Universidade Federal Fluminense

Instituto de Computação

Graduação em Ciência da Computação

Caio Cesar Abud Campos

Comandos para interação entre Cliente/Servidor em uma aplicação multimídia

Niterói

2011

Caio Cesar Abud Campos

Comandos para interação entre Cliente/Servidor em uma aplicação multimídia

Monografia apresentada ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obter o Grau de Bacharel em Ciência da Computação

Orientadora: D.Sc. Anna Dolejsi Santos

Niterói

2011

Caio Cesar Abud Campos

Comandos para interação entre Cliente/Servidor em uma aplicação multimídia

Monografia apresentada ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obter o Grau de Bacharel em Ciência da Computação

Aprovada em agosto de 2011

Banca Examinadora

_____.

Anna Dolejsi Santos, D.Sc. - Orientadora

_____.

Maurício Kischinhevsky, D.Sc.

_____.

Regina Célia Paula Leal Toledo, D.Sc.

Agradecimentos

Aos meus pais e familiares pelo apoio e força durante este curso, à professora orientadora, Anna Dolejsi Santos pela paciência e confiança, e agradecer também à todas as pessoas que de alguma forma contribuíram com esse projeto.

Resumo

Áudio e vídeo armazenados atendem a muitas situações atuais, especialmente as ligadas à educação (ex.: vídeo-aulas), ao trabalho (ex.: vídeo-conferências) e ao entretenimento (ex.: *sites* de vizualização de videos). No último caso, servidores multimídia especializados no atendimento de grande número de clientes se fazem necessários. Neste trabalho, adicionamos controles de reprodução aos servidores multimídia implementados no projeto final do Paulo Roberto P. Malafaia Junior e do Rafael Esteves Mansano[3].

Para a adição dos controles de transmissão aos servidores multimídia, estudamos o que são e como funcionam estes servidores. Com base neste estudo, implementamos os controles de transmissão nos protótipos de servidores/clientes multimídia, refatorando os códigos fornecidos pelo projeto final em [3]*, que utilizam* transmissão *unicast* e *multicast*.

Nos clientes, foram adicionados comandos que informam ao servidor as ações do usuário. Nos servidores, foram adicionados controles para atender aos pedidos dos clientes. Com o objetivo de otimizar a utilização de recursos nos servidores e diminuir o fluxo de dados pela rede, foram adicionados controles para desativar *threads* sem "ouvintes" e para encontrar *threads* ativas que sirvam às necessidades do cliente. O resultado é uma maior escalabilidade. Através de experimentos, essas implementações foram avaliadas quanto ao número de fluxos de dados na rede e quantidade de dados transmitidos pelos servidores.

Sumário

Índice de Figuras

Índice de Tabelas

1 Introdução

Nas diversas situações em que o usuário controla a reprodução do vídeo, as aplicações multimídia são responsáveis por controlar as relações de tempo que existem entre a transmissão de um fluxo contínuo de mídia e a sua exibição no destinatário. Como a internet oferece o serviço do "melhor esforço" – que não provê as condições necessárias para as aplicações multimídia – cabe às aplicações a tarefa de solucionar os problemas de atrasos e perdas de pacotes; variação de atraso na entrega de pacotes; e ausência de suporte à transmissão *multicast.*

Como exemplo de aplicações multimídia, podemos destacar: o Skype (www.skype.com), que possibilita conversas de áudio e vídeo entre dois usuários ou uma vídeo-conferência entre até 4 usuários; e o Youtube (www.youtube.com), que disponibiliza na internet milhares de vídeos para milhares usuários, permitindo a estes o controle sobre a reprodução do vídeo escolhido.

Nesse trabalho, apresentamos um estudo sobre aplicações multimídia para transmissão de áudio e vídeo de fluxo contínuo, armazenados entre um servidor e um cliente. São levadas em conta diversas situações em que o cliente pode pausar, avançar, retroceder, parar e seguir normalmente a reprodução. O objetivo é aumentar a escalabilidade da aplicação, através da diminuição da utilização de recursos de transmissão na rede e de recursos no lado do servidor.

Este trabalho está organizado em sete capítulos. No Capítulo 2, apresentamos alguns conceitos básicos necessários para a compreensão do trabalho desenvolvido. No Capítulo 3, descrevemos algumas classes de aplicações multimídia. No Capítulo 4, apresentamos técnicas para compartilhamento de recursos que utilizam o protocolo *multicast,* minimizando os requisitos de largura de banda passante dos servidores. No Capítulo 5, detalhamos as implementações do cliente e do servidor multimídia, que visam a redução do uso de banda passante e utilização de recursos do servidor. No Capítulo 6, mostramos os experimentos realizados e os resultados obtidos. No Capítulo 7, fazemos as conclusões e damos sugestões para trabalhos futuros.

2 Conceitos Básicos

Neste capítulo apresentamos o protocolo de tempo real RTP e o IP *multicast* utilizados na implementação do cliente e do servidor multimídia.

2.1 RTP (*Real Time Protocol*)

O RTP (R*eal Time Protocol*) [2] é um dos padrões que disponibilizam uma estrutura de pacotes padronizada que possuem campos para dados de áudio/vídeo, número de sequência e marcas de tempo, bem como outros campos potencialmente úteis às aplicações multimídia, pois estas anexam campos de cabeçalho às porções de áudio/vídeo que enviam aos clientes, antes de passa-las à camada de transporte. Estes cabeçalhos contém números de seqüência e marcas de tempo [1]. As Porções de dados de áudio e vídeo gerados por um servidor de uma aplicação multimídia são encapsuladas em pacotes RTP. Cada pacote, por sua vez, é encapsulado em um segmento UDP.

O desenvolvedor acredita que o RTP não faz parte da camada de transporte, mas sim da camada de aplicação. Isso porque o desenvolvedor tem de integrar o RTP à aplicação. No servidor, o desenvolvedor deve escrever o código da aplicação para criar os pacotes de encapsulamento RTP. A aplicação então envia os pacotes RTP para uma porta de interface UDP. No cliente da aplicação, os pacotes RTP entram na aplicação através de uma porta de interface UDP. Portanto, o desenvolvedor deve programar um código na aplicação para extrair as porções de mídia dos pacotes RTP.

O RTP em si não fornece nenhum mecanismo que assegure a entrega de dados a tempo nem fornece outras garantias de qualidade de serviço. Ele não garante a entrega dos pacotes e nem evita a entrega dos pacotes fora de ordem. Os roteadores não distinguem os datagramas IP que carregam pacotes RTP dos demais datagramas que trafegam na rede. Deixando assim a implementação desse mecanismo para os servidores e clientes.

2.1.1 Cabeçalho do RTP

Os quatro principais campos do cabeçalho de um pacote RTP são os campos de tipo de carga útil, número de seqüência, marca de tempo e identificador da fonte(Tabela 2.1).

Tipo de Carga Útil	Número de Seqüência	Marca de Tempo	Identificador de Sincronização da Fonte

Tabela 2.1- Principais campos do cabeçalho RTP

O campo de **tipo de carga útil** do pacote RTP tem 7 bits de comprimento, e serve para indicar o tipo de codificação de áudio/vídeo que está sendo utilizada. A Tabela 2.2 apresenta alguns tipos de carga útil suportados pelo RTP.

Número do tipo de carga útil	Formato
0	PCM
1	1016
3	GSM
7	LPC
9	G.722
14	áudio MPEG
15	G.728
26	Motion JPEG
31	H.261
32	Vídeo MPEG1
33	Vídeo MPEG2

Tabela 2.2- Tipos de Carga Útil

O campo do **número de seqüência** tem 16 bits de comprimento. Ele é incrementado de uma unidade a cada pacote RTP enviado e pode ser usado pelo cliente para detectar perda de pacotes e restaurar a seqüência perdida

O campo de **marca de tempo** tem 32 bits de comprimento. Ele indica o instante da amostragem do primeiro bit do pacote RTP (*timestamp*). A marca de tempo é derivada de um relógio de amostragem no servidor.

O campo **identificador de sincronização da fonte** tem 32 bits de comprimento e identifica a fonte do fluxo RTP. Em geral, cada fonte de uma sessão RTP tem um SSRC (*Synchronization Source Identifier*) distinto. O SSRC é um número atribuído aleatoriamente quando uma nova corrente é iniciada. Caso seja atribuído o mesmo SSRC a duas correntes, se isso acontecer, as fontes escolherão novos valores.

2.2 *Transmissões de dados*

As transmissão *unicast* é descrita por uma transmissão de um servidor para um cliente. Durante alguns anos transmissões *unicast* eram suficientes na Internet, até 1993, quando a primeira implementação de *multicast* foi feita numa distribuição do Unix BSD 4.4 [18]. A transmissão *multicast* pode ser utilizada para reduzir o tráfego de pacotes na rede. Aplicações multimídia mostram um cenário onde a transmissão *multicast* pode ser utilizada. A Figura 2.1 exibe a principal vantagem de uma transmissão *multicast* sobre uma transmissão *unicast*. A unicast utiliza-se de 3 fluxos para transmitir o dados a 3 clientes enquanto a multicast com 1 fluxo consegue atender a quem estiver ouvindo. Ela funciona como uma estação de radio onde somente quem sintoniza no seu canal recebe seus dados.

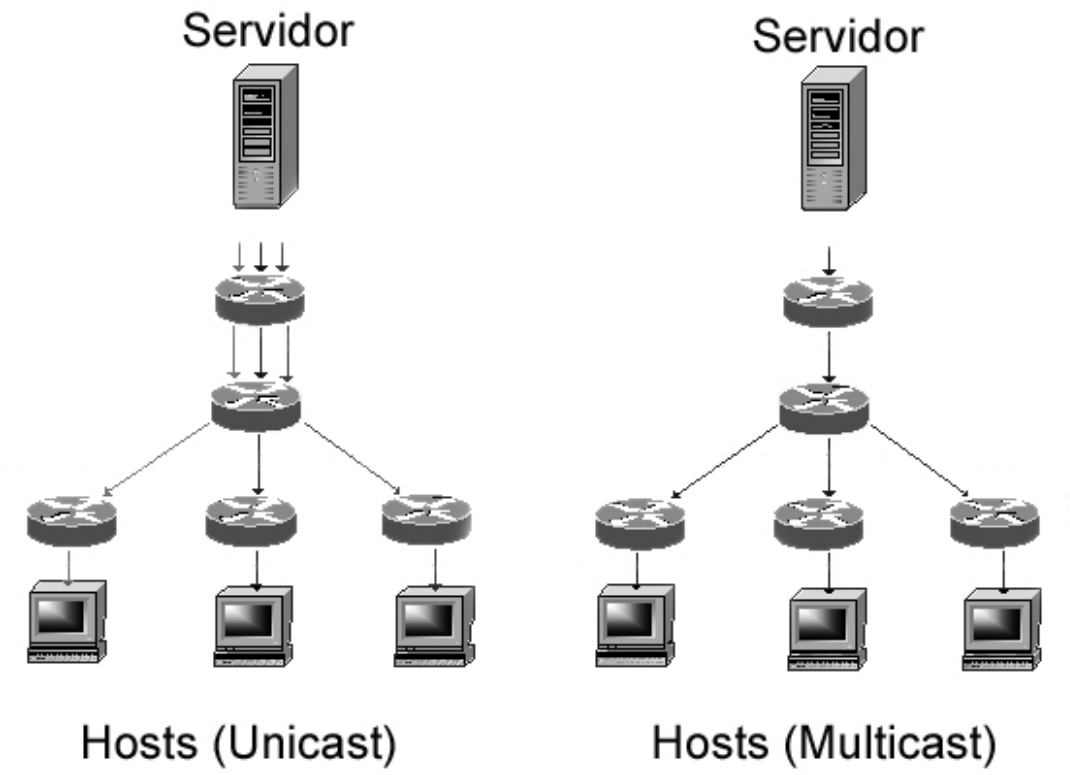

Figura 2.1- Transmissões unicast e multicast

A tecnologia atual viabiliza o "custo" de se estabelecer conexões *unicast* com todos que desejam visualizar uma página na *web*, porem, se a aplicação em questão for transmissão de áudio ou vídeo, se torna necessário uma largura de banda muito maior do que aplicações *web*, inviabilizando o "custo" de se estabelecer um canal de comunicação dedicado para cada receptor.

Como alternativa para minimizar esse problema envia-se pacotes para um endereço especial(similiar a uma freqüência de radio) aonde todos os clientes que decidirem se juntar a essa "freqüência" receberão os pacotes com esse destino. Esses endereçoes especiais são representados por faixas de IP. Essas faixas são divididas em classes baseadas nos bits mais significativos dos 32 bits do endereço IP. Todo datagrama IP cujo endereço de destino inicia com "1110" é um datagrama IP *multicast*, os demais 28 bits identificam o "grupo *multicast*" ao qual o datagrama está associado.

Existem alguns grupos *multicast* especiais, seus endereços não devem ser usados em aplicações multimídia devido aos seus propósitos bem definidos. Entre eles temos:

- 224.0.0.1 é o grupo *all-hosts*. Todos os *hosts* da rede que são capazes de enviar ou receber datagramas *multicast* devem se juntar a esse grupo.
- 224.0.0.2 é o grupo *all-routers*.Todos os roteadores *multicast* devem se juntar a esse grupo com suas interfaces disponíveis.
- 224.0.0.4 é o *all DVMRP routers*, o 224.0.0.5 como *all OSPF routers* , o 224.0.013 como *all PIM routers*, etc.

Todos esses grupos especiais são publicados em [4].

Resumidamente, a faixa de IP 224.0.0.0 até 224.0.0.255 é reservada para propósitos locais (tarefas administrativas e de manutenção) e os datagramas com esse IP de destino em seu cabeçalho não são repassados por roteadores. Assim como a faixa 239.0.0.0, até 239.255.255.255, é reservada para "escopo administrativo".

2.2.1 Enviando datagramas *multicast*

Para que uma aplicação envie datagramas *multicast* ela precisa apenas abrir um *socket* UDP, e preencher o endereço de destino com um IP de classe D como mostra a Figura 2.2 .

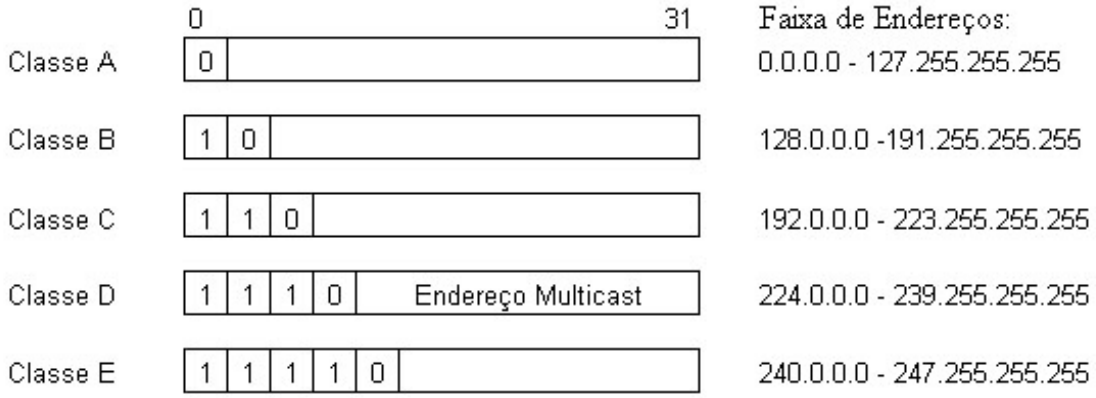

Figura 2.2 – Classes de IP

Os hosts podem ter três níveis de conformidade de acordo com a especificação *multicast* e com os requerimentos que eles atendem.

Nível 0 não suporta IP *multicast,* muitos *hosts* e roteadores na Internet possuem essa classificação, já que o suporte a *multicast* não é obrigatório no IPv4 (IPv6 o torna obrigatório).

Nível 1 suporta o envio mas não o recebimento de datagramas IP *multicast*.

Nível 2 suporta completamente IP *multicast*. São capazes tanto de enviar como receber datagramas *multicast*; precisam incluir uma implementação do IGMP (*Internet Group Management Protocol*) [5] na sua pilha TCP/IP, e os protocolos MOSPF (*Multicast* OSPF) ou PIM (*Protocol Independent Multicast*).

2.2.2 TTL (Time To Live)

O campo TTL (*Time To Live*) no cabeçalho IP é muito importante para *multicast*, ele controla o "tempo de vida" de um datagrama na rede para evitar que ele seja processado infinitamente devido a erros de roteamento. Roteadores decrementam o TTL de cada datagrama cada vez que processado, quando esse valor chega a zero o datagrama é descartado.

O TTL no IPv4 *multicast* também significa um limite para o repasse desses pacotes pela rede. Existe uma necessidade de limitar como o tráfego *multicast* se expandirá em uma rede. Cada roteador possui um limite de TTL associado a cada uma de suas interfaces, e apenas repassa os datagramas que possuírem um TTL menor ou igual ao que suas interfaces permitem. Desta forma é possível controlar o escopo de uma aplicação.

Uma lista de limites TTL e seu escopo associado está descrita na Tabela 2.3.

TTL	**Escopo**
0	Restrito ao mesmo *host*. Não é enviado para nenhuma interface.
1	Restrito à mesma sub-rede. Não é repassado por um roteador.
<32	Restrito a uma mesma região, departamento ou organização.
<64	Restrito à mesma região geográfica
<128	Restrito ao mesmo continente.
<255	Escopo sem restrição. Global

Tabela 2.3- Escopo de TTL multicast

O significado real de "mesma região" ou "região geográfica" não é bem delimitado, cabe aos administradores decidir até onde esses limites se aplicam.

2.2.3 Unindo-se a um grupo *multicast*

Para que um *host* possa receber datagramas *multicast*, este precisa se unir a um grupo *multicast*. Datagramas *multicast* são filtrados pelo hardware ou pela camada IP ou até mesmo por ambos. Apenas os que pertencerem ao grupo previamente registrado serão aceitos.

Essencialmente, quando um *host* se une a um grupo ele avisa ao sistema operacional (que ignora datagramas *multicast* por padrão), que está interessado em um grupo específico e que datagramas com aquele endereço de destino devem ser entregues aos processos interessados. O sistema operacional então utiliza o protocolo IGMP (*Internet Group Membership Protocol*) [6] para informar à rede que ele deseja fazer parte de um determinado grupo.

2.2.4 Saindo de um grupo *multicast*

Quando um processo não está mais interessado em um determinado grupo, ele informa ao sistema operacional que deseja sair desse grupo. O sistema operacional então utiliza o protocolo IGMP para informar à rede que ele deseja sair desse determinado grupo.

Como a entrada em um grupo não é por processo e sim por *host,* o *host* deixa de aceitar esses datagramas somente quando todos os processos interessados em um grupo deixarem o grupo.

3 Aplicações Multimídia

Existem diferentes tipos de aplicações cliente/servidor, como *Web*, e-mail, FTP e as aplicações multimídia. As primeiras são tidas como aplicações elásticas, pois o tempo em que os dados ficam à disposição ou se apresentam para o usuário tem menos importância que a recepção correta dos mesmos. As aplicações multimídia possuem diferentes requisitos, a perda de alguns pacotes durante a transmissão pode ser tolerada, porem o atraso na recepção dos dados pode comprometer o desempenho dessas aplicações.

Na maioria das aplicações multimídia, pacotes que sofrem atrasos de transmissor a receptor de mais do que algumas centenas de milissegundos são inúteis. Por outro lado, perdas ocasionais causam apenas pequenas perturbações na recepção de conteúdo multimídia [1]. As características dessas aplicações precisam ser observadas pelos seus desenvolvedores, especialmente quando a transmissão das informações trocadas entre os processos cliente e servidor é feita através da Internet que implementa somente o "serviço do melhor esforço". Na última seção desse capítulo são mostrados os obstáculos que as aplicações multimídia enfrentam ao usar o serviço de melhor esforço.

Segundo Kurose [1], as aplicações multimídia podem ser agrupadas em três classes: áudio e vídeo de fluxo contínuo armazenados, áudio e vídeo de fluxo contínuo ao vivo e áudio e vídeo interativos em tempo real. Nas próximas seções serão apresentadas essas três classes de aplicações.

3.1 Áudio e vídeo de fluxo contínuo ao vivo

Da visão do cliente, esta classe de aplicações é semelhante à transmissão de rádio e televisão, com uma diferença, a transmissão é realizada pela Internet. Como a abrangência da grande rede é mundial, isso permite que um usuário receba um conteúdo multimídia em praticamente qualquer parte do mundo.

Como o fluxo multimídia neste caso é gerado ao vivo (e não armazenado), o cliente não pode avançar a apresentação da mídia que está recebendo, porém, caso haja algum tipo de armazenamento local, ele pode realizar ações como pausa e retrocesso.

A distribuição de multimídia para vários usuários simultaneamente pode ser feita usando algumas técnicas de IP *multcasting*. O método mais utilizado atualmente é a transmissão *peer-to-peer*. A trasmissão *peer-to-peer* é um formato de rede de computadores cuja principal característica é descentralização das funções convencionais de rede, ou seja, o computador de cada usuário conectado acaba por realizar funções de servidor e de cliente ao mesmo tempo. *SopCast [7] e PPLive [8]* são exemplos a serem considerados.

Aplicações que utilizam *peer-to-peer* como método de distribuição de mídia para vários clientes ao mesmo tempo podem suportar atrasos de alguns segundos antes do início da reprodução sem que atrapalhe de forma vital o fluxo multimídia.

3.2 Áudio e vídeo interativos em tempo real

Essa classe compreende aplicações que usam áudio e vídeo para permitir a comunicação em tempo real entre pessoas. Basicamente esta classe se divide em duas categorias: áudio interativo em tempo real, também conhecido como telefone por Internet e, vídeo interativo em tempo real, denominada videoconferência [1].

As aplicações de telefone por Internet são atrativas porque podem oferecer serviços de central privada de comutação telefônica, serviços locais e de longa distância a um custo baixo. Além disso, dado que seu local de execução é em sistemas computacionais, este tipo de aplicação pode oferecer serviços extras, como integração telefone/*Web* e serviços de listagem telefônica, entre outros. Há ainda uma importante consideração: com esta aplicação, usuários podem fazer tanto chamadas de computador para computador como de computador para telefone convencional.

No caso de aplicações de videoconferência, os usuários se comunicam tanto visualmente quanto oralmente. Ao usar um programa de videoconferência, o usuário, que provavelmente estará junto a outros tantos numa sessão, pode mover-se ou falar a qualquer instante, isso traz a esta classe de aplicações requisitos bastante severos quanto à temporização. Como conseqüência, numa interação entre vários usuários, o atraso entre o momento em que um deles realiza algum movimento ou pronuncia alguma palavra e o momento em que a ação é percebida pelos outros usuários deve ser menor do que algumas centenas de milissegundos.

3.3 Áudio e vídeo de fluxo contínuo armazenados

Nesta classe, clientes solicitam arquivos de áudio e vídeo que estão armazenados em servidores. Na grande maioria das vezes estes arquivos são primeiramente comprimidos antes de serem gravados em disco nos servidores. Essa classe tem três características fundamentais que são mostradas a seguir [1].

Mídia armazenada: Essa característica permite que os usuários avancem a apresentação do conteúdo multimídia, já que os arquivos foram previamente gravados e armazenados em servidores. Ações de pausa, retrocesso e reinicio também são comuns nesse tipo de aplicação, e cada uma delas deve ter um tempo de resposta menor que 10 segundos para que seja considerada aceitável.

Fluxo contínuo *(Streaming)***:** Esse mecanismo faz com que o cliente inicie a reprodução do arquivo de áudio e vídeo alguns instantes após começar a recebê-lo do servidor. Geralmente alguns segundos são dados antes que o cliente comece a reproduzir o conteúdo multimídia. Evita-se assim, que o receptor tenha que aguardar o descarregamento do arquivo por completo antes de começar a exibi-lo.

Reprodução contínua: O conteúdo multimídia, como vimos anteriormente, está previamente gravado no servidor. Conseqüentemente, essa mídia possui uma temporização original de gravação que deve ser observada e seguida durante a reprodução no cliente. Isso torna o atraso na entrega de dados um obstáculo a ser superado na Internet, pois esses dados devem chegar ao cliente a tempo de serem reproduzidos.

Aplicações de áudio e vídeo de fluxo contínuo armazenados são muito comuns atualmente. Dentre as classes de aplicações multimídia, essa é a que obteve maior sucesso ao longo dos últimos anos. A constante queda no custo de armazenamento, melhorias estruturais na Internet – tais como acesso residencial de alta velocidade, técnicas de distribuição de conteúdo e novos protocolos voltados para QoS (*Quality of Service*) – e uma demanda reprimida por vídeo de alta qualidade podem ser apontadas como causas dessa popularidade.

Os servidores usados em sistemas multimídia geralmente são voltados exclusivamente para a transmissão de áudio e vídeo de fluxo contínuo. Servidores *Web*, no entanto, também podem atender a este tipo de demanda, caso em que a solicitação do arquivo é geralmente feita através de um navegador (*browser*) que

lança o reprodutor de mídia. Essas aplicações recebem a requisição do cliente e enviam para portas de saída o arquivo multimídia solicitado. Na prática são usadas tanto portas TCP (*Transport Control Protocol*) quanto UDP (*User Datagram Protocol*).

Antes de ser enviado pela rede, o arquivo multimídia é dividido em pacotes e estes então são encapsulados com cabeçalhos apropriados para o tráfego de áudio e vídeo. O RTP *(Real Time Protocol)* é um protocolo de domínio público para esta finalidade. Outro protocolo de domínio público usado nas aplicações de fluxo contínuo é o RTSP *(Real Time Streaming Protocol)*, é ele que permite ao usuário realizar as ações interativas, como pausar, voltar ou mesmo avançar na apresentação do arquivo multimídia.

A aplicação que efetivamente irá reproduzir o arquivo multimídia deverá possuir uma interface que permita ao usuário interagir com a mídia, além disso, ela deverá desempenhar algumas funções específicas para que funcione adequadamente.

A **descompressão** de áudio e vídeo enquanto são reproduzidos deve ser realizada, uma vez que o conteúdo geralmente é comprimido para não ocupar muito espaço de armazenamento e diminuir a demanda por largura de banda para transmissão.

A **eliminação da variação de atraso**, que é a variação dos atrasos fim-a-fim dos pacotes dentro de um mesmo fluxo, deve ser feita, já que os dados transportados possuem restrições de temporização, tendo que ser reproduzidos à mesma taxa que foram gravados. Para tanto, os pacotes são colocados durante um pequeno período de tempo em um *buffer*.

Esse pequeno período de tempo pode ser estimado, com base na variação de atraso de pacotes de testes enviados do servidor para o cliente antes do início da transmissão da mídia. Além disso, esse intervalo pode ir se adaptando de acordo com estimativas realizadas durante a transmissão. Nesse caso, a variação de atraso dos próprios pacotes do fluxo multimídia é usada para atualizar o valor do intervalo.

A **correção de erros** devido à perda de pacotes, que é causada na maioria das vezes pelo congestionamento na Internet, é essencial a sistemas de aplicações multimídia de fluxo contínuo armazenado. Esses sistemas tentam se recuperar de perdas, reconstruindo pacotes perdidos por meio de transmissão de pacotes redundantes, fazendo com que o cliente solicite a retransmissão do pacote perdido ou atenuando os efeitos da perda através de técnicas de ocultamento de erro.

3.4 Obstáculos para aplicações multimídia

A Internet utiliza em sua camada de rede o protocolo IP que provê um serviço de melhor esforço a **todos** os datagramas que transporta. Esse serviço implica que não são dadas as aplicações garantias de atraso fim-a-fim, da variação de atraso e tão pouco é dada a garantia que os datagramas serão entregues no destino. Como os protocolos da camada de transporte TCP e UDP executam sobre o IP, nenhum deles pode dar garantia quanto a atraso às aplicações a que servem. Esses fatores fazem com que no desenvolvimento de aplicações multimídia seja necessária a aplicação de técnicas que minimizem os efeitos do serviço de melhor esforço.

A seguir, nos aprofundamos um pouco mais nos obstáculos que as aplicações multimídia têm que superar para garantir um serviço satisfatório para o seu usuário.

O **atraso fim-a-fim** é a soma dos atrasos de processamento em roteadores e *hosts* de transmissão e propagação nos enlaces e de formação de filas em roteadores. O atraso fim-a-fim tem diferentes implicações para cada classe de aplicações como veremos a seguir.

Aplicações de áudio e vídeo interativos em tempo real são altamente sensíveis a atrasos fim-a-fim. Para que a interatividade entre os usuários seja aceitável, os atrasos de origem a destino não devem ultrapassar os 400 milissegundos [1]. Na prática, um pacote que sofra um atraso maior que este limite é inútil para a aplicação e acaba sendo descartado.

No caso de aplicações de áudio e vídeo de fluxo contínuo ao vivo, as restrições quanto ao atraso fim-a-fim não são tão severas. Um atraso de poucos segundos entre a geração do pacote no transmissor e a exibição do mesmo no receptor não é crítico. Entretanto, se esse atraso ultrapassar a casa das dezenas de segundos ele começa a comprometer a *performance* da aplicação, já que um fluxo de mídia que teoricamente estaria sendo exibido ao vivo, na realidade estaria consideravelmente defasado.

Para aplicações de áudio e vídeo de fluxo contínuo armazenados, as restrições quanto ao atraso fim-a-fim são menores ainda em relação às outras classes de aplicações. Como o fluxo de mídia não é exibido ao vivo, por estar gravado e armazenado no servidor, esse tipo de aplicação possui uma tolerância maior quanto ao

atraso inicial para a exibição. Uma situação onde o atraso fim-a-fim é realmente um empecilho para esta classe de aplicações, é quando esse atraso atrapalha a interatividade do usuário com a apresentação da mídia. As ações de pausa, reinicio e retrocesso, entre outras, devem ter um tempo de resposta adequado para que a reprodução da mídia não se torne frustrante.

A **variação de atraso *(jitter)***, que é na realidade a variação dos atrasos fim-a-fim de pacotes pertencentes a um mesmo fluxo, é decorrente dos atrasos aleatórios das filas de roteadores pelas quais os pacotes passam durante o percurso do transmissor ao receptor.

Essa variação de atraso pode ser tanto positiva quanto negativa. Isso quer dizer, que pacotes consecutivos enviados periodicamente podem chegar ao destino com uma diferença temporal entre eles maior ou menor que o intervalo de envio original.

A **perda de pacotes** (datagramas neste caso) pode ser compreendida olhando-se a camada de rede da Internet. Nessa camada, os datagramas enviados pelo servidor passam por várias filas de roteadores antes de chegarem ao cliente. Se uma dessas filas estiver cheia, os datagramas não poderão ser aceitos pelo roteador, e conseqüentemente, os dados transportados serão perdidos.

Uma forma de tratar a perda de dados seria utilizar o TCP para o transporte de mídia, uma vez que esse protocolo retransmite pacotes perdidos pela camada de rede. No entanto, a retransmissão de pacotes traria um efeito indesejado para aplicações multimídia: o aumento do atraso fim-a-fim [9]. Além disso, o TCP entra na fase de partida lenta após a perda de pacotes, o que causaria um impacto significativo no desempenho de sistemas multimídia. Por essas razões, as aplicações multimídia, em especial as aplicações de áudio e vídeo interativos em tempo real, utilizam o UDP como protocolo de transporte.

Como mencionado no início dessa seção, a perda de pacotes pode ser tolerada desde que ocorra ocasionalmente. Para conseguir isso, as aplicações multimídia fazem uso de alguns mecanismos, como o envio de informações redundantes, o encobrimento de erros e a intercalação dos dados. A utilização dessas técnicas possibilita a manutenção da qualidade da apresentação da mídia mesmo que alguns pacotes sejam perdidos.

Como os conteúdos multimídia possuem restrições de temporização quanto a sua reprodução, são necessários mecanismos que eliminem a variação de atraso e as

perdas de pacotes. Pacotes com **número de seqüência** e **marcas de tempo** e o **atraso da reprodução** são usados para este fim. Mais adiante, no próximo capítulo, mostraremos como esses mecanismos podem ser usados para atenuar os efeitos da variação de atraso.

4 Técnicas para compartilhamento

Os servidores multimídia podem fornecer dados aos seus clientes através de fluxos distintos, consequentemente, ocorre um esgotamento dos recursos do sistema, por exemplo a banda disponível.

Para minimizar esse problema podemos fazer com que vários clientes compartilhem um mesmo fluxo de dados, utilizando técnicas de compartilhamento de recursos e *multicast*, com o objetivo de reduzir a demanda por largura de banda, espaço em disco e velocidade de processamento, e ainda garantindo uma QoS (Qualidade de Serviço) aos clientes.

Nesta literatura estudamos as técnicas de compartilhamento de recursos orientada a requisições de clientes. Essas técnicas são baseadas na transmissão de um fluxo contínuo de mídia em resposta às requisições de múltiplos clientes.

4.1 Técnica *Batching*

A abordagem da técnica *batching* [11] é bem simples, tentar agrupar os clientes sempre que possível. Assim que o servidor recebe uma requisição, uma janela de *batching* é aberta e todas as requisições seguintes são enfileiradas até o término dessa janela. Quando essa janela expira, um único fluxo de mídia é iniciado e compartilhado por todos os clientes, como mostra a Figura 5.1. Com isso reduzimos o consumo de banda, mas no entanto introduzimos uma latência de início de reprodução para os clientes, pois o primeiro cliente alocado a uma janela terá que esperar essa

janela expirar para começar a receber a mídia requisitada.

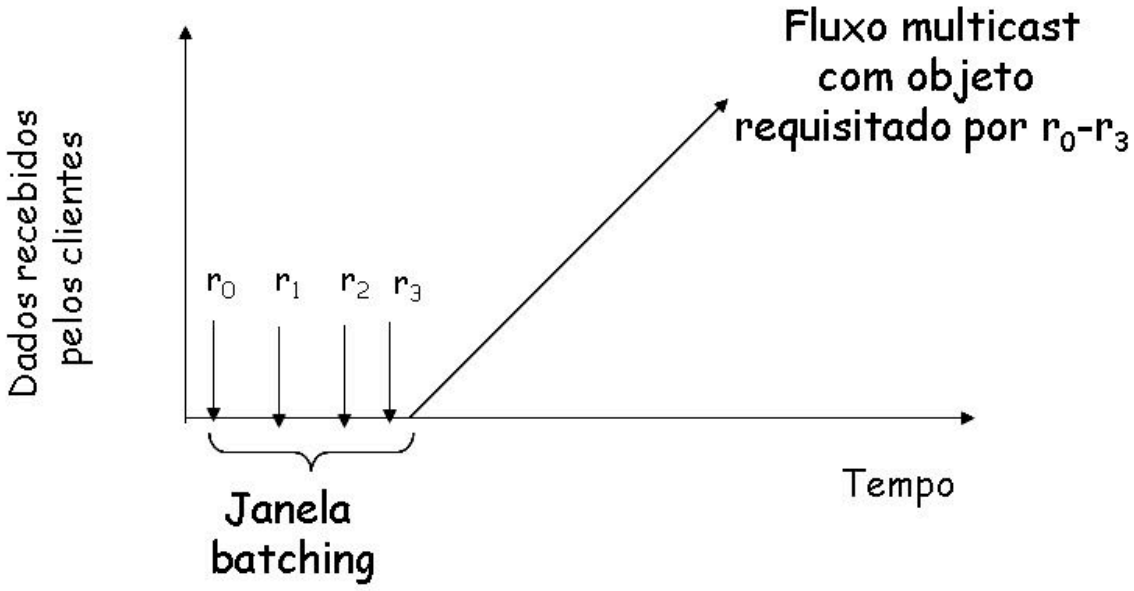

Figura 4.1- Técnica de batching[3]

4.2 Técnica *Patching*

A abordagem da técnica *patching* [12] é mais elaborada, se novos clientes chegarem e requisitarem a mídia que já está sendo transmitida, eles automaticamente são agrupados à transmissão em andamento, e começam a armazenar em *buffer* os dados transmitidos. Para obter a parte inicial (chamada de *patch*), que já foi transmitida ao grupo, um canal fica disponível e todos os clientes que requisitaram uma determinada mídia são atendidos, assim como acontece na técnica de *batching*. Quando essa parte inicial é completamente exibida, o cliente começa a consumir de seu *buffer* e o canal utilizado para transmitir o *patch* é fechado, o fluxo de dados continua normalmente em apenas um canal como está ilustrado na Figura 4.2. Nessa abordagem o cliente fica responsável por manter espaço suficiente de *buffer* e ser capaz de receber dados em dois canais.

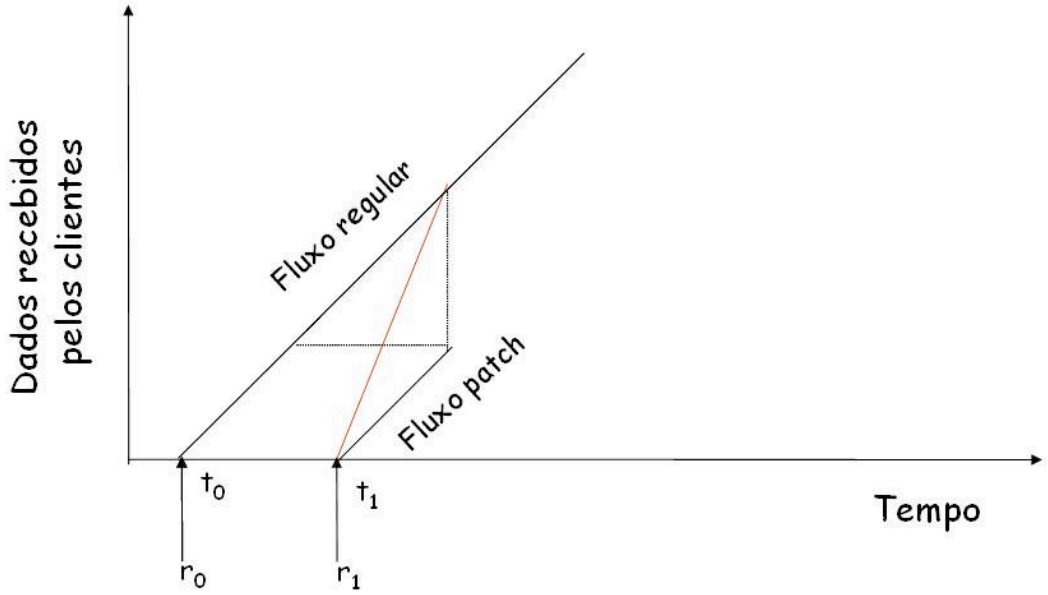

Figura 4.2- Técnica de patching[3]

O tamanho da janela é muito importante para o melhor desempenho dessas técnicas, pois se for configurada muito grande, a maioria dos canais do servidor estarão sendo utilizados para enviar *patchs*, se for configurada muito pequena não teremos um bom aproveitamento, pois poucos clientes serão agrupados em cada janela e mais canais serão necessários do lado do servidor. Modelos matemáticos para o cálculo desse tamanho ideal de janela são propostos em [14] e em [15].

4.3 Técnica *Hierarchical Stream Merging* (HSM)

A idéia básica desta técnica é a junção hierárquica dos clientes em grupos. O cliente recebe simultaneamente dois fluxos: um iniciado por ele e outro iniciado pelo cliente mais próximo. A grande questão desta técnica é escolher qual fluxo anterior cada cliente irá escutar, e para isto é desenvolvida uma árvore de *merges*. Existem diversas abordagens para a construção desta árvore e algumas delas são apresentados em [16]. Neste trabalho é determinado, através de programação dinâmica, a árvore ótima para a junção dos fluxos, considerando que os instantes de todas as chegadas são conhecidos.

Como estas chegadas de clientes não são conhecidas, então a busca por outras formas de determinação da árvore se faz necessária. São apresentadas 3 heurísticas para a construção da árvore, onde a primeira abordagem apresentada é a *Earliest Reachable Merge Target* (ERMT) [13], onde um novo cliente ou um novo grupo recém unido escuta o próximo cliente, com o qual possa se unir mais rapidamente, considerando que não haja chegadas futuras que possam ser unidas com eles. Esta decisão é feita através de uma simulação de todas as uniões a fim de determinar qual fluxo deve ser escutado para não haver desperdício de escuta. Esta simulação pode ser muito custosa computacionalmente e para minimizar este problema, foi definida a política *Simple Reachable Merge Target* (SRMT) [13], onde se determina o fluxo alcançável mais próximo considerando que cada fluxo ativo termina em seu *target merge point*, que é o instante onde cada fluxo irá se juntar com o seu fluxo mais próximo. A técnica SRMT tem uma desvantagem, pois pode haver desperdício de escuta, uma vez que um cliente poderá estar escutando um segundo fluxo que se juntará a um terceiro antes mesmo do primeiro conseguir se unir ao segundo, levando o primeiro cliente a escutar todos os dados restantes para alcançar o terceiro.

É definido, também, uma terceira abordagem chamada *Closest Target* (CT)[13], bem mais simples que as anteriores, onde cada cliente, ao chegar no sistema, escuta o fluxo ativo anterior a chegada dele, sem se preocupar se irá alcançá-lo em tempo hábil ou não.

Em [17] conclui-se que a ERMT se aproxima do resultado obtido com a árvore ótima, porém conclui, também, que a técnica CT, apesar de bem mais simples, apresenta resultados satisfatórios em termos de economia de banda.

A Figura 4.3 mostra um exemplo das uniões executadas quando utilizada a técnica de *merge* ótimo. O Cliente B escuta o fluxo do A e os clientes C e D escutam o fluxo iniciado por B. o cliente D escutará B pois ele não conseguirá se juntar com C antes de C se unir a B (pois a união C/B ocorre em 0.5), o que torna B o *merge target* alcançável mais próximo para D se chegadas futuras não forem se juntar a D.

A complexidade de implementação e de gerência do sistema multimídia utilizando essa técnica pode se tornar inviável, pois a estrutura de união de fluxos das técnicas baseadas em HSM é potencialmente uma floresta de árvores sem limite de

profundidade, impactando diretamente na complexidade do número de mensagens trocadas entre clientes e servidor. Daí, a depender da aplicação e do cenário de atendimento considerado, a eficiência absoluta atingida pode não justificar o nível de complexidade associado. Uma discussão detalhada sobre técnicas HSM pode ser encontrada em [17].

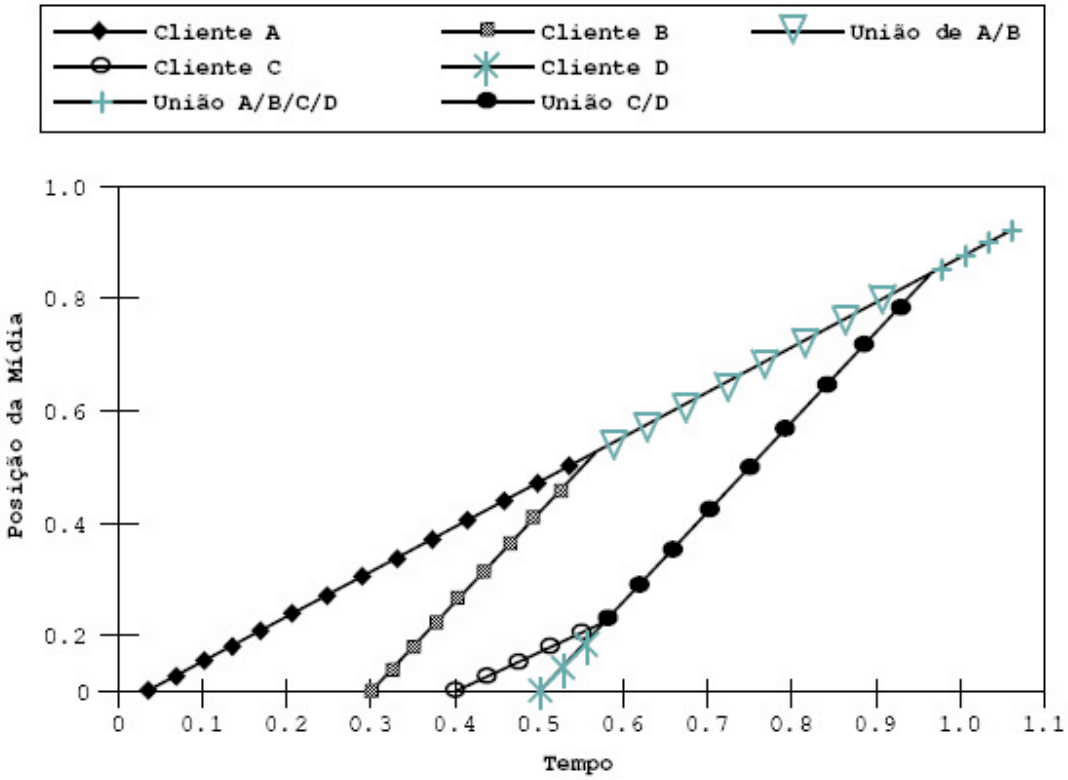

Figura 4.3- Hierarquical stream merge[3]

5 Implementação da aplicação multimídia Cliente/Servidor

Para implementarmos nos módulos servidor multimídia e cliente o controle de transmissão utilizamos a linguagem Java com o auxílio da IDE NetBeans[9], que é um projeto *open-source* dedicado a prover um bom desenvolvimento de software. A linguagem Java [10], que é orientada a objeto, foi escolhida pela sua boa documentação, maior portabilidade e por oferecer facilidades no uso de transmissão *multicast*.

Nas três diferentes versões do servidor e do cliente multimídia: uma empregando a transmissão *unicast*, outras duas usando a transmissão *multicast*, sendo que à primeira delas está associada a técnica de *batching* e à segunda a técnica de *patching*, foram desenvolvidas as ações para poder pausar, avançar, retroceder, parar e seguir normalmente a reprodução de um arquivo multimídia, para avaliar comparativamente, o número de pacotes transmitidos, isto é, a quantidade de banda passante necessária em cada uma das versões implementadas, bem como o número máximo de fluxos de dados mantidos abertos simultaneamente no servidor para atender os clientes em diversas situações onde o cliente altera a reprodução.

O esforço foi concentrado na implementação das ações nos três tipos diferentes de servidor implementados: simple, *patching* e *batching*, e em refatorar o código existente assim o tornando manutenível e mais legível. O código fonte dos três servidores e dos três clientes está disponível no Anexo A.

5.1 **Cliente/Servidor *Unicast***
Servidor:

Inicio

Enquanto o servidor estiver ativo faça;

Espere por uma requisição TCP;

Ao chegar uma requisição faça;

Inicie nova thread e envie o número de porta da thread para o cliente receber os frames;

Fim enquanto

Fim

**thread*

Inicio

Enquanto o cliente não desligar

Espere cliente definir uma ação;

Inicie o timer para transmitir os frames de acordo com a ação definida;

Fim enquanto

Fim

Cliente:

Inicio

Faça uma conexão TCP com o servidor;

Receba o número de porta para se conectar;

Enquanto cliente ligado

Espere usuário definir uma ação;

Comunique o tipo de ação ao servidor

Iniciar timer para receber e exibir os frames

Fim enquanto

Fim

Após o servidor e o cliente serem iniciados, uma troca de mensagens é feita no momento em que o cliente se conecta ao servidor. Esta comunicação é realizada inicialmente por uma conexão TCP para que o servidor através da mensagem informe ao cliente em qual porta ele deve estar escutando pela transmissão dos *frames* do vídeo. Logo em seguida, outra comunicação é realizada trocando mensgens para definir a da ação a ser executada, então a thread dedicada ao cliente iniciada pelo servidor fica escutando por novas ações do cliente assim mudando o fluxo de transmissão de acordo com a ação, o servidor continua escutando por novas conexões para atender outros clientes.

5.2 Cliente/Servidor *multicast* com *Batching*

Servidor:

Inicio

Enquanto o servidor estiver ativo faça;

Espere por uma requisição TCP;

Ao chegar uma requisição faça;

Se o cliente estiver escutando alguma trhead

Informe a trhead que o cliente não está mais escutando

Se a trhead não possuir mais ouvintes desligue-a

Fim se

Se houver uma thread reproduzindo vídeo na mesma ação que o cliente pediu e em posição próxima

Então agrupe essa requisição nessa thread;

Senão inicie uma nova "janela" de batching que ira disparar uma thread para transmitir os frames (multicast);

Fim se

Envie um número da porta para o cliente receber os frames;

Fim enquanto

Fim

Cliente:

Inicio

Enquanto cliente ativo

Espere por uma ação definida pelo usuário

Faça uma conexão TCP com o servidor informando ação requisitada pelo usuário e posição do frame corrente;

Receba o número de porta para se conectar;

Se está em um grupo multicast então saia do grupo;

Una-se ao grupo multicast na porta recebida pelo servidor;

Inicia timer para receber e exibir os frames;

Fim enquanto

Fim

Após o servidor e o cliente serem iniciados, o cliente espera por uma definição de ação pelo usuário, após a definição da ação o cliente se conecta ao servidor e passa uma mensagem informando a ação e o a posicão do frame no momento que o usuário definiu a ação. Esta comunicação é realizada inicialmente por TCP para que o servidor informe ao cliente em que porta este deve escutar pelos *frames*. O servidor inicia uma janela de *batching* sempre que um cliente se conecta e não existe nenhuma thread aberta que o atenda no momento.

5.3 Cliente/Servidor multicast com *Patching*

Servidor:

Inicio

Enquanto o servidor estiver ativo faça;

Espere por uma requisição TCP;

Ao chegar uma requisição faça;

Extraia informação sobre ação e número de frames;

Decremente a thread que o cliente está escutando se ele estiver escutando alguma thread;

Caso thread não tenha ouvintes desligue-a;

Se existir uma thread que atenda ao cliente

Envie o número de porta para o cliente se juntar à transmissão;

Senão

Inicie uma nova thread para a transmissão e envie o número da porta para o cliente se juntar a transmissão

Se existe outra thread transmitido o vídeo requisitado

Envie o número de porta para o cliente se juntar a transmissão e ir armazenando os frames

Fim se

Fim enquanto

Fim

Cliente:

Inicio

Enquanto cliente ligado

Espere por uma ação definida pelo usuário

Faça uma conexão TCP com o servidor informando ação requisitada pelo usuário e posição do *frame* corrente;

Extraia da resposta do servior o número de porta para se conectar e reproduzir os *frames* e o número de porta para ir armazenando *frames* futuros;

Caso segundo número de porta seja fornecido inicie uma thread para receber os frames futuros;

Una-se ao grupo multicast correspondente;

Enquanto não receber todos os frames faça

Receba o primeiro frame;

Se o frame recebido possui número de seqüência = número de sequencia do primeiro frame futuro recebido, então

Disconecte do servidor e consuma os frames do buffer;

Senão

Receba os frames multicast e os exiba;

Fim se

Fim enquanto

Fim enquanto

Fim

Após o servidor e o cliente serem iniciados, uma troca de mensagens é feita no momento em que o usuário define uma ação no lado cliente. Esta comunicação é realizada inicialmente por TCP para que o servidor informe ao cliente em que porta este deve escutar pelos *frames* e em qual porta deve fazer o *patch*. O servidor então inicia a transmissão dos *frames* e qualquer outro cliente que requisitar o mesmo vídeo se juntará à essa transmissão e receberá o *patch* relativo em outro canal.

6 Os Experimentos

Para testar a implementação dos aplicativos, vamos avaliar o desempenho dos servidores medindo a quantidade de fluxos de dados mantidos abertos ao mesmo tempo enquanto clientes são atendidos e a quantidades de pacotes transmitidos.

Os experimentos ocorreram no ambiente que está ilustrado na Figura 6.1, que possui nove computadores, sendo um servidor e os oito restantes clientes conectados através de um *switch* 10/100 Mbps.

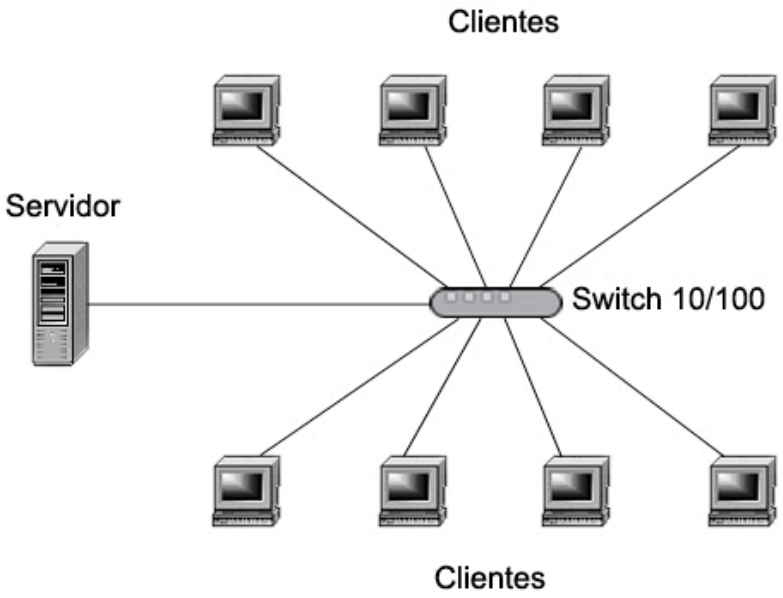

Figura 6.1- Ambiente de Testes

O vídeo utilizado nos testes possui as seguintes características: 320x240 *pixels*; 24 bits por *pixel*; 500 *frames*; duração de aproximadamente 35 segundos; Mjpeg, sem compressão.

Para realização dos testes utilizamos três situações (cenários) diferentes e um total de nove experimentos. Comparamos o desempenho das técnicas para compartilhamento de recursos de transmissão acrescidas dos controles de transmissão que foram implementados.

6.1 Cenários

Os cenários de teste que simulam uma distribuição de requisições dos clientes dentro de um espaço de tempo equivalente à transmissão de todo arquivo de vídeo para o último cliente.

6.1.1 Pico de requisições

Neste cenário simulamos o comportamento de requisições durante um horário de pico de conexões, isto é, o período de tempo em que o servidor precisa atender o maior número de clientes. Em nosso teste as requisições dos oito clientes ocorreram dentro de um intervalo de 5 segundos sendo que 4 dos 8 clientes iniciaram a reprodução com o botão "*Forward*" e os outros 4 com o botão "*Play*".

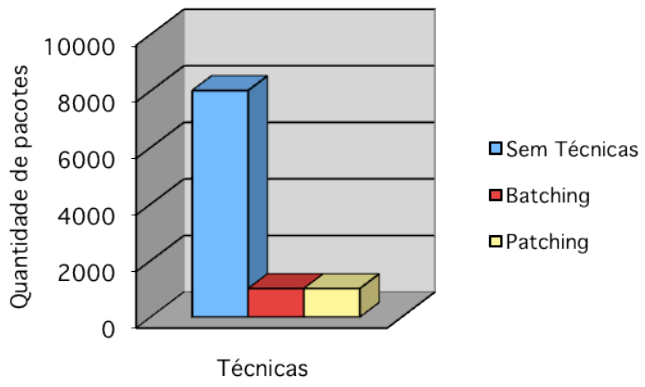

Figura 6.2- Quantidade de pacotes enviados no cenário 1

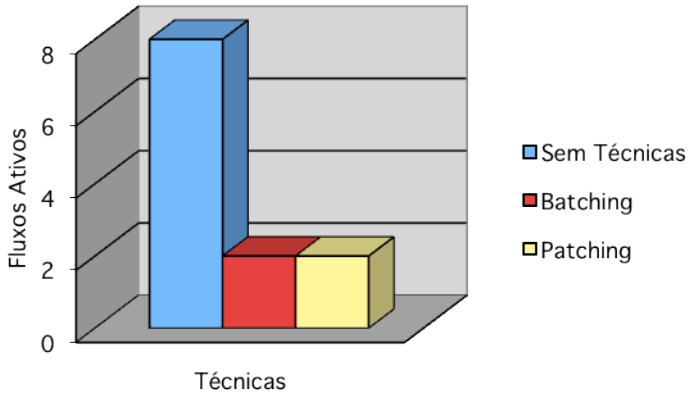

Figura 6.3- Número máximo de fluxos ativos no cenário 1

A Figura 6.2 nos mostra que, em um horário de pico, com muitas requisições, a utilização da técnica de *batching* ou de *patching* proporciona uma grande economia dos requerimentos de largura de banda permitindo ao servidor atender a um número substancialmente maior de clientes.

Ambas as figuras 6.2 e 6.3 nos mostram que a técnica de *patching*, em um momento onde todas as requisições chegam juntas ou muito próximas se torna equivalente a técnica batching.

6.1.2 Fora do pico de requisições

Neste cenário simulamos o comportamento de requisições fora do horário de pico de conexões, isto é, o período de tempo em que o servidor precisa atender o menor número de clientes. Em nosso teste fizemos com que as requisições dos oito clientes chegassem ao servidor com uma diferença de mais de 5 segundos entre elas, alternando entre requisições "*Play*" e "*Forward*".

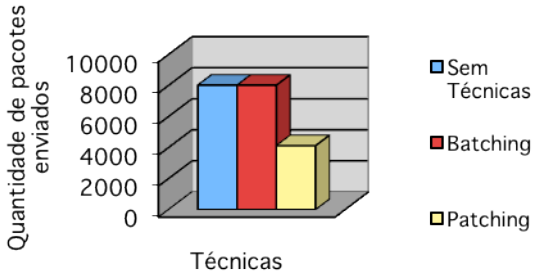

Figura 6.4- Quantidade de pacotes enviados no cenário 2

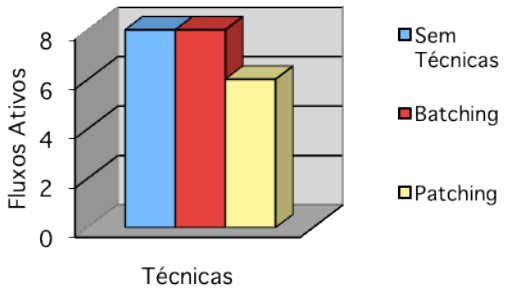

Figura 6.5- Número máximo de fluxos ativos no cenário 2

Como mostram as Figuras 6.4 e 6.5, o fato das requisições chegarem com intervalos maiores que 5 segundos, que é o tamanho da janela de *batching*, fez com que a técnica de *batching* tivesse um desempenho equivalente ao do servidor sem técnicas, pois em nenhum momento um mesmo fluxo de dados pôde ser utilizado por mais de um cliente.

Já o servidor com a técnica de *patching* teve o melhor desempenho pois enviou aproximadamente metade dos pacotes das outras técnicas e manteve menos fluxos de dados abertos simultaneamente.

6.1.3 Utilização dos controles de reprodução

Neste cenário simulamos uma situação na qual as requisições dos clientes chegam em grupos, intercalando momentos de muitas requisições com momentos de poucas requisições e uma utilização maior dos controles onde clientes param, pausam, avançam, retrocedem e seguem normalmente a reprodução do video até seu termino. Fizemos com que quatro clientes chegassem nos primeiros 5 segundos, logo após tivemos um intervalo de 10 segundos sem nenhuma requisição e então outros quatro clientes chegam. Após 20 segundos do video, os quatro primeiros retrocedem a reprodução durante 10 segundos do video e então pausam, depois de 10 segundos voltam a execução normal. O segundo grupo de clientes, depois de 25 segundos de video, retrocedem 20 segundos de video, avançam 10 segundos de video e voltam a execução normal.

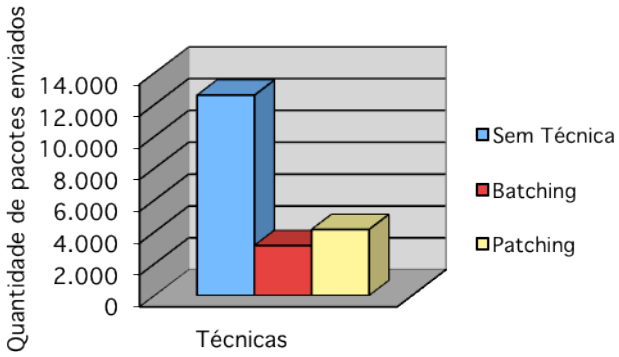

Figura 6.6- Quantidade de pacotes enviados no cenário 3

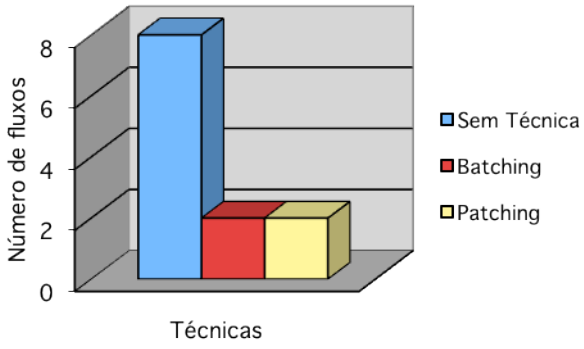

Figura 6.7- Número máximo de fluxos ativos no cenário 3

Como mostram as Figuras 6.6 e 6.7, a utilização dos controles de transmissão obteve o mesmo resultado, com relação ao máximo de fluxos ativos, que o caso de pico de requisições, pois se analisarmos, cada utilização de um controle é uma requisição ao servidor, com uma grande utilização dos controles temos muitas requisições. Observamos também que a quantidade de pacotes enviados aumentou em relação aos outros testes e a técnica de *batching* obteve um desempenho melhor que o do servidor sem técnicas.

Podemos observar que a técnica de *patching* teve o desempenho inferior a técnica de *batching*, com relação a quantidade de dados transmitidos, pois os *patchs realizados* eram descartados quando o usuário utilizava o controle para mudar a reprodução do video, iniciando novo patch.

7 Conclusão

Nesse trabalho estudamos as técnicas relacionadas ao desempenho de servidores multimídia. Vimos que a *Internet*, através do protocolo IP, oferece apenas o serviço do "melhor esforço" que significa que o IP não oferece garantias de atraso fim-a-fim, de banda passante e a maior parte dos seus provedores de serviço não implementa o IP *multicast*. Devido a esse "ambiente de trabalho" às aplicações multimídia se tornam responsáveis por solucionar os problemas de atrasos e perdas de pacotes, variação de atraso na entrega de pacotes.

Implementamos em três versões de servidor/cliente controles para reprodução do video, onde duas dessas versões empregam duas técnicas distintas para redução da necessidade de uso banda passante. Em um dos três servidores a transmissão *unicast* foi utilizada enquanto nos dois restantes a técnica de transmissão *multicast* associadas às técnicas *batching* e *patching*.

Os experimentos mostraram que a adição de controles de transmissão na reprodução nas técnicas de *patching* e *batching* que são técnicas que fornecem ao servidor economia de recursos, necessários para atender um maior conjunto de requisições, não alterou a eficacia das técnicas. Concluímos que não existe uma técnica melhor, mas conseguimos apontar qual técnica é mais apropriada para cada situação e com certeza é melhor utilizar alguma técnica do que não utilizar nenhuma pois podemos observar que o uso das técnicas traz beneficios aos servidores.

Observamos ainda, que a técnica de *batching e a técnica de patching*, onde a primeira mostra melhor desempenho quando a uma grande utilizacão dos controles de reprodução, que como vimos compreende o horário de pico de requisições, e a segunda mostra melhor desempenho em situações em que as transmissões compreendidas fora do horário de pico de requisições (ex.: mesma situação de um filme pouco popular), ficaram mais parecidas numa situação onde há o pico de requisições sem muita utilização dos controles de reprodução (ex.: mesma situação de um filme popular).

É possível estender esse trabalho, de modo a explorar outras técnicas usadas em servidores multimídia. Entre essas possibilidades, podemos destacar: implementar técnicas de recuperação de dados e adiciona-las aos módulos existentes verificando seus respectivos desempenhos; adicionar a possibilidade de reprodução de arquivos de video com diferentes formatos e verficar o impacto causado nos módulos existentes.

Bibliografia

[1] F.J. Kurose, e W.K Ross. Redes de computadores e a Internet. Addisson Wesley, 2ª edição, 2003.

[2] R. Frederick, V. Jacobson. *RTP: A Transport Protocol for Real- Time Applications*, RFC1889, Janeiro de 1996.

[3] Paulo Roberto P. Malafaia Junior e Rafael Eesteves Mansano Técnicas para Redução de Requisitos e Aumento de escalabilidade em Sistemas Multimídia, Universidade Federal Fluminense 2006.

[4] J. Reynolds. *Assigned Numbers*. RFC 1700, Outubro de 1994.

[5] R. Sharma and S. Deering. *Internet Group Management Protocol Version 2*. RFC 2236, Novembro de 1997.

[6] S. Deering. *Host Extensions for IP Multicasting*. RFC 1112, Agosto de 1989.

[7] http://www.sopcast.org/ consultado em agosto de 2010.

[8] http://www.pplive.com/en/index.html consultado em agosto de 2010.

[9] J.C. Bolot. E Andreas Vega-Garcia,Control Mechanisms for Packet Audio in the Internet, em Proceeding IEEE Infcom, páginas 232-239,199

[10] Silva, E. de Souza e et al. Performance issues of multimedia applications. em: *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*. London, UK: Springer - Verlag, 2002. p.374-404. ISBN 3-540-44252-9.

[11] S. Sheu, Kien. A. Hua, and W. Tavanapong. *Chaining: A generalized batching technique for video-on-demand*, páginas 110-117, Ottawa, Ontario, Canada, páginas

110-117, Junho de 1997.

[12] K. A. Hua, Y. Cai e S. Sheu. *Patching: A multicast techinique for true vídeo on demand services*. Em Proc.ACM Multimedia, páginas 191-200, 1998.

[13] D. L. Eager, M. K. Vernon, e J. Zahorjan. *Minimizing Bandwidth Requirements for On-Demand Data Delivery*, *Proc. MIS'99*, Indian Wells, CA, Outubro de 1999.

[14] Y. Cai, K. Hua e K. Vu. *Optimizing patching performance*. Em Proc. SPIE/ACM Conference on Multimedia Computing and Networking, 1999.

[15] L. Gao e D. Towsley. *Supplying Instantaneous vídeo-on-demand services using controlled multicast*. Em IEEE International Conference on Multimedia Computing and Systems, páginas 117-121, 1999.

[16] D. Eager, M. Vernon, e J. Zahorjan, *Optimal and e-cient merging schedules for video-on-demand servers*. Em Proc. ACM Multimedia, páginas 199-202, Janeiro de 1999.

[17] B. C. M. Netto, *Patching Interativo: um novo método para compartilhamento de recursos para transmissão de vídeo com alta interatividade*, Fevereiro de 2004.

[18] http://www.bsd.org consultado em 20 de agosto de 2011