IMPLEMENTAÇÃO E AVALIAÇÃO DE UM SISTEMA DE VÍDEO SOB DEMANDA BASEADO EM CACHE COOPERATIVA COLAPSADA DE VÍDEO

Leonardo Leal Sampaio Bragato

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

Aprovada por:

Prof. Claudio Luís de Amorim, Ph.D.

Prof. Edison Ishikawa, D.Sc.

Prof. Luís Felipe Magalhães de Moraes, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 2006

BRAGATO, LEONARDO LEAL SAMPAIO

Implementação e Avaliação de um Sistema de Vídeo Sob Demanda Baseado em Cache Co-operativa Colapsada de Vídeo [Rio de Janeiro] 2006

XIII, 53 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2006)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1 - Sistemas Multimídia

2 - Vídeo sob Demanda

3 - Cache

4 - Escalabilidade

I. COPPE/UFRJ II. Título (série)

À minha mãe Iza Leal
e aos meus tios Roberto Zucca e Silvana Leal

Agradecimentos

Ao professor e orientador Claudio Luis de Amorim, por ter me aguentado todo este tempo no Laboratório de Computação Paralela, por não ser arrogante, prepotente e ter o ego exacerbado como a maioria dos outros professores que encontramos pela ao longo de nossa jornada; e como o próprio Leonardo Pinho disse em uma ocasião, no bom sentido, "por ser uma mãe para todos os seus alunos e orientandos", e eu assino embaixo.

Ao Leonardo Bidese de Pinho, que está sempre disposto a ajudar em todos os momentos, e o qual foi imprescindível para a realização desta dissertação, sempre discutindo e dando as melhores idéias.

Ao João Maurício de Oliveira Alves, por sua disciplina, organização, conhecimento, experiência e disposição que transmite a todos, e assim acabamos evoluindo tanto no lado profissional quanto pessoalmente e ao qual peço desculpas por ter atraso o desenvolvimento de sua dissertação.

Ao professor Edison Ishikawa, por ter criado em sua tese de doutorado a memória cooperativa, dando assim a mim e a outros colegas as idéias para criação de nossos trabalhos de mestrado.

Ao Lauro Whately, por seu senso crítico apurado que sempre nos leva a um degrau acima.

Ao Luiz Maltar pelo apoio dado.

A todos os outros colegas do Laboratório de Computação Paralela (LCP) como o Alexandre Coser, Fábio Matos, Rafael Mendes e etc, por tantas discussões ao longo do curso.

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

IMPLEMENTAÇÃO E AVALIAÇÃO DE UM SISTEMA DE VÍDEO SOB DEMANDA BASEADO EM CACHE COOPERATIVA COLAPSADA DE VÍDEO

Leonardo Leal Sampaio Bragato

Setembro/2006

Orientador: Claudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

Neste trabalho é implementado e avaliado o funcionamento de um sistema de vídeo sob demanda baseado na técnica CVCC (Cache de Vídeo Cooperativa Colapsada). Dentre os principais problemas encontrados estão a limitação de banda disponível aos clientes de banda larga, a perda de pacotes na rede ou no sistema operacional e o atendimento as diversas demandas do sistema em tempo real. É implementado um sistema de escalonamento de pacotes que divide adequadamente a banda disponível entre os diversos usuários do sistema, adequando-se às necessidades de cada um deles. O sistema é capaz de alocar recursos para reparar possíveis perdas ocorridas na rede. Com intuito de atender adequadamente às restrições temporais é implementado um sistema de escalonamento de CPU que divide a mesma entre as diversas tarefas do sistema. O sistema é avaliado em uma operadora de *Internet* banda larga e comporta-se como esperado, mostrando continuidade, escalabiliade e eficácia na distribuição de mídia sob demanda.

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

IMPLEMENTATION AND EVALUATION OF A VIDEO ON DEMAND SYSTEM BASED ON COLAPSED COOPERATIVE VIDEO CACHE

Leonardo Leal Sampaio Bragato

September/2006

Advisor: Claudio Luis de Amorim

Department: Computing and Systems Engineering

In this work, the CCVC (Colapsed Cooperative Video Cache) technique is implemented and evaluated. Among the main problems we found are the limited bandwidth encountered on available broadband networks, lost packets in either the network or the operating system and in time execution of multimedia real time tasks. To tackle these problems we implemented a packet scheduling system capable of dividing the available system's bandwidth among the users, based on the particular communication resources each user has. Second, the resulting system is capable of saving resources to recover packets lost in the network. Third, to take care adequately of the temporal restrictions, we also implemented a CPU scheduling system that shares such restrictions among the system tasks. Our system is evaluated on a broadband Internet service provider and the results demonstrated that it works as predicted, showing continuity, scalability, and effetiveness on delivery of videos on demand.

Sumário

Lista de Acrônimos

ADSL	*Asymetric Digital Subscriber Line*
BRAS	*Broadband Remote Acess Server*
CBR	*Constant Bit Rate*
CDN	*Content Distribution Network*
CPU	*Central Processing Unit*
CVC	*Cache de Vídeo Cooperativa*
CVCC	*Cache de Vídeo Cooperativa Colapsada*
DOCSIS	*Data Over Cable Service Specification*
DSL	*Digital Subscriber Line*
DSLAM	*Digital Subscriber Line Access Multiplexer*
DVD	*Digital Versatile Disk*
FTP	*File Transfer Protocol*
FTTH	*Fiber-To-The-Home*
GloVE-LAN	*Global Video Environment for Local Area Networks*
GloVE-WAN	*Global Video Environment for Wide Area Networks*
HFC	*Hybrid Fiber-Coax*
HTTP	*Hypertext Transfer Protocol*
IP	*Internet Protocol*

LAN	*Local Area Network*
MMS	*Microsoft Media Server Protocol*
MPEG2	*Moving Pictures Experts Group version 2 standard*
MTU	*Maximum Transmission Unit*
QdS	*Qualidade de Serviço*
RAID	*Redundant Array of Independent Disks*
RDT	*Real Networks Data Transport*
RTC	*Real Time Clock*
RTCP	*Real Time Control Protocol*
RTP	*Real Time Transport Protocol*
RTSP	*Real Time Streaming Protocol*
TCP	*Transmission Control Protocol*
TELNET	*Teletype Network*
TSC	*Time Stamp Counter*
UDP	*User Datagram Protocol*
URL	*Uniform Resource Locator*
VBR	*Variable Bit Rate*
VsD	*Vídeo sob Demanda*
WAN	*Wide Area Network*
WiMAX	*Worldwide Interoperability for Microwave Access*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Visão geral

Esta dissertação objetiva avaliar o desempenho de sistemas de distribuição de *Vídeo sob Demanda* (VsD) escalável, sendo direcionado a usuários com acesso à banda larga. Em particular, o sistema avaliado é baseado na técnica *Cache de Vídeo Cooperativa Colapsada* (CVCC), que tem como elemento principal um distribuidor de vídeo (também denominado *proxy*), atuando como dispositivo intermediário entre o servidor e os usuários.

Este trabalho envolve, portanto, a criação de um protótipo do distribuidor, tanto funcionando como *proxy* ou como servidor, dos clientes, de um aplicativo de visualização e análise do desempenho do sistema em tempo real, além da implementação de um emulador de infra-estrutura de rede de acesso à Internet banda larga.

1.2 Motivações

O avanço das tecnologias das redes de acesso à Internet banda larga, bem como dos codificadores de áudio e vídeo, possibilitou uma alta qualidade nos fluxos multimídia distribuídos. A popularidade dos sistemas de distribuição de mídias contínuas, em particular áudio e vídeo, nunca foi tão grande, nem os investimentos crescentes que vem sendo feitos pelos estúdios de cinema. Quem nunca ouviu falar do *YouTube* [32], do *Globo Media Center* [8] ou mesmo do *iTunes* [2]?

A técnica CVCC, proposta na tese de doutorado [13], mostrasse eficaz para sistemas de distribuição de vídeo em larga escala. Entretanto, sua proposta original não prevê aspectos práticos importantes e que causam impacto em seu funcionamento original, tais como: gerenciamento de recursos de rede com escalonamento de pacotes tendo em mente a limitada banda disponível para os clientes; o suporte a reenvio de pacotes eventualmente

perdidos pela rede, que é importante para diversos codificadores e multiplexadores; a utilização de discos rígidos e limitação da banda entre o *proxy* e o servidor. Portanto, se fez necessária a implementação de um protótipo funcional para avaliar na prática os resultados obtidos por simulação.

1.3 Contribuições

Dentre as contribuições da presente dissetação, estão a implementação do distribuidor com o escalonamento dos recursos necessários para que seja mantida a qualidade de exibição do fluxo de vídeo pelo cliente. Assim foi necessária a criação de um protocolo de tratamento de perdas e duplicações de pacotes.

Foi tratado o escalonamento do envio dos pacotes para os clientes e também de recepção dos dados pelo servidor e do envio de pacotes perdidos, bem como os escalonamento das diversas tarefas do sistema de maneira a continuar atendendo às requisições em tempo real. Foi implementada uma janela de inclusão de clientes que agrupa pedidos para um mesmo trecho do vídeo, dependendo do seu tamanho, e atende aos clientes em cojunto economizando recursos [7].

A avaliação do sistema em uma rede *Asymetric Digital Subscriber Line* (ADSL) de uma operadora de telecomunicações brasileira na qual foi mostrado que o distribuidor consegue atender eficientemente diversos tipos de clientes ao mesmo tempo.

1.4 Organização

No capítulo 2 damos uma introdução aos sistemas de VsD, com uma visão geral dos principais componentes e desafios encontrados, detalhando as tecnologias e padrões utilizados.
No capítulo 3 relembramos o trabalho que originou esta dissertação.
No capítulo 4 descrevemos os principais aspectos relacionados com a implementação do CVCC que é denominada *Global Video Environment for Wide Area Networks* (GloVE-WAN).
No capítulo 5 avaliamos o funcionamento da implementação.
No capítulo 6 discutimos os trabalhos relacionados.
No capítulo 7 concluímos e apresentamos propostas para deselvolvermos no futuro.

Capítulo 2

Sistemas de Vídeo sob Demanda

No contexto deste trabalho, um sistema de vídeo sob demanda compreende os seguintes componentes fundamentais: Servidores, *Proxies* e Clientes. Para interconectá-los podemos definir duas redes principais: a rede de conexão entre os *proxies* e os servidores e a rede que dá acesso aos clientes à Internet. Para utilização das redes precisamos de um protocolo de comunicação, e através do qual fazemos a distribuição da mídia contínua armazenada nos servidores para os *proxies* e clientes.

Neste capítulo explicamos brevemente o funcionamento de cada uma das partes de um sistema de vídeo sob demanda, os protocolos mais utilizados e a mídia a ser distribuída. Tratamos em seguida dos desafios encontrados no desenvolvimento desses sistemas, definindo neste contexto o que significa *Qualidade de Serviço* (QdS) e depois discutimos o funcionamento de um *proxy*. Provisoriamente, entenda-se QdS a possibilidade de podermos garantir um limiar máximo para: as taxas de erro, os atrasos dos pacotes, o *jitter*, o tempo de início de exibição e etc; contratados com os usuários.

Inicialmente, descrevemos os principais problemas e soluções encontrados nos sistemas operacionais para a distribuição de mídia contínua.

2.1 Sistemas Operacionais

Nesta seção observamos os principais problemas encontrados em sistemas operacionais de propósito geral, representado nesta dissertação pelo Linux com kernel 2.6 [5], no desenvolvimento de um sistema de VsD.

Dentre os aspectos importantes encontramos a temporização com escalonamento dos pacotes de acordo com a largura de banda disponível na rede e também o escalonamento de *Central Processing Unit* (CPU) de modo a dividir seus recursos da maneira mais adequada para atender os requisitos da aplicação, dita Soft-Real Time.

2.1.1 Medidas do Tempo

Para que possamos enviar pacotes necessitamos inicialmente medir o tempo da maneira mais precisa e o com o mínimo de interferência possível.

No sistema Linux podemos persar em três formas principais de medir tempo. Podemos utilizar chamadas a função *gettimeofday*, através da recepção de interupções ou da leitura do registrador contador do número de ciclos da CPU, chamado de *Time Stamp Counter* (TSC).

Através da chamada *gettimeofday* podemos obter o tempo com uma precisão de microsegundos e com uma grande sobrecarga.

Todos os computadores pessoais possuem um relógio chamado *Real Time Clock* (RTC). Ele é capaz de gerar interrupções periódicas com frequências entre 2 Hz e 8192 Hz.

Os computadores *x86* possuem um registrador chamado TSC. Este registrador conta cada ciclo a partir da sua inicialização. Desta forma podemos utilizá-lo para as medidas de tempo, bastando somente termos posse do relógio (*clock*) da CPU, que é calculada na inicialização do sistema operacional.

Como vemos, por sua baixa sobrecarga e alta resolução o mais indicado para utilização em sistemas de tempo real é a leitura do registrador *tsc*.

2.1.2 Atraso

Além da obtenção do tempo, precisamos conseguir atrasar a execução das próximas instruções de maneira a não sobrecarregar os recursos disponíveis em rede e CPU. Descrevemos em seguida estratégias utilizadas para gerar o atraso desejado.

Através da função *usleep* podemos atrasar a execução, passando o controle para uma outra *thread* que esteja pronta para executar. No entando a resolução desta função é de 10ms (na arquitetura x86), que é muito alta para distribuição de mídia contínua.

Outra forma de gerarmos atraso é através da execução de instruções, como por exemplo, a leitura contínua do registrador TSC até que obtenhamos o atraso desejado. O problema deste método é que sobrecarregamos a CPU, sem darmos a chance de que outra *thread* execute.

O outra meneira mais eficiente é conseguida através do escalonamento da CPU com monitoramento do tempo através de leituras ao registrador TSC. Desta forma toda vez que desejamos gerar um atraso, chaveamos para uma próxima *thread* até que o controle

volte à inicial que testará se o atraso já foi suficiente, continuando ou não a sua execução. É importante mencionar que o chaveamento deve ser limitado, senão aumentará o uso da CPU e ainda que devemos baixar a granularidade de trocas de contexto de maneira que consigamos a precisão desejada.

2.1.3 Escalonamento

Para que consigamos o efeito descrito no parágrafo anterior, devemos entender o funcionamento do escalonador do sistema operacional Linux. Neste sistema existem 3 políticas de escalonamento diferentes: uma delas para funcionar no modo convencional (*SCHED-NORMAL* ou *SCHED-OTHER*) e outras duas (*SCHED-FIFO* e *SCHED-RR*) para o modo tempo real (*real time*) [5]. Segue abaixo a explicação do funcionamento de cada uma delas.

- SCHED-FIFO: Com esta política, um processo se mantém ativo até que o mesmo decida passar a CPU para outro, neste momento o processo é colocado no fim da fila de execução. Mesmo que existam outros processos prontos para executar, os mesmos se mantém parados até que o processo em execução decida passar a CPU para outro.

- SCHED-RR: Funciona de modo análogo ao anterior, diferenciando-se por definir um intervalo de tempo máximo com que os processos podem estar executando. A partir deste intervalo, o próximo próximo processo é colocado em execução.

- SCHED-NORMAL: Este é o padrão para todos os processos que não são de tempo real. Neste modo de funcionamento, o próximo processo a ser executado é aquele que tiver recebido, por um número maior de vezes, uma negativa ao pedir pelo uso da CPU.

Para que se consiga executar diversas *threads* ao mesmo tempo e garantir as temporizações corretas para a distribuição dos vídeos é necessário dividir o tempo de CPU adequadamente, dando a cada uma delas uma parte deste tempo. Para isto é necessário que coloquemos o sistema em um dos modos de tempo real. E dessa forma não necessitamos utilizar o modo kernel, ou seja, nosso sistema pode funcionar em modo usuário (super usuário). Assim o componente de distribuição de vídeo implementado neste trabalho, adota a política de escalonamento *SCHED-FIFO* como será visto no Capítulo 4.

2.2 Sistemas de Distribuição de Mídia Contínua

Os sistemas de distribuição de mídia podem ser de três tipos principais.

O primeiro é o *download* comum. O cliente entra em determinado site na Internet e seleciona o conteúdo que deseja, e têm que baixar o filme até o final para que se inicie a exibição com QdS.

Outro tipo comum de distribuição é o chamado *pseudo-streaming*, que ao contrário do que ocorre no caso anterior, o cliente inicia a conexão e não necessita aguardar que o conteúdo inteiro seja pré-carregado para que comece a exibí-lo mas não garante QdS.

O terceiro caso, conhecido como VsD e chamado técnicamente de IVoD – Interactive Video on Demand, ao contrário do que ocorre nos anteriores, o usuário possui operações de vídeo cassete, como avanço, pausa e recuo, e como no caso anterior somente necessita completar um *buffer* de poucos segundos para começar a exibição do conteúdo, com garantia de QdS.

2.3 Componentes Fundamentais de Sistema de VsD

Explicamos a seguir, a função de cada um dos componentes de um sistema de VsD e suas características principais. Na Figura 2.1 apresentamos uma arquitetura genérica para um sistema de vídeo sob demanda em larga escala, que consistem em um servidor, *proxies* nas bordas da rede e os clientes.

2.3.1 Servidores

Os servidores são os componentes responsáveis pelo armazenamento dos vídeos e seu envio para os clientes ou *proxies*. Normalmente são compostos por uma ou diversas máquinas, cada uma podendo possuir um ou vários discos rígidos, tendo portanto uma capacidade bem elevada [23].

Eles podem funcionar em dois modos fundamentais : *push* e *pull* [18]. A diferença entre os dois modos de funcionamento está no controle necessário para envio e recebimento do vídeo, ou seja, na interação necessária entre o cliente e o servidor para que seja iniciada a transmissão do fluxo de vídeo.

No modo *pull* a granularidade de obtenção dos pacotes de vídeo é menor que no *push* e em conseqüência disto a QdS é potencialmente maior, assim como a facilidade de implementação. Ou seja, no modo *pull*, tendo o cliente um controle maior sobre o fluxo,

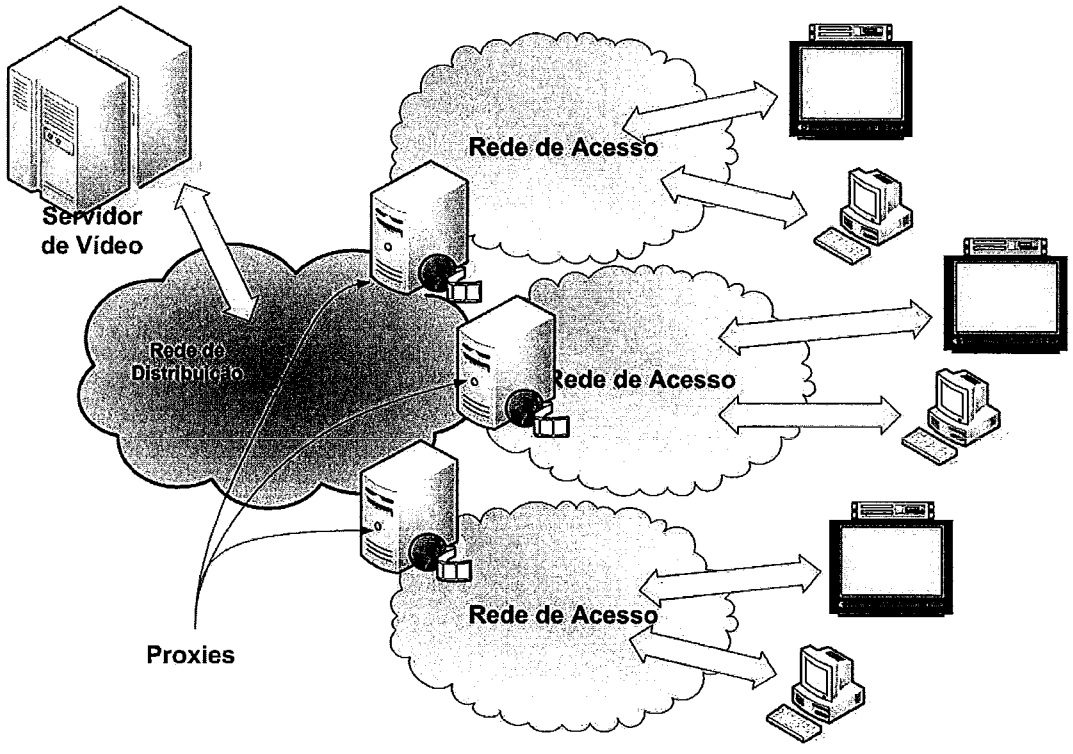

Figura 2.1: Arquitetura genérica de um sistema de vídeo sob demanda em larga escala

fica mais fácil lidar com variações no seu *buffer*, bem como com as possíveis perdas de pacotes. Neste modo o cliente busca os pacotes de vídeo no servidor. No entanto a escalabilidade não é boa, ou seja, a maior interação entre os componentes do sistema o torna menos escalável.

No modo *push* ocorre o contrário. Neste modo o servidor aguarda por um pedido do cliente e em seguida envia os pacotes na taxa necessária, sem que o cliente necessite fazer pedidos para cada uma das partes do vídeo. Com um menor número de pacotes de controle trafegando entre os componentes, facilíta-se o trabalho do servidor, que pode antever as taxas de distribuição do conteúdo, dividindo e controlando os recursos do sistema com maior precisão. Isso faz com que seja mais fácil fazer uma implementação escalável neste modo do que no descrito anteriormente, no entanto com potencial menor de QdS.

2.3.2 Proxies

O tipo de *proxy* atualmente mais utilizado é o denominado *proxy* com conteúdo estático [24]. Estes são utilizados principalmente para servidor de páginas *web*, e neste caso as páginas mais acessadas têm seu conteúdo totalmente mantido na *cache*. Como os vídeos

são, normalmente, muito grandes enchem rapidamente a *cache* estática [19]. Utilizando *cache* dinâmica, o vídeo é dividido em diversas partes e cada uma delas pode ser mantida em *cache* independentemente das outras.

Os *proxies* são responsáveis pelo gerenciamento do conteúdo que deve ser mantido em *cache* para uso futuro. Eles aguardam por pedidos de vídeo feitos por um determinado cliente, requisitando-o em seguida ao servidor se for necessário. Em seguida tratam de enviá-lo ao clientes e manter na *cache* aquilo que é mais acessado ou tem maior probabilidade de ser acessado no futuro. Assim, os principais objetivos da utilização de *proxies* em um sistema de vídeo sob demanda são [23]:

– aumento da capacidade do servidor: ao invés de servir diretamente os milhares de usuários do sistema com vários fluxos, são necessários apenas alguns poucos fluxos para servir os *proxies*.

– redução da latência de início de exibição: como os *proxies* são colocados nas bordas da rede de comunicação, para aumentar a sua eficiência, os clientes estão mais próximos dos mesmos do que do servidor. Assim, para a parte do conteúdo que é mais popular, o início de exibição é muito mais rápido, já que o *proxy* deve manter em sua memória a parte mais popular do acervo do servidor.

– redução da largura de banda necessária entre o servidor e os clientes: além de aliviar a carga no servidor, a utilização de *proxies* faz com que seja necessária uma largura de banda menor entre o servidor e os clientes, já que, mantendo em sua memória as partes mais acessadas dos vídeos, o *proxy* tem menor necessidade de acessar o servidor requisitando este conteúdo.

Na topologia utilizada normalmente (Figura 2.2), é contratada uma *Content Distribution Network* (CDN) e nas suas bordas são colocados os *proxies* (que na figura são chamados de distribuidor), como mostrado.

2.3.3 Clientes

Os clientes têm a função de requisitar os vídeos, armazená-los em um *buffer* local, decodificá-los e exibí-los na tela para o usuário. Sob este aspecto, o processo é extremamente simples, o cliente cria um *buffer* temporário onde armazena uma pequena parte do conteúdo recebido e evita assim os efeitos causados pela latência e o *jitter* da rede, pagando o preço de ter que aguardar um tempo inicial de bufferização.

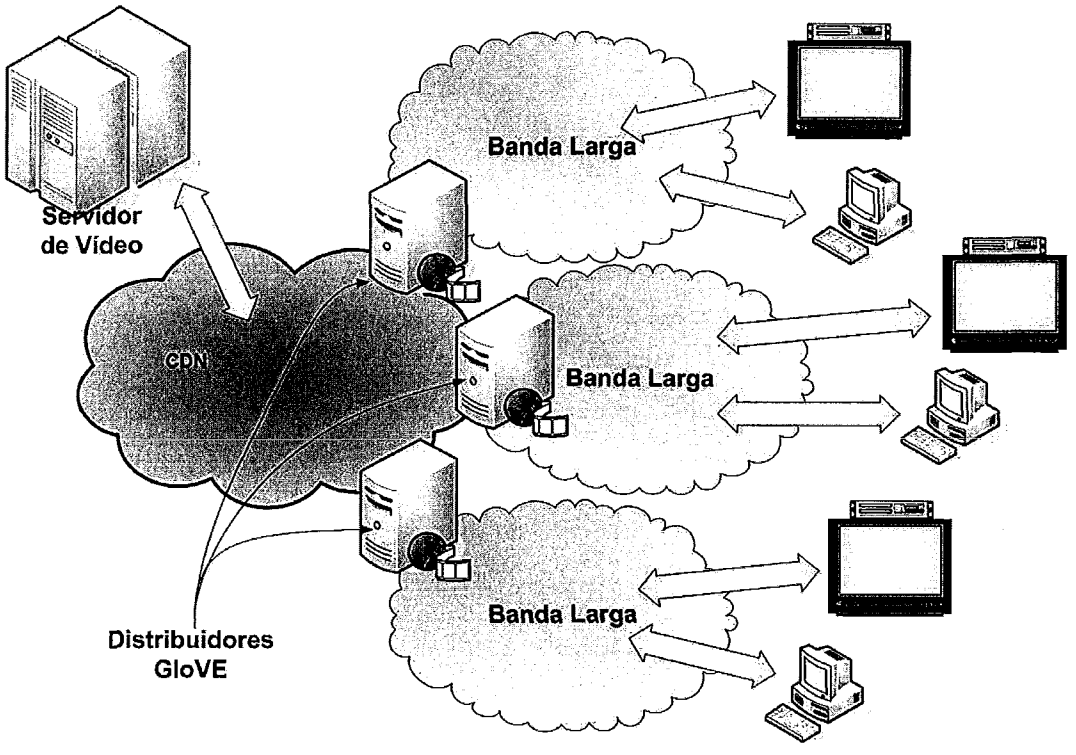

Figura 2.2: Sistema de vídeo em uma CDN

2.3.4 Rede de Conexão entre os *Proxies* e os Servidores

Na rede que conecta os *proxies* aos servidores deve haver alguma QdS que garanta uma largura de banda suficiente tal que possamos estipular o número máximo de fluxos entre os servidores e os *proxies* e dessa forma poder limitar o número de clientes no sistema quando estivermos próximos deste limite e assim garantirmos qualidade na exibição dos vídeos para aqueles clientes que já estiverem sido admitidos previamente ao sistema.

Dessa forma são necessárias as chamadas redes de distribuição de conteúdo (CDN) que são redes implementadas especificamente com o objetivo de possibilitar que seus usuários tenham melhor QdS na recepção do fluxo de vídeo, o que não ocorre quando os usuários possuem uma conexão de Internet comum. Ou seja, para disponibilizar fluxos de vídeo para um grupo de usuários pode-se contratar um serviço que estabeleça um conexão entre o servidor de vídeo e os clientes, evitando ou limitando desta forma problemas de congestionamento, atraso, perda de pacotes, ataques ao servidor e etc.

2.3.5 Redes de Acesso à Internet

Atualmente nas redes de acesso à Internet podemos encontrar, na última milha, alguns tipos principais de tecnologias de comunicação. A mais utilizada atualmente no Brasil , é a tecnologia *Digital Subscriber Line* (DSL) enquanto que nos Estados Unidos são as redes *Hybrid Fiber-Coax* (HFC).

Podemos encontrar ainda as conexões via cabo coaxial, originalmente utilizados para a transmissão de canais de TV unidirecionais e atualmente tendo sua rede reformulada para a transmissão bidirecional, com a conseqüente possibilidade de transmissão de dados (downloads e uploads). Existe ainda a possibilidade de se alugar um plano de fibra ótica (*Fiber-To-The-Home* (FTTH)) ou obtermos uma conexão sem fio através do *Worldwide Interoperability for Microwave Access* (WiMAX). Outro tipo de conexão ainda pouco utilizada e em desenvolvimento se dá através das redes elétricas.

Tanto no Brasil como no mundo as duas formas mais comuns encontradas são as redes DSL e as redes via cabo coaxial. Assim descreveremos nas seguintes subseções a arquitetura de cada uma delas.

2.3.5.1 Redes ADSL

As redes ADSL se caracterizam por serem assimétricas, ou seja, ou a largura de banda de descida é maior do que a de subida. A topologia consiste em vários concentradores de acesso (*Digital Subscriber Line Access Multiplexer* (DSLAM)), que se encontram localizados na última milha em direção ao usuário. O DSLAM tem a função de multiplexar a voz e os dados e transmitir o sinal para os clientes através de cabos telefônicos comuns. Outro componente é o *Broadband Remote Acess Server* (BRAS) que tem a função de autenticar os clientes e assim liberar os seus respectivos endereços *Internet Protocol* (IP) para acesso à Internet. A topologia é mostrada na Figura 2.3 [9] [15] [25].

2.3.5.2 Redes HFC

A tecnologia que prevalece nas redes HFC é a chamada *Data Over Cable Service Specification* (DOCSIS). Esta tecnologia especifica os métodos de modulação, sincronização, multiplexação, acesso ao meio, segurança, resolução de colisões e etc [27].

Na Figura 2.4 podemos observar a topologia deste tipo de rede. O concentrador de acesso (conhecido como *headend*) recebe tanto os dados da Internet, como os próprios sinais de TV e de servidores internos. Ele multiplexa os sinais e envia por uma rede de fibra ótica que é convertido para cabo coaxial e enviado para as todas as residências

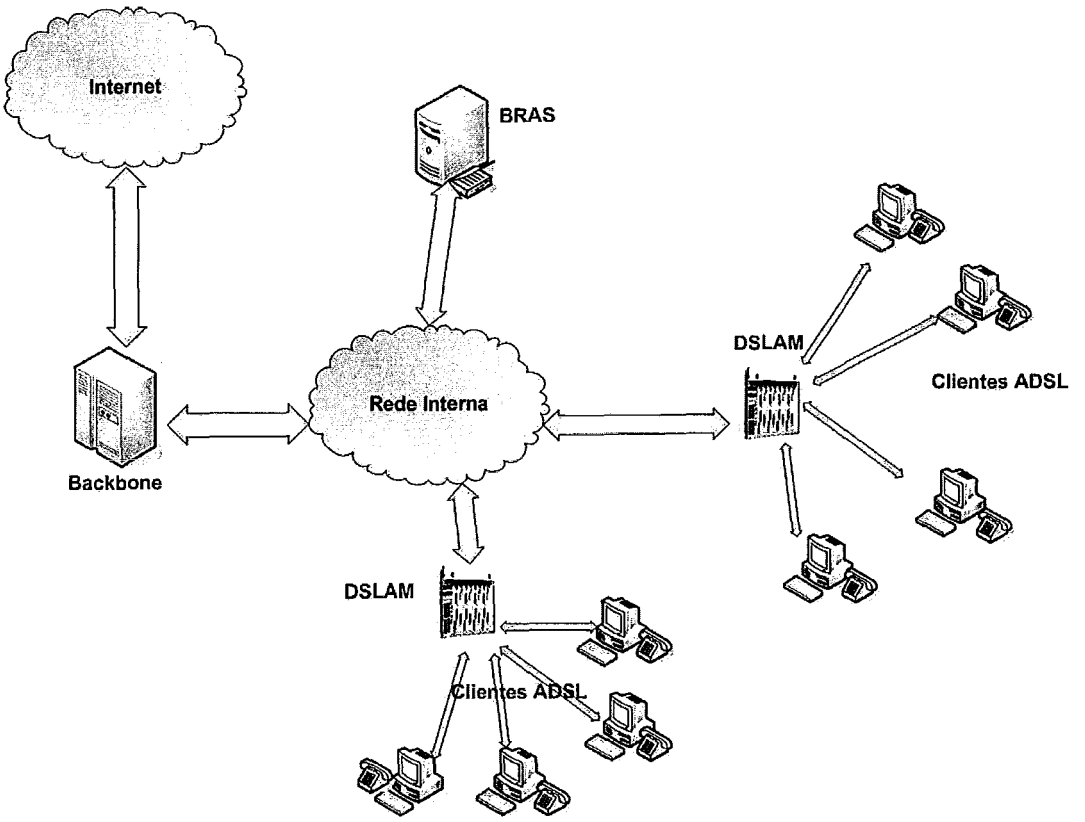

Figura 2.3: Topologia de uma rede ADSL

passando por amplificadores, divisores e etc. Chegando até o domicílio do usuário o sinal pode ser conectado a um computador pessoal, passando por um modem ou ligado diretamente à TV, passando ou não por um decodificador de canais. Em toda a rede são necessários amplificadores para reforçar a qualidade do sinal, que degrada com a distância [1].

As redes via cabo coaxial para transmissão de dados surgiram para agregar valor à infra-estrutura montada para a transmissão dos sinais de TV a cabo. O maior problema encontrado na reestruturação da rede para transmissão de dados, foi o fato das redes de TV a cabo terem sido criadas somente com intuito de enviar o sinal para os clientes. Assim além de parte da infra-estrutura de rede ter que ser trocada ainda foi necessário que se pensasse no protocolo de colisão que seria utilizado para que os clientes enviassem seu sinal em direção ao concentrador de acesso [15].

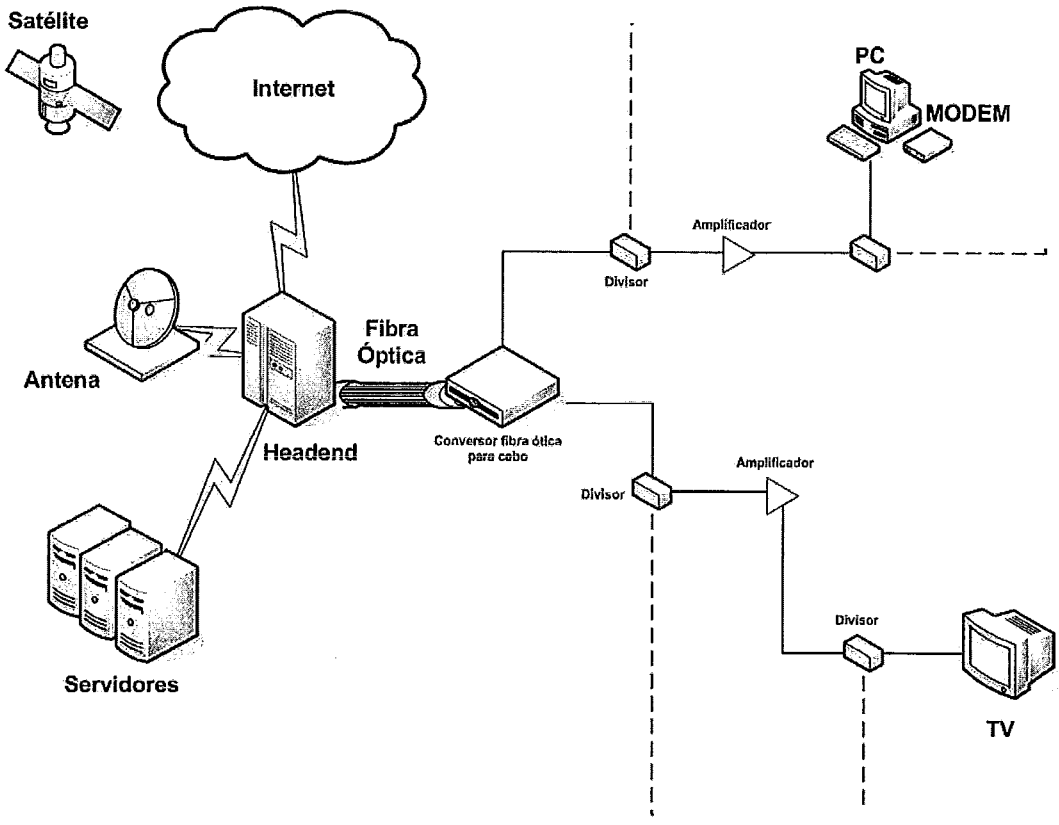

Figura 2.4: Topologia de uma rede HFC

2.3.6 Mídia

Normalmente os filmes são compostos de três tipos de dados principais: o vídeo, o áudio e as legendas. Quando comparamos a necessidade de banda para se transmitir um vídeo de boa qualidade observamos que esta é muito maior do que a necessária para transmitir tanto o áudio como as legendas. Um filme pode ser caracterizado por dois parâmetros principais: o primeiro, o seu tamanho, um vídeo pode ser curto, como um pequeno trailer de 2 minutos ou um longa metragem de 2 horas; o segundo, a variabilidade de sua taxa de compressão: sua taxa pode ser constante (*Constant Bit Rate* (CBR)), variável (*Variable Bit Rate* (VBR)), além de outras variações de ambas [26].

O áudio pode também ter taxa constante ou variável, no entanto, o mais comum é codificar o áudio em taxa constante. Pode haver ainda filmes com mais de um áudio, no entanto, isto não é recomendável para streaming já que aumentará a banda de transmissão necessária para enviarmos uma seqüencia de dados que nem todos os clientes utilizarão. O melhor é que o próprio cliente selecione o seu áudio antes do início da exibição.

As legendas ocupam muito menos espaço do que os outros dois componentes mesmo

quando várias são acrescentadas, já que são somente seqüências de caracteres com *times-tamps* que definem o momento de exibição das mesmas.

Os três componentes mencionados são multiplexados em algum formato conhecido. Devemos saber definir bem o multiplexador, já que alguns deles acrescentam uma quantidade grande de dados, aumentando assim a taxa total.

O que ocorre é que quando desejamos editar um filme o melhor é que todos os *key-frames* estejam indexados, ou seja, que exista um ponteiro para a posição de cada um deles o filme, e assim facilitar e aumentar a velocidade de busca das informações. O caso do vídeo estar sendo transmitido em vídeo sob demanda não existe esta necessidade e podemos poupar o espaço e banda necessários para transmissão desses índices. Existem até multiplexadores que, para poupar tempo na multiplexação, colocam estes índices no fim do vídeo, o que torna impossível o envio pela rede. Utilizando essas características da mídia podemos definir e configurar tanto o software como o hardware necessário para servir este conteúdo.

Como visto anteriormente, as redes de acesso disponíveis atualmente fornecem uma banda de descida limitada (*download*), e assim precisamos saber exatamente a taxa média do vídeo para que possamos distribuí-lo sem que haja *overflow* ou *undeflow* nos *buffers* dos clientes. Existem duas formas de verificarmos se uma determinada mídia esta no padrão que desejamos. A primeira delas é escolhendo o codificador desejado e recodificando a mídia escolhida com os parâmetros desejados. A segunda é através de uma rápida análise de toda seqüencia de dados. No caso da mídia estar sendo distribuída ao vivo, o codificador deve ser configurado adequadamente.

2.4 Protocolos de Comunicação

Quando pensamos nos protocolos de comunicação fim-a-fim que devemos utilizar em nosso sistema, pensamos logo nos protocolos largamente utilizados na Internet : *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP).

2.4.1 TCP

O TCP é o protocolo mais utilizado pelos aplicativos ligados à Internet atualmente, como por exemplo o *File Transfer Protocol* (FTP), o *Teletype Network* (TELNET) e etc. Este protocolo possui controle de erros, de fluxo e de congestionamento. Dessa forma, quando o utilizamos, simplificamos nossa aplicação, deixando por conta dele a parte mais com-

plexa da comunicação. No entanto, será que conseguimos transmitir vídeo utilizando este protocolo? Pensamos que isto depende de diversos fatores. Como por exemplo: a largura de banda disponível para cada um dos clientes quando comparada a taxa em que o vídeo deve ser transmitido, se a banda disponível e a taxa forem próximas, certamente será improvável; a topologia da rede, que influenciará no trabalho do protocolo; a capacidade dos servidores, porque, provavelmente, para que consigamos o mesmo *throughput* do protocolo UDP com uma máquina simples, necessitaremos de uma máquina ou pouco mais poderosa para termos o mesmo desempenho com o TCP.

Apesar disto alguns sistemas utilizam somente um servidor web comum (*Hypertext Transfer Protocol* (HTTP)) para disponibilizar seus vídeos. Estes, certamente, terão problemas de escalabilidade. Se houver banda suficiente, para a distribuição do vídeo mesmo com a sobrecarga do protocolo, não só o servidor estará sendo sobrecarregado desnecessariamente, já que estará enviando os dados a uma taxa mais alta do que é realmente necessário, como o próprio cliente necessitará de uma máquina melhor equipada, com maior quantidade de memória para manter em um *buffer* o conteúdo recebido até que chegue o momento da exibição.

Um dos maiores problemas do HTTP é que ele não mantém o estado das requisições, ou seja, cada requisição que chega não é tratada como seqüencia de uma anterior e dessa forma não há como priorizar ou temporizar uma sessão ou outra.

2.4.2 UDP

O protocolo UDP, por ser muito mais simples, não sofre destes problemas encontrados no TCP, ou sejam este protocolo não tem controle de fluxo, de congestionamento e ainda não possui nenhum controle de erros. Algumas vezes temos que lidar com este tipo de controle na distribuição de vídeo, no entanto, nem sempre, e dessa forma não necessitamos arcar com o custo necessário ao TCP para fazer este tipo de trabalho que é pago no tempo que perdemos e no consumo de CPU. Com a utilização deste protocolo podemos controlar a banda disponível tanto do lado do servidor como do lado dos clientes no nível da aplicação, além de poder tratar, quando for necessário, perdas de pacotes ocorridas na rede, nos *buffers* dos *sockets* do sistema operacional ou mesmo dentro da nossa própria aplicação.

2.4.3 Protocolos Mistos

Existem alguns protocolos que foram feitos especialmente para o controle e transporte de mídia contínua. O mais comum é utilizar protocolos mistos com ambos TCP e UDP.

O TCP geralmente é usado na parte de controle do fluxo, onde são enviados pacotes requisitando o início da exibição, um avanço, retrocesso e etc. Como neste caso os pacotes são pequenos e as taxas bem menores do que a do vídeo não existem grandes problemas na utilização da banda disponível, pelo menos por parte do cliente. Para o envio dos pacotes de vídeo propriamente ditos é utilizado o UDP.

Citamos abaixo alguns dos protocolos mais conhecidos e utilizados atualmente.

RTP

O *Real Time Transport Protocol* (RTP) [20] foi criado justamente para realizar o transporte de áudio e vídeo em aplicações de tempo real. Este protocolo foi proposto de maneira que ele fosse independente do protocolo de camada mais baixa, no entanto encontramos com mais freqüência o protocolo implementado através de sockets UDP. O RTP multiplexa fluxos em diferentes portas UDP e suporta tanto a distribuição unicast como multicasting.

Para dar a implementação do envio de dados em tempo real, o RTP inclui em seu cabeçalho informações sobre o tempo de exibição do conteúdo (*timestamping*). No entanto ele não é reponsável pela sincronização dos pacotes de áudio e vídeo. Os *timestamps* devem ser utilizados pela aplicação para fazê-lo.

O RTP numera seus pacotes para que seja determinada a ordem dos mesmos e ainda facilite a detecção da ocorrência de perda de pacotes.

Em seu pacote encontramos ainda informações sobre os codificadores utilizados, tanto pelo áudio quanto pelo vídeo e informações sobre a fonte que enviou os dados.

RTCP

Junto com o RTP, também foi definido o *Real Time Control Protocol* (RTCP) [20], com intuito de trabalhar em conjunto com ele. Periodicamente são trocados pacotes RTCP para que ambos comunicadores tenham informações sobre a qualidade do envio dos dados, como o número de pacotes perdidos, número do último pacotes recebido, jitter

além de informações sobre o momento de envio dos pacotes (*timestamp*) para que possa ser calculado o atraso. A quantidade de pacotes RTCP gerada é limitado pelo RTP a um máximo de 5% do tráfego evitando assim que o mesmo ocupe toda a banda disponível.

Devemos notar que nem mesmo o conjunto RTP/RTCP não garante QdS, e dessa forma não reserva recursos de rede nem de CPU para determinado fim, nem faz o controle dos erros ocorridos por perdas de pacotes e dessa forma deixa para que a aplicações de nível superior façam este tratamento, ou seja, o conjunto dos protocolos não dá nenhum tratamento ao ocorrido, somente informa à camada de nível superior o que ocorreu.

RTSP

O *Real Time Streaming Protocol* (RTSP) [21] é um protocol que define as operações de controle remoto como *play*, *pause*, *resume*, *forward*, *rewind* e etc. Foi desenvolvido para trabalhar em conjunto com os outros protocolos de mais baixo nível como o RTP – e dessa forma também com o RTCP. Ele pode controlar tanto um único fluxo, como também vários fluxo. Além das operações de obtenção de fluxo mencionadas, o protocolo também define a adição de um novo conteúdo ao servidor e suporta ainda convites a conferência no qual são definidos dois tipos de mensagens, o convite à visualização e o a gravação.

No RTSP cada fluxo é representado por uma *Uniform Resource Locator* (URL). As propriedades do mesmo são identificadas a partir de um arquivo que informa sobre os *codecs* utilizados, a linguagem, o endereço IP e porta onde o conteúdo pode ser obtido, além de outros parâmetros.

Microsoft

A Microsoft possui um conjunto de protocolos, tanto aqueles baseados em HTTP (Windows Media Services HTTP streaming protocol extensions), que simplesmente estendem o funcionamento do HTTP para que o mesmo fique mais eficiente na transmissão dos arquivos, como outros baseados no padrão RTSP e até uma solução própria *Microsoft Media Server Protocol* (MMS) que está caindo em desuso.

Os problemas encontrados na utilização desses protocolos são os inerentes aos protocolos proprietários, como a falta de documentação adequada e aprofundada sobre os mesmos, e a falta do código fonte. Dessa forma se quisermos incluir tais protocolos em

nosso sistema, temos que utilizar todos os componentes da solução Microsoft para Windows.

Observamos que a tendência atual é a convergência dos principais protocolos proprietários para o padrão RTSP com, por exemplo, a descontinuação em alguns protocolos proprietários da Microsoft e a criação de novos protocolos indo em direção do padrão definido como sendo o RTSP, como também descreve a definição do protocol de transporte da Real Networks chamado *Real Networks Data Transport* (RDT) [10].

Obviamente, não adianta que todas as empresas utilizem implementações próprias do RTSP se elas não forem compatíveis byte a byte. Sendo compatíveis, teoricamente, poderíamos utilizar clientes diferentes em servidores diferentes e vice-versa.

2.5 Desafios no Desenvolvimento

Para que pudéssemos implementar o sistema tivemos que descobrir inicialmente quais eram os principais problemas e desafios que tínhamos pela frente. Existem diversos problemas que temos que superar para garantir QdS a um cliente de vídeo sob demanda. Descrevemos em seguida vários aspectos que precisamos garantir para que exista uma boa QdS.

2.5.1 Continuidade

Um deles é o problema da continuidade, ou seja, uma vez iniciada a exibição, o cliente não deve perceber nenhuma variação na rede e em conseqüência o vídeo deve ser exibido continuamente sem que haja interrupções.

Apesar de as mídias serem codificadas em modo CBR quando queremos transmitir com a máxima qualidade possível com uma banda disponível fixa para os clientes, o que ocorre na prática é que as taxas dos vídeos não são realmente constantes, sempre havendo alguma variação em torno da mesma. Além disso, dependendo da carga em cima do sistema, a capacidade de E/S do sistema pode variar, tanto para acesso aos discos rígidos como à própria rede, além do que com os sistemas operacionais atuais temos que estar sempre atentos ao fato de que as aplicações estão compartilhando os mesmos recursos e em um sistema de vídeo sob demanda devemos atender aos clientes em tempo real, ou seja, não deve haver atrasos na entrega dos pacotes. No caso da mídia ser VBR estes problemas aumentam ainda mais.

2.5.2 Eficiência

Com o alto custo de todos os componentes de *hardware* e ainda dos *links* dedicados contratados para prover uma solução com uma baixa relação custo/benefício, devemos tentar economizar ao máximo todos os recursos do sistema, ou seja, temos que utilizar os equipamentos e infra-estrutura de rede de maneira mais eficiente possível, de forma a prover vídeo sob demanda para o maior número de clientes possível com a menor quantidade de recursos.

2.5.3 Escalabilidade

Em um sistema de vídeo sob demanda se conseguimos aumentar a capacidade do sistema para atender novos clientes através do acréscimo de novas máquinas então o sistema é considerado escalável. Ou seja, se ao investir novos recursos aumentamos em maior razão o número de clientes (crescimento sub-linear), o sistema é considerado escalável. Ou seja, se duplicando o número de recursos diponíveis, mais que duplicarmos o número de clientes, então o sistema é considerado escalável.

2.5.4 Confiabilidade

Outro problema encontrado no desenvolvimento de sistemas de vídeo sob demanda é a confiabilidade do sistema. Para que um sistema tenha confiabilidade é necessário que qualquer falha de hardware seja detectada e rapidamente este recurso que quebrou seja substituído pelo próprio sistema com componentes extras, previamente alocados para este fim. Dessa forma podemos garantir a QdS do sistema, já que os clientes conectados não perceberão o ocorrido. Por exemplo, se um servidor falhar, devemos ter outra máquina que assuma o lugar da primeira através da verificação de que a mesma parou. Podemos, por exemplo, utilizar harddisks configurados em *Redundant Array of Independent Disks* (RAID) 5 e suportar possíveis falhas nos mesmos.

2.5.5 Qualidade de Serviço

Um outro importante tópico, muito desafiador está em obter QdS no sistema em redes TCP/IP, uma vez que este foi projetado para suportar apenas o serviço de melhor esforço.

Para dizermos que um sistema de vídeo sob demanda possui QdS, ele deve ter algumas características, dentre elas podemos destacar as seguintes: a garantia de que os pacotes enviados pela rede chegarão, e mais importante, que chegarão a tempo de serem exibidos,

e quando não chegarem, serão repostos se necessário, ainda em tempo de serem exibidos; a garantia de que uma vez que o cliente foi incluído no sistema, ele assistirá seu fluxo até o final, mesmo que a carga em cima do sistema seja alterada, assim como as condições da rede.

Atualmente a Internet não garante QdS, uma vez que sua infra-estrutura não suporta serviços diferenciados ou integrados. No entanto, uma forma de melhorar a QdS de uma rede de melhor esforço é através de uma CDN, que procura através de *proxies* na borda da rede suprir as deficiências da rede de transmissão de dados.

Capítulo 3

Cache de Vídeo

O intuito deste capítulo é descrever os principais pontos que levaram à criação da CVCC em trabalho doutorado proposto anteriormente [13]. A CVCC é uma evolução da *Cache de Vídeo Cooperativa* (CVC). Para que se entenda esta evolução apresentamos uma visão geral da CVC e em seguida é visto o funcionamento da CVCC, apresentando em seguida os resultados conseguidos por simulação no trabalho em que a mesma foi proposta.

3.1 Cache de Vídeo Cooperativa

Com intuito de diminuir a carga sobre o servidor e aliviar a rede, aumentando assim a escalabilidade do sistema, foram propostas diversas técnicas e políticas de reutilização de fluxos, seja através do encadeamento de fluxos de clientes (*Chaining* [22]), com recepção ou não de remendos do servidor (*Patching* [11]) e através do agrupamento de diversos clientes por lote com a subseqüente utilização de *multicasting* para atendê-los (*Batching* [7]).

Apesar de todos esses esquemas terem sido propostos, nenhum explorou as memórias disponíveis nos clientes. A CVC faz justamente isso. Ela agrega as memórias dos *buffers* de todos os clientes, diminuindo assim a necessidade de acesso ao servidor, tornando o sistema mais escalável.

Na Figura 3.1 podemos observar o funcionamento básico da CVC. Em um instante inicial o primeiro cliente chega ao sistema e recebe o fluxo do servidor e segue armazenando o conteúdo em seu *buffer* local. Em seguida outros clientes vão sendo incluídos ao sistema, e como o conteúdo buscado já estava anteriormente disponível nos *buffers* locais dos clientes, não é necessário recorrer ao servidor. O fluxos são enviados diretamente de uns clientes para outros.

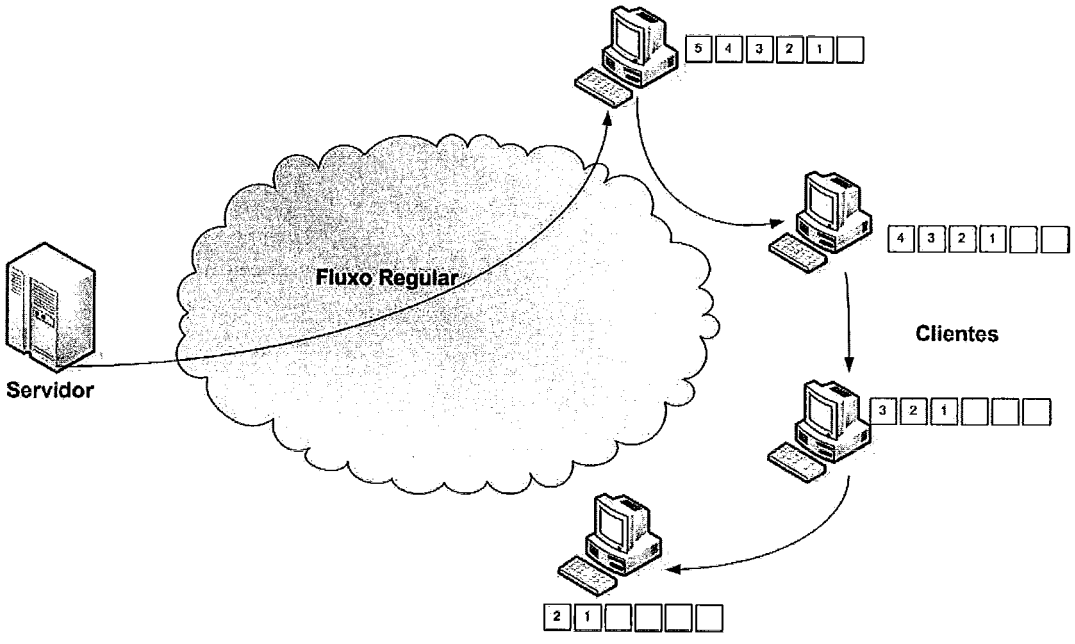

Figura 3.1: Cache de Vídeo Cooperativa (CVC)

3.2 Cache de Vídeo Cooperativa Colapsada

Em trabalhos desenvolvidos anteriormente foram mostrados alguns problemas do sistema CVCC. Como principal desvantagem foi destacado o tráfego gerado entre os clientes peer-to-peer que sobrecarregava a rede, sendo somente viável em redes *Local Area Network* (LAN) onde os recursos de rede disponíveis são mais abundantes.

Atualmente a Internet é disponibilizada aos clientes por meio de redes ADSL ou HFC, quase sempre na forma assimétrica, em áreas geograficamente muito maiores do que em uma LAN. Dessa forma era inviável a utilização do sistema de CVC em redes *Wide Area Network* (WAN). Por isso foi proposta a técnica de CVCC, na qual foi conseguida a agregação dos conceitos da CVCC de maneira que fosse possível seu uso em redes maiores.

Os princípios da CVC são utilizados na implementação da CVCC, tentando preservar a sua vantagem de escalabilidade e eliminando as suas principais desvantagens. Pode-se citar, por exemplo, a sua dependência no hardware, drivers e sistema operacional dos clientes que não podem ser controlados e certamente interferirão na QdS da distribuição do vídeo, que são as desvantagens.

De maneira similar à CVC os *buffers* de cada um dos clientes são agregados, mas agora em uma memória única, em um componente que chamaremos de Distribuidor (*Proxy*).

Com os *buffers* dos clientes concentrados no Distribuidor, evita-se o uso da dorsal da rede no compartilhamento do fluxo de vídeo entre os clientes, havendo, é claro, uma distribuição de diversos Distribuidores em pontos estratégicos da rede.

Assim, para que os Distribuidores sejam alimentados pelo Servidor surge a necessidade de um ambiente propício entre os mesmos, ou seja, uma rede especializada com banda suficiente disponível e baixa latência, que com visto anteriormente, são chamadas de rede de distribuição de conteúdo.

3.2.1 Arquitetura da Cache de Vídeo Coopetativa Colapsada

Em abordagens anteriores à CVCC foram utilizados objetos separados, que representam cada um, o *buffer* de um cliente. Cada *buffer* é implementado com um vetor circular. Essa abordagem dificulta o gerenciamento de memória e pode ser ineficiente, utilizando memória extra para representar um mesmo conteúdo em *buffers* diferentes.

Assim, para que sejam gerenciados os *buffers* dos clientes colapsados no Distribuidor, serão utilizadas duas estruturas englobando todos os *buffers* colapsados dos clientes. As duas estruturas utilizadas são dois vetores circulares: o vetor de vídeo e o vetor de *slots* (ver Figura 3.2). Um *slot* nada mais é do que uma associação a um determinado intervalo de tempo do vídeo. Na Figura 3.2 observamos um *buffer* colapsado inserido no vetor de *slots* e o vetor de vídeo com intervalos numerados de 1 a N.

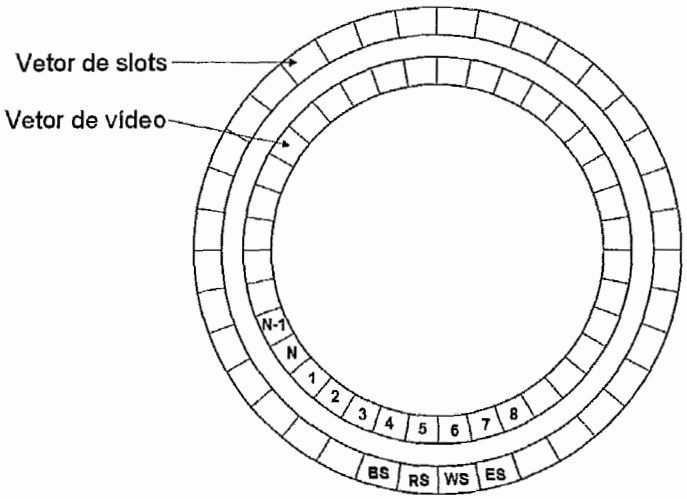

Figura 3.2: Vetor de slots e vetor de vídeo

O vetor de vídeo é a estrutura utilizada para gerenciar as unidades de vídeo. Uma unidade de vídeo é a estrutura mínima que pode ser manipulada. Cada unidade do vetor de vídeo possui um ponteiro para a unidade de vídeo correspondente além do *timestamp* correspondente.

O vetor de *slots* é a estrutura utilizada para gerenciamento dos *buffers* colapsados. Cada *buffer* colapsado pode ser dividido em quatro partes principais, BS, RS, WS e ES (ver Figura 3.3); cada uma delas limitada por um ponteiro. Dois ponteiros indicam o início (B) e o fim do *slot* (E). Dois outros ponteiros indicam onde estão sendo feitas a leitura (R) e escrita (W) em um determinado momento. Outros dois indicam o risco de *Overflow* (O) e *Underflow* (U). Os *slots* que pertencem a um *buffer* colapsado não têm sua prioridade alterada e não podem ser descartados, já que serão enviados. Três prioridades indicam o estado das unidades de vídeo, com prioridade 0 são marcadas aquelas que não estão presentes na cache, com 1 aquelas que estão presentes na cache mas não estão reservadas e com prioridade 2 são marcadas aquelas unidades que estão reservadas. Na medida em que o vídeo vai sendo enviado o vetor de *slots* gira por cima do vetor de vídeo na mesma velocidade alterando as prioridades das unidades de vídeo que entram e que saem dos *buffers* colapsados.

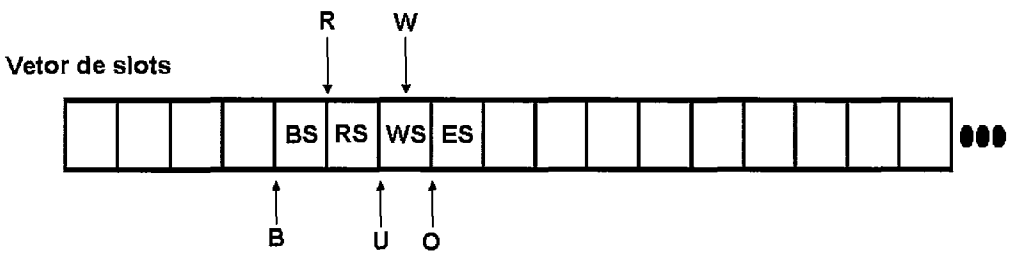

Figura 3.3: Estrutura de um buffer colapsado

Utilizando estas estruturas precisamos saber como serão feitas as manipulações de alocação e desalocação dos *buffers* e ainda a manipulação dos ponteiros dentro do *buffer*. Para que isto seja conseguido precisamos de um método para descobrir em qualquer instante que parte do vídeo um determinado *slot* está mapeando. A CVCC utiliza os time stamps do vídeo para determinar a qual *slot* uma determinada unidade de vídeo pertence. Assim, quando um pedido é recebido inicialmente a CVCC marca o tempo de início (ST). Se um pedido chega em um tempo CT requisitando que o primeira unidade de vídeo a ser enviada seja a que possui time stamp UTS, precisamos calcular qual é o *slot* RS para que possamos começar a enviar o vídeo. Os outros *slots* (BS,WS e ES) podem

ter sua posição calculada a partir de RS. Sendo SL o tamanho de cada *slot* e NS o número de *slots* podemos calcular RS como mostrado em 3.1.

$$RS = \left(\frac{CT - ST + UTS}{SL}\right) modNS \tag{3.1}$$

Dados RS da Equação 3.1, ST, CT e sendo PT o período de rotação que coincide com a duração do vídeo, pode-se calcular NC, o número de voltas dada pelo vetor de *slots* da pela expressão 3.2.

$$NC = \left[\frac{CT - ST - (RS * SL)}{PT}\right] \tag{3.2}$$

E o inicio do *slots* (IS) e término do *slot* (TS) são dados pelas equações 3.3 e 3.4.

$$IS = CT - ST - NC * PT - RS * SL \tag{3.3}$$

$$TS = IS - SL \tag{3.4}$$

3.2.2 Algoritmo de substituição

Com o agrupamento de *buffers* em uma única memória cria-se um problema de gerenciamento que não havia antes. Anteriormente, na CVC, cada cliente gerenciava o seu próprio *buffer* que fazia parte de uma memória global que poderia ser utilizada por outros clientes. Esse *buffer* local do cliente não podia ser controlado pelo sistema, ou seja, ele dependia exclusivamente da vontade do usuário.

Cria-se então o conceito de *link slots* (ou *slots* de ligação), que nada mais são do que marcações feitas para que o conteúdo apontado pelos *slots* marcados desta maneira, não sejam descartados pelo sistema e dessa forma liguem dois *buffers* colapsados, podendo economizar um fluxo do servidor. Na Figura 3.4 podemos observar que em um instante T1

um usuário chegou ao sistema e para isso foi criado um *buffer* colapsado para comportá-lo. Alguns instantes depois (T2) um novo usuário chega ao sistema e da mesma forma outro *buffer* colapsado é criado, no entanto desta vez, como os dois *buffers* estão próximos, para evitar que o conteúdo entre eles seja descartado, os *slots* entre eles são marcados como *link slots* e desta forma têm seu conteúdo preservado, evitando assim a criação de um novo fluxo do servidor.

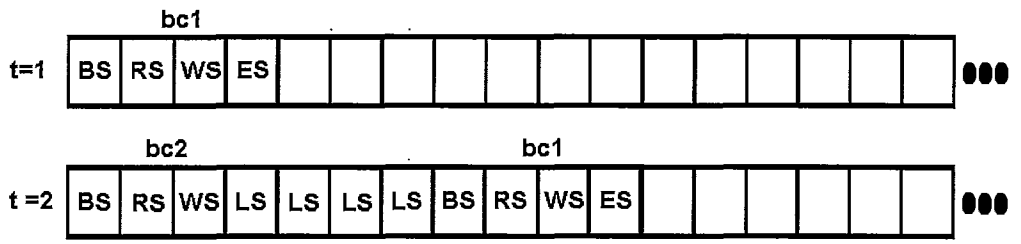

Figura 3.4: Link slots

Na CVCC, na medida em que novos clientes chegam podem ocorrer duas situações, ou o cliente é incluído em um *buffer* colapsado existente, ou é criado um novo *buffer* colapsado. Em algum momento se esgota a memória do distribuidor de vídeo, e conseqüentemente para que um novo *buffer* colapsado seja criado é necessário que alguma memória seja liberada através da liberação de *link slots*.

Assim foi criado o algoritmo de liberação de *link slots*, que é executado toda vez que a cache está cheia e necessita-se incluir um novo cliente no sistema. No algoritmo procura-se liberar a seqüencia de *link slots* cujo custo seja o menor possível, ou seja, a seqüencia escolhida deve ser trocada por um fluxo de vídeo do servidor que dure o mínimo possível, tendo o menor tamanho, e ainda esteja o mais próximo possível do final do vídeo, já que é menos provável que clientes comecem a assistir o vídeo pelo seu fim. A seqüencia liberada deve ser, obviamente, maior ou igual ao número de *slots* necessários.

O algoritmo é mostrado em 3.5. As variáveis utilizadas no algoritmo são as seguintes: f é o número de *slots* que precisam ser liberados para alocar espaço para inclusão de novo cliente, *queue* é uma fila de prioridades de seqüencias de *link slots* cuja prioridade é dada por 3.5, onde d é distância até o fim do vídeo e t é o tamanho da seqüencia de *link slots*.

$$F(d, f, t) = \frac{d}{min\,(f, t)} \tag{3.5}$$

```
getFreeSlot( f ) {
   freeSlots=0;

   while(freeSlots < t or queue.empty()==TRUE) {
      linkSlotSequence=queue.getFirst();
      freeSlots += linkSlotSequence.size;
      freeSlot(linkSlotSequence);
   }

   return freeSlots;
}
```

Figura 3.5: Algoritmo de substituição

3.2.3 Resultados da CVCC por simulação

No trabalho original [13] foram realizados experimentos utilizando o simulador NS-2 e comparados indiretamente os resultados obtidos com outros trabalhos realizados anteriormente.

As simulações foram feitas para três versões da CVCC. A LS- que não faz a utilização de *link slots*, a LS que utiliza *link slots* mas sem o uso da política de substituição e a LS+ que utiliza o algoritmo de substituição. A seguir faremos um breve resumo dos principais resultados obtidos para cada uma das políticas, com o intuito de comparar os resultados com os conseguidos a partir da implementação, que será descrita no próximo capítulo e avaliada em capítulo posterior.

3.2.3.1 Ambiente

Nesta simulação foram considerados os seguintes componentes: um servidor, um *proxy* e diversos clientes. A rede de comunicação entre o servidor e o *proxy* CVCC era uma CDN que era capaz de suportar 1000 fluxos simultâneos do vídeo escolhido para a simulação. O vídeo estava codificado no formato MPEG2 com uma taxa média de 8 Mbps e de 1 hora de duração. Cada um dos clientes era conectado ao *proxy* através de um enlace ADSL cuja largura de banda era suficiente para suportar o fluxo escolhido.

O intervalo de chegada entre clientes variou de 3.1 a 150 segundos.

O tamanho da cache foi variado de 10% a 100%.

Um teste inicial foi realizado somente com um vídeo e outro com 100 vídeos iguais ao primeiro. No último caso foi utilizada uma distribuição Zipf igual a 0, 0,271 e 1 para a escolha do vídeo, ou seja, para definicão da popularidade de cada um dos vídeos. É importante mencionar que apesar dos 100 vídeos serem iguais, a distribuição Zipf foi calculada tal que eles fossem diferentes, com a vantagem de não ter-se que gerar traces de 100 vídeos diferentes.

3.2.3.2 Métricas de desempenho

As principais métricas de desempenho utilizadas foram as seguintes:

Taxa de bloqueio : Percentagem de requisições não atendidas por falta de recursos do sistema.

Taxa de utilização : Taxa média de utilização do servidor.

Taxa de pico de utilização : é a maior taxa atingida entre o servidor e o *proxy*.

Latência : é o tempo médio de espera que o cliente se sujeita entre uma requisição do vídeo e o início da exibição.

3.2.3.3 Resultados

Tamanho dos slots

Para o tamanho dos *slots*, foi observado o tempo de 8 segundos como sendo o melhor valor. Neste caso foi obtida uma taxa de bloqueio quase igual a quando foram utilizados *slots* menores, que exigem maior processamento, e foi conseguida, obviamente, uma latência bem menor do que com *slots* de tamanho maior.

Tamanho do prefixo

Se chega a conclusão de que devemos prefixar os 32 *slots* iniciais de todos os vídeos. Desta forma tanto os vídeos impopulares quanto os populares manterão o prefixo em cache. Se nenhuma requisição chegar a prioridade de descarte é aumentada, e se futuramente for necessário, o descarte será iniciado pelos menos populares.

Capítulo 4

Sistema GloVE-WAN

Neste capítulo definimos o GloVE-WAN, nosso protótipo que implementa a CVCC e descrevemos seu funcionamento. Este nome tem origem em dissertação relacionada[16], onde foi criado um protótipo da CVC chamado *Global Video Environment for Local Area Networks* (GloVE-LAN). Iniciamos descrevendo as características principais do sistema e em seguida tratamos de algumas extensões. Posteriormente, fazemos algumas observações quanto aos problemas encontrados nas redes de acesso à Internet, dando ênfase às redes ADSL, na qual fizemos testes de campo junto a uma operadora de telecomunicações.

Na seqüência, descrevemos o protocolo que utilizamos. Apresentamos então o modelo utilizado no sistema, tratando do problema de escalonamento das threads, da recepção dos pacotes com controle de admissão de clientes, da obtenção dos pacotes de vídeo e ainda da distribuição dos mesmos.

4.1 Características do Sistema

O GloVE-WAN consiste em uma implementação da CVCC para redes de larga escala com redes de acesso de banda larga, em particular, ADSL.

O sistema é composto por um servidor que armazena os vídeos e os transfere para o distribuidor (*proxy*), que é o elementoresponsável pelo cacheamento do vídeo. Os clientes se comunicam com o distribuidor requisitando os vídeos. Na prática o que acontece é que temos um elemento único que pode ler dados do disco e enviar para o distribuidor, sendo neste caso chamado de servidor, ou pode obter os dados do servidor, gerenciá-los na cache e enviar para os clientes, sendo então chamado de distribuidor.

Assumimos que o enlace disponível entre o distribuidor e o servidor possui largura de banda conhecida e garantida durante todo tempo.

A rede de acesso conecta os clientes a um concentrador de acesso (DSLAM) que está localizado próximo ao distribuidor, o que evita o uso da rede até o servidor, diminui a latência dos pacotes até os clientes e possívelmente aumenta a banda disponível até os clientes, pois com o distribuidor próximo ao DSLAM o mesmo tem capacidade suficiente para atender usuários conectados a todas as portas.

Quanto a mídia, deve ser codificada, preferencialmente, em taxa constante, de acordo com o limite de banda disponível a cada um dos clientes. Se a taxa for variável o cliente necessitará esperar um tempo muito maior, para que possa iniciar a exibição sem que haja interrupções, por uma possível falta de blocos – um bloco corresponde a um conjunto de *bytes* fixo) – de vídeo em seu *buffer* local, lembrando que o distribuidor sempre transmite usando CBR, isto é, os vídeos VBR serão suavizados.

4.2 Extensões

Na implementação do protótipo da CVCC foram necessárias mudanças para adequar o sistema ao mundo real. As principais ocorreram tanto no vetor de vídeo com no vetor de *slots*.

Nos formatos gerados pelos codificadores de vídeo e áudio e, conseqüentemente, naqueles obtidos pela sua multiplexação, os quadros de vídeo e áudio são obtidos em tamanhos variáveis, mesmo que na configuração dos codificadores seja escolhida uma taxa constante. Isto se deve às características das imagens de um vídeo que variam de cenas simples a outras extremamente complexas. Assim, cada quadro é codificado com tamanhos diferentes uns dos outros e os menores são codificados parcialmente em função dos maiores.

Assim, como os tamanhos dos quadros não são fixos, teríamos que criar ferramentas de análise dos fluxos para dividí-los nos GoFs (*Group of Frames*). No vetor de vídeo proposto na *CVCC*, o mesmo é dividido em diversos *GoFs*. Na implementação dividimos o vetor de vídeo em diversos blocos de tamanho fixo, assim como foi feito em um trabalho realizado anteriormente [16].

No modelo original da *CVCC*, cada *slot* do vetor de *slots* possui um intervalo de tempo fixo. Assim dois *slots* com tamanho igual a 1 segundo podem possuir um número diferente de *bytes*, o que dificulta o envio dos mesmos a uma taxa constante. No modelo implementado, cada *slot* possui um tamanho fixo de blocos e em conseqüência, também em *bytes*. Dessa maneira, cada um dos *slots* não possui um tempo de vídeo fixo e sim um

tempo de envio fixo, e, ainda, cada um possui um tempo médio fixo. Com este modelo, onde os *buffers* colapsados giram síncronamente, só precisamos de um parâmetro do vídeo para que possamos enviá-lo, seu tamanho em segundos, já que seu tamanho em *bytes* é obtido automaticamente do arquivo. A partir destes dois parâmetros podemos calcular fácilmente a taxa média em que o vídeo deve ser enviado, através de um mecanismo de escalonamento da interface de rede que será apresentado a frente.

Estendemos ainda o funcionamento da CVCC para podermos disponibilizar vídeos ao vivo. Neste modo de funcionamento o tamanho do vídeo não é conhecido inicialmente, ou melhor, pode ser infinito. No entanto precisamos saber quanto tempo, anterior ao ponto que estiver sendo exibido, iremos manter na cache do *proxy*. Outro ponto é que a maioria dos clientes estará assistindo o vídeo ao vivo e desta forma devemos facilitar o uso de *multicast* do *buffer* colapsado mais adiantado.

4.3 Problemas das Redes de Acesso

Em uma LAN, atualmente não encontramos redes com banda inferior a 10 Mbps, o que é mais do que suficiente para a transmissão de qualquer fluxo gerado com as técnicas disponíveis atualmente, mesmo em qualidade de *Digital Versatile Disk* (DVD), a não ser que queiramos distribuir vídeos em HD (High Definition) [17] e neste caso seriam necessários 20 Mbps codificados em *Moving Pictures Experts Group version 2 standard* (MPEG2). No entanto, isto está fora da realidade atual já que são necessárias máquinas de última geração para conseguir decodificar vídeos com tamanha qualidade e mesmo porque seria necessário usar monitores ou TVs de altíssima qualidade para que seja notada alguma diferença de qualidade nesses vídeos. Dessa forma neste tipo de rede não se faz necessária a temporização dos pacotes (um pacote é um bloco de vídeo com metadados necessários para o envio e controle na rede) com uma granularidade muito fina, já que com um banda muito superior à taxa do vídeo dificilmente ocorreriam perdas de pacotes, a não ser que haja falha no *hardware* ou o *driver* da placas de rede não esteja configurado apropriadamente.

Um dos principais problemas encontrados na implementação do GloVE-WAN para redes ADSL, assim como a implementação de sistemas de distribuição de vídeo sob demanda com o servidor trabalhando no modo push, encontra-se na limitação de banda disponível aos clientes. Com uma banda limitada (ainda é comum encontrarmos conexões ADSL inferiores a 300 Kbps), o vídeo tem que ser enviado com uma taxa o mais próxima

possível da banda de descida disponível, dando apenas uma pequena margem para reposição de pacotes perdidos mas o suficiente para melhorar a qualidade para o fluxo de áudio e vídeo.

Em redes WAN a perda de pacotes ocorrida é muito maior, visto que o que caracteriza uma rede deste tipo é que ela é geograficamente maior. Devido aos pequenos *buffers* encontrados nos dispositivos que interligam os clientes ao distribuidor, as taxas de erro podem ser muito grandes quando enviamos dados a uma taxa levemente superior à banda disponível para o cliente. Desta maneira, para facilitar o trabalho dos DSLAMs, tratamos de enviar os dados o mais próximo possível à taxa do vídeo, que deve ser, é claro, menor que a banda disponível para o cliente.

Mesmo tomando estas medidas, ocorrem perdas. E, dependendo do formato do vídeo estas perdas não podem ser ignoradas, tornando necessário repor os pacotes perdidos na rede, o mais rápido possível de maneira que o usuário não tome ciência do ocorrido através de pausas na exibição.

4.4 Protocolo de Comunicação

O protocolo utilizado em toda a comunicação do sistema é o UDP, tanto nas mensagens de controle como no fluxo de vídeo propriamente dito. A escolha de tal protocolo se deu principalmente devido à maior possibilidade de se obter um sistema escalável e também para evitarmos a necessidade de termos que adaptar o sistema a um determinado protocolo específico. Se fossem utilizados protocolos acima da camada de transporte, comprometeríamos escalabilidade do sistema, além de termos que levar o sistema à sofrer adaptações que levam tempo e com certeza comprometeriam seu melhor funcionamento.

Quando consideramos uma Rede Ethernet para transmissão dos vídeos, não podemos ter pacotes maiores que 1492 *bytes*. Se o fizermos, teremos que tratar a fragmentação dos pacotes, ou seja, pode ser que na tentativa de enviar um pacote maior que um quadro Ethernet, os mesmos cheguem em ordem diferente da enviada, ou pior ainda, que alguns dos pacotes sejam perdidos.

No primeiro caso teríamos que ter marcações a cada, por exemplo, N *bytes*. No último caso teriamos que enviar o pacote inteiro novamente. Como podemos observar isto não compensa, o melhor é fazermos a nossa própria "fragmentação", limitando o tamanho dos pacotes do sistema ao tamanho de um quadro Ethernet, no nível da aplicação. No outro extremo, se utilizarmos pacotes muito pequenos, não conseguiremos aproveitar toda a

banda disponível, uma vez que parte dela será utilizada para transmitir outras informações como o cabeçalho dos pacotes. Em redes Ethernet de até 100 Mbps não teremos perdas significativas quando utilizarmos pacotes menores, mas em uma rede Gigabit Ethernet temos que utilizar pacotes com o tamanho maior possível, de maneira a não subutilizar a rede.

Dessa forma definimos o tamanho de cada pacote transmitido no sistema como sendo igual ao *Maximum Transmission Unit* (MTU), que em uma rede de acesso à Internet como a ADSL é um dado conhecido, subtraído dos cabeçalhos IP e UDP, ou seja, $PPayloadSize = (MTUSize - IPv4Header - UDPHeader)$. Os valor para o MTU em uma rede Ethernet é 1500 *bytes*, incluindo o cabeçalho *IPV4* de 20 *bytes* e o cabeçalho do UDP, de 8 *bytes*.

Todos os pacotes utilizados no sistema têm em seu cabeçalho um identificador de tipo que é seguido por um indentificador de mensagem. O identificador de mensagem ser para que possamos dar tratamento às mensagens duplicadas, controle de perda de pacotes e ordenamento. O distribuidor mantém para cada cliente um identificador da última mensagem processada. Se recebermos uma mensagem com identificador menor ou igual que a última mensagem recebida podemos descartar este pacote já que o mesmo esta sendo recebido de forma duplicada, o identificador da mensagem nunca é maior que a mensagem esperada já que uma mensagem só é reenviada quando não for recebido uma resposta confirmando o pacote recebido (*ack*). Se esta resposta não for recebida em um tempo pré-determinado a mensagem é reenviada, e caso não não tenha sido a própria mensagem perdida e sim o *ack*, a mesma será tratada como mensagem duplicada.

O protocolo de comunicação é composto por requisição e repostas, que consistem em 30 tipos de pacotes diferentes. Citando alguns exemplos temos:

- Bloco de vídeo.
- Handshake (cálculo de banda, nível de perdas) - São utilizadas para descobrir a banda de descida disponível ao cliente para distribuição do vídeo, bem como o número de perdas e o jitter entre o distribuidor e o cliente.
- Controle do fluxo (pausa, recúo, etc) - Tem a função de alterar ou parar o ponto em que o vídeo está sendo enviado.
- Problemas encontrados (ex. perda de bloco) - Se uma perda de um bloco ocorre, então o cliente avisa o distribuidor, que em seguida tenta repor os blocos perdidos.

4.5 Modelo do Sistema

Com o lançamento do kernel 2.6 do Linux, com destaque para o novo esquema de escalonamento, obtivemos grandes melhorias no suporte a threads. Dessa forma pensamos na possibilidade da utilização de um modelo multithread no qual cada *buffer* colapsado possuiria duas threads, sendo uma de escrita e outra de leitura, tal modelo teria maior semelhança com a *CVCC* proposto originalmente. No entanto com uma implementação inicial, verificamos que os tempos de distribuição do vídeo não eram atendidos adequadamente quando mais de 25 *buffers* colapsados eram inicializados e, consequentemente, 50 *threads*.

Além dos tempos de envio não serem atendidos adequadamente, o número de perdas ocorridas no próprio sistema operacional no envio aumentava quanto maior fosse o número de *threads*, devido à concorrência pela rede e CPU. É importante mencionar que estas perdas ocorriam devido ao kernel 2.6 estar em suas versões iniciais, não suportando um mecanismo, incorporado nas versões recentes, de verificação de que quantos *bytes* estão aguardando na fila de envio dos sockets UDP para serem enviados. De posse desse valor poderíamos verificar antes do envio se existia espaço vago na fila de envio e caso contrário aguardar mais alguns instantes, evitando assim as perdas.

Abandonamos este modelo *multithread* e tentamos implementar um novo modelo com um número menor de threads que não varie com a carga sobre o sistema, tal que tornamos o sistema mais escalável do ponto de vista do sistema operacional e ao mesmo tempo tornar mais fácil a implementação, quando comparado com um modelo de uma única thread.

No novo modelo implementado o sistema é divido em 4 threads. A primeira delas é responsável pela obtenção e envio de dados do servidor ou pela obtenção dos dados do disco, a segunda é responsável pela leitura dos dados da cache e envio para os clientes, a terceira serve para recepcionar os pacotes de controle enviados pelos clientes e incluí-los em uma fila de prioridades. A última thread é utilizada para sincronização do sistema, ela recebe os pedidos da fila de prioridades e os trata adequadamente, além de liberar a memória da cache quando for necessário e girar os *slots*.

A Figura 4.1 mostra os dois estados do sistema e as 4 *threads*. Em um dos estados existem 3 *threads* executando, somente a de sincronização está parada. Neste estado fluxos estão sendo recebidos do servidor e enviados para os clientes e as requisições sendo atendidas. No outro estado, os *slots* estão sendo giradas e novas requisições estão sendo

enfileiradas para que futuramente sejam atendidas.

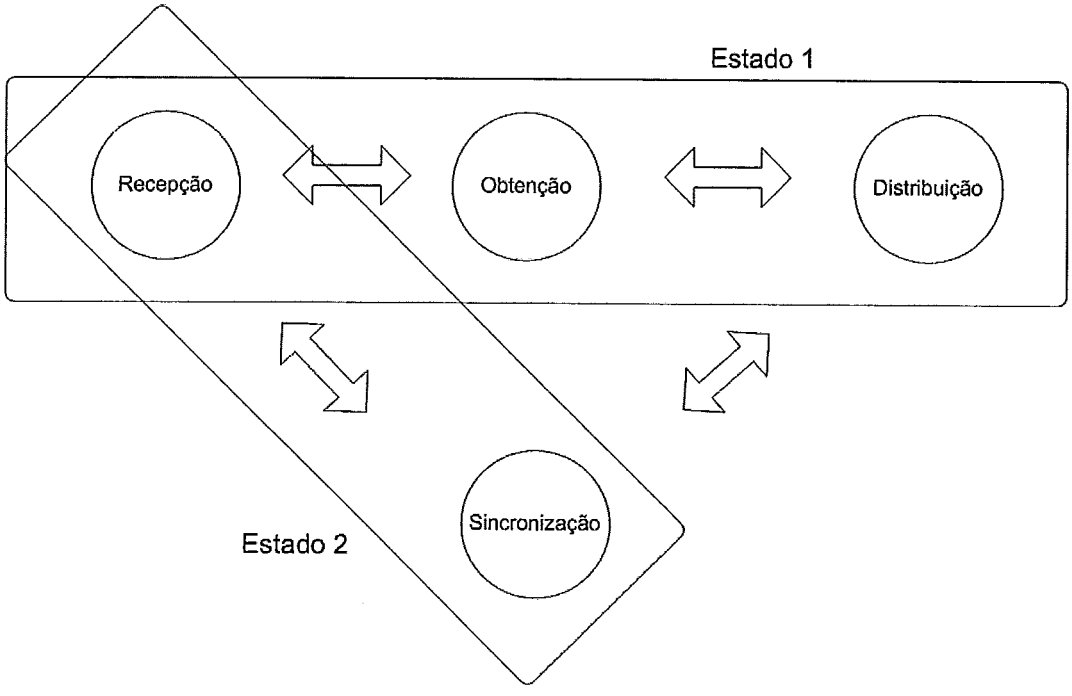

Figura 4.1: Estados do sistema

Dessa forma a sincronização do sistema foi simplificada significativamente e além disso o sistema operacional não se sobrecarrega com a execução de inúmeras threads. O consumo de tempo de CPU é dividido entre as duas threads principais, a de recepção e a de envio de pacotes. A maior parte do tempo é dado para a *thread* de recepção de pacotes de controle quando as duas estão ociosas.

Mesmo utilizando esta solução, temos problemas com as taxas de envio dos vídeos, já que o sistema operacional pode chavear para outro processo em qualquer momento, quando estamos utilizando o escalonador "normal" do Linux. Isto ocorre mesmo quando diminuímos ao máximo a granularidade (*timeslice*) do escalonador. Devido a isso, foi necessário criar estratégias de escalonamento especiais explicadas a seguir, em que foi utilizado o escalonador de tempo real.

4.5.1 Escalonamento de Rede e CPU

Desta forma a solução que encontramos para enviar os vídeos em tempo real com a garantia de que os pacotes seriam enviados no tempo previsto, sem sairmos do modo usuário [1],

[1]Uma aplicação pode rodar em modo usuário ou em modo kernel. Para funcionar em modo kernel devemos compilá-la como um módulo ou chamada de sistema, o que complica a implementação e realização de testes

foi alterar as prioridades das threads para funcionamento em tempo real, tendo que utilizar para isto uma conta de super-usuário. Neste modo de funcionamento podemos chavear de uma thread para outra sempre que for necessário, sendo somente interrompidos por interrupções.

Nesta abordagem torna-se necessário definir com qual peso cada uma das *threads* irá executar. Analisando o funcionamento de cada uma vemos que: a de obtenção dos pacotes de vídeo precisa de uma prioridade (P_o) proporcional à banda disponível entre o distribuidor e o servidor (B_s b/s), a de distribuição dos pacotes de vídeo precisa de uma prioridade (P_d) proporcional à banda entre o distribuidor e os clientes (B_d b/s) e finalmente, a *thread* de recepção de pacotes, necessita de uma prioridade (P_r) tal que ela consiga atender a um número máximo de pedidos por segundo ($requestsPerSecond$). Dessa forma, considerando o pior caso, no qual todas as *threads* estão utilizando a rede no limite podemos calcular as prioridades como mostram as equações 4.1. Definindo o tamanho máximo de um pacote como sendo $maxPacketSize$ a banda máxima utilizada na recepção de pacotes de controle é $B_r = requestsPerSeconds \cdot maxPacketSize$.

$$
\begin{aligned}
P_o &= \frac{B_s}{B_s + B_d + B_r} \\
P_d &= \frac{B_d}{B_s + B_d + B_r} \\
P_r &= \frac{B_r}{B_s + B_d + B_r}
\end{aligned}
\tag{4.1}
$$

4.5.1.1 Recepção dos Pacotes

A recepção de pacotes no sistema acontece através de um socket UDP que é utilizado por uma única thread, cuja função é única e exclusivamente a recepção dos pacotes da rede e a inclusão dos mesmos em uma fila de prioridade, como mencionado anteriormente. Dentre os principais tipos de pacotes de controle recebidos pelo sistema, e importantes para a transmissão dos vídeos, estão os seguintes: inclusão, exclusão, avanço ou retrocesso, troca de vídeo e pausa. Dentre estes pacotes, são dadas prioridades maiores para os pacotes cujos clientes já estão conectados ao sistema, ou seja, já conseguiram se incluir ao mesmo através do envio de um pacote de inclusão com *foward*, *rewind* e etc. Os pacotes de pedido de reposição de alguma parte do vídeo perdida tem prioridade sobre qualquer outro e dessa

forma não precisam aguardar sincronização, sendo incluídos na fila de pacotes perdidos do cliente que requisitou. Na próxima vez que ocorrer o envio de um pacote perdido para este cliente, o pacote será enviado.

A inserção nesta fila, para os pedidos de inclusão, avanço, retrocesso ou troca de vídeo, é feita de maneira que os clientes dentro de uma mesma janela de tempo ocupem a mesma posição, ou seja, agrupamos aqueles que estiverem requisitando um mesmo momento do vídeo de acordo com a janela de tempo definida (WT). Quando uma determinada janela de tempo de um pedido estiver batendo com outras, o pedido será incluído junto com aquele que estiver mais a frente na fila.

Como não sabemos a priori a quantidade de pacotes que iremos receber, calculamos o tempo de execução nos baseando no número máximo de pedidos por segundo ($maxRequestsPerSecond$). Isto significa que no caso extremo esta *thread* deve executar no pior caso ($maxRequestsPerSecond$) vezes. Assim de posse do tempo de envio e recepção de *slots* T e da prioridade P_r desta *thread* podemos calcular o tempo após o qual devemos passar a execução para a próxima *thread* t_r, como mostra a Equação 4.2.

$$t_r = \frac{T \cdot P_r}{maxRequestPerSecond} \tag{4.2}$$

4.5.1.2 Controle de Admissão dos Clientes e Gerenciamento

Periodicamente colocamos em prática as ações solicitadas através da retirada dos pacotes da fila de inclusão e o processamento dos mesmos. Este tempo deve ser proporcional ao tempo de envio de cada *slot*. Dessa forma podemos processar os pedidos da fila a cada T segundos, onde T é o tempo de envio do *slot*. Pode ser feito o processamento de todos os pedidos incluídos na fila, ou aguardar por mais um tempo T, dependendo de qual é a política determinada para a inclusão.

Com a utilização desta fila de pacotes, dependendo do QdS determinado para o sistema, podemos aguardar por mais ou menos tempo pela chegada de um maior número de clientes com pedidos de recebimento do mesmo *slot*, por exemplo, para o primeiro *slot* dos vídeos. Assim, se for determinado um tempo grande o suficiente para que mais de um cliente possa ser incluído no mesmo *slot*, conseguiremos economizar memória, e além disso estaremos facilitando a utilização de *multicast* para que sejam economizados fluxos de vídeo entre o distribuidor e os clientes. É claro que, isto só e possível se os com-

ponentes entre os clientes e o distribuidor suportarem *multicast*, sejam eles roteadores, DSLAM, comutadores e etc.

Duas políticas são possíveis. Podemos incluir o cliente no sistema imediatamente, ou podemos aguardar por um determinado tempo. Para tomar esta decisão temos como informação a fila que contém os conjuntos de pedidos a serem atendidos e o tempo de chegada do pacote ao sistema. Testamos o primeiro pacote da fila verificando se o tempo expirou, ou seja, se o tempo em que o pedido chegou somado ao tempo máximo de inclusão for maior ou igual ao tempo atual, então o pedido é atendido. Assim definimos uma nova variável para o sistema, o tempo máximo de espera (MTW).

A *thread* de controle executa a liberação de memória, a liberação de *links slots* e o giro dos blocos de vídeo para o próximo *slot*. No momento em que ela está executando, tanto a de envio quanto a de obtenção de dados do servidor ficam paradas aguardando que ela termine seu serviço. Pode parecer a princípio que o tempo de gasto nas operações desta *thread* (TO) irão influênciar no funcionamento do sistema. No entanto, isto não ocorre. Testes realizados com 26 vídeos e *slots* de 1 segundo, TO foi igual a 2 microsegundos. Isto significa que temos que obter (quando necessário) e enviar todos os *slots* contidos nos *buffers* colapsados 2 microsegundos mais rápido. No caso mencionado teremos que aumentar a nossa taxa de envio e recepção de dados em 2 Kbps.

Critérios para cache cheia

Quando incluímos um novo cliente no sistema precisamos garantir que o mesmo conseguirá assistir o vídeo escolhido até o final sem nenhuma interrupção, ou seja, os recursos do sistema têm que estar disponíveis desde o instante em que se admite o cliente até o momento em que é terminada a exibição ou o mesmo sai do sistema por vontade própria.

Para que um cliente ou um grupo de clientes não sejam retirados do sistema até o fim da exibição, precisamos descobrir se existem recursos disponíveis no sistema. Se não existirem, o controle de admissão de novos clientes bloqueia o pedido do cliente.

Se houver banda disponível entre o servidor e o *proxy*, então mais um *buffer* colapsado pode ser criado e conseqüentemente pelo menos um cliente pode ser admitido. Se esta condição não for satisfeita devemos observar para qual ponto do vídeo o cliente requisitante deseja ir e observarmos se dentro da janela de inclusão não existe algum *buffer* colapsado previamente criado no qual o mesmo possa ser incluído. Se essas duas condi-

ções não forem satisfeitas devemos verificar se é possível a criação de *link slots* para encadear o *buffer* colapsado criado para o novo cliente no ponto requerido. Se nenhuma das três condições forem satisfeitas então o cliente não pode ser incluído e está, portanto, bloqueado, devendo aguardar na fila de inclusão por mais alguns instantes. Além dos requisitos necessários ao distribuidor é necessário também que o cliente possua recursos para receber o fluxo. Discutiremos este caso em seção posterior.

4.5.1.3 Obtenção dos Vídeos

Para cada um dos *buffers* colapsados criados temos que preencher seu *slot* de escrita na medida em que os dados contidos no *slot* de envio são mandados para os clientes. Para que isto ocorra, deve haver banda disponível entre o servidor e o distribuidor de modo que os dados de todos os *slots* de escrita sejam recebidos antes que os *slots* de leitura sejam enviados para os clientes.

Para cada vetor de *slots* é mantida uma lista de *buffer* colapsados, desta forma sabemos quais *slots* devem ser preenchidos com blocos, já que a partir do *buffer* colapsado obtemos o *slot* de escrita. Assim sabemos quantos blocos teremos que obter na próxima recepção de *slots*. De posse da prioridade de execução da *thread* P_o e do número de blocos que devem ser obtidos ($numberOfBlocksToGet$) podemos calcular o tempo após o qual devemos passar a execução para a próxima *thread* t_o, mostrado na Equação 4.3.

$$t_o = \frac{T \cdot P_o}{numberOfBlocksToGet} \tag{4.3}$$

4.5.1.4 Distribuição dos Vídeos

Como podemos observar existem duas estruturas principais para manipulação dos vídeos. São elas os vetores de vídeo e os vetores de *slots*. A cada vetor de *slots* está associado um vetor de vídeo. Para o cálculo do tamanho do vetor de *slots* precisamos conhecer duas variáveis relativas ao vídeo, o tamanho e o tempo se o vídeo estiver sendo disponibilizado para vídeo sob demanda ou a taxa e o tempo que se deseje manter o vídeo na cache se estivermos fazendo uma transmissão ao vivo. De posse desses dados, além do tempo determinado para o *slot*, podemos inicializar as duas estruturas. A função do vetor de

vídeo é armazenar os blocos de vídeo, descartando-os e alocando-os quando solicitado. Os vetores de *slots* tem função de gerenciar o que deve ser mantido na cache ou não.

Além destas duas estruturas criamos mais uma, chamada de vetor de *slots* de distribuição. A função dos *slots* de distribuição é dividir a banda de descida do distribuidor de maneira a não ultrapassarmos a banda disponível entre o distribuidor e os clientes, e mais importante ainda, garantir que tempo de envio de cada bloco para cada um dos clientes seja determinado e não ultrapasse a banda de descida do cliente. Além dos pacotes de vídeo os *slots* de distribuição também gerenciam o pacotes de controle e também a reposição de erros ou pacotes perdidos.

Nem todos os *slots* de distribuição estão alocados para os clientes. Parte deles é alocada para que possamos responder a clientes que ainda não foram incluídos no sistema e portanto não possuem nenhuma estrutura associada a eles. Os três tipos de mensagens que são respondidas desta forma são : as mensagens avisando ao cliente que ele está bloqueado; as mensagens que requisitam informações sobre o sistema que é respondida com um pacote que contém dados como o número de arquivos disponíveis para a exibição, o tempo de resposta máximo que deve ser aguardado antes que seja feito um novo pedido e ainda de quanto em quanto tempo o cliente deve enviar uma mensagem informando que continua assistindo ao conteúdo e evitar assim que seja excluído do sistema e por último as informações sobre cada um dos arquivos disponíveis. Assim calculamos o número necessário de *slots* (r_{slots}) de distribuição que devem ser reservados com esta finalidade, considerando o número máximo de inclusões previstas por segundo ($maxInclusionsPerSecond$) como mostra a Equação 4.4.

$$r_{slots} = \Big(maxInclusionsPerSecond \cdot T\Big)\Big(\frac{maxStreams}{streamFilesInfoPerPacket} + 1\Big) \quad (4.4)$$

Quando um cliente requisita inclusão no sistema, precisamos verificar se o mesmo possui banda suficiente para receber o fluxo e reservarmos alguns *slots* de distribuição para ele, de acordo com a sua banda de descida. A banda de descida do cliente (C_b) é calculada no momento de instalação do software cliente e tem o resultado armazenado em disco para que futuramente não seja necessário novo cálculo. De posse da banda disponível podemos calcular o número de *slots* (ns) que devem ser reservados para o cliente no tempo de um *slot* T, considerando além do fluxo de vídeo, os blocos de controle e aqueles utilizados para a reposição de blocos perdidos. Como veremos posteriormente, no próximo capítulo, a taxa de erros que ocorre em uma rede ADSL é inferior a 0,1%.

O número de pacotes de controle que necessitam de resposta, podem ser calculados a partir de outra variável do sistema, o número máximo de mensagens de controle por segundo ($maxControlMessagesPerSecond$), que representa o número de mensagens de controle por segundo esperadas para cada cliente. Assim podemos calcular a banda necessária ao cliente (C_{bn}) na Equação 4.5. Se a banda do cliente C_b for maior ou igual à banda necessária, então o cliente pode ser admitido.

$$C_{bn} = (1,001 \cdot streamBandwidth) + (maxControlMessagesPerSecond \cdot PPayloadSize) \tag{4.5}$$

De posse dos *slots* de distribuição reservados podemos calcular o número de pacotes que devem ser enviados no tempo de um *slot* T, $numberOfPacketsToSend$. Possuindo em mãos ainda a prioridade da *thread* de distribuição podemos calcular o tempo, após o qual, esta *thread* deve passar a execução para a próxima (t_d), como mostra a Equação 4.6.

$$t_d = \frac{T \cdot P_d}{numberOfBlocksToSend} \tag{4.6}$$

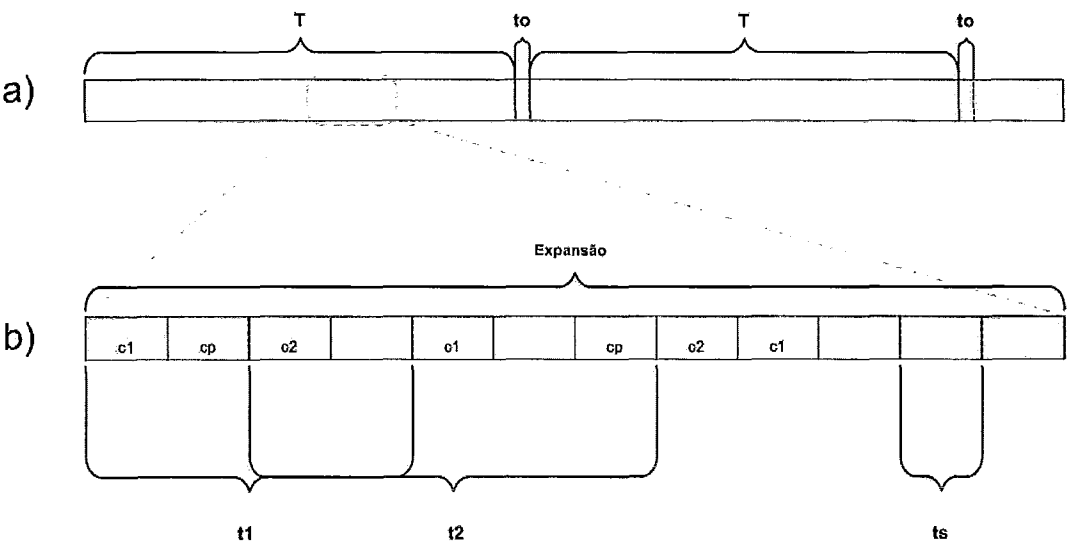

Figura 4.2: Slots de distribuição

Na Figura 4.2 (a), mostramos o tempo de envio de um T que é distribuídos entre os clientes. A cada intervalo de tempo T temos o *overhead to* para girar os slots associados as *buffers* colapsados. Experimentalmente vemos que este tempo não é superior a 2 microsegundos, e dessa forma não interfere na temporização dos pacotes do sistema. Na

Figura 4.2 (b), mostramos *slots* de distribuição alocados para dois diferentes clientes, marcados com *c1* e *c2*. O intervalo de tempo de envio entre um pacote e o próximo (*t1* e *t2*) e calculado a partir da banda disponível a cada um dos clientes. O tempo de um *slot* de distribuição *ts* é o tempo necessário para o envio de um pacote do tamanho do MTU.

Multicast

A utilização de *multicast* é interessante quando queremos diminuir o uso da rede e não necessitamos ter uma alta QdS, no que se refere a latência no início de exibição. De qualquer forma, em períodos de alta utilização do sistema a ocorrência de clientes requisitando um mesmo ponto do vídeo aumentará e neste caso a latência de início de exibição será a mesma, tanto com o uso de multicast como sem. Neste caso é necessário que a infra-estrutura de rede suporte *multicast*.

Não existindo suporte, podemos utilizar *multicast* a nível de aplicação, no qual não é necessário que a rede o suporte e dessa forma não existem vantagens na sua utilização para o desempenho da rede. No entanto, podemos economizar memória com seu uso, já que podemos alocar, por exemplo, somente um *buffer* colapsado para diversos clientes e além disso, diminuímos o número de leituras à memória *RAM* ou ao disco. Mas, como mencionado anteriormente, para agrupar clientes em um mesmo ponto do vídeo diminuímos a QdS do sistema na medida em que aumentamos a latência de início de exibição do vídeo para aguardar que mais clientes requisitem um mesmo ponto do vídeo. Portanto, com o uso de *multicast* no sistema mantemos o mesmo esquema de alocação dos *slots* de distribuição, no entanto, a leitura da memória pelo *slot* de escrita do *buffer* colapsado é feita uma única vez através do agrupamento dos clientes.

Capítulo 5

Avaliação Experimental

Neste capítulo apresentamos alguns resultados obtidos em uma rede telefônica brasileira. Na seção inicial descrevemos o ambiente de testes utilizado. Em seguida os resultados obtidos.

5.1 Ambiente Utilizado

O ambiente de testes utilizado foi constituído por componentes do Laboratório de Computação Paralela (LCP) e da operadora de telecomunicações que são mostrados na Tabela 5.1.

Foi feito um link interligando os dois laboratórios através de fibra óptica e desta forma utilizamos um cluster de 20 computadores para gerar carga para o sistema. A topologia utilizada é mostrada na Figura 5.1.

5.2 Resultados

Os testes foram realizados com 4 filmes de longa duração e 4 diferentes taxas ao mesmo tempo, totalizando 16 vídeos. Cada vídeo foi codificado tendo como base para a sua taxa de compressão a banda disponível aos clientes ADSL, conforme Tabela 5.2.

O servidor utilizado trabalhava em modo *pull* e não limitava a banda disponível até o distribuidor, ou seja, para cada pedido de bloco requisitado, era imediatamente enviada uma resposta, após lido do disco. Tínhamos somente 4 fluxos entre o servidor e o distribuidor, um para cada vídeo, e dessa forma ele conseguia dar vazão, sem se tornar o gargalo.

O distribuidor foi configurado com tamanho do *slot* igual a 3 segundos para que tivéssemos uma baixa latência no início de exibição. Não foi utilizada nenhuma política de

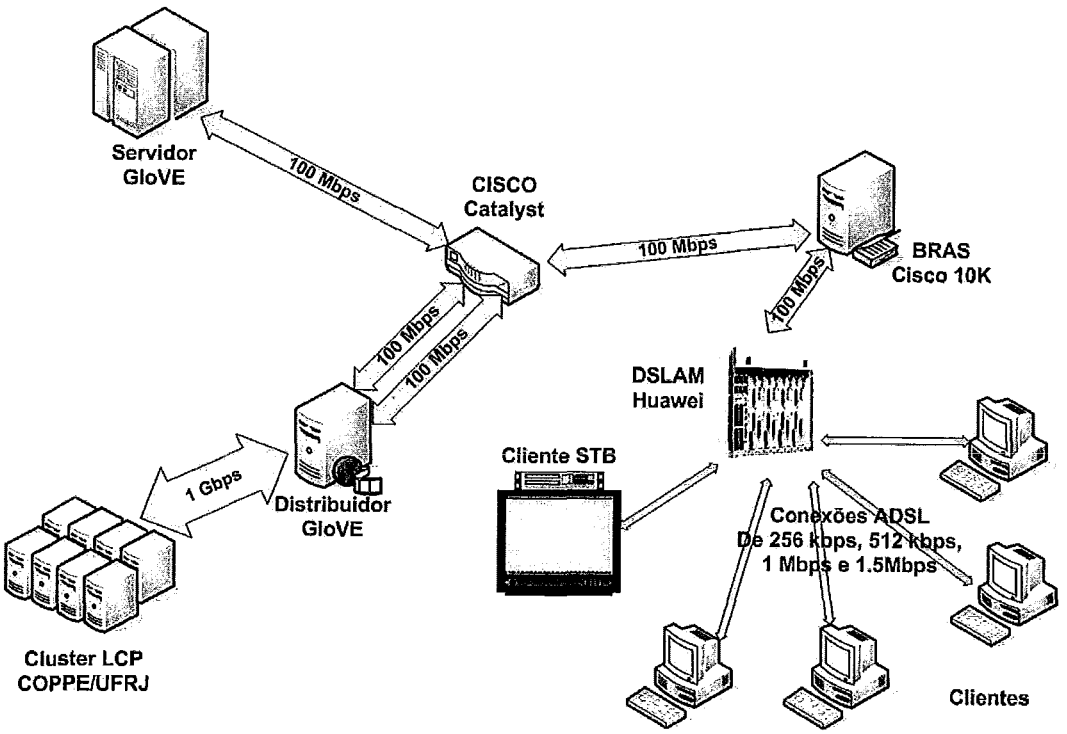

Figura 5.1: Topologia utilizada nos testes

substituição de slots na cache.

Os únicos resultados avaliados foram a vazão do distribuidor, a taxa de erros entre os distribuidores e o DSLAM, a capacidade de recuperação de blocos perdidos pelos clientes e as operações de vídeo cassete realizadas pelos clientes reais ADSL.

O sistema se portou bem e verificamos que a vazão do distribuidor foi limitada apenas pela rede como mostra a Tabela 5.2 que possuía uma vazão efetiva[1] de 403 Mbps entre o distribuidor e uma máquina do cluster do LCP. A taxa de erro máxima foi de 0,1%. Este erro deve aumentar quando todas as portas do DSLAM estiverem ocupadas, já que neste teste foram utilizados somente 4 clientes reais ADSL. Os clientes conseguiam se recuperar dos erros sem que o usuário percebesse o ocorrido, ou seja, sem interrupção na exibição do vídeo. As operações de vídeo cassete funcionaram perfeitamente. A latência de início de exibição dos vídeos, como era esperado, ficou entre 10 e 16 segundos, já que o *prefetch* necessário era de 10 segundos.

[1]Medida com a ferramenta de medição de desempenho de rede Iperf [12]

Equipamento	Descrição
Concentrador (DSLAM)	Huawei
Autenticador (BRAS)	Cisco Série 10000
Roteador	Cisco Catalyst Série 12000
Comutador	3COM 2224 Gigabit
Modems ADSL	Efficient Networks SS5200
	Efficient Networks SS5400
Servidor	Pentium 4 2.8 Ghz – 1GB RAM – kernel 2.6
Distribuidor	Pentium 4 2.8 GHz – 1 GB RAM – kernel 2.6
20 Computadores para gerar carga	Pentium 4 2.8 GHz – 1 GB RAM – kernel 2.6
1 Set-Top Box	Mobile Celeron 1.266 GHz – 256 MB RAM
4 Computadores	Windows XP – mínimo 256 MB RAM

Tabela 5.1: Componentes utilizados

Taxa de conexão	Número de clientes
256 Kbps	1512
512 Kbps	756
1,0 Mbps	400
1,5 Mbps	260

Tabela 5.2: Número máximo de clientes simultâneos

Capítulo 6

Trabalhos Relacionados

Neste capítulo discutimos os trabalhos encontrados que tratam de cache dinâmica em sistemas de distribuição de mídia.

Cache

O trabalho desenvolvido por E. Bommaiah [4] usa em sua política de gerenciamento de cache dinâmica *buffers* circulares . No entanto eles são utilizados somente para armazenamento em curto prazo. A longo, é utilizada a política de cache estática em disco. Desta forma um *buffer* circular pode ser alimentado tanto pela cache estática em disco quanto por um fluxo do servidor, dependendo dos recursos disponíveis do sistema. É implementado um esquema como o LS discutido nesta dissertação. Este esquema não é tão eficiente quanto o LS+, já que o esquema não gerencia as memórias principal e secundária em conjunto. Com a CVCC é considerada a memória como um todo.

Uso semelhante aos *buffers* circulares é dado em outro trabalho [28], no qual é enfatizada a coordenação entre os *proxy* e o servidor através dos mesmos. Neste trabalho o *proxy* só serve um cliente quando for possível alocar um *buffer* para ele, garantindo desta forma uma QdS na entrega do vídeo. Neste trabalho o tamanho do prefixo é estabelecido a partir da popularidade dos vídeos. A gerência dos *buffers* colapsados associados aos clientes é feita com a utilização de um algoritmo específico e é feita dinâmicamente a partir da banda na dorsal da rede que estiver sendo utilizada no momento, o que é um problema, já que de tempos em tempos o algoritmo deve ser executado.

Outro trabalho [29] propõe um serviço com QdS, no qual não é previsto o uso de uma cache secundária, em disco por exemplo, com o objetivo de atender mais clientes. Todo o pedido que chega é atendido ou bloqueado. Se um fluxo de vídeo é entregue pelo

servidor com determinada QdS ao *proxy* então este será entregue com a mesma QdS ao cliente. Este trabalho possui uma política que funciona de maneira semelhante à LS.

Escalonamento

O trabalho [6] trata o problema da utilização de um mesmo canal por diversas fontes de vídeo de maneira a otimizar a distribuição dos pacotes para que seja dada prioridade àqueles, digamos, mais importantes. É um trabalho realizado por simulação, claro, por que como podemos observar é extremamente difícil implementar tal sistema, que em termo real determina quais pedaços do vídeo devem ter prioridade sobre outros. Esta tarefa involve múltiplos aspectos, como entender os codificadores de áudio e vídeo, os multiplexadores e ainda como eles se relacionam entre si. Poderia ter sido pensado algo mais simples como por exemplo identificar os quadros independentes e dar prioridade a estes.
Em [31], para facilitar o escalonamento dos pacotes é necessário tentar agrupar os clientes de modo a formar grupos de *multicasting*, diminuindo assim o número de fluxos. Trabalho extremamente teórico.

Outros trabalhos encotrados tratam o escalonamento/atraso de pacotes focando no fato de que diversas aplicações estarão utilizando a rede ao mesmo tempo [14] [30], diferente do nosso caso que temos somente uma.

Nenhum trabalho que implemente algum esquema de escalonamento de pacotes para distribuição de mídia contínua sob demanda foi encontrado até o momento.

Capítulo 7

Conclusões

Introduzimos os principais conceitos de sistemas de vídeo sob demanda e discutimos as vantagens de utilização dos *proxies* nas bordas da rede. Descrevemos as topologias e tecnologias mais utilizadas nas redes de acesso atualmente. Apresentamos em seguida os protocolos mais utilizados e conhecidos para distribuição de mídia em tempo real e mostramos os desafios encontrados no desenvolvimento de um sistema de vídeo sob demanda.

No capítulo 2, mostramos os conceitos da memória cooperativa, dando ênfase à memória cooperativa colapsada, fazendo um resumo do trabalho que a criou.

Decrevemos então a implementação do sistema com os principais problemas encontrados como o escalomento de pacotes através dos *slots* de distribuição e o escalonamento de CPU através da divisão por prioridade, de acordo com a tarefa a ser executada. Então apresentamos os resultados conseguidos em uma rede de telefonia e discutimos em seguida os trabalhos relacionados.

Conseguimos mostrar com os experimentos realizados que o distribuidor consegue suportar um número de clientes proporcional à banda disponível, servindo reparos aos clientes quando necessário, e com uma latência aceitável, dentro do esperado. Os resultados dos testes foram bastante positivos. A qualidade da transmissão para todas as taxas em questão (256 Kbps, 512 Kbps, 1,0 Mbps e 1,5 Mbps) foram muito boas, tanto com uso do modem no computador, quanto no *set-top box*. Tudo isto, quando o servidor funciona no modo *pull* e a leitura do disco não limita o envio a uma taxa que prejudique o funcionamento do distribuidor.

Mostramos ainda que o esquema de escalonamento das *threads* funcionou efetivamente, conseguindo manter as taxas de envio e recebimento de pacotes através da divisão do tempo de CPU. O protocolo funcionou perfeitamente, sem sobrecarregar o distribuidor.

7.1 Trabalhos Futuros

É interessante analisar o sistema em outros cenários mais realistas. Para tal poderemos utilizar clientes simulados que interpretem o formato do vídeo, consumindo na taxa adequada cada um dos quadros do vídeo. Dessa forma conseguiremos prever com realidade a variação nos buffers dos clientes e evitar assim a ocorrência de *overflows* ou *underflows* após a ocorrência de saltos, já que quando o cliente começa a exibir o vídeo desde o início, não teremos grandes diferenças, já que o cabeçalho é pequeno em relação às taxas de exibição.

É relevante ainda criar interfaces virtuais que limitem a banda tanto de download, quanto de upload dos clientes dependendo da capacidade do cliente ADSL que queremos simular. Se possível a interface virtual deve ser capaz de simular atrasos e jitter nos pacotes. As interfaces virtuais devem ser utilizadas também para limitar a banda entre o servidor e o distribuidor, já que não deve haver picos neste link.

Verificamos que conseguimos simular poucos clientes por máquina do cluster. Dependendo da taxa do vídeo conseguíamos simular entre 50 e 70 clientes por máquina, com um cliente que consumia os dados recebidos em uma taxa constante e sem a utilização de interfaces virtuais. Dessa forma, com o intuito de diminuir o uso de CPU, diminuímos a concorrência pela CPU através da implementação de um cliente com uma única *thread.*

Testamos neste trabalho o funcionamento do distribuidor quando o servidor envia os dados no modo *pull.* No entanto, servidores funcionando do modo *pull*, não são escaláveis, potencialmente. Dessa forma, como o distribuidor responde às requisições no modo *push*, pensamos em utilizá-lo como servidor, lendo blocos a partir do disco, e dessa forma preparar o distribuidor para funcionar com os servidores reais.

É importante também implementar e testar o distribuidor funcionando como servidor e o cliente *single thread*, que já foi implementado, em conjunto com as interfaces virtuais. Os *scripts* para criação das interfaces virtuais já foram implementados e no momento limitam somente a banda de descida, falta implementar a de subida e incluir o suporte a erros, atrasos e *jitter.*

Deve-se também realizar os principais testes variando os seguintes parâmetros:

- *unicast* e *multicast* em nível de aplicação
- vídeos contidos totalmente em disco ou ao vivo
- sem interatividade com

- um vídeo ou vários
- Tamanho da cache
- Políticas LS- , LS, LS+
- Intervalo de chegada entre os clientes
- Popularidade dos vídeos
- Tamanho dos *slots*
- Banda disponível limitada ou infinita entre o distribuidor e o servidor
- Janela de inclusão
- Tempo de espera na inclusão
- Realização de operações de vídeo cassete (definindo a probabilidade de ocorrência de cada uma) ou não

Como trabalhos futuros podemos propor ainda, um esquema que consiga dar escalabilidade ao sistema visto sob o ponto de vista do servidor. Para tal devemos tratar o escalonamento do acesso aos diversos discos rígidos no servidor de maneira que ele consiga manter a vazão, e dessa forma diminuir a influência da vazão do disco na de rede. Para podermos aumentar ainda mais a escalabilidade do sistema para que no futuro seja utilizado em rede de 10 Gbps podemos dividir cada um dos vídeos em diversas seqüencias de *slots*. Cada seqüencia de *slots* corresponde a um seqüencia de blocos no vetor de vídeo. No vetor de vídeo é sempre mantida a mesma distribuição nos discos e máquinas alocados inicialmente. E as seqüencias de vetores de *slots* são giradas por sobre o vetor de vídeo e dessa forma diminuem a vazão tanto sobre os discos como sobre as próprias máquinas. Neste trabalho precisaremos criar novas políticas de gerenciamento de cache com um controle global [3].

Podemos ainda incluir no sistema suporte a outros protocolos como o RTP e verificar a sua influência na escalabilidade do sistema.

Fazer uma avaliação matemática do sistema, comparando os resultados que aqui foram exibidos e os obtidos anteriormente por simulação.

Um trabalho já em andamento, sendo desenvolvido pelo colega João Alves, visa garantir a confiabilidade do sistema. Neste trabalho tenta-se garantir que, mesmo com a

falha de algum componente do sistema, os fluxos continuam a ser transmitidos de maneira que os clientes não percebam o ocorrido, ou pelo menos sofram a mínima interferência possível. Este trabalho ainda trata da migração de clientes para outros distribuidores em redes *wireless*. A parte de segurança, neste contexto, esta sendo tratada pelos alunos do laboratório de redes de alta velocidade (RAVEL), e coordenada pelo Prof. Luís Felipe Magalhães de Moraes.

Por fim, deve-se analisar a cooperação entre diversos *proxies* e avaliar o sistema em funcionamento no modo multicasting real, através da criação de interfaces virtuais que o suportem e/ou aquisição de novos comutadores.

Referências Bibliográficas

[1] Anttalainen, T. *Introduction to Telecommunications Network Engineering.* Artech House, 2003.

[2] Apple i-Tunes. http://www.apple.com/itunes.

[3] Bolosky, W., Draves, J., Fitzgerald, R., Gibson, G., Jones, M., Levi, S., Myhrvold, N., Rashid, R. "The Tiger Video Fileserver", 1996.

[4] Bommaiah, E., Guo, K., Hofmann, M., Paul, S. "Design and Implementation of a Caching System for Streaming Media over the Internet". In: *IEEE Real Time Technology and Applications Symposium*, pp. 111–, 2000.

[5] Bovet, D. P., Cesati, M. *Understanding the linux kernel.* O Reilly, 2005.

[6] Chakareski, J., Frossard, P. "Rate-Distortion Optimized Distributed Packet Scheduling of Multiple Video Streams Over Shared Communication Resources". *IEEE Transactions on Multimedia*, v. 8, n. 2, pp. 207–218, 2006.

[7] Dan, A., Sitaram, D., Shahabuddin, P. "Dynamic Batching Policies for an On-Demand Video Server". *Multimedia Systems*, v. 4, n. 3, pp. 112–121, 1996.

[8] Globo Media Center. http://www.globo.com.br.

[9] Golden, P., Dedieu, H., Jacobsen, K. S. *Fundamentals of DSL Technology.* Auerbach Publications, 2006.

[10] Helix Community. http://protocol.helixcomunity.org/.

[11] Hua, K. A., Cai, Y., Sheu, S. "Patching: A Multicast Technique for True Video-on-Demand Services". In: *Proceedings of the ACM Multimedia*, pp. 191–200, 1998.

[12] Iperf. http://dast.nlanr.net/Projects/Iperf.

[13] Ishikawa, E. *Memória Cooperativa para Distribuição de Vídeo sob Demanda.* PhD thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, October 2003.

[14] Kim, J., Tode, H., Murakami, K. "Network Adaptative Packet Scheduling for Streaming Video over Error-prone Networks". *IEEE*, 2004.

[15] Moulton, P. *Telecommunications Survival Guide.* Prentice Hall, 2001.

[16] Pinho, L. B. *Implementação e Avaliação de um Sistema de Vídeo sob Demanda Baseado em Cache de Vídeo Cooperativa.* Master's thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Maio 2002.

[17] Poyton, C. *Digital Video and HDTV Algorithms and Interfaces.* Morgan Kaufmann, 2003.

[18] Rao, S., Vin, H., Tarafdar, A. "Comparative Evaluation of Server-push and Client-pull Architectures for Multimedia Servers". In: *Proceedings of the 6th International Workshop on Network and Operating System Support for Digital Audio and Video*, April 1996.

[19] Rodriguez, P., Ross, K., Biersack, E. W. "Improving the WWW: caching or multicast?". *Computer Networks and ISDN Systems*, v. 30, n. 22–23, pp. 2223–2243, 1998.

[20] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V. "RTP: A Transport Protocol for Real-Time Applications". *RFC 3550, Network Working Group*, January, 1996.

[21] Schulzrinne, H., Rao, A., Lanphier, R. "Real Time Streaming Protocol (RTSP)". *RFC 2326, Network Working Group*, April, 1998.

[22] Sheu, S., Hua, K. A., Tavanapong, W. "Chaining: A Generalized Batching Technique for Video-On-Demand". In: *Proceedings of the International Conference on Multimedia Computing and Systems*, pp. 110–117, 1997.

[23] Sitaram, D., Dan, A. *Multimedia Servers: Applications, Environments, and Design*, chapter 7. Morgan-Kaufmann, 2000.

[24] Squid Web Proxy Cache. http://www.squid-cache.org.

[25] Summers, C. K. *ADSL: Standards, Implementation, and Architecture.* CRC Press LLC, 1999.

[26] Symes, P. *Video Compression Demystified.* McGraw-Hill, 2001.

[27] Tzerefos, P., Smythe, C., Stergiou, I., Cvetkovic, S. "Standards for High Speed Digital Communications Over Cable Tv NetWorks".

[28] Venkatramani, C., Verscheure, O., Frossard, P., Lee, K. "Optimal proxy management for multimedia streaming in content distribution networks", 2002.

[29] Verscheure, O., Frossard, P., Boudec, J. "Joint Smoothing and Source Rate Selection for Guaranteed Service Networks". In: *INFOCOM*, pp. 613–620, 2001.

[30] Wu, E. H., L., H., Tsai, M., Chou, C. "Low Latency and Efficient Packet Scheduling for Streaming Applications". *IEEE Communications Society*, pp. 1963–1967, 2004.

[31] Wu, M., Ma, S., Shu, W. "Scheduled Video Delivery – A Scalable On-Demand Video Delivery Scheme". *IEEE Transactions on Multimedia*, v. 8, n. 1, pp. 179–187, 2006.

[32] You Tube. http://www.youtube.com.