MINISTÉRIO DA DEFESA

DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA

INSTITUTO MILITAR DE ENGENHARIA

CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

Fabrício Guedes Bissoli

Gerência de memória cooperativa em proxies para distribuição de vídeo sob demanda

Rio de Janeiro

2006

INSTITUTO MILITAR DE ENGENHARIA

FABRÍCIO GUEDES BISSOLI

Gerência de memória cooperativa em proxies para distribuição de vídeo sob demanda

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientador: Edison Ishikawa, D.Sc.

Rio de Janeiro
2006

C2006

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro - RJ CEP: 22290-270

B623 Bissoli, Fabrício Guedes
Gerência de memória cooperativa em proxies para distribuição de vídeo sob demanda / Fabricio Guedes Bissoli. - Rio de Janeiro : Instituto Militar de Engenharia, 2006.
76 p. : il., tab.

Dissertação (mestrado) - Instituto Militar de Engenharia, Rio de Janeiro, 2006.

1. Engenharia de Sistemas. 2. Sistemas Multimídia. 3. Vídeo sob Demanda.
I. Instituto Militar de Engenharia. II. Título

CDD 004.6

INSTITUTO MILITAR DE ENGENHARIA

FABRICIO GUEDES BISSOLI

GERÊNCIA DE MEMÓRIA COOPERATIVA EM PROXIES PARA DISTRIBUIÇÃO DE VÍDEO SOB DEMANDA

Dissertação de Mestrado apresentada no Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientador: Prof. Edison Ishikawa – D.Sc..

Aprovada em 07 de agosto de 2006 pela seguinte Banca Examinadora:

_____

Prof. Edison Ishikawa – D.Sc. do IME – Presidente

_____

Prof. Cláudio Luis de Amorim - Ph.D. da UFRJ

_____

Prof. Ronaldo Moreira Salles - Ph.D. do IME

Rio de Janeiro
2006

A Deus.

A minha querida esposa, Ana.

A meus pais.

AGRADECIMENTOS

A Deus, a quem tudo devo.

A minha querida esposa, Ana Paula, companheira e amiga que muito me incentiva e compreende os momentos de ausência.

A meus pais, por sempre terem acreditado e investido em mim, mesmo nos momentos mais difíceis.

A Edison Ishikawa, por ter sido meu orientador e hoje um amigo. Sempre pude contar com sua sabedoria e paciência oriental até mesmo em seus momentos de lazer, pois sempre esteve presente e disposto a me ajudar.

A todos os amigos que sentiram minha ausência por esse período e souberam compreendê-la. Aos amigos que torceram por mim e deram força e principalmente aos que acreditaram!

Aos membros da banca, Cláudio Luis de Amorim e Ronaldo Moreira Salles, por terem aceitado o convite e honrar com suas presenças.

A professora Claudia Justel, pela ajuda que foi tão importante.

A Nelson Antonio Borges Garcia, pela disponibilização do cluster que foi fundamental para minhas simulações.

Aos professores Roberto Ades e Fernanda Campos, pelo especial voto de confiança.

Aos amigos Fabio Suim e Paulo Renato Costa Pereira, pelas vezes em que me ajudaram a distância em problemas técnicos durante meus trabalhos.

Nunca ande pelo caminho traçado, pois ele conduz somente até onde os outros foram.

Alexandre Graham Bell

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

CBR	Constant Bit Rate
CDN	Content Distribution Network
DVD	Digital Versatile Disk
GOF	Group of Frames
GOP	Group of Pictures
HD	Hard Disk
IEC	International Eletrotechnical Commission
ISO	International Organization for Standardization
LAN	Local Area Network
MAN	Metropolitan Area Network
MCC	Memória Cooperativa Colapsada
MPEG	Moving Picture Experts Group
NS	Network Simulator
ODPC	On Demandad Proxy Collaboration
RMI	Remote Method Invocation
RTP	Real-time Transfer Protocol
RTCP	Real-Time Control Protocol
SBTVD	Sistema Brasileiro de TV Digital
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VBR	Variable Bit Rate
VOD	Video On Demand

RESUMO

No momento em que a popularização de vídeo em DVDs e transmissões digitais para TV tornam-se realidade, uma outra tecnologia se aproxima do sucesso: o vídeo digital transmitido sob demanda, ou simplesmente VoD (Video on Demand). Sistemas que envolvem VoD necessitam de uma infraestrutura especial em relação à transmissão live (ao vivo, como ocorre nas TVs digitais) devido a um alto tráfego ocasionado por diversos pedidos simultâneos ao servidor. Por essa razão várias pesquisas buscam soluções para a redução de carga no servidor e largura de banda utilizada por clientes na dorsal da rede. Nesse trabalho é apresentada a proposta de um protocolo de gerência de memória cooperativa em proxies entitulado ODPC (*On Demand Proxy Collaboration*). Seu objetivo é gerenciar o buffer de dois ou mais proxies de maneira a formar um sistema colaborativo entre os mesmos.

ABSTRACT

At the time when video in DVDs becomes popular and transmissions for digital TV become reality, another technology approaches success: the digital video transmitted on demand, or simply VoD (Video on Demand). Systems that involve VoD needs a special infrastructure if compared to live transmission (as in the digital TVs) due to the high traffic created from simultaneous requests to the server. Because of that, some researchers try to find solutions for the load reduction in the server and bandwidth reduction used by clients in the dorsal of the network. This work is aimed at presenting the proposal of a cooperative memory management protocol in proxies named ODPC (On Demand Proxy Collaboration). Its objective is to manage the buffer of two or more proxies to create a cooperative system among them.

1 INTRODUÇÃO

1.1 CONTEXTO E MOTIVAÇÃO

No momento em que o governo brasileiro pronuncia o decreto que estabelece o sistema aberto brasileiro de TV Digital (SBTVD) e que a largura de banda oferecida a usuários residenciais ultrapassa 1 Mbps, alem da evolução dos codecs de áudio/vídeo propiciando qualidade em transmissões de *streaming* de vídeo via Internet, o termo “vídeo sob demanda” (VoD) ressurge, podendo oferecer bem mais que a precária qualidade do passado aliada à viabilidade técnica de um sistema ao alcance de usuários domésticos.

Sistemas de transmissão de vídeo digital com qualidade não são novidades, sendo inclusive que diversas empresas oferecem soluções comerciais há anos, geralmente em canais por assinatura. O SBTVD será um sistema aberto e com alguns incrementos que se refletem na qualidade da imagem, som e interatividade, mas continuará sendo um sistema de transmissão “ao vivo” (*live*). A diferença básica entre um sistema de vídeo sob demanda e um sistema *live* é que no primeiro caso o conteúdo gerado é difundido por broadcast apartir de uma única fonte, diferente de um sistema sob demanda onde o conteúdo é servido no momento de sua solicitação, o que ocasiona vários fluxos simultâneos na rede.

Os termos “vídeo sob demanda” e “ao vivo” podem ser confundidos, principalmente, quando recursos de hardware possibilitam certa flexibilidade de horários ao incorporar uma unidade de disco (HD) no dispositivo de recepção, possibilitando que determinada programação seja gravada e assistida posteriormente (como um videocassete). Um exemplo disso é a TIVo [TIVO], a qual oferece o aluguel de um equipamento capaz de gravar até 180 horas de vídeo em formato digital, possibilitando recursos de videocassete (retrocesso, pausa, etc).

O ATM Forum [ATM] define o Vídeo sob Demanda da seguinte forma:
"Serviço assimétrico que envolve várias conexões, transferindo informações de vídeo digital, comprimido e codificado, de um servidor (tipicamente um servidor de vídeo), para um cliente (tipicamente um "Set Top Terminal" ou um PC). No destino o vídeo é descomprimido, decodificado, convertido de digital para analógico e apresentado em um monitor."

A aplicabilidade de um sistema de VoD é ampla, estando associada a situações onde o horário e a própria programação (vídeo) são flexíveis, ou seja, o cliente escolherá o horário e conteúdo. O ensino à distância pode se beneficiar desse sistema, assim como o hábito de ir a uma locadora de vídeo e escolher um filme pode ser substituído pela comodidade de uma poltrona e alguns cliques em menus oferecidos por uma eventual "locadora de vídeo sob demanda". Aplicações em telemedicina, monitoramento remoto e muitos outros exemplos podem ser beneficiados com o VoD.

Em nível de distribuição, transmitir *streaming* de vídeo *live* requer menos infra-estrutura que no caso de sob demanda, pois em geral apenas um fluxo é criado no enlace do servidor de vídeo para atender a muitos clientes, enquanto que no VoD convencional seria necessário 1 fluxo por cliente, o que tornaria o sistema pouco escalável. Diversos trabalhos propõe soluções de infraestruturas que visam auxiliar a distribuição de VoD e aumentar sua escalabilidade, sendo que o ponto comum a todos é a reutilização de conteúdo que circula na rede através de sistemas de cache, seja a mesma dos clientes ou de seus respectivos proxies (formando uma CDN – *Content Distribution Network*). Algumas soluções serão apresentadas no capítulo 3.

1.2 OBJETIVOS

Reduzir a carga em servidores de vídeo e dorsal da rede através de uma política de reuso de conteúdos previamente fornecidos a outros usuários é a solução para aumentar o número de clientes atendidos e, com isso, a viabilidade de um sistema VoD. Alternativas tradicionais de cache para conteúdo estático (textos e imagens) não são adequadas para o caso de vídeo digital devido ao espaço ocupado pelo mesmo, sendo recomendável uma fragmentação do conteúdo para gerenciar o que se deve ou não armazenar em cache.

A contribuição deste trabalho é o desenvolvimento e a aplicação de um protocolo de gerência de conteúdo para distribuição de *streaming* de vídeo sob demanda através da colaboração de caches de proxies, objetivando a redução de solicitações ao servidor de vídeo e consequentemente aumentando a escalabilidade do sistema. O ponto de partida foi o protocolo de gerência de memória cooperativa colapsada (MCC) [ISHIKAWA, 2001], o qual trata principalmente da gerência de conteúdo local (em um *proxy*), sendo que o objetivo deste trabalho é estender essa pesquisa simulando uma colaboração entre

caches de *proxies* remotos. Para isso foi desenvolvido proposta de um protocolo de gerência de memória cooperativa em proxies entitulado ODPC (*On Demand Proxy Collaboration*).

1.3 ORGANIZAÇÃO DO TRABALHO

O texto está dividido em 5 capítulos, da seguinte forma:

- Capítulo 2: Introdução ao vídeo digital. Conceitos básicos de vídeo digital e alguns dos principais formatos atualmente utilizados. Definição de streaming de vídeo e como vídeo digital pode ser distribuído na Internet.
- Capítulo 3: Trabalhos relacionados: Soluções apresentadas para infraestrutura de uma CDN distribuindo vídeo sob demanda. Memória Cooperativa Colapsada: a base do protocolo proposto nesse trabalho.
- Capítulo 4: Proposta de protocolo colaborativo. Modelagem do problema, apresentação de protocolo de partida às simulações (*naive)* e o protocolo final ODPC (On Demanda Proxy Collaboration).
- Capítulo 5: Resultados obtidos com os protocolos naive e ODPC comparados ao caso de não colaboração de proxies.
- Capítulo 6: Conclusões e trabalhos futuros.
- Capítulo 7: Referências bibliográficas.

2 INTRODUÇÃO AO VÍDEO DIGITAL

Serão apresentados nesse capítulo os conceitos básicos de vídeo digital necessários para o completo entendimento do trabalho proposto, assim como alguns dos principais formatos relacionados ao padrão MPEG necessários à compreensão do presente trabalho.

Vídeo digital pode circular por uma rede através de download ou streaming. Quando o conteúdo não necessita ser entregue em “tempo-real”, a garantia de entrega de pacotes do TCP e a insensibilidade a *delays* e *jitters*, tornam o download a solução ideal. Todavia, caso seja necessário que o conteúdo seja transmitido em mídia contínua, o processo é chamado *streaming* e além de ser mais sensível a atrasos e variações de atrasos da rede, ainda está sujeito a admitir perdas de pacotes (com certa tolerância) oriundas de uma transmissão feita geralmente através do protocolo UDP. O escopo desse trabalho envolve streaming de vídeo, sendo que o caso de download não será abordado.

2.1 VÍDEO MPEG

MPEG é a sigla de *Moving Picture Experts Group*, um comitê formado sob o *Joint Technical Committee* de *International Organization for Standardization* (ISO) e *International Eletrotechnical Commission* (IEC), formado em 1988 sob a liderança de Leonardo Chiariglione (Italia) com o objetivo inicial de normatizar a codificação de vídeo para redes T1 e CD-ROM (aproximadamente a 1.5 Mbps). Essa norma ficou conhecida como MPEG-1 e desde então outras normas foram criadas, especificando áudio e vídeo para diferentes finalidades.

As especificações das normas MPEG descrevem ferramentas que podem ser utilizadas e trazem exemplos de como as mesmas podem ser implementadas, mas não definem o codificador/decodificador.

Um vídeo digital pode ser apresentado como uma seqüência de *frames* (ou imagens), sendo inclusive que o formato MJPEG (*Motion* JPEG) o trata dessa maneira. Todavia ao se tratar cada frame de maneira independente, não se está beneficiando de uma redundância chamada temporal, a qual parte do princípio de que *frames* consecutivos

podem apresentar grandes semelhanças. O MPEG considera esse fato e divide uma sequência de imagens em um subgrupo chamado GOP (*group of pictures*). O esquema completo de *layers* do MPEG pode ser visto na figura 2.1.

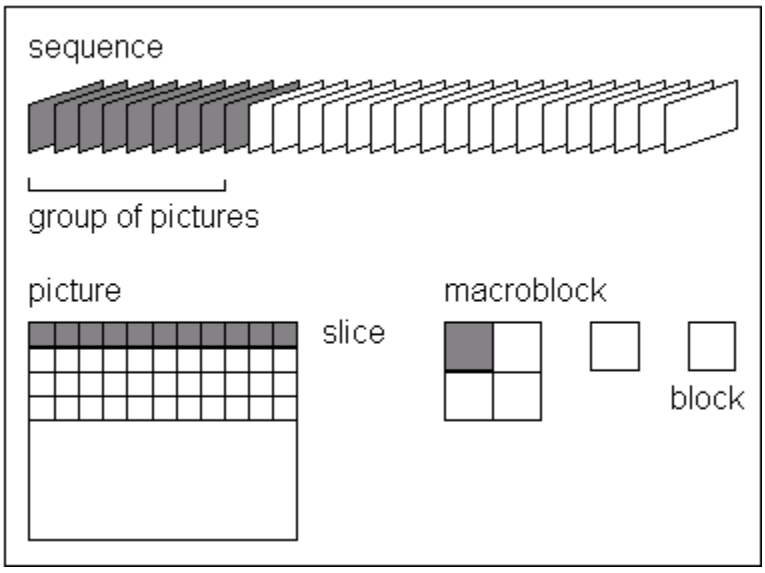
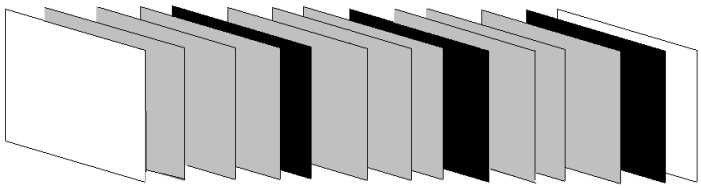


FIG. 2.1 – Esquema de layers do vídeo MPEG. O tamanho de um bloco (menor unidade processada) depende da norma (MPEG-1, 2, 4), mas terá um mínimo de 8x8 pixels [MPEG-1].

Um GOP possui geralmente entre 12 a 18 frames, sendo o primeiro frame do tipo I (*Intra coded*), sendo este um JPEG do quadro inteiro. Os demais *frames* de GOP visam utilizar-se da redundância temporal, ou seja, as semelhanças entre *frames* próximos temporalmente. Na norma MPEG existem ainda frames do tipo P (*Predictive coded*) e B (*Bi-directional coded*), sendo que o primeiro armazena apenas a diferença em relação ao I-frame anterior,enquanto que o B-frame faz a predição baseada em um I-frame (ou P-frame) anterior e um P-frame posterior. A figura 2.2 mostra uma sequência típica de frames de um GOP [SYMES, 2003].



I B B B P B B B P B B B P

FIG 2.2 – Exemplo de sequência de um GOP

É simples notar que um decodificador precisa receber o P-frame sucessor a um B-frame (além do I ou P anterior, obviamente) para que possa exibir tal frame bidirecional, pois o mesmo depende de dados “futuros” para ser montado.

Um vídeo MPEG pode ser CBR (*constant bit rate*) ou VBR (*variable bit rate*), dependendo se seu fluxo é constante ou não. Um vídeo CBR costuma sacrificar a qualidade a qualidade de cenas de ação extrema em pró de manter a taxa de bits constante, já que em tais cenas existe pouca redundância temporal e com isso a compressão é reduzida. Em uma rede de computadores é mais fácil trabalhar com vídeos CBR do que VBR pela facilidade de se estimar com mais precisão o tráfego na mesma. A figura 2.3 e 2.4 ilustram um comparativo entre o mesmo trecho de vídeo onde foram mantidas as mesmas configurações (exceto pelo CBR e VBR). Nota-se que mesmo no CBR existe uma pequena variação de fluxo, principalmente quando a cena muda (próximo ao frame 55).

O padrão MPEG-1 foi criado com a finalidade de oferecer uma qualidade equivalente ao VHS e um fluxo que não excedesse 1.8Mbps. Para o NTSC, a resolução utilizada é de 352x240 pixels a 30 fps (NTSC utiliza 29.97 Hz) e no caso do PAL, 352x288 pixels a 25 fps (PAL utiliza 25Hz). Sistemas de videoconferência utilizavam MPEG-1, sendo que atualmente seu uso ficou restrito a VCDs e aplicações específicas, já que a última atualização na norma foi em 1991 e novos padrões foram criados desde então

O MPEG-2 nasceu para oferecer mais flexibilidade e qualidade que o MPEG-1, ainda que a um custo computacional de cerca de 50% acima deste. Com um fluxo de 9.8Mbps, oferece resolução de 720x480 pixels para o NTSC e 720x576 para o PAL. Suas inovações em relação ao MPEG-1 são muitas, podendo-se citar codificação de cor (relação entre luminância e crominâncias) no formato 4:2:2 e 4:4:4 além da 4:2:0 ,

quantização para o coeficiente DC (coeficiente 0,0 do bloco de 8x8 após transformada discreta de cossenos – os demais blocos referenciam a esse) de um bloco DCT podendo chegar a 9 ou 10 bits em certos perfis (o MPEG-1 aceita somente 8 bits), diferentes *aspect ratios* (formatos de tela, por exemplo 4:3 e 16:9) e outras. O padrão MPEG-2 está presente nos atuais DVDs e nos sistemas de TV digital americano (ATSC) [www.atsc.org], europeu (DVB) [www.dvb.org] e japonês (ISDB) [www.nhk.or.jp/strl/open99/bs-1/].

O projeto da norma MPEG-3 foi iniciado com o objetivo de atender a sistemas de TV de alta definição (HDTV), mas abandonado por se concluir que o MPEG-2 já atendia às necessidades dos mesmos.

Em 1993 iniciou-se o projeto do MPEG-4, o qual foi formalizado em julho de 1995. Inicialmente o objetivo era codificar vídeo a taxas muito baixas, mas a norma acabou especificando três faixas de banda: abaixo de 64Kbps, de 64Kbps a 384Kbps e de 384Kbps a 4Mbps. Devido à sua flexibilidade, é possível utilizar MPEG-4 em *profiles* e *levels* que chegam a 1.2Gbps (qualidade de estúdio). Uma característica inovadora do MPEG-4 é a possibilidade de separar objetos de áudio e vídeo, gerando possibilidades interessantes. Cada objeto pode ser codificado de maneira diferente, sendo posteriormente unificado pelo decodificador. Essa proposta pouco exequível deu lugar a uma nova versão denominada H.264.

H.264 é o nome (publicado pela *International Telecommunications Union* - ITU-T) ao trabalho desenvolvido pela J*oint Video Team* (JVT). O mesmo trabalho recebeu o nome de MPEG-4 *Part* 10 e representa um grande passo em relação a compressão de vídeo. Possui uma compressão até 50% superior comparada ao H.263v2 ou MPEG-4. Mostrou-se bastante flexível em aplicações de streaming de vídeo e foi escolhida para ser o codec de vídeo para o sistema aberto brasileiro de TV Digital (SBTVD).

Apesar de o padrão H.264 ser mais eficiente que o MPEG-2 para streaming de vídeo, foi utilizado trace de vídeo no formato MPEG-2 nas simulações desse trabalho (visto em capítulos posteriores) devido à possibilidade de estabelecer-se comparativos com trabalhos relacionados mais antigos, todavia não existem restrições quanto a utilização do sistema aqui sugerido com outros padrões de vídeo diferentes do MPEG-2.

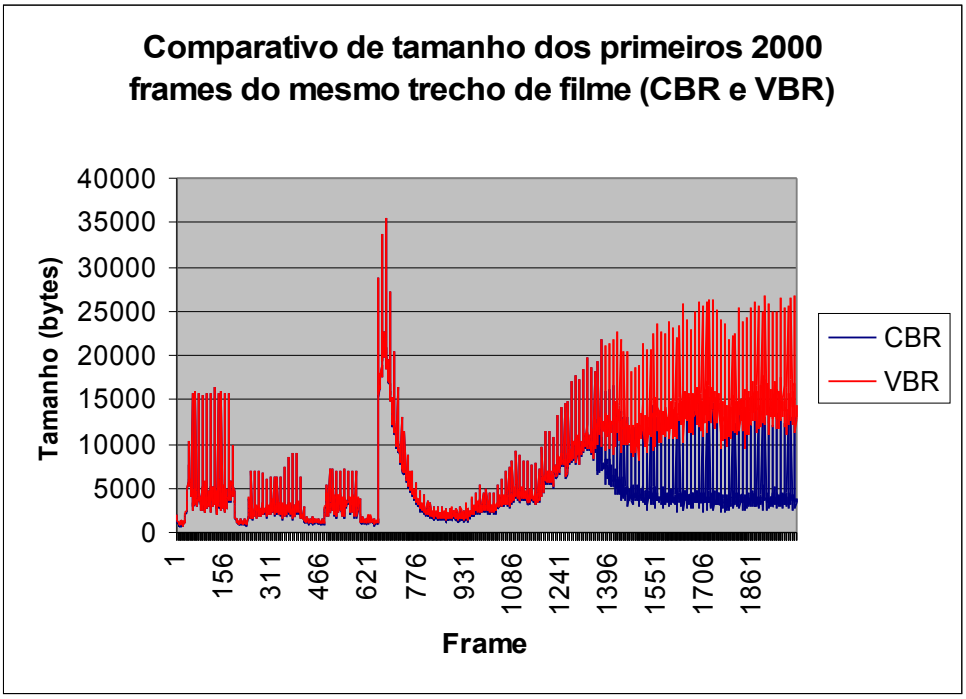


FIG 2.3 - Um trecho de 2000 frames de um vídeo MPEG-4

As figuras 2.3 e 2.4 representam o tamanho em bytes de cada unidade considerada (frame na 2.3 e GOP na 2.4). Apesar de um vídeo CBR não possuir tamanho constante, sua variação é bem inferior ao VBR. As maiores oscilações de tamanho ocorrem principalmente quando se dá uma mudança de cena, o que ocorre quando não existe nenhuma redundância temporal em relação ao último frame de referência da cena anterior.

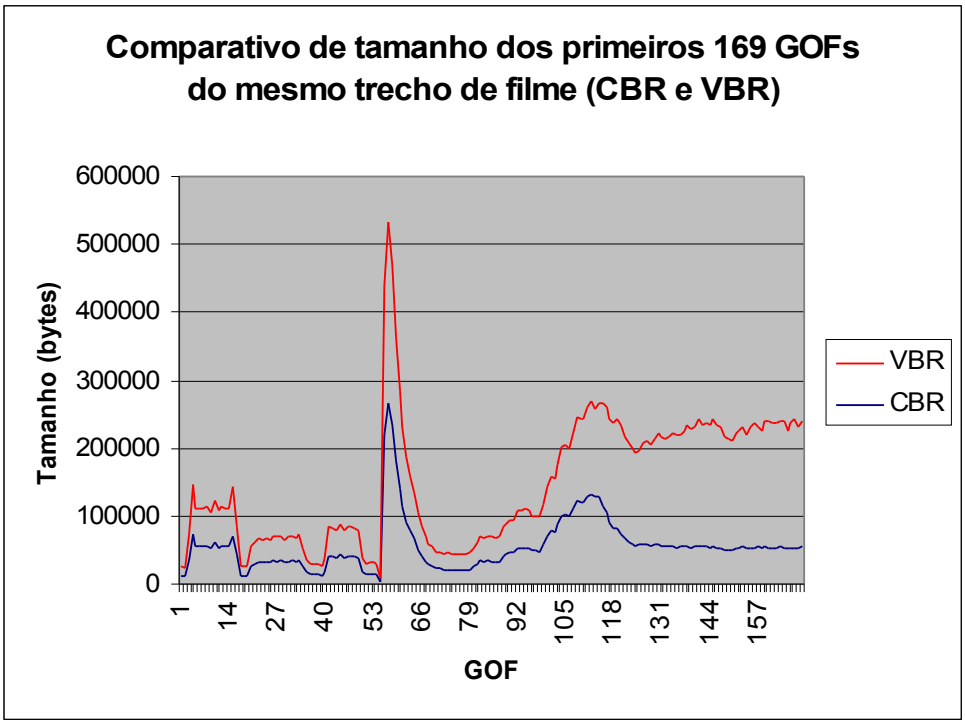


FIG 2.4 – O mesmo trecho, comparado por GOPs (ou GOFs) de 12 frames.

2.2 VISAO GERAL PARA *STREAMING* DE VÍDEO

A dificuldade em distribuir conteúdos contínuos via Internet não está associada apenas à disponibilidade de largura de banda, mas também aos tipos de serviços que a rede pode oferecer. A internet oferece um serviço considerado do tipo *best effort* (melhor esforço), o qual não é recomendável para a transmissão de dados contínuos como *streaming* de áudio e vídeo. Nesse tipo de mídia as informações são produzidas em tempo real e geram uma quantidade de dados a cada instante, sendo que o sistema de transporte, ou seja, a rede, deve ser capaz de transportar os dados continuamente da origem ao destino.

Nesses casos os dados devem ser tratados como *stream*, ou seja, um fluxo, e não como arquivo. *Streaming* é aplicável a mídias contínuas e utilizado tipicamente para transmissão de áudio e vídeo [TANENBAUM, 2003]. A continuidade da mídia faz com que o fluxo de dados necessitem ser transportados através de uma seqüência de pacotes enviados pela rede. No *streaming,* o cliente inicia o *playout* da mídia (seja áudio ou vídeo) antes da transmissão completa dos pacotes pelo servidor.

Em geral, sistemas de *streaming* operam sobre o protocolo UDP, o qual não garante integridade da transmissão e nem a ordem de chegada. Esse fato torna a transmissão mais rápida, pois dispensa troca de informações de controle, mas gera uma necessidade de reconstituir a ordem e adotar políticas de tolerância a erros em nível de aplicação. O erro é tolerável em certo nível, pois os dados transferidos são percebidos por um ouvinte (geralmente humano), o qual admite pequenas perdas.

Como dito anteriormente, uma vez que o controle de ordem e integridade não foi efetuado em nível de transporte, o mesmo deverá ser feito em nível de aplicação. A desordem produzida deve-se ao fato de que pacotes podem levar tempos diferentes para chegar ao mesmo destino. Este efeito é típico do serviço de melhor esforço do IP, e não é corrigido em nível de transporte pois o mesmo não é tratado no protocolo UDP. A variação estatística do atraso da transmissão é chamada *jitter*.

O mecanismo de uma transmissão contínua pode ser resumida da seguinte maneira:

- O cliente solicita determinado *streaming* (áudio ou vídeo).
- O servidor inicia transmissão do fluxo (enviando pacotes via UDP).
- O cliente recebe o primeiro pacote e antes de exibí-lo ao ouvinte, aguarda um certo período no qual tentará acumular pacotes com a finalidade de criar um *buffer*, tentando desta maneira previnir os efeitos de *jitter*.

O cliente deverá dispor de um espaço de memória, no qual armazenará os pacotes referentes ao *buffer* antes do *playout*. Tipicamente um *buffer* é uma estrutura de dados que é preenchido de um lado e esvaziado de outro, podendo sofrer variações em seu nível provenientes da ordem de chegada, atraso e variação de atraso dos pacotes recebidos. A dimensão de um *buffer* estará associada ao tempo de espera, ou seja, quanto maior o *buffer*, maior o tempo de espera para o início do *playout*, todavia se obtém um melhor controle do jitter em nível de aplicação dos pacotes recebidos com *buffers* maiores.

2.3 ARQUITETURAS PARA DISTRIBUIÇÃO DE VÍDEO EM UMA REDE

Conforme citado anteriormente, a distribuição de vídeo por download foge ao escopo desse trabalho, sendo que apenas a questão de streaming de vídeo será tratada.

Uma streaming de vídeo pode circular em uma rede via unicast, broadcast ou multicast. Unicast conecta um ponto a outro, em nosso caso, um servidor de vídeo ao cliente. Tal arquitetura é ineficiente para a distribuição de vídeos a vários clientes, pois o enlace do servidor rapidamente seria sobrecarregado. *Broadcast* é uma solução aplicável em casos específicos de uma LAN (e sistemas de radiodifusão), mas não considerável em nível de Internet devido a impossibilidade de se efetuar broadcast externamente, pois se todos os roteadores efetuassem broadcast na Internet, poderia haver um grande tráfego de dados inútil para a maioria. *Multicast* considera a comunicação de um ponto para outros que compartilhem o IP multicast. *Multicast* pode ser uma alternativa interessante na distribuição de streaming de áudio e vídeo *live*, mas sozinha não resolve o problema de vídeo sob demanda, uma vez que o conteúdo recebido por cada cliente pode não ser o mesmo (mesmo vídeo em posições diferentes ou até vídeos diferentes).

A idéia de se criar arquiteturas que viabilizem a distribuição de VoD pela Internet não é recente, diversos trabalhos (citados no capítulo 3) procuram criar uma infraestrutura que viabilize a distribuição de VoD, inclusive originando um subtermo chamado *near* VoD, o qual se difere do *true* VoD pelo fato de que clientes podem ter de esperar até alguns minutos para serem atendidos, pois o sistema pode gerar uma lista de clientes que tenham escolhido o mesmo vídeo para, através de um único fluxo multicast, atendê-los simultaneamente.

O capítulo 3 apresenta diversas soluções propostas, sendo que as mais complexas envolvem uma rede de distribuição de conteúdo (CDN).

3 TRABALHOS RELACIONADOS

Como foi citado, sistemas que envolvem VoD necessitam de uma certa infraestrutura para que seja reduzido o consumo de largura de banda do servidor e, consequentemente, mais clientes possam ser atendidos. No ítem 3.1 serão apresentadas soluções relacionadas a esse tópico. Na seção posterior (3.2) é apresentado

resumidamente o fundamento da memória cooperativa colapsada [ISHIKAWA, 2003], a qual é a base das soluções apresentadas nesse trabalho.

3.1 PROPOSTAS EXISTENTES PARA STREAMING DE VÍDEO

Uma solução existente é o chamado **vídeo quase por demanda** (*near video on demand*), onde o servidor criaria fluxos periódicos, fazendo com que clientes tenham de aguardar certo tempo (alguns minutos em geral) após ter solicitado o vídeo para que a transmissão se inicie. Dessa maneira o servidor conseguiria criar um único fluxo para atender a diversos clientes. Diversos trabalhos surgiram de maneira a incrementar essa técnica e oferecer mais benefícios, dentre os quais podemos citar: *Batching* [DAN, 1996], *Patching* [HUA, 1998], *PiggyBacking* [GOLUBCHIK, 1996], *Chaining* [SHEU, 1997].

No *Batching* os pedidos são enfileirados e atendidos após algum tempo (por multicast). O maior problema é a latência a que estão submetidos os clientes que chegaram no início do lote.

No caso do *Patching*, onde é suposto que a capacidade de *upstream* do cliente suporta o mesmo fluxo do vídeo e também que o enlace de *downstream* seja capaz de receber o dobro do fluxo do vídeo. O primeiro cliente seria o que receberia o fluxo completo do servidor; clientes posteriores solicitariam o fluxo a clientes que já o tivessem, sendo que no caso de o início do filme já ter sido perdido, ele seria solicitado diretamente ao servidor numa espécie de remendo (*patch*), o qual seria de curta duração se comparado à duração total do filme.

O *Piggybacking* possui uma visão bastante diferente, pois tenta reduzir fluxos similares (mas defasados por poucos minutos de exibição) adiantando a exibição do que entrar posteriormente em até 5% (o que seria imperceptível aos humanos) e quando atingisse a sincronia, poderia reduzir dois fluxos a um único.

Para aproveitar a capacidade de buffers de clientes, o *Chaining* cria um encadeamento onde clientes contribuem com fluxos a outros clientes. Os problemas desse sistema são a escalabilidade (tolerância a falhas) e a incapacidade de reutilização de fluxos já perdidos.

Outros trabalhos sugerem uma rede de distribuição de conteúdo (*Content Distribution Network – CDN*), onde proxies auxiliariam a distribuição de vídeos através de

gerência sobre sua cache. Sistemas tradicionais de gerenciamento de cache mostraram-se ineficazes para o caso de distribuição de *streaming* de vídeo, por essa razão novos trabalhos surgiram com o intuito de gerenciar a cache do proxy de maneira adequada e proporcionar maior economia de fluxos do servidor. Na arquitetura de CDN, podemos citar como exemplos os trabalhos relacionados a VoD [MA, 2000], [PARK, 2001], [REJAIE, 2001], [ZHU, 2003], [WANG, 2002], [PARK, 2000] além do trabalho [ISHIKAWA, 2003], o qual foi o pilar para o trabalho aqui proposto.

O trabalho entitulado *Dynamic proxy-cache multiplication inside LANs* [COBARZAN, 2005] foi publicado recentemente e propõe um sistema bastante semelhante, onde a cache de dois *proxies* colaboram para distribuição de VoD, todavia trabalha com uma política de tempo de último acesso e não com atribuição de custos de memória e fluxos. Por falta de informações no artigo, não foi possível recriar o sistema para estabelecer-se um comparativo, o que seria bastante razoável.

3.2 ESTRUTURA DA MCC

Os conceitos fundamentais da memória cooperativa colapsada foram mantidos nesse trabalho, como vetor de vídeo, vetor de slots, buffer colapsado e tipos de slots.

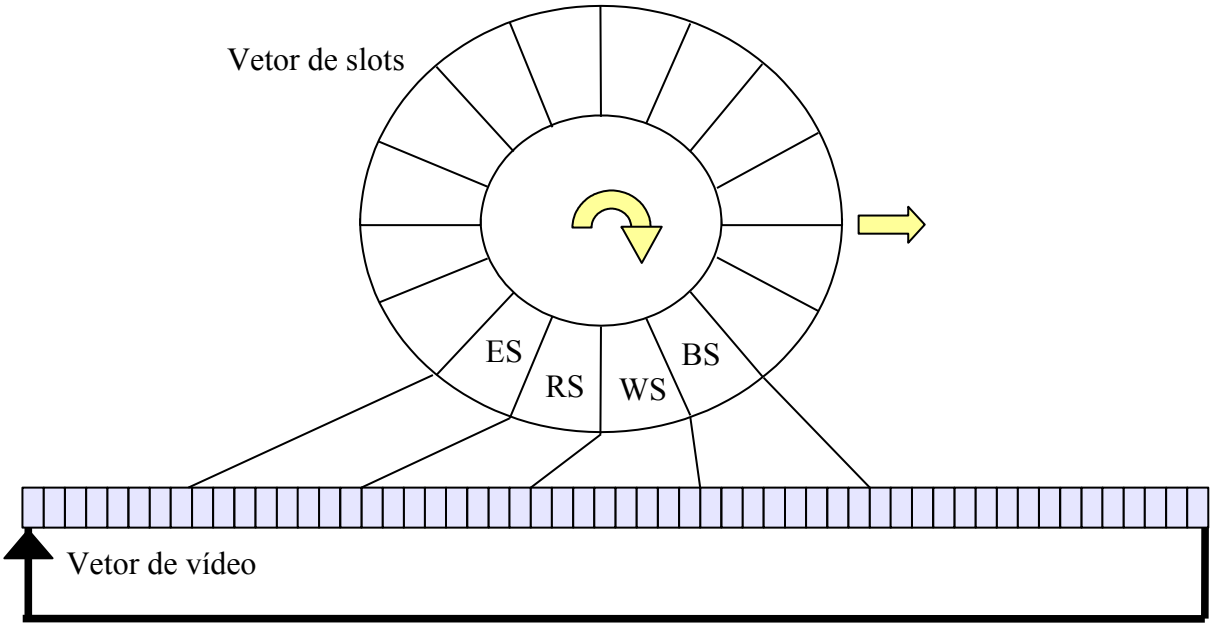


FIG 3.1 – Vetor de slots se projetando sobre o vetor de vídeo

De maneira resumida a estrutura de dados funciona da seguinte forma: existem duas estruturas utilizadas na MCC, designadas por vetor de vídeo e vetor de slots, ambas implementadas de maneira circular. A primeira contém apenas metadados referentes ao mapeamento da memória, ou seja, onde se encontra determinada unidade de vídeo. O vetor de slots se projeta sobre o vetor de vídeo mapeando as unidades de vídeo que o compõe. É composto de slots de tamanho fixo e serve para alocar buffers colapsados, podendo ter o número de slots reservados pela MCC. A figura 3.1 ilustra de maneira simplificada o vetor de slots se projetando sobre o vetor de vídeo. Um buffer colapsado tem ponteiros associados ao vetor de slots, os quais indicam o início, fim e limites de *underflow* e *overflow*.

Um *slot* é o menor fragmento de vídeo tratado pela MCC e contém 8 segundos de duração (estipulado a partir de conclusões obtidas em várias simulações da tese utilizada como referência). Um buffer colapsado terá um tamanho mínimo de 4 *slots*, sendo que eventualmente pode conter *slots* de ligação, os quais unem cadeias com o objetivo de atender mais clientes no mesmo *buffer* colapsado. A figura 3.2 ilustra um *buffer* colapsado mínimo e um *buffer* colapsado com *slots* de ligação (*link slots* – ls). Um maior detalhamento pode ser encontrado em [ISHIKAWA, 2003].

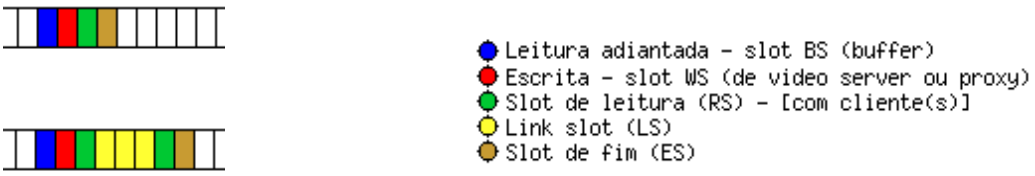


FIG 3.2 – Buffer colapsado mínimo e com 3 slots de ligação (ls)

O vetor de vídeo se projeta sobre o vetor de slots de maneira que o conteúdo do *buffer* colapsado é alterado dinamicamente, mas o número do *slot* (associado no vetor de *slots*) não se altera. Um cliente que esteja associado a um determinado *slot* de leitura (RS) permanecerá no mesmo até o final do vídeo, a não ser que utilize recursos de videocassete (pausa, retrocesso, etc).

4 NAIVE E ODPC

Nesse capítulo serão apresentados os protocolos de gerência de memória coopertiva entre proxies designados por *naive* e ODPC. A modelagem do problema é apresentada inicialmente e uma proposta de simplificação apresentada a seguir.

4.1 COLABORAÇÃO ENTRE PROXIES

4.1.1 PROPOSTA DE COLABORAÇÃO

Considerando-se que exista um *proxy* que atenda a clientes que podem, eventualmente, fazer requisições a um servidor de vídeo sob demanda geograficamente distante e escolher determinado conteúdo. É lógico imaginar a utilização da memória deste *proxy* como sistema de cache para armazenar esse tipo de conteúdo (integralmente ou parcialmente).

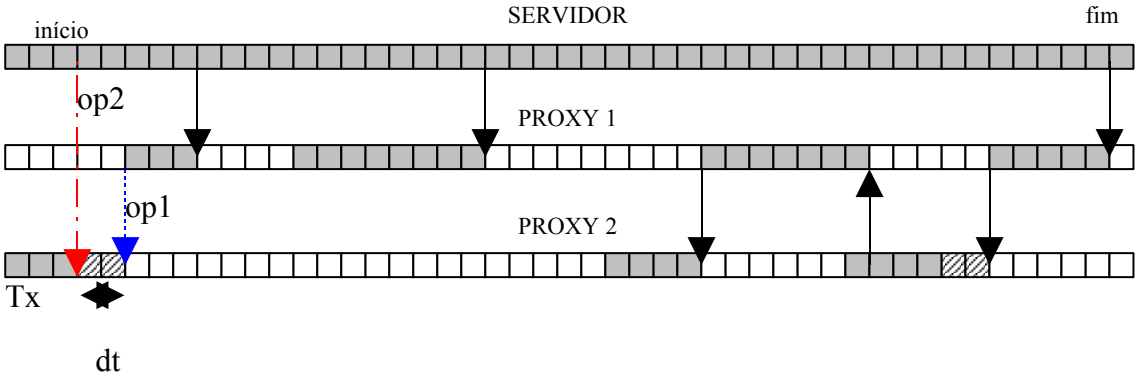
Se houvesse memória infinita na cache do referido *proxy*, o sistema funcionaria sem maiores problemas, pois uma vez que determinado vídeo tivesse sido solicitado por um cliente, seria armazenado na memória do *proxy* e para atender a requisições futuras (do mesmo conteúdo) de outros clientes, não haveria necessidade de uma nova solicitação ao servidor de vídeo. Esse tipo de sistema de cache é aplicável a conteúdos estáticos de tamanhos muito inferiores ao da cache, mas no caso de *streaming* de vídeo de alta qualidade, os quais possuem tamanho significativo em relação ao tamanho da cache, e se considerando que poderão existir centenas de vídeos no servidor, o armazenamento de um conteúdo completo em *proxies* torna-se pouco viável, sendo que a solução é o armazenamento de “fragmentos” desses vídeos. Diversos trabalhos apresentam soluções para a gerência interna de tais fragmentos na cache de *proxies* [FAHMI et al, 2001] [ZHU et al, 2003] [WANG et al, 2002] [ISHIKAWA, 2003], mas poucos tratam da questão de ser viável uma colaboração entre dois *proxies* de uma MAN [COBARZAN, 2005], desde que exista um enlace suficientemente grande entre os mesmos (o que pode ocorrer numa MAN).

Um sistema distribuído possui naturalmente uma maior complexidade e, no caso considerado, acrescentado o fato de que o conteúdo é totalmente dinâmico, tornando a questão da gerência da memória cooperativa uma tarefa que pode consumir tantos recursos computacionais que inviabilizaria o sistema.

Por isso faz-se necessária a busca de um sistema capaz de gerenciar a cooperação em tempo real, ainda que a solução adotada não seja a ótima (mas que se aproxime da mesma).

4.1.2 MODELAGEM DO PROBLEMA

Instância do problema: Um conjunto de M proxies, podendo cada um conter trechos de vídeos que mudam ao longo do tempo (por questões de simplificação da modelagem, considerar-se-á apenas um vídeo, podendo o mesmo estar fragmentado ao longo de diversos proxies e completo no servidor). O objetivo é reduzir a carga do servidor através de uma colaboração entre proxies, os quais passariam a fornecer uns aos outros, fluxos já contidos em suas caches e novos conteúdos que lhes estão sendo fornecidos continuamente (em direção ao término do filme).



Legenda:

- Slot vazio (área de memória livre).
- Slot de ligação (*link slot*) ocupado para poupar novo fluxo do servidor.
- Slots mínimos ocupados pela MCC.

Notas:

- 1 slot contém 8 segundos de vídeo (de acordo com a definição da MCC [ISHIKAWA, 2003]).
- op1 e op2 representam as duas opções para se atender a determinado cliente do proxy 2, cuja distância temporal é de 2 slots.
- dt indica a "distância temporal", ou seja, a quantidade de slots entre a posição de leitura considerada e o próximo trecho analisado. Se for optado pela solicitação a um proxy que possua dt > 0 em relação ao trecho considerado, os slots vazios são reservados, tornando-se slots de ligação (*link slots* - ls) e será necessário um *patch* (remendo) do servidor com a duração dos ls reservados.

FIG 4.1: Representação da cooperação entre dois proxies em seus slots de vídeo.

Ao se buscar reduzir a carga do servidor encaminhando solicitações a outros proxies, deve-se buscar, dentre várias opções disponíveis, a melhor solução (ou alguma próxima a

essa). Custos são originados basicamente pelo consumo de largura de banda e alocação de memória na cache do *proxy* que origina o pedido. Buscar um algoritmo que seja capaz de estimar uma boa opção em tempo real (online) é um dos objetivos do trabalho aqui apresentado.

A figura 4.1 representa de maneira simplificada duas possibilidades de atendimento a um (ou mais) cliente(s) do *proxy* 2, ali indicadas por *op1* e *op2*. O trecho *Tx* está distante 2 slots do trecho mais próximo (sempre analisado em direção ao fim do vídeo), contido no *proxy* 1. Nesse caso existirão duas opções: alocar mais 2 slots de memória no proxy 2 para receber o fluxo do *proxy* 1 enquanto se solicita um remendo (patch) do servidor com duração de 2 slots ou solicitar tudo diretamente ao servidor, economizando memória no *proxy* 2, evitando que em dado momento existam 2 fluxos para o mesmo trecho (*patch* + fluxo normal), mas não reduzindo carga no enlace do servidor. O cálculo dos custos, apresentado posteriormente, depende não somente da análise local de cada caso, mas também de uma visão global, a qual leve em conta a posição do trecho (em relação ao tempo de duração total do filme), taxa de utilização de memória dos proxies, taxa de utilização dos enlaces e eventualmente a possibilidade de realocar memória com a finalidade de buscar custo global mínimo.

Por razões de simplificação, as simulações apresentadas por este trabalho considerarão clientes assistindo vídeos do início ao fim (o que é bastante razoável em termos práticos, pois em um sistema real o cliente estaria pagando para assistir ao vídeo e raramente desistiria antes de seu término). Outra consideração visando simplificação é que não serão simuladas interações de clientes com o fornecimento do vídeo (recursos de videocassete – retrocesso, avanço rápido, pausa, etc.).

A figura 4.2 mostra uma tomada de decisão levando em conta uma visão global do sistema: um cliente entrou no *proxy* 2 e solicitou determinado vídeo que estava a somente 2 slots adiante no *proxy* 1. O fluxo Fs poderia ser reduzido a um remendo de duração 5 slots caso o *proxy* 1 colaborasse com o fluxo (2), mas para isso seria necessário alocar mais 2 slots na memória do *proxy* 2. Se a memória do *proxy* 2 já estiver saturada, uma visão local levaria à solicitação do fluxo completo ao servidor. Todavia o algoritmo com visão global poderia detectar que próximo ao final do vídeo existe uma colaboração do *proxy* 1 para o *proxy* 2 (1), a qual demanda 2 link slots de memória para existir.

Considerando que essa colaboração esteja próxima do fim do vídeo, é provável que nessa situação torne-se mais interessante criar um fluxo direto do servidor (3) e liberar a memória previamente ocupada pelos dois *link slots*, reutilizando-a para criar os 2 *link slots* necessários para que Fs seja apenas um remendo e não um fluxo com a duração integral do vídeo.

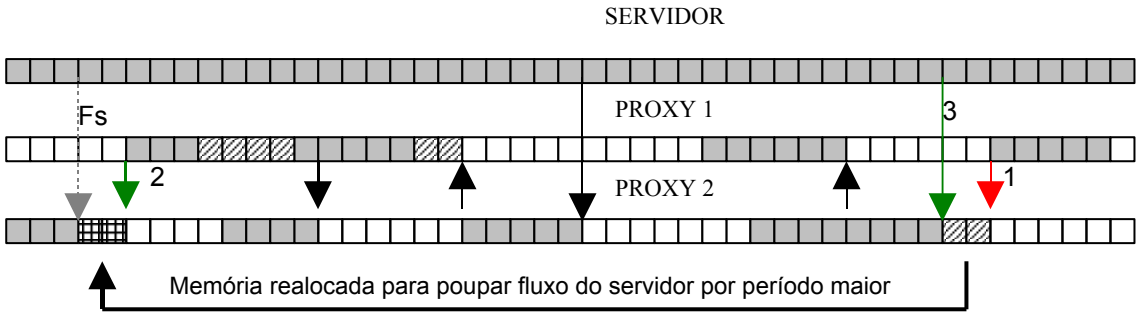


FIG 4.2 – Tomada de decisão envolvendo realocação de memória.

A modelagem inicial será em função desse sistema (visão global), sendo que deve-se buscar um algoritmo que seja capaz de aproximar da solução ótima em tempo polinomial. Posteriormente será apresentada uma modelagem simplificada, de visão local (sem realocação de memória). Um comparativo entre essas duas soluções será apresentado posteriormente.

Tomando-se uma visão global do sistema, pode-se escrever o seguinte:

$$C_{mínimo} = (X_{a1} * Op1 \ \vee X_{a2} * Op2) + ... + (X_{an} * Op1 \ \vee X_{an+1} * Op2) \qquad (1)$$

onde:

X_{a1} a X_{an+1} são variáveis booleanas.
Op1 e Op2 representam as opções referentes à solicitação de fluxo direto do servidor ou fluxo daquele *proxy* (+ *patch* do servidor) em relação ao trecho de vídeo analisado.

Considerando os 4 tipos de custo de cada trecho separadamente, tem-se que:

$$C_{mínimo} = \min \sum_{i=1}^{\text{num de trechos}} (x_{ai}c_{fi} + x_{bi}c_{mi} + x_{ci}c_{si} + x_{di}c_{pi}) \qquad (2)$$

Onde $\boldsymbol{c_{fi}}$ é o custo originado pelo consumo de largura de banda de determinado vídeo no proxy ***i***, $\boldsymbol{c_{mi}}$ o custo originado pela alocação de memória na cache do proxy que

origina o pedido,c_{si} o custo proveniente do fluxo do servidor (fluxo completo, até o final do vídeo) e c_{pi} o custo do *patch* (remendo) do servidor. Sejam x_{ai}, x_{bi}, x_{ci} e x_{di} variáveis booleanas a serem determinadas em função da minimização do somatório de trechos de um vídeo em determinado proxy.

Para o cálculo dos custos, considerar-se-á:

A capacidade do enlace é finita, ou seja, se um fluxo de 1,5Mbps ocupa 15% de um enlace de 10Mbps, faz sentido avaliar a taxa de utilização do enlace. Essa taxa de utilização é definida por:

Txe = (1 - largura de banda consumida no enlace) / largura de banda total do enlace.

Outra taxa de utilização a ser considerada diz respeito à memória. Assim como foi definida a taxa de utilização do enlace, pode-se definir a taxa de utilização da memória, de forma que a mesma pode ser escrita como:

Txm = (1- memória alocada no *proxy*) / total de memória do *proxy*.

Dessa forma, tem-se que:

cf = [largura de banda consumida pelo vídeo (em Mbps)] * [tempo de duração do fluxo (quantidade de slots **até o fim do vídeo** * duração em segundos de cada slot)] * [abs(1- largura de banda consumida no enlace do *proxy*) / largura de banda total do *proxy*]

cm = [quantidade de slots a serem reservados pela MCC / 2] * [quantidade de slots do ponto médio do trecho até o fim do vídeo] * [abs(1 – quantidade de slots de memória ocupados no *proxy* que fez o pedido) / total de slots do buffer do *proxy* que fez o pedido]

cs = [largura de banda consumida pelo vídeo (em Mbps)] * [tempo de duração do fluxo (quantidade de slots **até o fim do vídeo** * duração em segundos de cada slot)] * [abs(1- largura de banda consumida no enlace do servidor) / largura de banda total do servidor]

cp = [largura de banda consumida pelo vídeo (em Mbps)] * [tempo de duração do fluxo (quantidade de slots **até o instante necessário** * duração em segundos de cada slot)] * [abs(1- largura de banda consumida no enlace do servidor) / largura de banda total do servidor]

Conforme indicado em (1) e (2), ter uma visão global significa que a escolha de uma determinada opção pode influenciar em diversas outras opções previamente selecionadas, ou seja, para tornar o somatório mínimo, todos os trechos de vídeo presentes naquele *proxy* deverão ser considerados. As variáveis booleanas x_{ai}, x_{bi}, x_{ci} e x_{di} servem para calcular o custo do trecho analisado, indicando a existência (ou não) dos custos *cf*, *cm, cs e cd* para cada opção. A partir dessa informação, a escolha entre pedido de fluxo ao *proxy* e/ou servidor poderá ser tomada de maneira otimizada.

Como se pode notar, para cada trecho existirão 4 variáveis a serem determinadas. O número de soluções a considerar cresce exponencialmente com o número de variáveis, sendo que existirão $4*2^i$ soluções possíveis, sendo i o número de trechos de determinado vídeo presentes no *proxy*.

$$C_{mínimo} = \min_{i=1}^{\text{num de trechos}} \sum \left(x_{ai}c_{fi} + x_{bi}c_{mi} + x_{ci}c_{si} + x_{di}c_{pi}\right)$$

$1 \leq i \leq$ número de trechos do vídeo no *proxy*

x_{ai}, x_{bi}, x_{ci} e $x_{di} = 0$ ou 1

Pode-se acrescentar algumas restrições ao problema:

$(x_{ai} + x_{bi} + x_{ci} + x_{di}) \leq 3$

$(x_{ai} + x_{bi} + x_{ci} + x_{di}) \geq 0$

se $dt > 0$ então $x_{ci} = 1$ (distância temporal > 0 implica em patch do servidor)

se $dt = 0$ então $x_{ci} = 0$ (distância temporal = 0 indica que todo conteúdo pode ser conseguido sem a necesidade do servidor)

Quando não houver mais memória no *proxy* que originou o pedido, o que indicaria uma taxa de utilização de memória máxima (quantidade de slots alocados igual à

quantidade de slots totais de memória), um cliente que entrasse no sistema teria pedido negado a não ser que fosse liberada memória através de realocação de fluxos que atendem a outros trechos. O primeiro passo seria encontrar no mínimo 4 slots disponíveis para se criar um novo buffer colapsado, enquanto é verificado se o enlace do servidor conseguiria fornecer mais um fluxo (caso a distância temporal até o próximo trecho seja maior que zero). A origem dessa memória liberada seria de link slots que proporcionassem fornecimento de fluxo a partir de outro *proxy* a um trecho preexistente, sendo que no caso de haver liberação de memória, o fluxo preexistente passaria a ser fornecido pelo servidor ou eventualmente de outro *proxy* capaz de atendê-lo.

Seja uma cache em determinado *proxy* capaz de armazenar 12 minutos de vídeo no total (ou 60 x 12 / 8 = 90 slots). Esses 12 minutos representam 20% de um vídeo de 60 minutos, considerado em nossas simulações. O objetivo aqui é provar a complexidade de um algoritmo com visão global e a seguir propor uma solução simplificada, capaz de resolver em tempo real o problema de escolha de trechos em proxies.

A situação apresentada na figura 4.3 é hipotética, sendo que a realidade poderia assumir complexidade maior, pois não somente a quantidade de proxies poderia ser superior a 2, quanto o tamanho da cache poderia ser maior que 90 slots, como também poderia existir mais de 1 título de vídeo e até centenas de clientes poderiam estar presentes no sistema simultaneamente, o que se traduziria em um número mais elevado de trechos referentes ao mesmo vídeo.

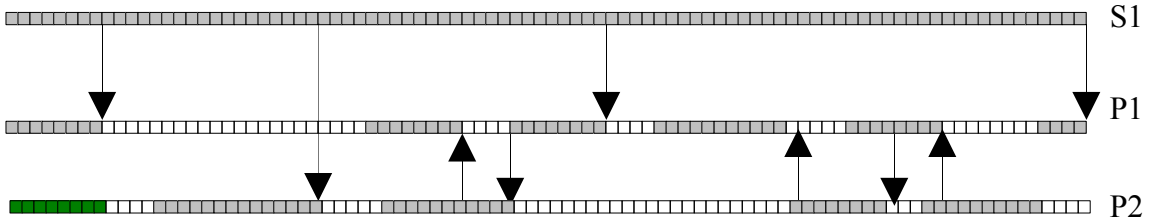


FIG 4.3 – Hipótese de um vídeo consumido por clientes em 2 proxies.

Sejam P1 e P2 proxies atendendo a seus respectivos clientes e eventualmente colaborando um com o outro e seja S1 o servidor de vídeo. Para efeito de demonstração numérica do algoritmo, serão considerados os seguintes parâmetros:

S1: Largura de banda do enlace: 10Mbps

P1: Largura de banda do enlace: 10Mbps

Memória total para cache da MCC: 90 slots (cada slot equivale a 8 segundos de vídeo).

P2: Largura de banda do enlace: 10Mbps

Memória total para cache da MCC: 90 slots.

De acordo com a figura e os parâmetros propostos, considerar-se-á que um novo cliente entra no sistema através do *proxy* P2. Ele deverá alocar um mínimo de 8 slots de memória, conforme definido na MCC. O algoritmo de custo mínimo é acionado nos proxies 1, 2 e no servidor.

Considerando-se um cliente chegando no *proxy* **P2** (indicado pelo buffer colapsado mais à esquerda de P2), calcula-se, como exemplo o que ocorre em termos de melhor escolha para o *proxy* **P1**:

$$C_{mínimo} = \min_{i=1}^{6} \Sigma \left(x_{ai}c_{fi} + x_{bi}c_{mi} + x_{ci}c_{si} + x_{di}c_{pi} \right)$$

x_{ai}, x_{bi}, x_{ci} e $x_{di} = 0$ ou 1

$(x_{ai} + x_{bi} + x_{ci} + x_{di}) \leq 3$

$(x_{ai} + x_{bi} + x_{ci} + x_{di}) \geq 0$

se $dt > 0$ então $x_{ci} = 1$

se $dt = 0$ então $x_{ci} = 0$

Σ (GOFs atendidos pelos 3 tipos de fluxo) = total de GOFs necessários

Calculando-se os custos, tem-se:

cf1 = 1.5 * 90 * 8 * abs((1 - 3) / 10)	⇒	cf1 = 216
cm1 = (8 / 2) * 86 * abs((1 - 44) / 90)	⇒	cm1 = 164.36
cs1 = 1.5 * 90 * 8 * abs((1 – 6) / 10)	⇒	cf1 = 540
cp1 = 0	⇒	cp1 = 0
cf2 = 1.5 * 60 * 8 * abs((1 - 3) / 10)	⇒	cf2 = 144
cm2 = (8 / 2) * 56 * abs((1 – 44) / 90)	⇒	cm2 = 107.02
cs2 = 1.5 * 60 * 8 * abs((1 – 6) / 10)	⇒	cs2 = 360
cp2 = 1.5 * 30 * 8 * abs((1 – 6) / 10)	⇒	cp2 = 180
cf3 = 1.5 * 48 * 8 * abs((1 - 3) / 10)	⇒	cf3 = 115.2
cm3 = (8 / 2) * 44 * abs((1 – 44) / 90)	⇒	cm3 = 84.08
cs3 = 1.5 * 48 * 8 * abs((1 – 6) / 10)	⇒	cs3 = 288
cp3 = 1.5 * 42 * 8 * abs((1 – 6) / 10)	⇒	cp3 = 252
cf4 = 1.5 * 36 * 8 * abs((1 - 3) / 10)	⇒	cf4 = 86.4
cm4 = (11/2) * 30.5 * abs((1 – 44) / 90)	⇒	cm4 = 80.14
cs4 = 1.5 * 36 * 8 * abs((1 - 6) / 10)	⇒	cs4 = 216
cp4 = 1.5 * 54 * 8 * abs((1 – 6) / 10)	⇒	cp4 = 324
cf5 = 1.5 * 20 * 8 * abs((1 - 3) / 10)	⇒	cf5 = 48
cm5 = (8 / 2) * 16 * abs((1 – 44) / 90)	⇒	cm5 = 30.57
cs5 = 1.5 * 20 * 8 * abs((1 - 6) / 10)	⇒	cs5 = 120
cp5 = 1.5 * 70 * 8 * abs((1 – 6) / 10)	⇒	cp5 = 420
cf6 = 1.5 * 4 * 8 * abs((1 - 3) / 10)	⇒	cf6 = 9.6
cm6 = (4 / 2) * 2 * abs((1 – 44) / 90)	⇒	cm6 = 1.91
cs6 = 1.5 * 4 * 8 * abs((1 - 6) / 10)	⇒	cs6 = 24
cp6 = 1.5 * 86 * 8 * abs((1 – 6) / 10)	⇒	cp6 = 516

$C_{mínimo}$ = min (x_{a1} * 216 + x_{b1} * 164.36 + x_{c1} * 540 + x_{d1} * 0) + (x_{a2} * 144 + x_{b2} * 107.02 + x_{c2} * 360 + x_{d2} * 180) + (x_{a3} * 115.2 + x_{b3} * 84.08 + x_{c3} * 288 + x_{d3} * 252) + (x_{a4} * 86.4 + x_{b4} * 80.14 + x_{c4} * 216 + x_{d4} * 324) + (x_{a5} * 48 + x_{b5} * 30.57 + x_{c5} * 120 + x_{d5} * 420) + (x_{a6} * 9.6 + x_{b6} * 1.91 + x_{c6} * 24 + x_{d6} * 516)

Sujeito a :

x_{ai}, x_{bi}, x_{ci} e x_{di} = 0 ou 1

xa1*90 + xc1*90 + xd1*0 = 90;
xa1*60 + xc1*60 + xd1*30 = 90;
xa1*48 + xc1*48 + xd1*42 = 90;
xa1*36 + xc1*36 + xd1*54 = 90;
xa1*20 + xc1*20 + xd1*70 = 90;
xa1*4 + xc1*4 + xd1*86 = 90;

xa1 + xb1 + xc1 + xd1 + xa2 + xb2 + xc2 + xd2 +xa3 + xb3 + xc3 + xd3 + xa4 + xb4 + xc4 + xd4 + xa5 + xb5 + xc5 + xd5 + xa6 + xb6 + xc6 + xd6 <= 3;

xa1 + xb1 + xc1 + xd1 + xa2 + xb2 + xc2 + xd2 +xa3 + xb3 + xc3 + xd3 + xa4 + xb4 + xc4 + xd4 + xa5 + xb5 + xc5 + xd5 + xa6 + xb6 + xc6 + xd6 >= 1;

Observações:

1- Dizer que o somatório de trechos presentes deverá ser igual a 90 GOFs (para o exemplo proposto) é devido ao cliente estar iniciando a sessão (vendo filme do início). Eventualmente ele poderia saltar o início do filme por alguma razão, nesse caso o somatório indicaria o número de GOFs da posição em que o mesmo iniciará o filme (seja onde for) até o final.

2- Dizer que o somatório das variáveis binárias deverá ser menor ou igual a 3 e maior ou igual a 1 é devido à impossibilidade de existirem mais de 3 tipos de custo simultaneamente e à certeza de que sempre existirá o conteúdo necessário no Servidor principal. Caso o servidor esteja saturado e negue pedidos, a indicação seria um custo excessivamente alto para c_s.

Resolvendo-se o problema formulado com o software de programação linear LINGO (http://www.lindo.com) versão 9.0, encontra-se os seguintes resultados:

```
Classe de modelo: Pure Integer Linear Program
Solver: Branch-and-Bound

Global optimal solution found.
Objective value:                              216.0000
Extended solver steps:                             0
Total solver iterations:                           0


Variable           Value        Reduced Cost
     XA1        1.000000            216.0000
     XB1        0.000000            164.3600
     XC1        0.000000            540.0000
     XD1        1.000000            0.000000
     XA2        0.000000            144.0000
     XB2        0.000000            107.0200
     XC2        0.000000            360.0000
     XD2        0.000000            180.0000
     XA3        0.000000            115.2000
     XB3        0.000000            84.08000
     XC3        0.000000            288.0000
     XD3        0.000000            252.0000
     XA4        0.000000            86.40000
     XB4        0.000000            80.14000
     XC4        0.000000            216.0000
     XD4        0.000000            324.0000
     XA5        0.000000            48.00000
     XB5        0.000000            30.57000
     XC5        0.000000            120.0000
     XD5        0.000000            420.0000
     XA6        0.000000            9.600000
     XB6        0.000000            1.910000
     XC6        0.000000            24.00000
     XD6        0.000000            516.0000
```

De onde se conclui que a melhor opção para o caso de um pedido ao *proxy* 1, teria custo 216 e corresponderia ao primeiro trecho, ou seja, o de menor distância temporal em relação ao novo cliente que solicitou o vídeo.

Conforme citado anteriormente, analisar o problema com uma visão local torna possível simplificá-lo drasticamente. Se for suposto que existindo mais de um trecho do mesmo vídeo em um mesmo *proxy*, o trecho de menor distância temporal (à direita dele) seria um candidato provável a atender determinada requisição (desde que não seja considerada realocação de memória de outros slots). Sendo assim, poderia-se avaliar apenas os custos dos trechos mais próximos (com menor "dt") entre diversos proxies que os contenham. Obviamente essa solução simplificada não seria a ótima, mas é provável que haja um significativo ganho de desempenho (custo computacional) pela simplificação do algoritmo.

O problema pode ser simplificado levando-se em conta apenas o trecho "mais provável" de cada *proxy, ou seja, o trecho de menor distância temporal em relação ao momento do pedido*. Esse proposto será considerado na modelagem simplificada a seguir.

$$\text{escolha i ótimo} = \min_{i=1}^{\text{num proxies}} \left[\quad \min\left[x_{ai}c_{fi} + x_{bi}c_{mi} + x_{ci}c_{si} + x_{di}c_{pi}\right] \quad \right]$$

onde:

$1 \leq i \leq$ número de proxies

$(x_{ai} + x_{bi} + x_{ci} + x_{di}) \leq 3$

$(x_{ai} + x_{bi} + x_{ci} + x_{di}) \geq 0$

x_{ai}, x_{bi}, x_{ci} e $x_{di} = 0$ ou 1

Σ (*slots* atendidos pelos 3 tipos de fluxo) = total de *slots* necessários

Observações:

1- A quantidade mínima de slots reservados pela MCC para o caso de um pedido (do início do vídeo) direto ao servidor será de 4 unidades de *slots* (parâmetro definido na simulação e herdado de [ISHIKAWA, 2003]).

2- Os casos de *patch* (remendo) do servidor ocorrem quando o trecho inícial do filme (ou a partir do momento necessário) não está presente em nenhum *proxy*, o

que implica em fluxo obrigatório do servidor com uma duração menor que o tempo total do vídeo. Dependendo da distância temporal do início do vídeo a algum trecho que tenha sido localizado em algum *proxy*, torna-se desvantajoso o *patch* + "fluxo de outro *proxy*", pois além de memória extra, eventualmente serão criados dois fluxos simultâneos (um para o *patch* e outro entre proxies, caso o trecho futuro não esteja no mesmo *proxy*).

Conforme a modelagem apresentada, o problema pode ser dividido em dois sub-problemas:

1- Encontrar os valores booleanos de Xai, Xbi, Xci e Xdi, os quais, dependendo de seus valores (1 ou 0) indicarão a existência (ou ausência) de cada um dos custos referentes aos trechos de vídeo procurados em determinado *proxy*.
2- Comparar o valor do somatório dos custos em cada *proxy* (e também no servidor, se for o caso), buscando-se o *proxy* que contenha a melhor escolha, ou seja, o de menor custo.

A primeira parte do problema pode ser escrita como:

$$Z = \min[x_a c_f + x_b c_m + x_c c_s + x_d c_p]$$

mantendo-se:

$(x_a + x_b + x_c + x_d) \leq 3$

$(x_a + x_b + x_c + x_d) \geq 0$

x_a, x_b, x_c e $x_d = 0$ ou 1

Σ (*slots* atendidos pelos 3 tipos de fluxo) = total de *slots* necessários

O que indica ser um caso de Programação Linear Binária, podendo ser resolvido utilizando-se *Branch and Bound*.

Para minimizar $[x_a c_f + x_b c_m + x_c c_s + x_d c_p]$ será necessário conhecer-se os custos (c_f, c_m, c_f e c_p), calculados em tempo real (online) pelas expressões referentes, além de

satisfazer simultaneamente a questão de que o somatório dos slots obtidos com a opção analisada deve totalizar o número de slots necessários (do ponto analisado até o final do vídeo).

A segunda parte do problema consiste apenas em varrer todos os proxies de maneira a localizar o de menor valor para o valor da variável custo.

A escolha do "i ótimo" será processada no *proxy* que originou o pedido, em posse da solução da "parte 1 do problema", retornada por cada *proxy* e servidor.

$$\text{escolha i ótimo} = \min_{i=1}^{\text{num proxies}} [\text{“}\textit{parte 1 do problema”}]$$

O algoritmo para a primeira parte do problema pode ser escrito da seguinte maneira:

Parâmetros utilizados para cálculo do custo:

lbv → largura de banda consumida pelo vídeo (em Mbps)

qsfv → quantidade de slots até o fim do vídeo

dss → duração em segundos de cada slot

lbd → largura de banda disponivel (no *proxy* ou servidor) – em Mbps

lbt → largura de banda total (do *proxy* ou servidor)

qsmcc → quantidade de slots que serão reservados pela MCC

qstrechcons → quantidade de slots até o trecho considerado (distância temporal dt)

qsinstnec → quantidade de slots até o instante necessário (tamanho do *patch* ou fluxo completo)

qsocup → quantidade de slots ocupados no *proxy* que originou o pedido

totslotsbuff → total de slots do buffer do *proxy* que originou o pedido

Pode-se escrever o seguinte algoritmo para o cálculo dos custos:

```
custo (lbv, qsfv, dss, lbd, lbt, qsmcc, qstrechcons, qsinstnec) {
if (qstrechcons == 0) {
// distancia temporal = 0 -> nao necessita nenhum fluxo do servidor
// Nesse caso existirão:
// -- custo do consumo de largura de banda do outro proxy (cf).
// -- custo de memoria alocada (ainda que qstrechcons seja zero, alocar-se-a mais
// memoria por estar-se considerando OUTRO proxy, ou seja, um novo cliente não
// pode utilizar da memoria ja alocada por estar em outro proxy.
cf = lbv * qsfv * dss * ( 1 – lbd ) / lbt;
cm = (qsmcc / 2) * qsfv * (abs( 1 – qsocup) / totslotsbuff ) ;
custoPriTrecho = cf + cm;
} else {
// distancia temporal > 0 -> implica em patch ou fluxo completo do servidor

        // opcao 1: fluxo do proxy + patch
        cf = lbv * qsfv * dss * ( 1 – lbd ) / lbt;
        cm1 = (qsmcc / 2) * qsfv * (abs( 1 – qsocup) / totslotsbuff );
        cp = lbv * qstrechcons * dss * ( 1 – lbd ) / lbt;
        custoPriTrechoOP1 = cf + cm1+ cs + cp;

        // opcao 2: fluxo completo do servidor
        cm2 = (8 / 2) * qsfv * (abs( 1 – qsocup) / totslotsbuff );
        cs = lbv * qsfv * dss * (1 – lbd) / lbt;
        custoPriTrechoOP2 = cm2+ cs;

if (custoPriTrechoOP1 >= custoPriTrechoOP2) {
        custoPriTrecho = custoPriTrechoOP1} else {
        custoPriTrecho = custoPriTrechoOP2}
        }
return custoPriTrecho;
}
```

4.2 O PROTOCOLO NAIVE

Inicialmente será proposto um protocolo simplificado sugerindo colaboração entre dois *proxies*. Do inglês, *naive* significa “ingênuo” e a razão desse nome deve-se ao fato de a colaboração sugerida ocorrer de maneira ingênua, sem avaliação de custos apresentados na modelagem, avaliando-se somente a disponibilidade de memória e largura de banda para existir colaboração.

A topologia utilizada, também simplificada, envolve um servidor de vídeo, 2 *proxies* e até 1000 clientes para cada *proxy*. A figura 4.4 ilustra a topologia sugerida.

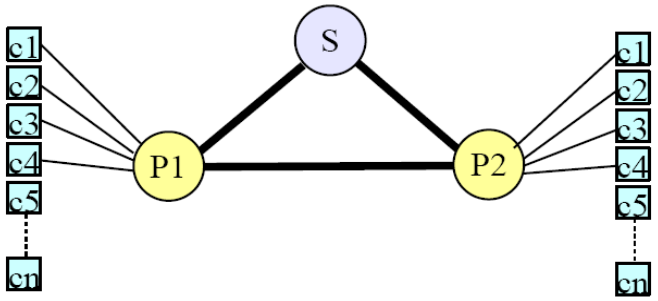


FIG 4.4 – Topologia utilizada nas simulações. S representa o servidor de vídeo, P1 e P2, os *proxies* e c1 a cn, os clientes de cada *proxy*.

Na topologia proposta pelo *naive*, clientes chegam conforme uma distribuição de Poisson [TRIVEDI, 2002] e solicitam vídeos ao servidor de acordo com sua popularidade (distribuição de Zipf [ZIPF, 1946]). É importante observar que modelar a entrada através de uma distribuição de Poisson só é possível observando certas faixas de horário, pois se forem consideradas as 24 horas do dia, as chegadas poderiam apresentar comportamento diferente. Por esse motivo e também com o objetivo de “estressar” o sistema, optou-se por simular o horário nobre (*prime time*) que corresponde ao horário das 20:00 às 22:00 horas (simulando pelo tempo de 1 hora).

No exemplo simplificado proposto existem apenas 2 proxies, sendo que clientes próximos a seu respectivo proxy fazem a solicitação do vídeo ao mesmo e se este não tiver todo o conteúdo requerido em sua cache, faz a solicitação a outro proxy supondo que o mesmo o possua (ingenuamente). Caso o conteúdo requerido não seja encontrado em nenhum dos dois proxies, a solicitação é encaminhada ao servidor de vídeo, o qual

criará um objeto servidor para atender à solicitação (novo streaming).

O protocolo naive, assim como o protocolo apresentado na próxima seção (ODPC), baseia-se em chamadas a métodos de clientes para proxies, de proxies a proxies, de proxies a servidor e de servidor a proxies. Se fosse implementado em “mundo real”, essas chamadas poderiam ser efetuadas como invocações a métodos remotos do Java (RMI), chamadas a procedimentos remotos (RPC) ou soluções equivalentes. Os métodos utilizados, bem como os parâmetros passados pelo protocolo são apresentados na tabela 4.1.

MÉTODO	**PARÂMETROS**	***ORIGEM***	***DESTINO***	***DESCRIÇÃO***
VodCli	cli, gofNr, gof->time, gof->size, gof->duration	cliente ou proxy	proxy	Envia gof para quem solicitou
getSlotStatus	slot	proxy	-	Verifica status do slot
getNearestBusy SlotDistance	gof	proxy	-	Obtém a distância (em slots) do slot mais próximo
in_pedePatch	video_->getVideoId(), gofNr	proxy	Servidor de vídeo	Solicita remendo de tamanho especifico
allocCollapsedB uffer	slot	proxy ou cliente	proxy	Aloca buffer colapsado
sendPatch	TPatchTimer* pt_, int& gofIni, int gofFim	servidor	proxy	Envia o patch solicitado
insertClient	int cliId, int slotNr, int gof	proxy	-	Insere cliente no slot especificado
searchVideo	id	proxy	proxy	Busca pelo vídeo (ou fragmentos do mesmo

TAB 4.1 – Alguns métodos, parâmetros recebidos, origem do pedido, destino e descrição. Aplicável em *naive* e ODPC.

Conforme já citado, o protocolo *naive* avalia apenas questões referentes à possibilidade de atender ou não a um pedido, ou seja, havendo memória e largura de banda disponível, um proxy atenderá ao pedido. Esse tipo de procedimento pode causar um desperdício de memória no *proxy* que recebe o pedido de cooperação de outro *proxy*, pois eventualmente deverá alocar *link slots* até o ponto de solicitação, conforme pode-se observar nas figuras 4.5 e 4.6.

As figuras 4.5 e 4.6 representam duas situações envolvendo 2 proxies que procuram

estabelecer uma colaboração. Cada retângulo representa um slot de 8 segundos, sendo que os retângulos brancos indicam que não há conteúdo, vermelhos representam posição de escrita, amarelo claro os *link slots* e amarelo escuro o slot de fim. Slots verdes indicam a presença de clientes naquele ponto, sendo que é possível existir mais de 1 cliente por slot.

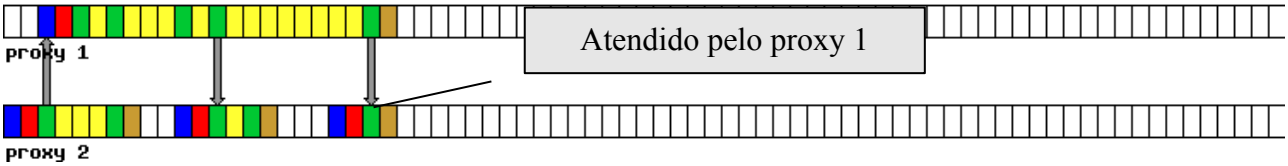


FIG 4.5 – *Proxy* 1 recebe um fluxo de colaboração do proxy 2 e colabora com 2 fluxos ao *proxy* 2. Para fornecer o segundo fluxo ao *proxy* 2, foi necessário alocar 8 link slots (amarelos).

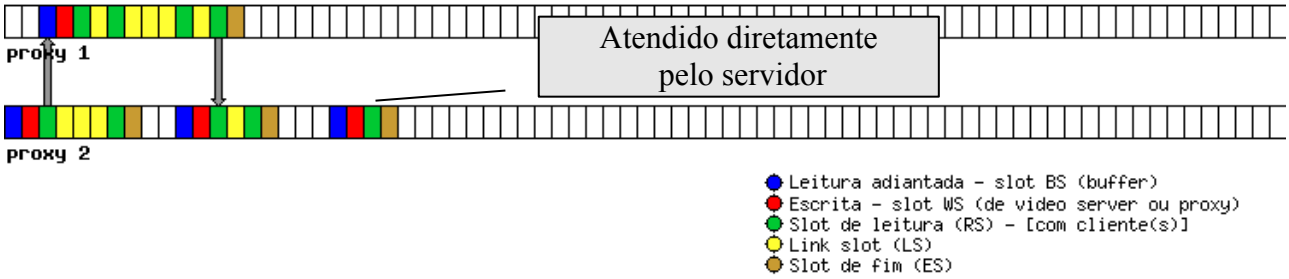


FIG 4.6 – A mesma situação de 8.a, porém optou-se por não criar um segundo fluxo do *proxy* 2 devido ao custo de alocação de memória no *proxy* 1. O *naive* não avalia esse tipo de custo, mas negaria a colaboração se não houvesse memória ou banda disponíveis.

Resultados de simulações preliminares obtidas com o *naive* são apresentadas na figura 4.7 e 4.8. No cenário simulado em 4.7 e 4.8, são comparadas as mesmas topologias (2 proxies, seus clientes e 1 servidor de vídeo) com variação de tamanho de cache (5% e 10% do tamanho total do vídeo) em ambiente colaborativo (*naive*) ou não (sem colaborar), variando o intervalo médio entre chegadas de 1 a 40 segundos. A figura 4.7 avalia a taxa de utilização do servidor *versus* distância média entre chegadas (distribuição de Poisson). A taxa de utilização do servidor refere-se à taxa de ocupação do enlace do servidor de vídeo durante o tempo da simulação e a distância média entre chegadas equivale ao tempo médio entre chegadas de clientes aos *proxies*.

Em 4.7 a taxa de utilização do servidor é contrastada nos casos de naive e sem colaborar, com caches de 5% e 10% do tamanho do vídeo. Nota-se que para intervalos médios entre chegadas muito baixos (alta taxa de chegada indica vídeo muito popular) a

colaboração não se destaca. A razão desse comportamento é que grande popularidade de vídeos está associado a baixa fragmentação do conteúdo na cache dos *proxies.* À medida em que o intervalo médio entre chegadas aumenta, a colaboração tende a tornar-se mais vantajosa. Posteriormente será demonstrado que para intervalos médios mais elevados (não mostrados na figura em questão, a qual representa somente até 40 segundos de intervalo médio) o comportamento do ambiente cooperativo vai perdendo a eficácia.

Em 4.8 o cenário de simulação se repete, mas a relação de pedidos bloqueados/chegados é avaliada, sendo observado que a escalabilidade do sistema aumenta para intervalos maiores que 6 segundos (também até certo limite, como será mostrado posteriormente nesse trabalho).

Em simulações posteriores, parâmetros referentes a ajustes finos do protocolo *naive* foram efetuados com o objetivo de aumentar ainda mais a escalabilidade do sistema, pois os primeiros resultados foram obtidos considerando-se apenas 1 vídeo no servidor. Posteriormente será demonstrado que no caso de existirem mais vídeos o comportamento se altera e os ajustes ideais para somente 1 vídeo não se aplicam a um número maior de vídeos.

As definições das métricas utilizadas (presentes nos gráficos) encontram-se no apêndice 1.

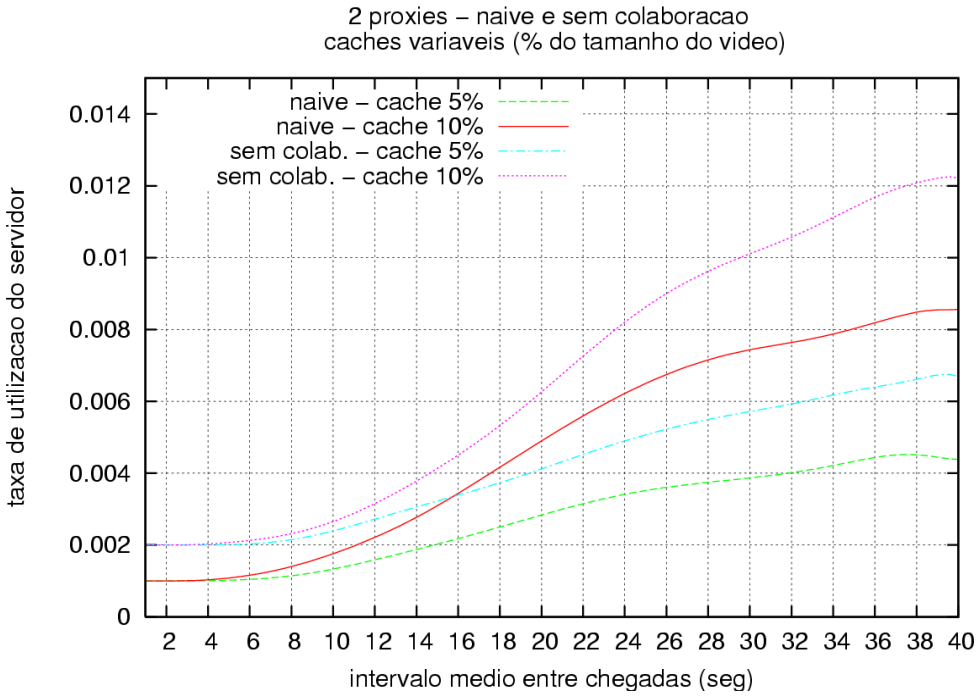


FIG 4.7 – Taxa de utilização do servidor para caches de 5% e 10% do tamanho total do video, avaliando *naive* e sem colaborar. Apenas 1 vídeo está presente no servidor nessas

simulações.

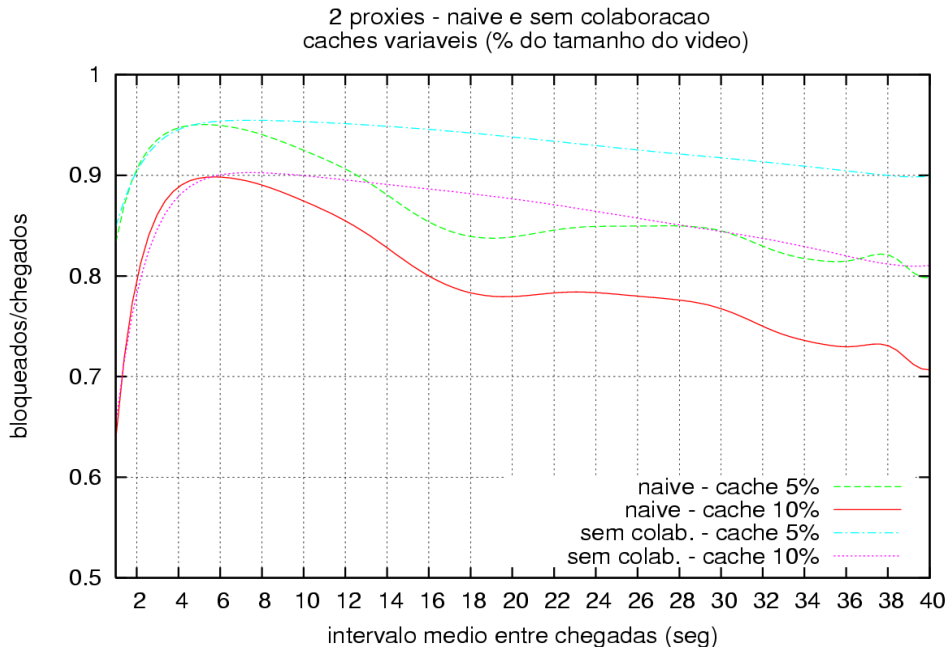


FIG 4.8 – A relação de pedidos bloqueados/chegados é menor quando existe colaboração. Simulações para 1 vídeo no servidor e caches de 5% e 10% do tamanho total do vídeo.

Resultados obtidos com o protocolo *naive* demonstraram que colaboração entre caches de *proxies* pode ser uma boa alternativa para aumentar a escalabilidade de um sistema de VoD. O próximo passo foi incrementar o protocolo *naive* para que o mesmo deixasse de se portar de maneira “ingênua” e passasse a considerar certos custos de cooperação antes de tomar decisões de atender ou não. No capítulo 4.3 é apresentado o protocolo ODPC, uma evolução do *naive* que opera respeitando a modelagem proposta em 4.1, mas de maneira simplificada.

4.3 O PROTOCOLO ODPC

Como foi visto, o naive apenas avalia se “pode ou não” atender a um pedido, enquanto um sistema que tomasse decisões do tipo “deve ou não” atender, poderia se mostrar mais eficiente. No capítulo 4.1 foi apresentada a modelagem do problema, sendo inclusive que a resolução recairia sobre *branch-and-bound* [ROSS, 1973]. Em um sistema no “mundo-real”, seria exigir muito da capacidade computacional de um *proxy* se o

mesmo tivesse de ser capaz de encontrar centenas (e até milhares) de soluções por segundo, pois poderão existir diversos *proxies* com centenas de vídeos e milhares de clientes, o que implicaria em dezenas de milhares de trechos de vídeos fragmentados.

Na busca de uma solução simplificada que oferecesse ganho de desempenho em relação ao naive, foi considerado inicialmente que o trecho de menor custo tende a estar mais próximo do instante solicitado (menor distância temporal “dt”). O passo seguinte foi buscar os limites dessa distância, ou seja, uma distância máxima onde um *proxy* deveria atender a um pedido de outro *proxy.* O protocolo proposto foi entilulado ODPC, derivado de *On Demand Proxy Collaboration.*

A determinação de tal distância foi obtida a partir de simulações com variações do trecho máximo capaz de aceitar colaboração. As figuras 4.9 e 4.10 mostram testes para determinação da melhor distância temporal para se atender a um pedido de outro *proxy*.

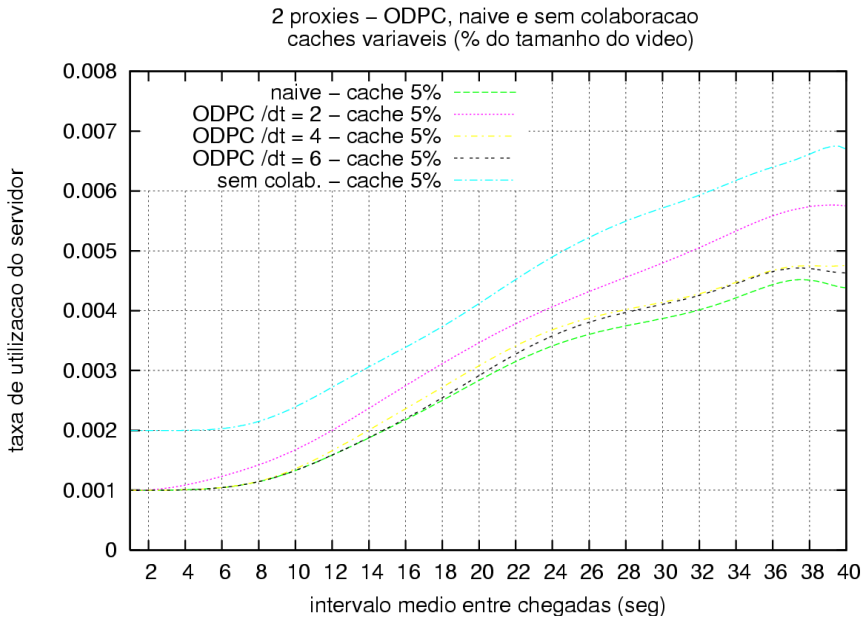


FIG 4.9 – Taxa de utilização do servidor para sem colaborar, naive e ODPC, sendo variadas as distâncias temporais para haver colaboração para o ODPC.

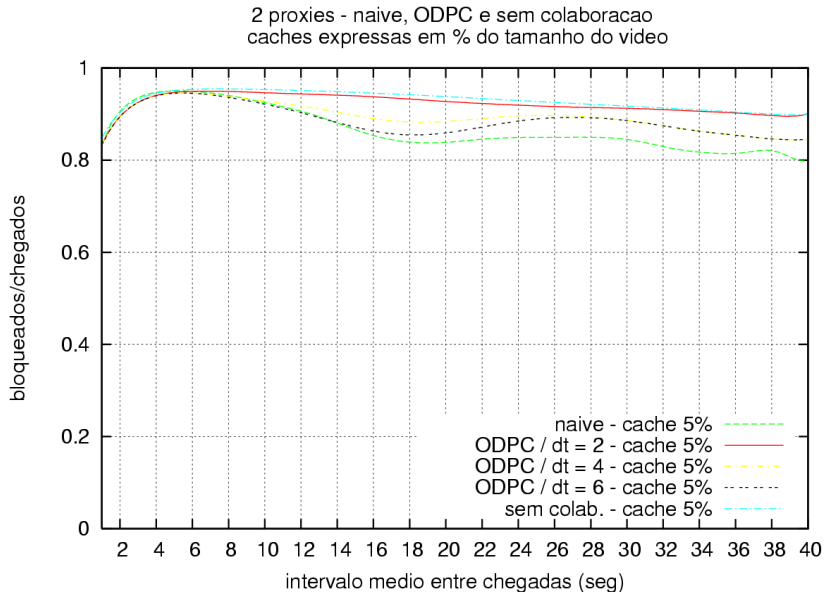


FIG 4.10 – Bloqueados/chegados para os mesmos casos da figura 4.9.

A distância temporal é medida em *slots e*, no caso, um *slot* possui 8 segundos. Em termos práticos, dizer que a distância temporal máxima para haver colaboração é de 4, significa dizer que se um cliente pedir um vídeo a seu *proxy* e esse (por não ter o conteúdo) direcionar o pedido a um *proxy* que já esteja atendendo a seus clientes com o mesmo vídeo, haverá colaboração do segundo para o primeiro *proxy* somente se a distância entre os pedidos for inferior a 32 segundos (4 *slots* de distância temporal). O valor de 4 *slots* foi obtido a partir de resultados de diversas simulações e exibidos nas figuras 4.10 e 4.11.

O que se observa nas figuras 4.10 e 4.11 é que se *dt* for muito pequeno (menor ou igual a 2), o sistema não colaborará para distâncias médias entre chegadas acima de 6 segundos. Para distâncias entre 4 e 6, o sistema se aproxima do *naive*. Acima disso o sistema se comporta bastante semelhante ao *naive* na faixa de intervalo considerada. Optou-se em adotar a distância temporal de 4 e 6 *slots*, sendo que posteriormente foi adotado apenas o valor 4 por ter apresentado melhores resultados em outras simulações.

As figuras 4.11 a 4.25 representam o estado de vetor de slots após 1 hora de simulação para os 3 tipos de rede de distribuição de conteúdo avaliados (sem colaborar, naive e ODPC). Como as figuras representam situações reais de um sistema distribuído, pequenas variações de posições de fluxos (setas da figura) podem parecer incorretos devido à dificuldade natural de se sincronizar tal sistema, todavia apenas as representações estão incorretas, pois o sistema demonstrou estabilidade em seu

funcionamento.

Vetor de *slots* é definido no capítulo referente a trabalhos relacionados e seus conceitos são originados na MCC [ISHIKAWA, 2003]. Um vetor de slots se projeta sobre o vetor de vídeo e uma maneira simples de compreender seu funcionamento é visualizar que um *buffer* colapsado contém no mínimo 4 slots. Esses slots, quando ocupados, tem seus *status* alterados. Assim, pela sequência de figuras citadas, o *slot* colorido em verde indica que um ou mais cliente está presente (podendo ser um cliente normal, outro proxy ou ambos). O slot de cor vermelha indica posição de escrita e o amarelo um *slot* de ligação (link slot). Clientes estão associados a determinados slots e tal estado não se altera sob condições normais (como, por exemplo, o cliente não assistir ao vídeo de maneira normal, utilizando recursos de videocassete como pausa e avanço rápido), mas o vetor de vídeo se desloca de maneira contínua sobre o vetor de slots, de maneira que em dado instante o *slot* "A" pode conter os 8 segundos iniciais de um vídeo e 1 minuto após o mesmo *slot* conter o trecho entre 60 e 68 segundos. Por característica da MCC, quando um cliente entra no sistema e o slot a ele associado não possua previamente clientes, o *proxy* tentará atendê-lo estendendo um *buffer* colapsado até o slot do cliente (através de slots de ligação – indicados na cor amarela da figura) ou alocar um novo *buffer* colapsado (de 4 slots). Em ambiente de colaboração entre *proxies*, o pensamento é análogo, o que muda é o fato de um cliente poder ser outro *proxy,* o que demanda uma atenção especial.

As figuras 4.11 a 4.23 demonstram o comportamento dos 3 casos em função da variação de taxa de entrada de clientes. Um vídeo com grande popularidade possuirá uma taxa de entrada maior (intervalo médio entre chegadas menor) e como citado anteriormente, casos de alta popularidade (intervalos inferiores a 10 clientes por segundo) já são tratados de maneira interna em cada *proxy* pela MCC, enquanto casos de baixa popularidade não merecem atenção especial por serem ocasionais. Como se pode observar pelas figuras, para casos de intervalo médio entre chegadas de 10 a 20 segundos (figuras 4.11 e 4.12), apenas um fluxo de colaboração foi criado devido a proximidade dos clientes. Os comportamentos do *naive* e ODPC são idênticos, pois nesses casos a distância temporal não ultrapassa ao valor que se convencionou. O número de clientes atendidos foi idêntico ao do caso de não colaboração, mas fluxos diretos do servidor foram poupados. Com intervalo médio de 40 segundos ocorreu uma falha no sistema ao não serem aceitos clientes referentes ao últimо *buffer* colapsado (presente no caso de sem colaboração). A razão de tal falha deve-se possivelmente ao sistema ter considerado que a memória era escassa e tentar criar um *batch* de clientes

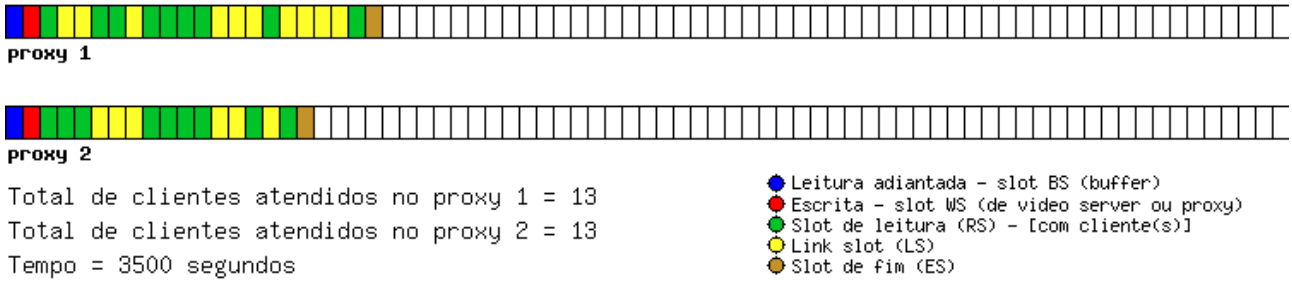
("fila de espera"). Esse fato se repete em algumas outras simulações, o que demonstra uma pequena diferença em relação ao número de clientes atendidos por parte do *naive* e ODPC, mas aqui a simulação considera apenas 1 vídeo no servidor e posteriormente será demonstrado que a memória ociosa decorrente desse fato, pode ser utilizada para outros *slots* de vídeos diferentes (quando simular com mais vídeos no servidor).

Nas figuras 4.17 e 4.18, nota-se a diferença entre o *naive* e ODPC em relação ao atendimento de pedidos de colaboração: enquanto o *naive* atendeu a todos os pedidos possíveis (e com isso alocou diversos *link slots* (amarelos) no *proxy* colaborador, o ODPC limitou o atendimento considerando-se a distância temporal. Ainda que a taxa de utilização do servidor para esses casos tenha sido maior para o ODPC, a memória poupada ficou livre para atender a clientes de outros vídeos (ou, para um trabalho futuro, outros *proxies).* Para intervalos superiores a 90 segundos, *naive* e ODPC voltam a apresentar comportamento semelhante, o que confirma o escopo do ODPC para casos de média popularidade.

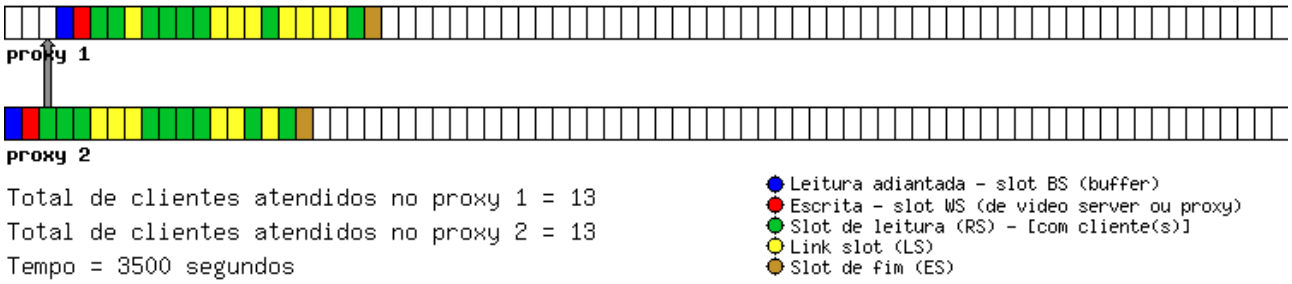
Para plotagem das figuras a partir dos resultados gerados pelo ns-2.28, foram utilizados os módulos GD e GD::Arrow para a linguagem Perl.

As figuras citadas (de 4.11 a 4.23) foram obtidas com apenas 1 semente na simulação. Os resultados exibidos por elas servem para demonstrar o funcionamento dos protocolos, mas não servem como dados de validação estatística. Os resultados finais, apresentados no próximo capítulo, obedecem a critérios estatísticos para validação com 95% de confiança.

SEM COLABORAR:



NAIVE:



ODPC:

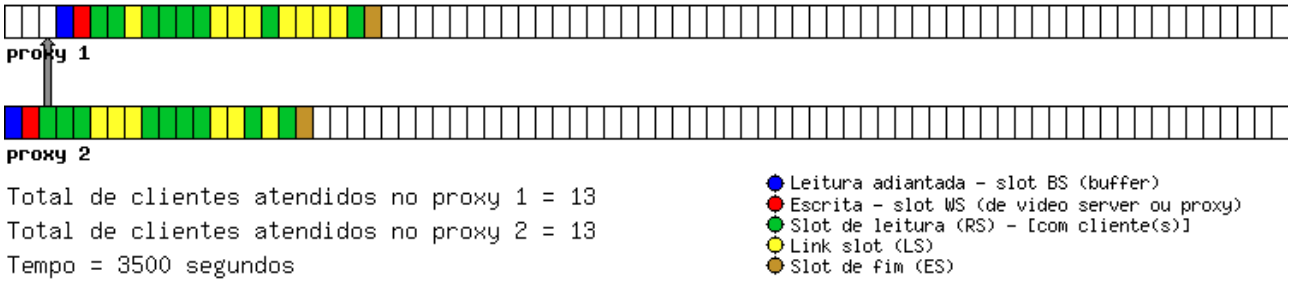
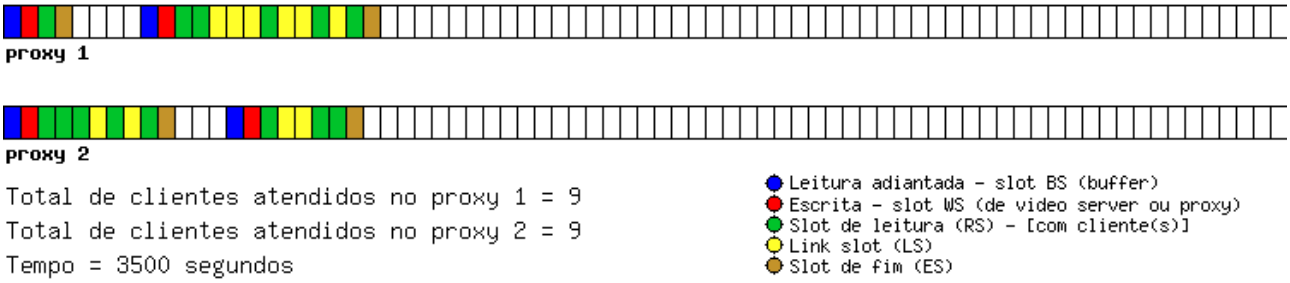
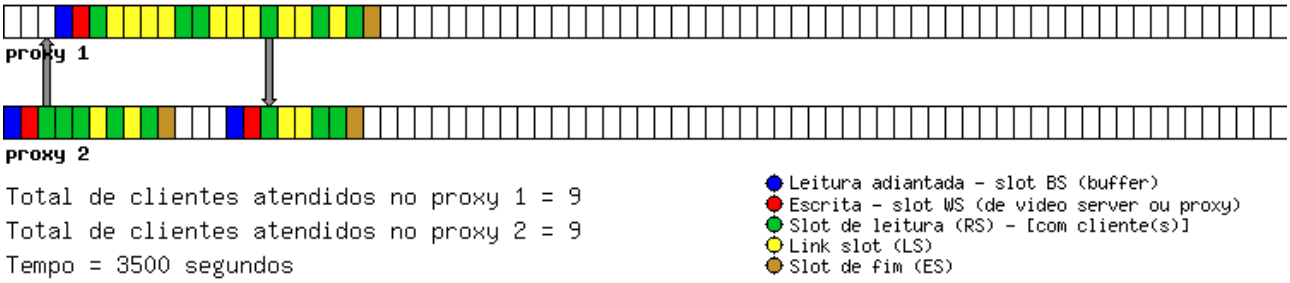


FIG 4.11 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 10 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

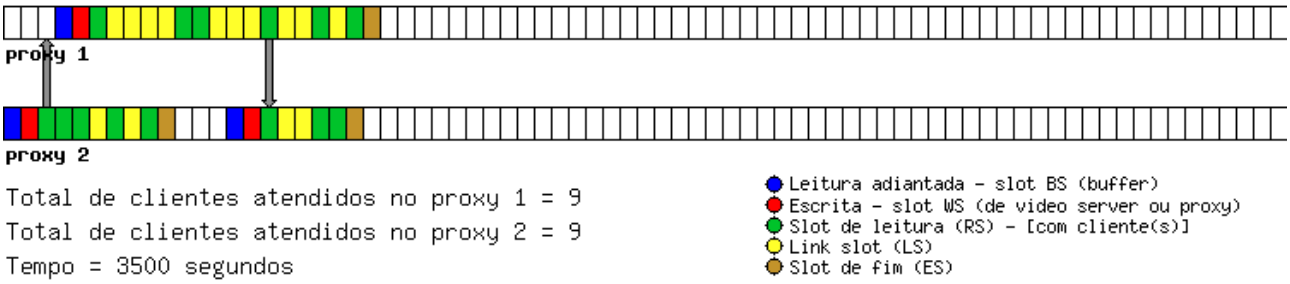
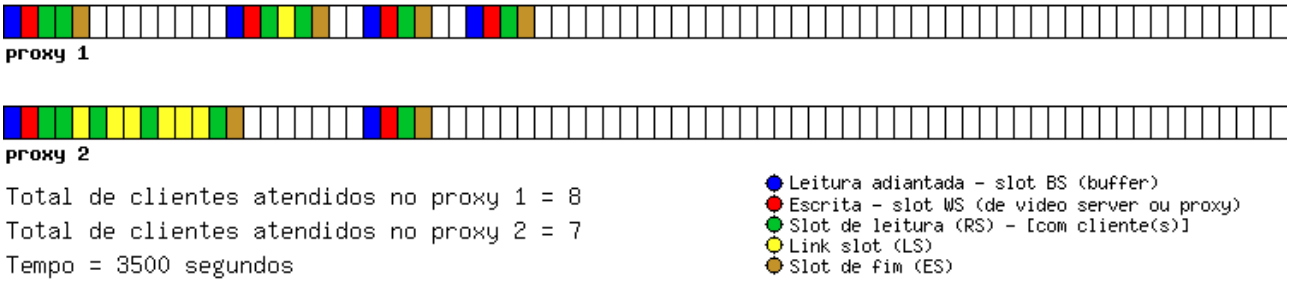
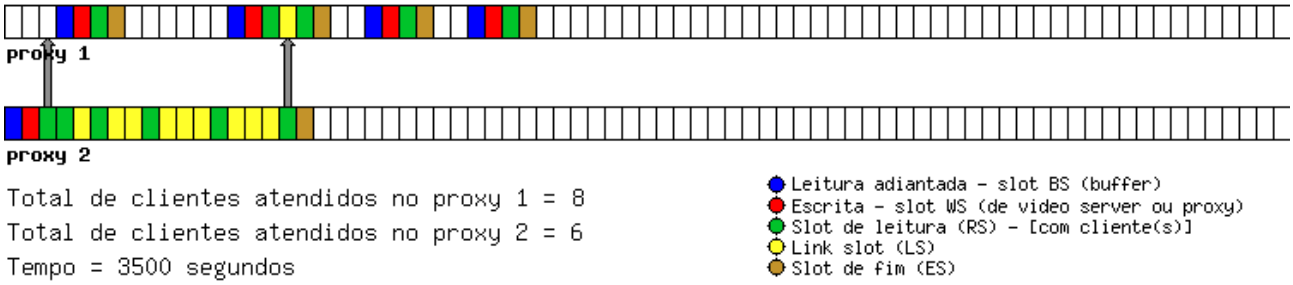


FIG 4.12 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 20 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

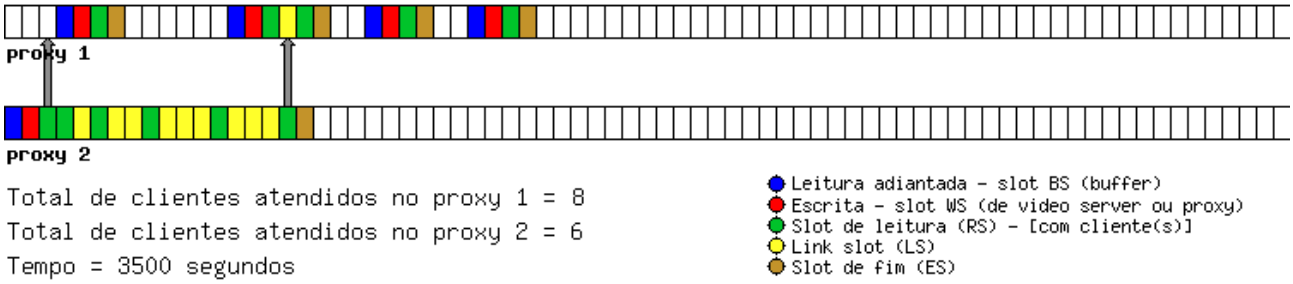
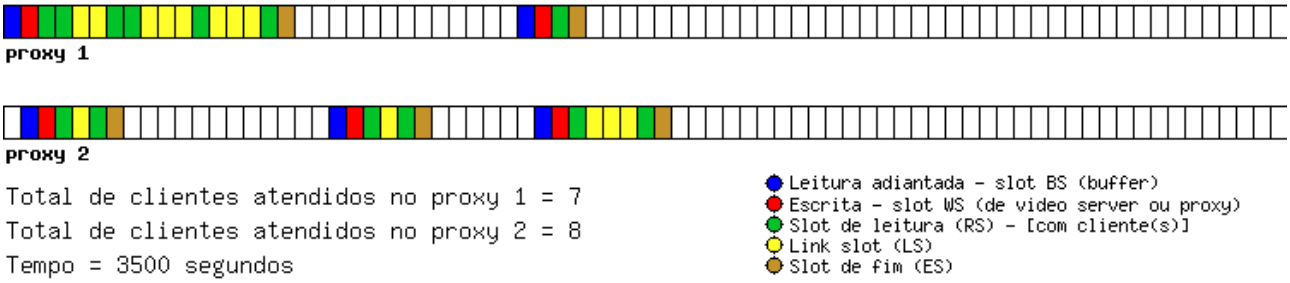
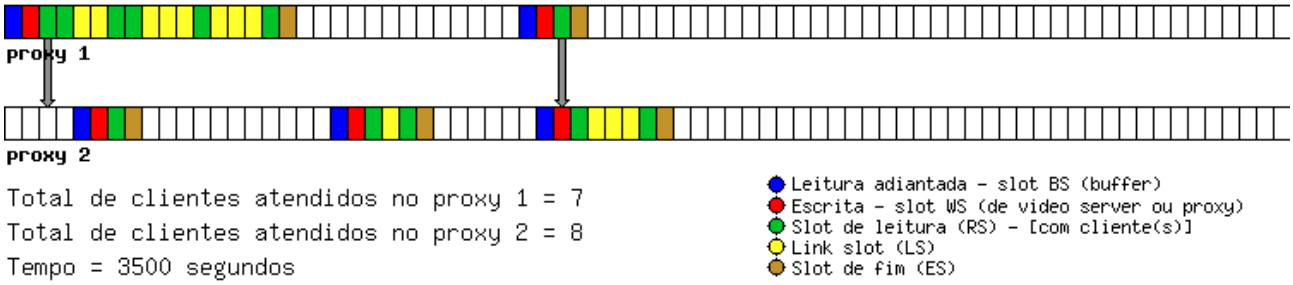


FIG 4.13 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 30 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

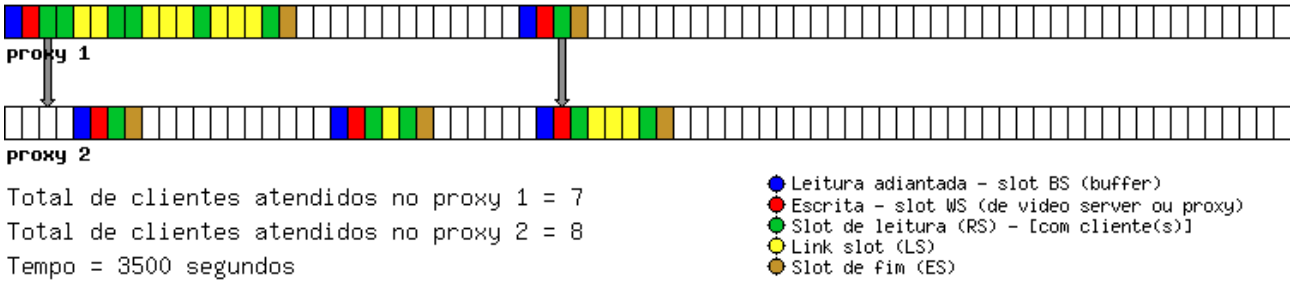
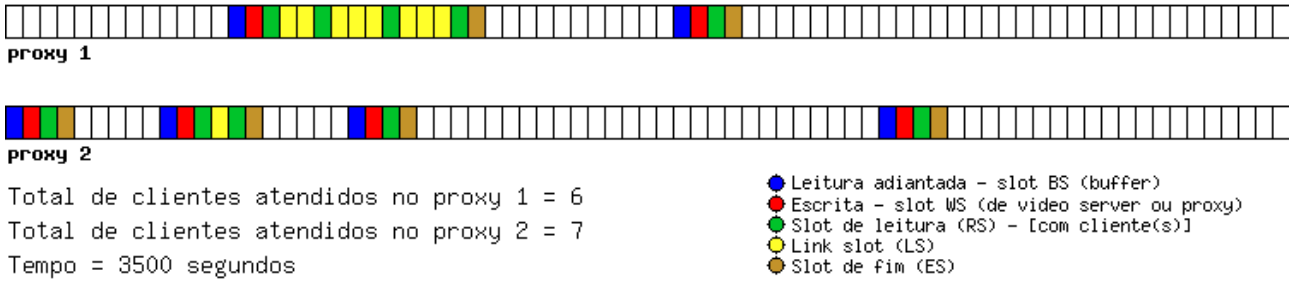
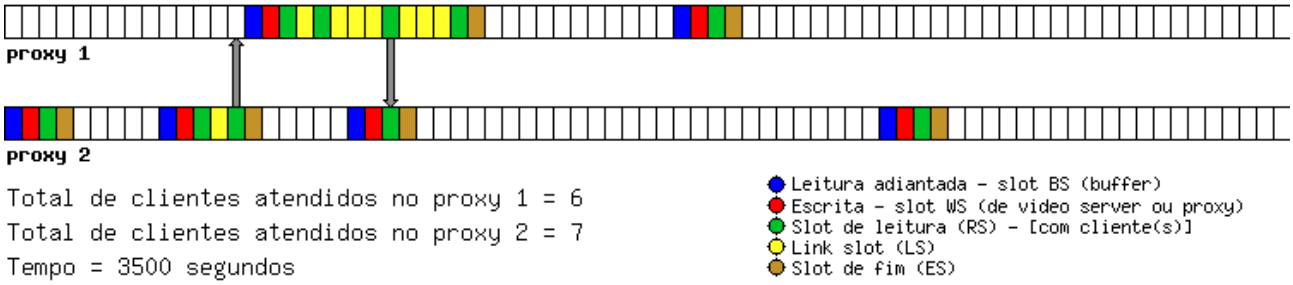


FIG 4.14 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 40 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

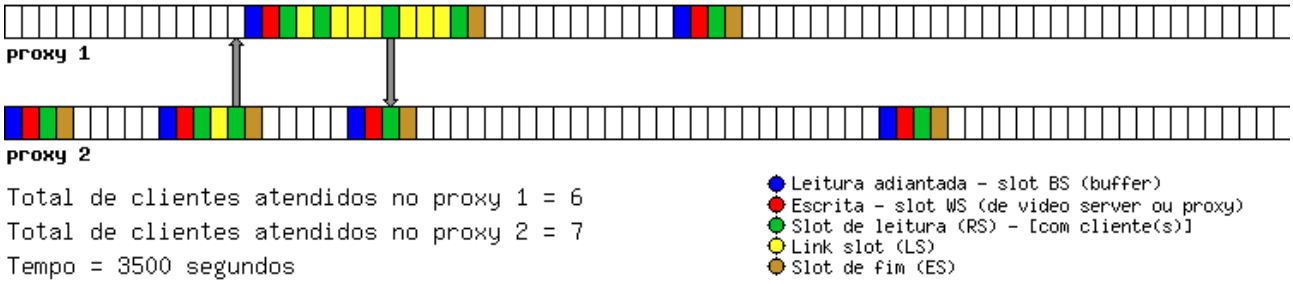
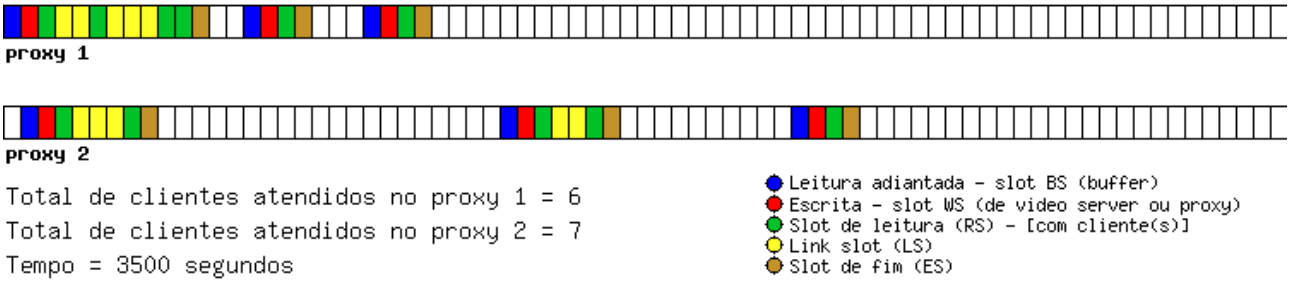
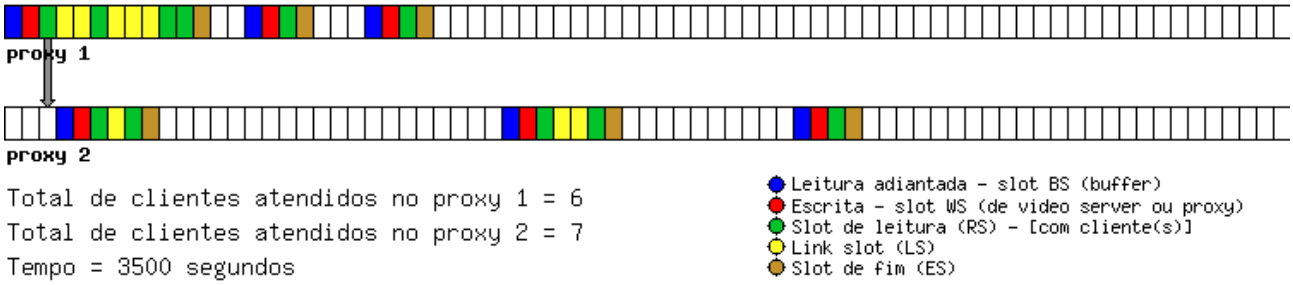


FIG 4.15 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 50 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

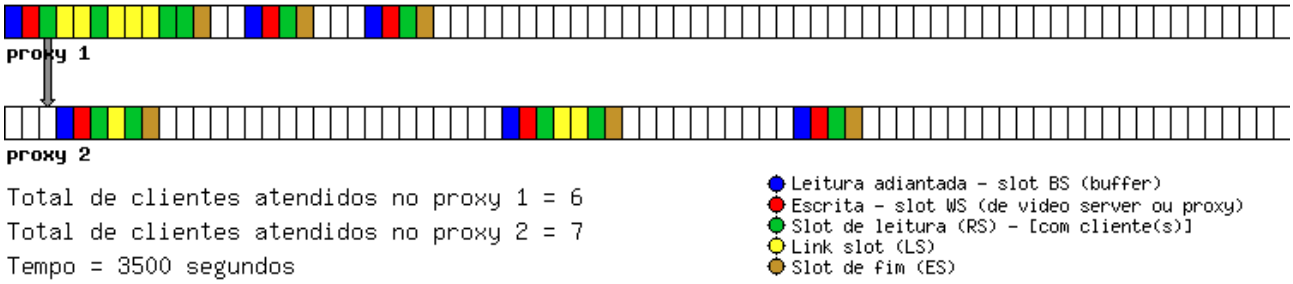
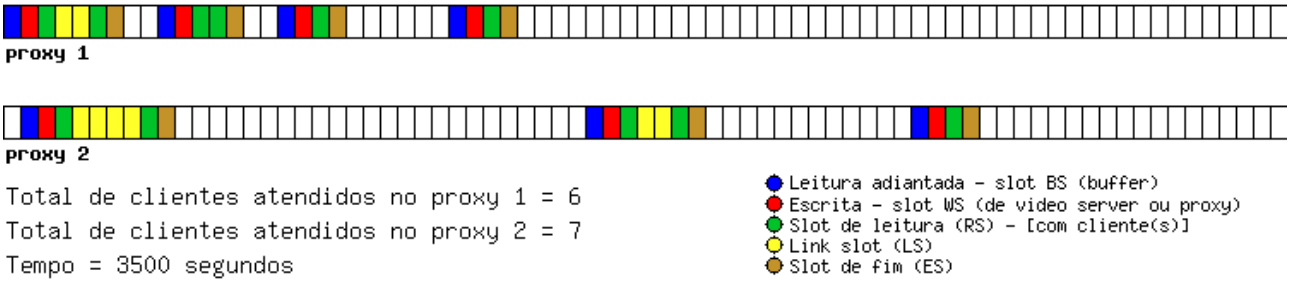
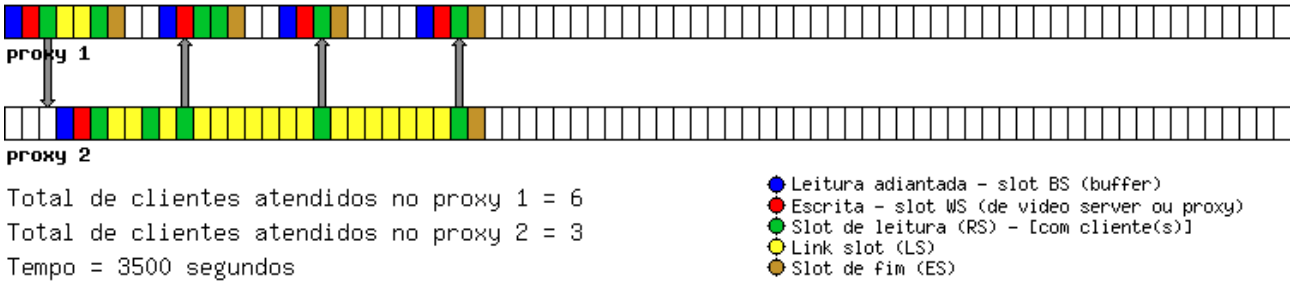


FIG 4.16 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 60 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

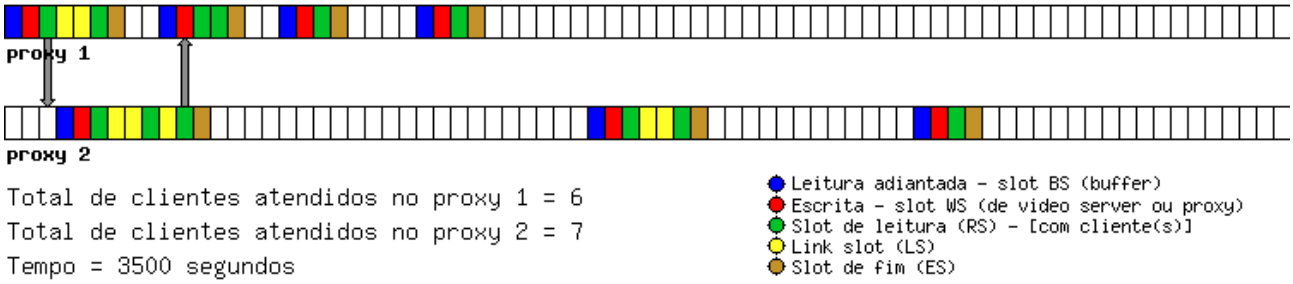
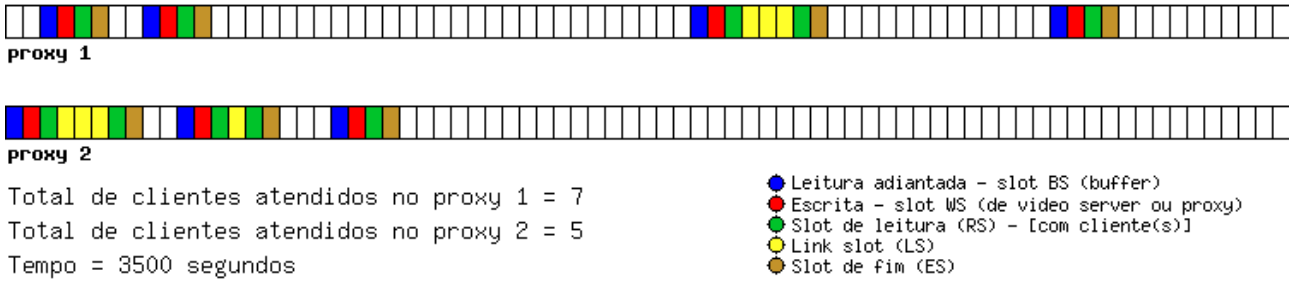
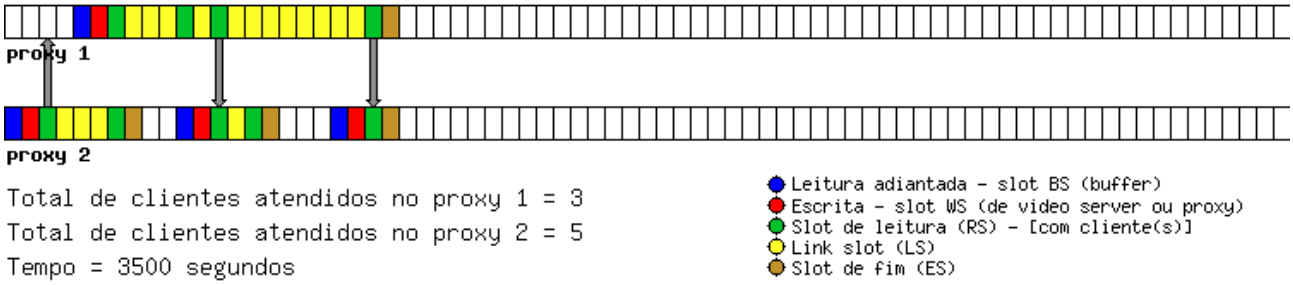


FIG 4.17 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 70 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

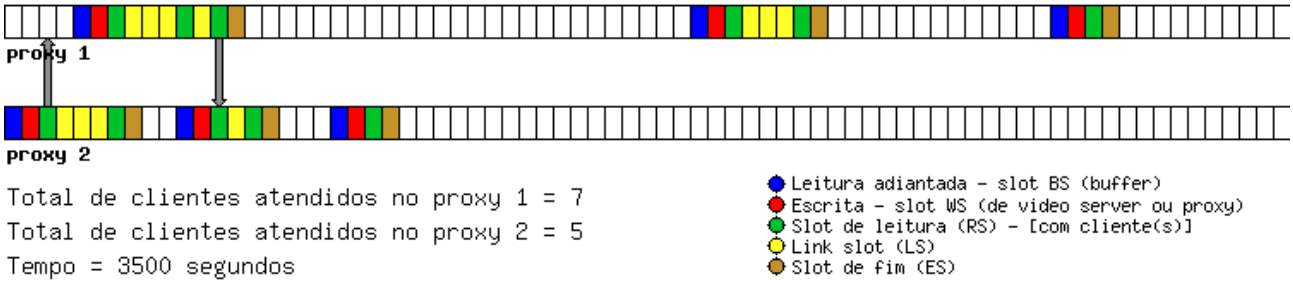
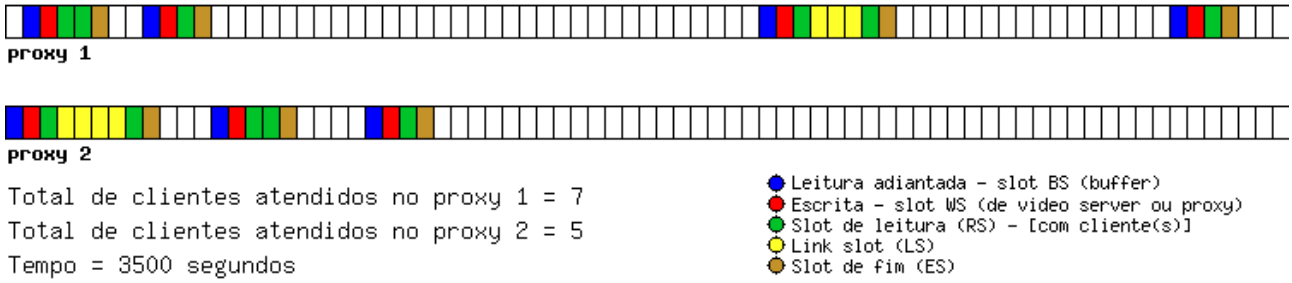
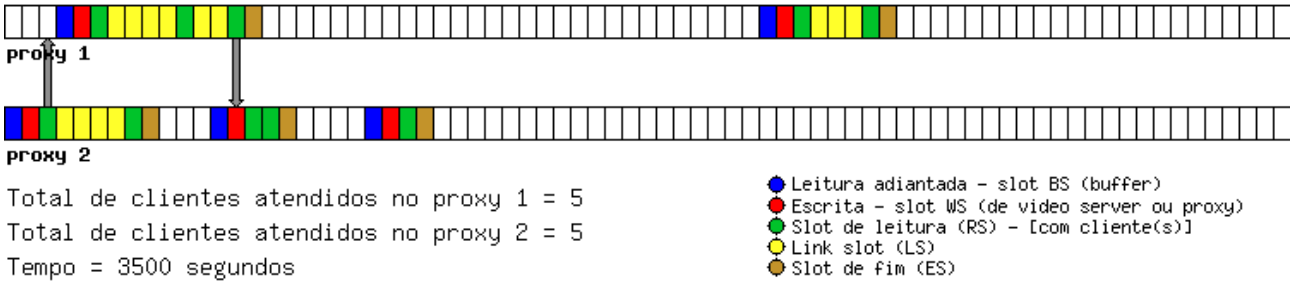


FIG 4.18 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 80 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

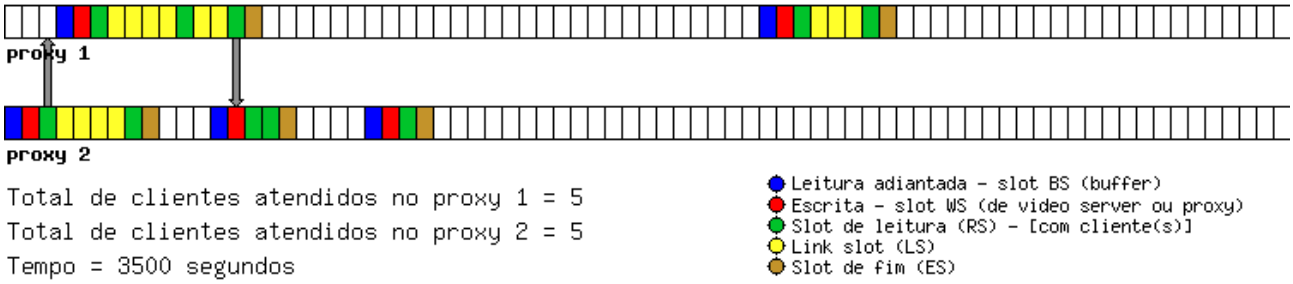
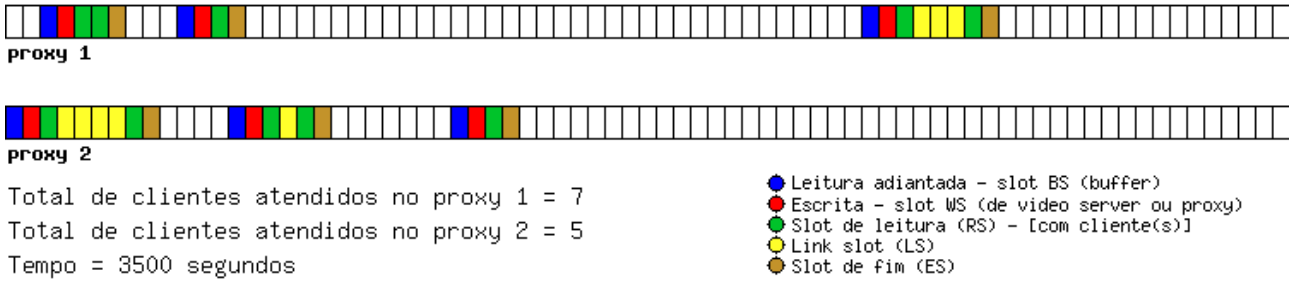
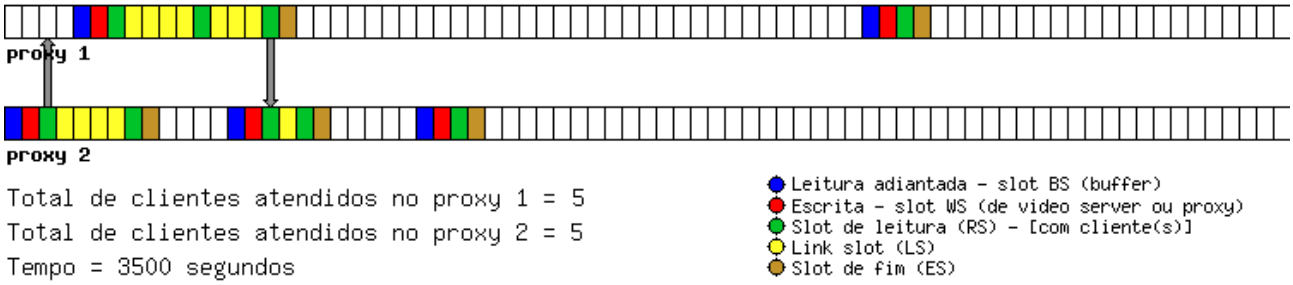


FIG 4.19 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 90 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

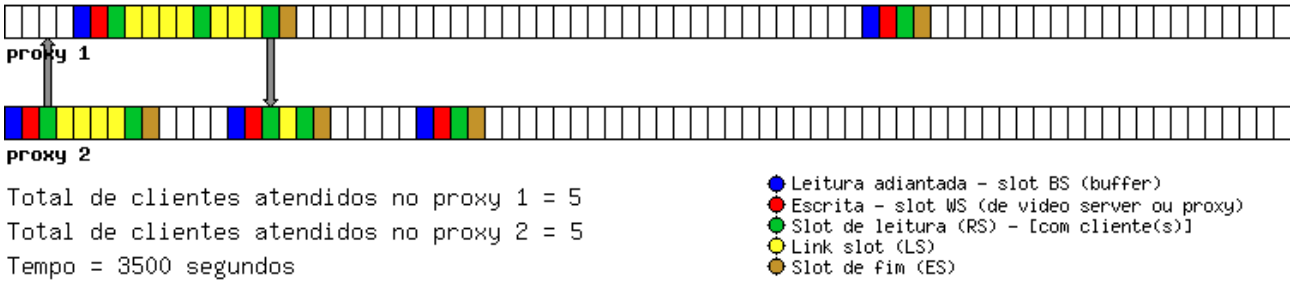
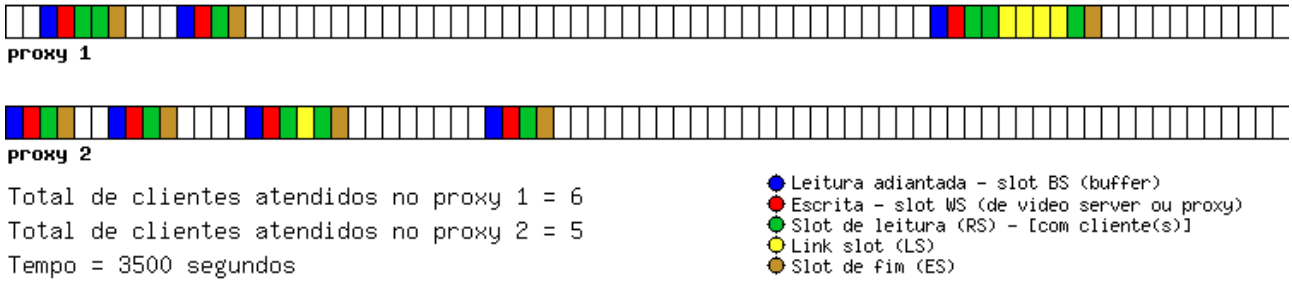
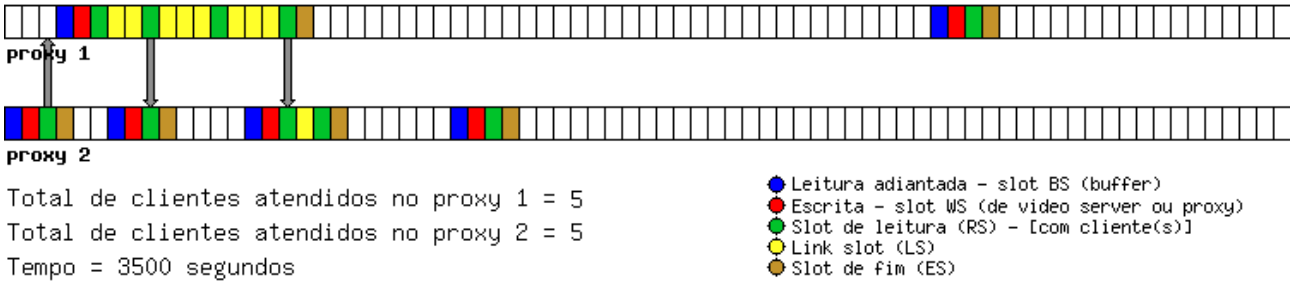


FIG 4.20 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 100 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

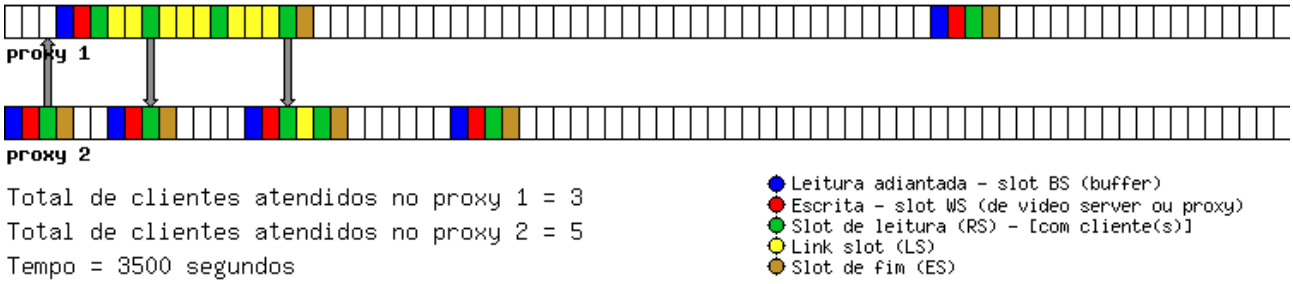
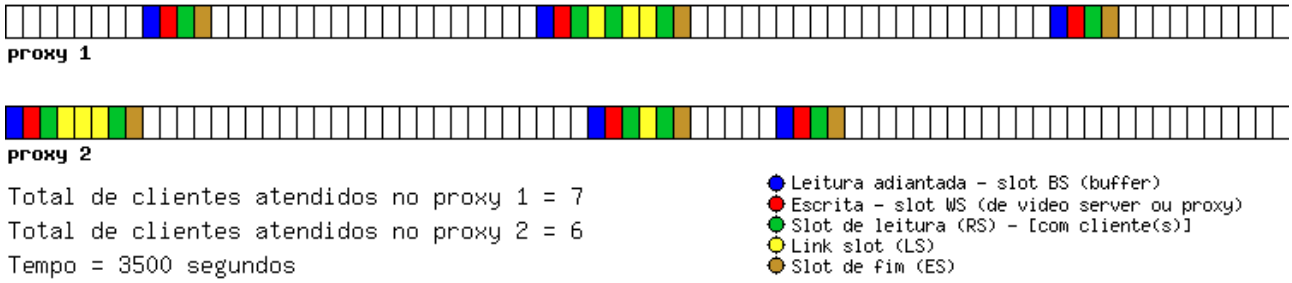
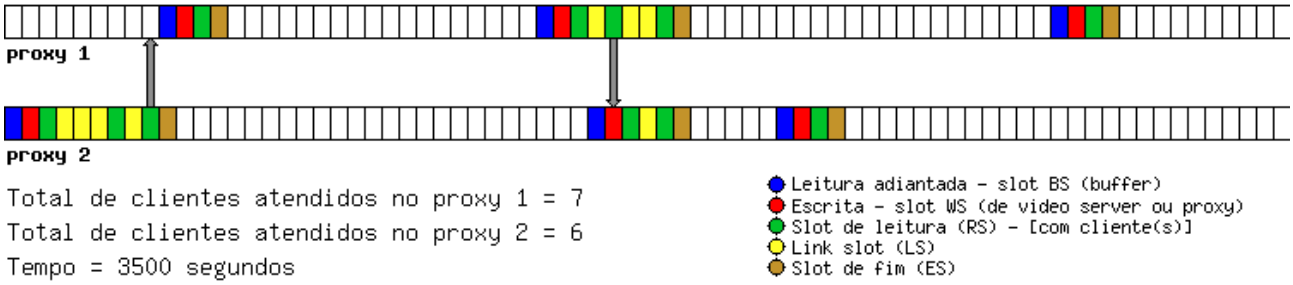


FIG 4.21 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 110 segundos.

SEM COLABORAR:



NAIVE:



ODPC:

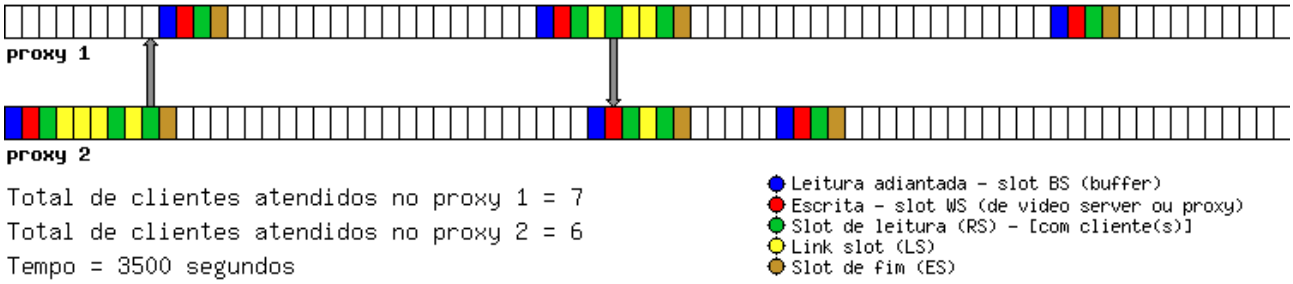


FIG 4.22 – Representação do vetor de slots para sem colaborar, naive e ODPC após 3500 segundos de simulação e intervalo médio entre chegadas de clientes igual a 120 segundos.

5 RESULTADOS

Os parâmetros utilizados nas simulações, assim como as definições das métricas, encontram-se no APÊNDICE 1. Os resultados apresentados (com exceção da simulação para 100 vídeos) foram obtidos a partir de 30 simulações (variando-se as sementes de geração de aleatórios) o que garantiu confiança de no mínimo 95% em um intervalo de confiança inferior a 1% do valor medido [SHELDON, 1990].

As simulações foram executadas com o simulador NS-2 nas versões 2.27 e 2.28 [NS], tendo sido utilizadas as linguagens Tcl e ANSI C++ para o desenvolvimento e Perl para interpretação de resultados. Os gráficos foram plotados com a ferramenta GNUPLOT (suavizadas com Berzier) e, no caso dos vetores de *slots*, utilizando-se os módulos GD e GD::Arrow no Perl.

A figura 5.1 apresenta a taxa de utilização do servidor para os 3 casos comparados (sem colaborar, *naive* e ODPC) com variação de intervalo médio entre chegadas de 10 a 100 segundos. O resultado obtido demonstra uma redução na carga do servidor do ODPC em relação ao *naive.* Observa-se que os 3 sistemas apresentam comportamentos semelhantes em intervalos menores (vídeos muito populares), mas mesmo em casos de média popularidade a taxa de utilização ficou acima do “sem colaborar”. O fenômeno ocorrido não deve ser avaliado de maneira isolada, assim como deve-se considerar o fato de tais simulações utilizarem somente 1 vídeo no servidor (por razões de tempo demandado: 3 a 4 dias para 1 vídeo, sendo que mais vídeos implicam em mais tempo de simulação). As figuras 5.2 a 5.3 consideram a mesma simulação, mas com outras métricas avaliadas.

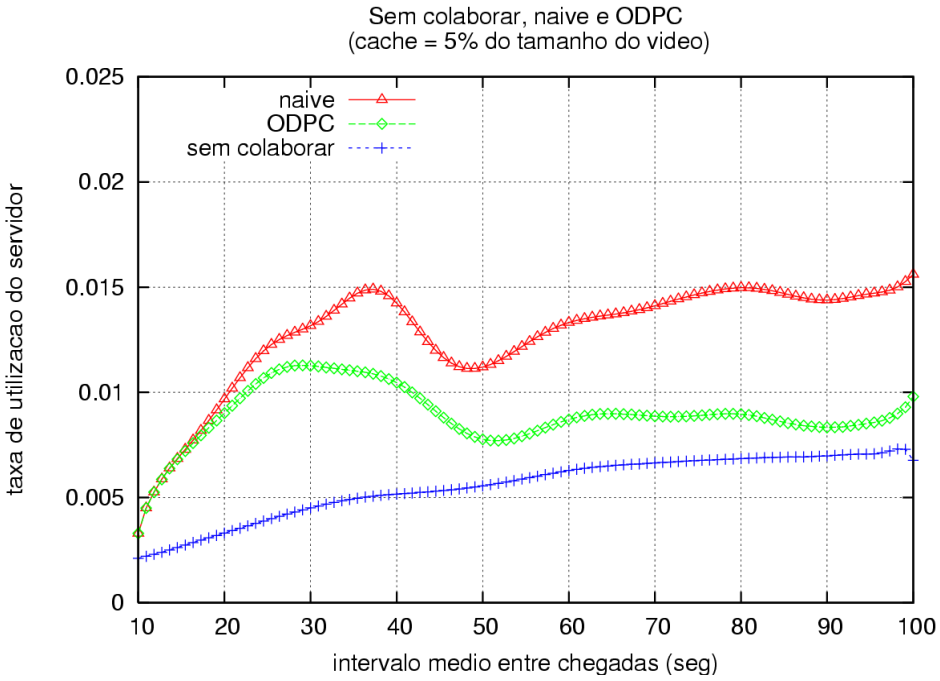


FIG 5.1 – Taxa de utilização do servidor para o caso de 1 vídeo.

A figura 5.2, obtida na mesma simulação, avalia a taxa de utilização da memória dos *proxies.* Nota-se claramente que no caso de não haver colaboração, a memória é quase totalmente utilizada, enquanto que nos dois casos de cooperação, justamente na faixa média de popularidade, a economia é notável. O protocolo ODPC ainda consegue manter uma taxa de utilização menor que o *naive,* pois aloca memória de maneira mais racional.

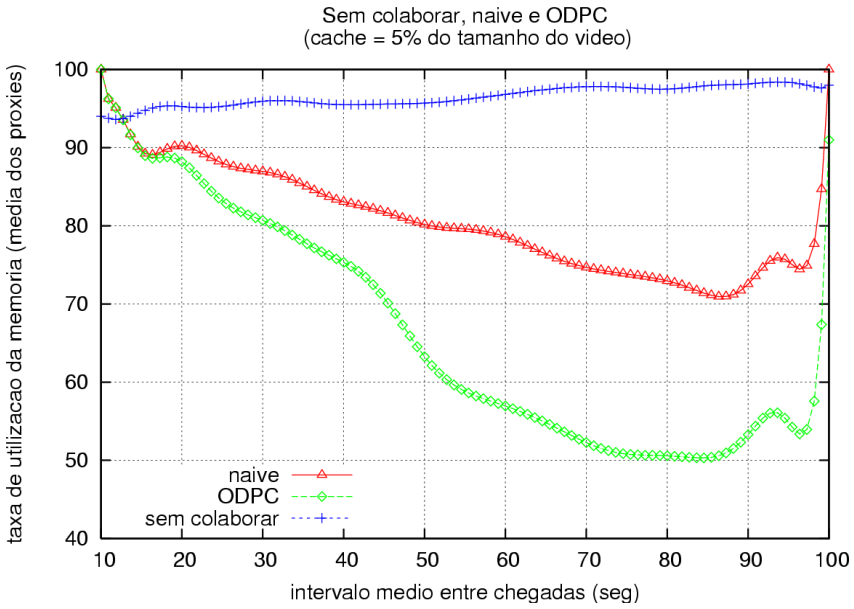


FIG 5.2 – Taxa de utilização média de memória nos proxies para o caso de 1 vídeo.

Nota-se que para vídeos muito populares e pouco populares, a utilização de memória pelos *proxies* é muito semelhante. Tal fato é comprovado pela sequência das figuras 4.12 a 4.24, as quais exibem a utilização da memória entre 10 a 150 segundos de intervalo médio entre chegadas.

A quantidade de clientes atendidos após 3500 segundos (somatório dos *proxies*) é apresentada na figura 5.3. Apesar de o gráfico indicar mais clientes atendidos somente até aproximadamente 20 segundos (e mantendo-se ligeiramente abaixo do "sem colaborar" acima desse intervalo), é importante ressaltar que muita memória ficou disponível (*vide* figura 5.2), indicando que mais clientes poderão ser atendidos no caso de existirem outros vetores de *slots* (mais de 1 vídeo presente no servidor).

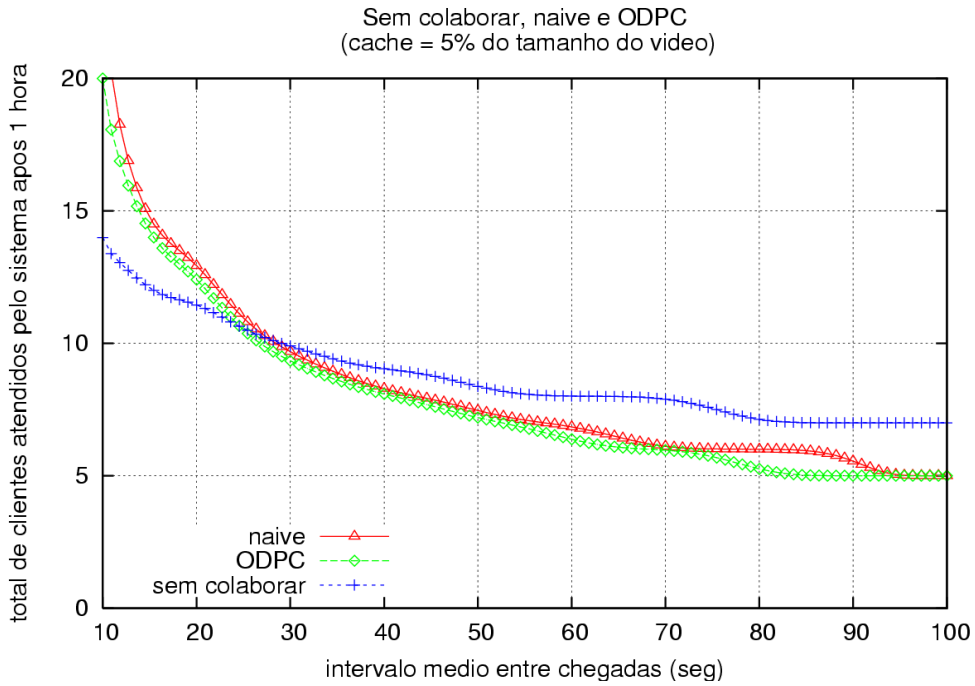


FIG 5.3 – Total de clientes atendidos pelo sistema aos 3500 segundos de simulação.

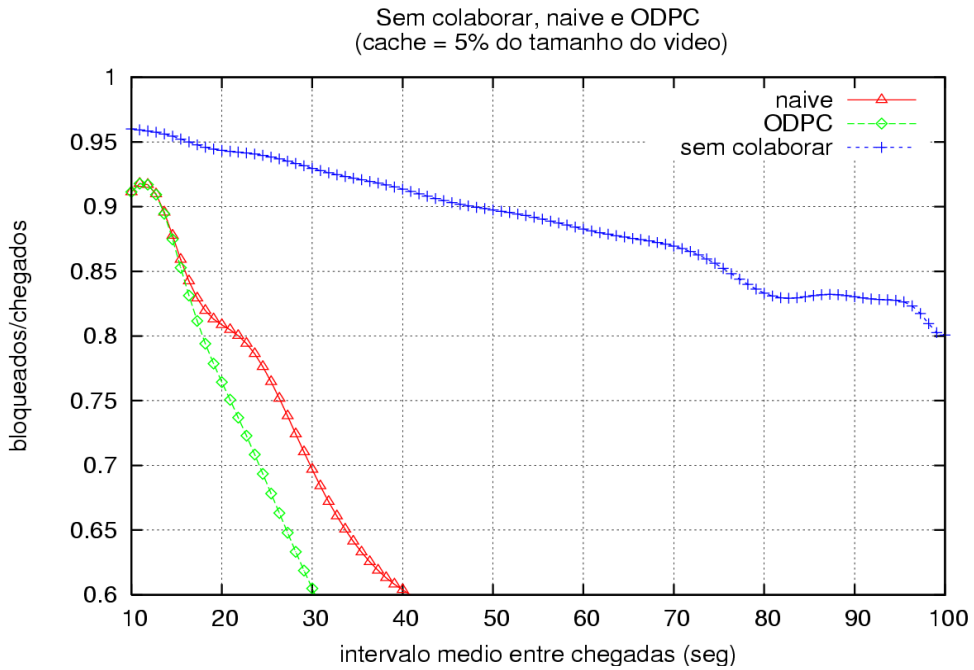


FIG 5.4 – pedidos bloqueados/chegados para 1 vídeo

A figura 5.4 mostra que a escalabilidade do sistema aumenta havendo colaboração em relação a pedidos que deixariam de ser atendidos por um sistema não cooperativo. Tal ocorrência é explicada através da gerência de memória mais eficiente, o que se traduz em mais pedidos atendidos.

Para comprovar a eficiência do sistema com 100 vídeos e verificar a hipótese de que a memória economizada pelo ODPC (nas simulações com 1 vídeo) poderia atender a outros clientes (se houvesse mais vídeos no servidor), efetuou-se uma simulação para 100 vídeos avaliando-se a taxa de utilização do servidor. O resultado é apresentado na figura 5.5 e pode ser analisado da seguinte forma: apesar de as curvas parecerem semelhantes, a diferença de cota (segmento vertical medido entre as duas curvas) à medida em que a curva (semelhante a uma exponencial) decresce, aumenta em valor proporcional à cota total. Em outras palavras, a medida da distância entre as curvas no caso de 20 segundos de intervalo é próxima da mesma medida para o caso de 50 segundos, mas o valor proporcional em 50 segundos é bem mais significativo, logo, pode-se dizer que o ganho de escalabilidade em relação à taxa de utilização do servidor cresce em função do crescimento do intervalo médio entre chegadas, tendendo a se estabilizar para vídeos de baixa popularidade.

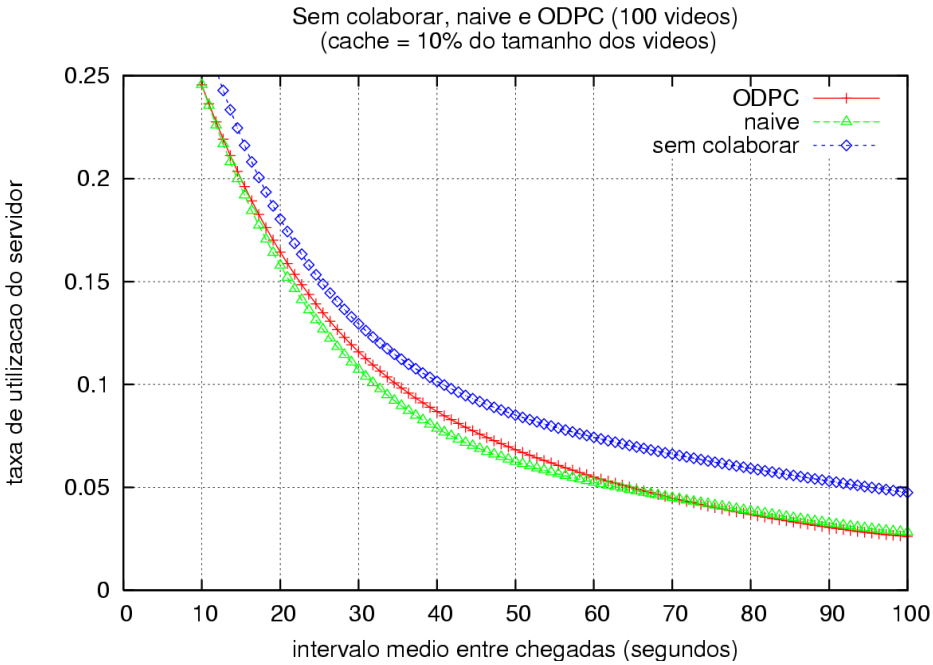


FIG 5.5 – Taxa de utilização do servidor para o caso de 100 vídeos.

A figura 5.6 demonstra que o número de clientes atendidos aumenta com a presença de mais vídeos no sistema, todavia existe uma convergência de número total de clientes atendidos com intervalos médios entre chegadas muito elevados. Isso se deve ao fato de que com taxas de chegada de clientes muito baixas, a colaboração deixa de fazer sentido. É importante lembrar que taxas de chegada muito baixas indica vídeos pouco populares, o que não é o objetivo do sistema proposto.

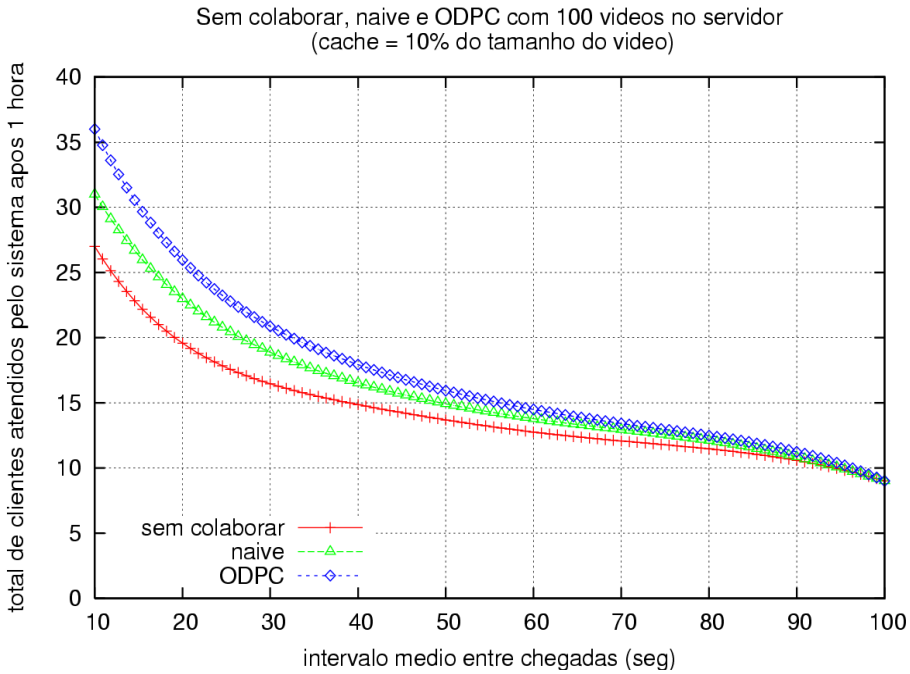


FIG 5.6 – Total de clientes atendidos para o caso de 100 vídeos.

6 CONCLUSÕES

Redes de distribuição de conteúdo permitem aumentar a escalabildade de um sistema de vídeo sob demanda. Nesse trabalho foram apresentadas duas alternativas de protocolos para CDN, entituladas *naive* e ODPC. Ambas sugerem colaboração de caches de *proxies* e trabalham a partir da estrutura de memória cooperativa colapsada [ISHIKAWA, 2003].

O protocolo *naive* opera de modo a colaborar ao máximo, e seu nome refere-se mesmo a uma colaboração considerada "ingênua", pois não avalia custos antes de colaborar; enquanto que o protocolo ODPC (*On Demand Proxy Collaboration*) procurou implementar uma política de avaliação de pedidos em função da distância entre o conteúdo existente no *proxy* e o momento do pedido (entitulada "distância temporal"). Tanto o protocolo *naive,* quanto o ODPC, demonstraram ganho de escalabilidade num sistema envolvendo 2 *proxies* e um servidor de vídeo, sendo que o ODPC apresentou melhor desempenho.

Em simulações com apenas 2 proxies e 1 vídeo, o ganho de desempenho se refletiu principalmente sobre a taxa de utilização de memória, sendo que a taxa de utilização do servidor apresentou um resultado desfavorável. O aumento da taxa de ocupação do enlace do servidor deve-se a pedidos de *patch* irregulares, pois o conteúdo solicitado encontra-se geralmente em fragmentos. Sendo que a taxa de utilização de memória chegou a níveis 50% inferiores em relação ao sistema não colaborativo, pode-se supor que em um sistema com mais vídeos tenderia a ser melhor atendido pelo sistema devido a essa memória livre.

O capítulo 5 apresentou resultados de simulações em diversas métricas avaliadas, a maioria considerou apenas 1 vídeo no servidor por razões de tempo consumido pelas simulações. A simulação envolvendo 100 vídeos no servidor (figuras 5.5 e 5.6) mostrou o que já havia sido previsto: a escalabilidade e o desempenho do sistema aumentam com o número de vídeos. A Figura 5.6 demonstra que o ODPC é capaz de atender a um maior número de clientes no caso de 100 vídeos, havendo uma convergência entre os 3 sistemas avaliados à medida em que os vídeos tornam-se menos populares.

Também espera-se que o ganho de desempenho cresça significativamente com um maior número de proxies, pois serão encontradas novas opções em relação à localização de conteúdo em caches havendo um número maior de *proxies*.

Em trabalhos futuros seria válido obter-se simulações exaustivas com 100 vídeos ou

mais, como também variar-se o parâmetro Zipf [ZIPF, 1946] e o tamanho das caches dos *proxies.*

Também para trabalhos futuros seria válido alterar a topologia, considerando-se mais de 2 *proxies* (o que seria muito viável em uma MAN). Alguns problemas referentes a topologias mais complexas poderiam surgir, principalmente na questão da localização do conteúdo e diversos trabalhos poderiam servir de base de estudo para a solução de tal problema.

Por último, restaria a implementação em “mundo real” de tal protocolo. Imagina-se a utilização de chamadas a métodos remotos como o RMI (Remote Method Invocation) do Java ou RPC (Remote Procedure Call) de diversas linguagens para a troca de mensagens entre *proxies* e clientes. Também os protocolos RTP (Real-time Transport Protocol) e RTCP (Real-time Transport Control Protocol) poderiam ser utilizados para transporte e controle dos fluxos de vídeo.

Espera-se que esse trabalho tenha contribuído para o estudo e desenvolvimento de novas alternativas para distribuição de vídeo sob demanda.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ATM - **ATM Forum**. Disponível: http://www.atmforum.com. [capturado em 05 de abril de 2006).

BISSOLI, F. G., ISHIKAWA, E., **ODPC: Protocolo para Gerência de Memória Cooperativa em Proxies para Distribuição de Vídeo sob Demanda**. SEMISH – XXXIII Seminário Integrado de Software e Hardware, Campo Grande, 2006.

BOMMAIAH, E., GUO, K., HOFMANN, M., PAUL, S., **Design and Implementation of a Caching System for Streaming Media over the Internet**. In: Proceeding of IEEE Real-time Technology and Applications Symposium – RTAS, Washington, DC, maio 2000.

CHAE, Y., GUO, M., BUDDHIKOT, M., SURI, S., ZEGURA, E. **Silo, Rainbow and Caching Token: Schemes for Scalable, Fault Tolerant Stream Caching**. IEEE Journal on Selected Areas in Communications, Special Issue on Internet Proxy Services, 2002.

COBARZAN, C., BOSZORMENYi, L., **Dynamic Proxy-cache Multiplication inside LANs**. Technical Report, Institute of Information Technology, University Klagenfurt, 2005.

DAN, A., SITARAM, D., SHAHABUDDIN, P., **Dynamic Batching Policies for an On-demand Video Server**. Multimedia Systems, v.3, n.4, pp. 112-121, 1996.

DESHPANDLE, H., BAWA, M., GARCIA-MOLINA, H. **Streaming Live Media over a Peer-to-Peer Network**, Technical Report, Stanford University, março 2002.

GOLUBCHIK, L., LUI, J.C.S., MUNTZ, R.R., **Adaptative piggybacking: a Novel Technique for Data Sharing in Video-on-demand Storage Servers**. Multimedia Systems, v.4, n.3, pp. 140-155, 1996.

GUO, Y., SUH, K., TOWSLEY, D. **A Peer-to-Peer On-Demand Streaming Service and Its Performance Evaluation**. IEEE International Conference on Multimedia, ICME – 2003, Baltimore, MD, julho 2003.

HUA, K., CAI, Y., SHEU, S. **Patching: A Multicast Technique for True Vídeo-On-Demand Services** In Proc. Of ACM Multimedia 98, Bristol, UK, setembro 1998.

ISHIKAWA, Edison. **Memória Cooperativa para Distribuição de Vídeo sob Demanda**.

2003. Tese (Doutorado em Ciências em Engenharia de Sistemas e Computação). Universidade Federal do Rio de Janeiro, COPPE, 2003.

JIN, S., BESTAVROS, A. **GreedyDual* Web Caching Algorithm – Exploiting the Two Sources of Temporal Locality in Web Request Streams**. International Journal on Computer Communications, v.24, n.2, pp. 174-183, fevereiro 2001.

KHAN, S. et al, **A Performance Comparison of Multiple Description Video Streaming in Peer-to-Peer and Content Delivery Networks**. Institute of Communication Net works, Technische Universität München, 2003.

MA, W-H. DU, D.H.C.: **Reducing Bandwidth Requirement for Delivering Video over Wide Area Networks with Proxy Server**. In: IEEE International Conference on Multimedia and Expo, 2000.

MEYER, P. L., P**robabilidade – Aplicações à Estatística**. 1ª.Edição, Rio de Janeiro, Livros Técnicos e Científicos S/A, 1980.

MPEG-1 - **MPEG-1 description**. Disponível: http://www.chiariglione.org/mpeg/standards /mpeg-1/mpeg-1.htm, [capturado em julho 2006].

NS - **Network Simulator 2**. http://www.isi.edu/nsnam/ns/ns-build.html, [capturado em 7 julho 2006].

PARK, H.S.,CHUNG, K.D., Lim, E.J.: **Popularity-based Partial Caching for VOD Systems using a Proxy Server**. In: Workshop on Parallel and Distributed Computing in Image Processing, Video and Multimedia, 2001.

PINHO, L. B., ISHIKAWA, E., AMORIM, C. L. **Glove: A Distributed Environment for Scalable Vídeo-On-Demand Systems**. International Journal of High Performance Computing Applications, v. 17, n. 2, pp. 000-000, Spring 2003.

ROSS, G. TERRY, **A Branch and Bound Algorithm for the Generalized Assignment Problem**. Mathematical Programing 8, North-Holland Publishing Company, p. 91-103, 1975.

SEN, S., REXFORD, J., TOWSLEY, D. **Proxy Prefix Caching for Multimedia Streams**. In: Proc. Of the IEEE INFOCOM'99, abril de 1999.

SHELDON, M. Ross. **A Course in Simulation**:Macmillan Publishing Company, p. 97-100, 1990.

SHEU, S., HUA, K., TAVANOPONG, W. **Chaining: A Generalized Batching Technique for Vídeo-On-Demand Systems**. In: Proc. IEEE International Conference on Multimedia Computing and Systems, Ottawa, Canada, junho 1997.

SYMES, Peter. **MPEG**. In: Digital Video Compression, p. 153-170, New York, 2003.

TANEMBAUM, Andrew. **Redes de Computadores**. 4ª. Edição, Ed. Campus, p. 745-757, 2003.

TIVO - **The TIVo Home Page**. Disponível: http://www.tivo.com [capturado em julho 2006].

TRIVEDI, K. Shridharbhai. **Poisson**. In: Probability and Statistics with Reliability, Queuing, and Computer Science Applications. John Wiley & Sons, INC, p. 297-304, 2002.

VENKATA, N. P., HELEN, J. W., PHILIP, A. C. **Distributing Streaming Media Content Using Coopeartive Networking**. Flash Crowd, Miami, FL, USA, maio 2002.

VENKATA, N. P., KUNWADEE, S., **The Case for Cooperative Networking**. IPIPS – 1st International Workshop a Peer-to-Peer Systems, Cambridge, MA, USA, março 2002.

VENKATRAMANI, C., VERSCHEURE, O., FROSSARD, P., LEE, K-W. **Optimal Proxy Management for Multimedia Streaming in Content Distribution Networks**. In: Proc. NOSSDAV'02, Miami, FL, USA, maio 2002.

ZHU, LI, **Proxy Caching for Video on Demand Systems in Multicasting Networks**.

ZIPF, G. K., **Some Determinants of the Circulation of Information**. Amer. J. Psycologhy 59 (1946) 401-421.

8 APÊNDICE

8.1 APÊNDICE 1: MÉTRICAS E PARÂMETROS UTILIZADOS NAS SIMULAÇÕES

Nas avaliações de desempenho e comparativos, foram utilizadas as seguintes métricas:

- Taxa de utilização do servidor: média de utilização do tempo utilizado por cada canal do enlace do servidor.
- Bloqueados/chegados: relação entre pedidos que não puderam ser atendidos (por razão de falta de memória ou largura de banda insuficiente) dividido pelo total de pedidos chegados. Pode varia entre zero e um.
- Taxa de utilização da memória dos *proxies*: equivale a uma média obtida entre o estado da memória de cada *proxy* ao término da simulação (mais precisamente aos 3500 segundos).
- Total de clientes atendidos pelo sistema: Quantidade de clientes atendidos por cada *proxy* aos 3500 segundos da simulação.

Os parâmetros utilizados foram os seguintes:

- Tamanho de um slot: 8 segundos.
- Tamanho mínimo de um buffer colapsado: 4 slots
- Trace de vídeo: Platoon, de Oliver Stone. Formato MPEG-2, 30fps, duração de 1 hora, *wide screen* e taxa média 8Mbps.
- Capacidade do enlace dos *proxies*: ADSL capaz de sustentar um fluxo de vídeo.
- Parâmetro Zipf para 100 vídeos: 0.271 (nas simulações com 1 vídeo a popularidade será sempre 1 independente desse parâmetro).

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo