission scheme for video streaming in overlay networks. *ACM Symposium on Applied Computing - SAC*, 2010.

[Levine e Garcia-Luna-Aceves 1998]LEVINE, B.; GARCIA-LUNA-ACEVES, J. A comparison of reliable multicast protocols. *Multimedia Systems*, 1998.

[Liang, Chanson e Neufeld 1990]LIANG, L.; CHANSON, S.; NEUFELD, G. Process groups and group communications. *IEEE Computer*, 1990.

[Lin e Paul 1996]LIN, J.; PAUL, S. RMTP: A reliable multicast transport protocol. In: *IEEE INFOCOM*. [S.l.: s.n.], 1996.

[Liu 1999]LIU, C. Multimedia over IP: RSVP, RTP, RTCP, RTSP. In: CRC PRESS, INC. *Handbook of emerging communications technologies*. [S.l.], 1999.

[Lua et al. 2005]LUA, K. et al. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, 2005.

[MacAulay, Felts e Fisher 2005]MACAULAY, A.; FELTS, B.; FISHER, Y. *WHITEPAPER–IP Streaming of MPEG-4: Native RTP vs MPEG-2 Transport Stream*. October 2005.

[Mase et al. 1983]MASE, K. et al. Go-back-n arq schemes for point-to-multipoint satellite communications. *Communications, IEEE Transactions on*, 1983.

[Nichols et al. 1998]NICHOLS, K. et al. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. [S.l.], dez. 1998. 20 p. Disponível em: <http://www.rfc-editor.org/rfc/rfc2474.txt>.

[Peng 2004]PENG, G. CDN: Content Distribution Network. *Arxiv preprint cs.NI/0411069*, 2004.

[Pereira, Rodrigues e Oliveira 2003]PEREIRA, J.; RODRIGUES, L.; OLIVEIRA, R. Semantically reliable multicast: Definition, implementation, and performance evaluation. *IEEE Transactions on Computers*, 2003.

[Postel 1981]POSTEL, J. *Transmission Control Protocol*. [S.l.], set. 1981. 85 p. Disponível em: <http://www.rfc-editor.org/rfc/rfc793.txt>.

[Postel 1982]POSTEL, J. *Simple Mail Transfer Protocol*. [S.l.], ago. 1982. 68 p. Disponível em: <http://www.rfc-editor.org/rfc/rfc821.txt>.

[Pouwelse et al. 2005]POUWELSE, J. et al. The bittorrent p2p file-sharing system: Measurements and analysis. *Peer-to-Peer Systems IV*, 2005.

[Queiroz et al. 2009]QUEIROZ, W. et al. Semantically reliable multicast based on the (m-k)-firm technique. *Parallel and Distributed Processing with Applications, International Symposium on*, 2009.

[Rasti, Stutzbach e Rejaie 2006]RASTI, A.; STUTZBACH, D.; REJAIE, R. On the Long-term Evolution of the Two-Tier Gnutella Overlay. In: *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*. [S.l.: s.n.], 2006.

[Ripeanu e Foster 2001]RIPEANU, M.; FOSTER, I. Peer-to-peer architecture case study: Gnutella network. In: *Proceedings of International Conference on Peer-to-peer Computing*. [S.l.: s.n.], 2001.

[Rosenberg et al. 2002]ROSENBERG, J. et al. *SIP: Session Initiation Protocol*. [S.l.], jun. 2002. 269 p. Disponível em: <http://www.rfc-editor.org/rfc/rfc3261.txt>.

[Schulzrinne et al. 2003]SCHULZRINNE, H. et al. *RTP: A Transport Protocol for Real-Time Applications*. [S.l.], jul. 2003. 104 p. Disponível em: <http://www.rfc-editor.org/rfc/rfc3550.txt>.

[Schulzrinne, Rao e Lanphier 1998]SCHULZRINNE, H.; RAO, A.; LANPHIER, R. *Real Time Streaming Protocol (RTSP)*. [S.l.], abr. 1998. 92 p. Disponível em: <http://www.rfc-editor.org/rfc/rfc2326.txt>.

[Schulzrinne e Rosenberg 1998]SCHULZRINNE, H.; ROSENBERG, J. A Comparison of SIP and H. 323 for Internet Telephony. In: *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*. [S.l.: s.n.], 1998.

[Singh K.D. 2006]SINGH K.D., R. D. T. L. V. C. Improving multimedia streaming over wireless using end-to-end estimation of wireless losses. *Vehicular Technology Conference*, 2006.

[Tran, Hua e Do 2003]TRAN, D.; HUA, K.; DO, T. Zigzag: An efficient peer-to-peer scheme for media streaming. In: *IEEE INFOCOM*. [S.l.: s.n.], 2003.

[Verscheure, Frossard e Hamdi 1999]VERSCHEURE, O.; FROSSARD, P.; HAMDI, M. User-oriented QoS analysis in MPEG-2 video delivery. *Real-Time Imaging*, 1999.

[Wang, Xiong e Liu 2007]WANG, F.; XIONG, Y.; LIU, J. mTreebone: A hybrid tree/mesh overlay for application-layer live video multicast. In: *27th International Conference on Distributed Computing Systems*. [S.l.: s.n.], 2007.

[Wenger 2003]WENGER, S. H. 264/avc over ip. *IEEE Transactions on Circuits and Systems for Video Technology*, 2003.

[Wenger et al. 2005]WENGER, S. et al. *RTP Payload Format for H.264 Video*. [S.l.], fev. 2005. Disponível em: <http://www.rfc-editor.org/rfc/rfc3984.txt>.

[Wiegand et al. 2003]WIEGAND, T. et al. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology*, 2003.

[Zhang et al. 2005]ZHANG, M. et al. A Peer-to-Peer network for live media streaming using a push-pull approach. In: *Proceedings of the 13th ACM international conference on Multimedia*. [S.l.: s.n.], 2005.

[Zhang et al. 2005]ZHANG, X. et al. CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. In: *Proceedings of IEEE Infocom*. [S.l.: s.n.], 2005.