Implementação e Avaliação de um Mecanismo Adaptativo para Redução de Jitter para o Sistema RIO

Jefferson Elbert Simões[1], **Daniel Ratton Figueiredo**[2], **Rosa Maria Meri Leão**[2]

[1]Departamento de Engenharia Eletrônica e de Computação – POLI/UFRJ
Caixa Postal 68.529 – 21.945-970 – Rio de Janeiro, RJ – Brasil

[2]Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ
Caixa Postal 68.511 – 21.941-972 – Rio de Janeiro, RJ – Brasil

elbert@land.ufrj.br, {daniel, rosam}@land.ufrj.br

Abstract. *Video applications on the Internet are being increasingly adopted. Many of these applications use streaming, making them more interactive and scalable. However, the high variability of the Internet can lead to a significant degradation of the quality of service. In this work, we propose and implement an adaptive mechanism to reduce the variability of transmission delay, designed for the VoD transmission system RIO. This mechanism is evaluated and compared with the mechanism previously used in RIO. Our results improve user interactivity without preventing a high quality transmission.*

Resumo. *Aplicações de vídeo vêm se tornando cada vez mais utilizadas na Internet. Muitas destas aplicações utilizam a técnica de streaming, que torna estes sistemas mais interativos e escaláveis. No entanto, a alta variabilidade dos canais de comunicação da Internet pode levar a uma grande degradação da qualidade de serviço oferecida pela aplicação. Neste trabalho, propomos e implementamos um mecanismo adaptativo para redução da variabilidade do retardo de transmissão, projetado para o sistema de transmissão de vídeo sob demanda RIO. O mecanismo proposto é avaliado e comparado com o mecanismo já existente no sistema RIO. Nossos resultados sugerem que é possível melhorar a interatividade do usuário sem impedir uma transmissão de alta qualidade.*

1. Introdução

As aplicações de transmissão de vídeo na Internet estão entre as aplicações mais difundidas dos últimos anos. O interesse por este tipo de aplicação tem sido cada vez maior, devido à sua grande abrangência, que vai desde educação até entretenimento. Um dos desafios deste cenário é garantir aos sistemas de transmissão de vídeo não somente vídeo com alta qualidade, mas também uma alta escalabilidade do serviço, que deve ser capaz de suportar grandes quantidades de usuários.

Diversas técnicas foram propostas com o objetivo de desenvolver sistemas de transmissão de vídeo escaláveis e capazes de oferecer uma alta qualidade de serviço. Podemos citar, em particular, a técnica de *streaming*, na qual o vídeo é transmitido sob demanda conforme é reproduzido pelo usuário. Esta técnica permite que usuários assistam vídeos antes de eles serem completamente transmitidos e torna mais eficiente a utilização dos canais de transmissão. No entanto, o desempenho deste tipo de técnica é

diretamente afetado pela variabilidade de métricas da rede, como retardo de transmissão e taxa de perda. Em particular, a variação no retardo de transmissão, ou *jitter*, possui grande influência na qualidade do vídeo transmitido. Além disso, como as métricas da rede variam no tempo, é importante que este problema seja tratado de forma dinâmica.

O objetivo deste trabalho é desenvolver um mecanismo adaptativo para diminuir os efeitos do *jitter* na Internet. O mecanismo tem como principais características: i) coleta de estatísticas da rede; ii) uso de um sistema de dois *buffers*; iii) uso de duas taxas distintas para preenchimento destes *buffers*; iv) ter sido projetado para um sistema que opera em modo "*pull*", ou seja, os clientes fazem as requisições dos dados. Este mecanismo foi implementado e incorporado ao sistema de transmissão de vídeo sob demanda RIO [RIO 2010]. Nossos resultados apresentam uma melhora na interatividade com o usuário, sem que isto prejudique a qualidade do vídeo reproduzido.

Este trabalho está estruturado da seguinte forma: inicialmente apresentaremos na seção 2 os principais conceitos utilizados neste trabalho. Em particular, discursaremos sobre *streaming*, explicaremos como esta abordagem é afetada por *jitter*, e apresentaremos como solução para este problema a técnica de atraso de *playout*. A seção 3 será dedicada a apresentar trabalhos relacionados a esta técnica, e apresentaremos o sistema RIO na seção 4. O mecanismo proposto para redução de *jitter* será descrito na seção 5. Por fim, apresentaremos uma avaliação de desempenho na seção 6, seguida de uma conclusão na seção 7, incluindo possibilidades de trabalhos futuros.

2. Streaming e Jitter

Uma das abordagens mais utilizadas para transmissão de vídeo é o uso da técnica de *streaming*. Nesta abordagem, os dados a serem reproduzidos pelos clientes serão transmitidos pelos servidores ao longo de sua reprodução. Assim, o usuário precisa aguardar menos tempo antes de iniciar a reprodução do vídeo. Outro benefício da transmissão por *streaming* é tornar os sistemas mais escaláveis, já que o servidor evitará transmitir dados que não serão reproduzidos (caso, por exemplo, o usuário interrompa a transmissão antes do final do vídeo), e também poderá distribuir a transmissão destes dados ao longo do tempo, o que diminui a carga instantânea no servidor [Kurose and Ross 2009].

Uma das características mais marcantes dos canais de comunicação da Internet é a sua variabilidade no tempo. Podemos perceber isto observando o retardo de transmissão dos pacotes, que definiremos como o tempo decorrido entre a emissão de um pacote e a sua recepção. Mais especificamente, estamos interessados no *jitter* — a variação do retardo de transmissão. Ele é causado pelo fato de que cada pacote transmitido irá atravessar a rede em uma situação diferente. Para alguns pacotes, as filas nos roteadores estarão mais vazias, levando a um retardo inferior. Outros encontrarão filas mais cheias nos roteadores e sofrerão um retardo superior. Com isto, os pacotes não serão recebidos com a mesma regularidade com que eles foram enviados — como ilustrado na Figura 1.

Para aplicações tradicionais, sem restrições estritas de tempo, o *jitter* é pouco relevante para o sucesso da aplicação. No entanto, em aplicações de tempo real (como VoIP ou *streaming* de vídeo), o *jitter* pode levar a uma degradação significativa da qualidade da mídia reproduzida. Isto se deve ao fato de que este tipo de aplicação necessita que a recepção dos dados seja realizada de forma constante, sem grandes variações. Caso contrário, a reprodução da mídia é interrompida abruptamente por falta de dados. Neste

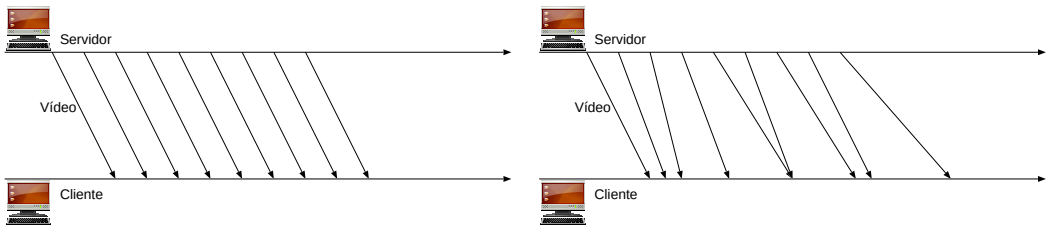
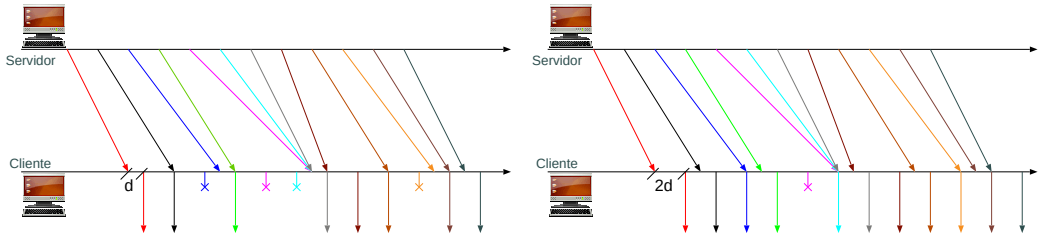


Figura 1. Transmissão de pacotes em uma rede ideal (sem a presença de *jitter*) e em uma rede com presença de *jitter*.

aspecto, a presença de um valor de *jitter* alto impede que estas aplicações possam oferecer um serviço de qualidade.

O problema de *jitter* é geralmente resolvido inserindo na reprodução um atraso de *playout* [Kurose and Ross 2009]. Nesta técnica, em vez de o cliente reproduzir os pacotes de dados assim que recebê-los, ele irá armazená-los em um *buffer*, chamado *buffer de playout*, e escalonar a reprodução destes pacotes com algum atraso. Todos os pacotes recebidos serão armazenados neste buffer até serem reproduzidos. Por outro lado, se o cliente ainda não tiver recebido um pacote no instante de sua reprodução, ele será considerado perdido e não será reproduzido, o que acarreta em perda de qualidade na reprodução da mídia. O objetivo desta técnica é permitir que uma quantidade maior de pacotes chegue antes do instante em que devem ser reproduzidos. Em compensação, devido ao atraso inicial inserido, o cliente irá aguardar mais tempo para iniciar a reprodução.

A Figura 2 ilustra este mecanismo. No primeiro exemplo (Figura 2(a)), a reprodução dos pacotes foi iniciada com um atraso igual a d. Com isto, quatro pacotes chegaram após o instante em que deveriam ser reproduzidos, ou seja, ocorreram quatro perdas. Já no segundo exemplo (Figura 2(b)), foi inserido um atraso igual a $2d$ para iniciar a reprodução dos pacotes. Neste caso, apenas um pacote não pôde ser reproduzido. Note o *trade-off* que ocorre com a utilização do atraso de playout: quanto maior for o atraso escolhido, maior é a chance de cada pacote chegar antes do instante de sua reprodução, porém mais tempo o cliente deve aguardar para iniciar a reprodução da mídia. Isto é particularmente relevante em aplicações que envolvem interatividade, como aplicações de videoconferência. Assim, podemos dizer que há um *trade-off* entre qualidade do vídeo e interatividade.



(a) Atraso de *playout* igual a d (b) Atraso de *playout* igual a $2d$

Figura 2. Técnica de atraso de *playout*.

Devemos notar também que o valor de atraso de *playout* a ser utilizado em uma determinada aplicação irá depender do *jitter* presente na rede durante a reprodução do

vídeo — quanto maior for o *jitter*, maior deve ser o atraso de *playout* para que a aplicação possa oferecer uma mesma qualidade de vídeo.

3. Trabalhos relacionados

A maior parte dos trabalhos que tratam de atraso de *playout* situam-se no contexto de aplicações de voz [Pinto and Christensen 1999]. No entanto, os conceitos utilizados neste domínio são válidos para mídia contínua em geral, como vídeo.

O primeiro trabalho, descrito em [Ramjee et al. 1994] e citado em [Kurose and Ross 2009], vem tratar do *trade-off* entre qualidade de vídeo e interatividade. Ele apresenta a idéia de atraso de *playout* adaptativo: motivada pela natureza desconhecida do retardo inserido pela rede na transmissão, esta abordagem tenta prever este retardo e responder à previsão adaptando o atraso de *playout* dinamicamente, ou seja, durante a reprodução da mídia. Assim, é possível evitar atrasos demasiadamente longos, que causariam perda de interatividade, e atrasos demasiadamente curtos, garantindo assim a qualidade da mídia. Também são apresentados alguns algoritmos para adaptação de atraso de *playout* na transmissão de voz. Estes algoritmos diferem apenas na forma de realizar a estimativa do retardo e, para realizar esta adaptação, eles utilizam o fato de o fluxo de voz ser composto de rajadas de voz alternadas com períodos de silêncio. Estes períodos de silêncio são ideais para adaptar o atraso de playout, já que as modificações realizadas nestes instantes são praticamente imperceptíveis pelo usuário.

Um outro algoritmo para adaptação de atraso de *playout* aplicado a transmissão de voz é apresentado em [Moon et al. 1998]. Este algoritmo coleta estatísticas dos pacotes previamente recebidos e, a partir delas, gera uma distribuição empírica do retardo baseado nas amostras de retardo mais recentes. Para realizar a estimativa, o algoritmo calcula um percentil superior q nesta distribuição. O valor do retardo correspondente ao percentil q, que chamaremos de d_q, será considerado pelo algoritmo o retardo máximo tolerável. O atraso de *playout* p_d é obtido, então, subtraindo-se de d_q o menor retardo observado, que é uma aproximação para o retardo de propagação do pacote, assumindo que não haja outros pacotes na rede. O parâmetro q age como um valor de tolerância a perdas, pois ele define a parcela de pacotes que, segundo a previsão do algoritmo, irá chegar ao cliente após o momento de sua reprodução. Assim, o parâmetro q atua como parâmetro de regulagem no *trade-off* entre qualidade e interatividade.

Para exemplificar, consideremos a distribuição empírica do retardo de transmissão apresentada na Figura 3. O eixo horizontal representa os retardos observados, e o eixo vertical representa a probabilidade de se observar um atraso igual ou superior a estes valores de retardo. Suponhamos um valor de q igual a 0.05 — ou seja, estamos interessados em inserir um atraso de *playout* para tentar acomodar os 95% de retardos inferiores. Pelo gráfico, percebemos que este valor de q corresponde a um retardo d_q igual a 1.736 s. Isto significa que, de todas as amostras de retardo coletadas, 5% representam retardos de transmissão iguais ou superiores a 1.736 s. Deste valor, subtraímos o menor retardo observado nestas amostras — também pelo gráfico, notamos que este valor é igual a 1.709 s. O resultado desta operação, 27 ms, será o atraso de *playout* inserido na reprodução.

Uma característica comum aos algoritmos descritos acima é a suposição de que a comunicação entre o servidor e o cliente funciona no modo "*push*" — o servidor é responsável por enviar pacotes para o cliente conforme estes pacotes vão sendo gerados,

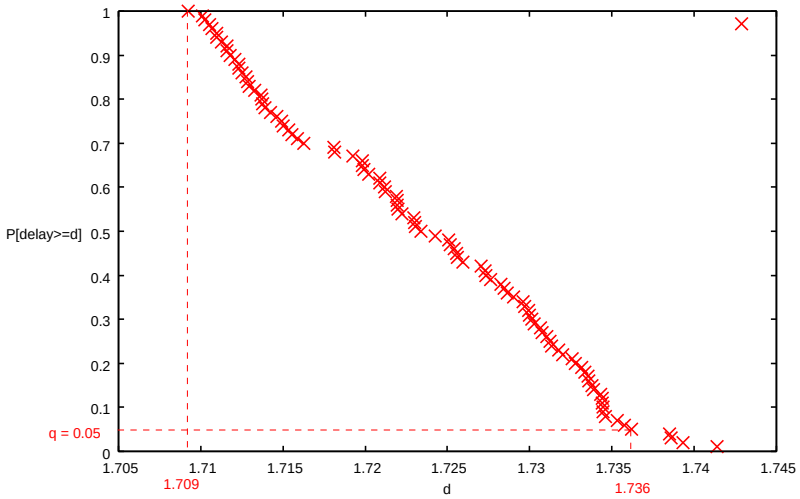


Figura 3. Exemplo de distribuição empírica do retardo de transmissão.

seja esta geração uma captura de dados ao vivo ou uma recuperação de dados em disco. Isto difere fundamentalmente do modo "*pull*" de funcionamento utilizado pelo sistema RIO, como veremos na seção seguinte. Não foram encontradas na literatura propostas que contemplassem este modo de funcionamento, o que nos exigiu uma adaptação das propostas estudadas para modo "*push*".

4. Sistema RIO

O sistema RIO [Muntz et al. 1997] [Netto et al. 2005] (*Random I/O Multimedia Storage System*) é um sistema distribuído de armazenamento de conteúdo multimídia, atualmente em desenvolvimento pelo grupo de pesquisas LAND [LAND 2010]. Ele possui uma arquitetura cliente-servidor distribuída, composta por três entidades: os clientes, os servidores de gerenciamento (ou apenas servidores) e os servidores de armazenamento (ou *storages*).

No sistema RIO, os vídeos são divididos em blocos de tamanho fixo e armazenados nos *storages*. Clientes se conectam ao servidor e passam a requisitar blocos. O servidor, ao receber uma requisição, encaminha-a a um *storage* que possua o bloco requisitado. Este *storage* irá, então, fragmentar o bloco em pacotes de tamanho fixo e enviá-los ao cliente, como ilustrado na Figura 4(a). É importante ressaltar que o sistema RIO funciona no modo "*pull*", visto que os clientes são os responsáveis por requisitar os dados conforme necessário. Caso o servidor não seja capaz de servir estas requisições à mesma taxa com que elas são enviadas, as requisições serão armazenadas em uma fila de requisições pendentes no servidor, até que sejam servidas. Outra característica do sistema RIO é um mecanismo de controle de fluxo, utilizado para suavizar o envio de pacotes dos *storages* para os clientes, evitando uma sobrecarga instantânea na rede e o consequente descarte de pacotes. Ele funciona limitando a taxa de envio destes pacotes a uma taxa máxima λ, superior à taxa de reprodução do vídeo para evitar que ocorram interrupções no cliente por falta de dados. Tais aspectos influenciam diretamente o projeto do mecanismo que apresentaremos mais adiante.

Antes deste trabalho, o cliente de visualização do sistema RIO possuía um mecanismo de buffer estático, constante durante a reprodução de um vídeo. Ao iniciar a

reprodução e após cada salto realizado pelo cliente, são requisitados blocos suficientes para preencher completamente este buffer, em uma operação conhecida como *prefetch*. Depois que todos estes blocos forem recebidos, a reprodução é iniciada. Deste ponto em diante, o cliente irá requisitar um novo bloco após algum bloco ser reproduzido e a posição que ele ocupava no buffer estiver vazia — ou seja, os blocos serão solicitados à mesma taxa com que eles serão reproduzidos.

5. Mecanismo adaptativo

O mecanismo proposto é formado por três componentes: i) um algoritmo de adaptação de atraso de *playout* baseado em coleta de estatísticas; ii) um sistema de *buffers* para implantação deste atraso de *playout*; e iii) um modo diferenciado de requisição para pre-enchimento deste *buffer*. Estes três componentes diferem do mecanismo de atraso de playout estático utilizado pelo sistema RIO, e iremos detalhá-los a seguir.

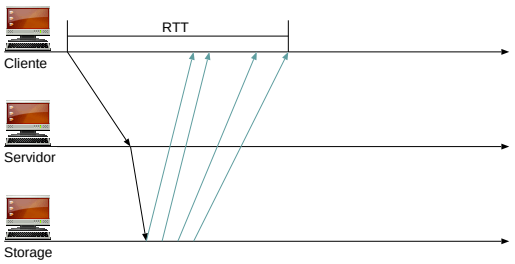
O primeiro elemento do mecanismo é o algoritmo de adaptação de atraso de *playout*. Ele realiza coleta de estatísticas, de forma equivalente ao algoritmo de [Moon et al. 1998]. No entanto, a arquitetura do sistema RIO exige que utilizemos uma métrica diferente, para capturar todos os retardos de transmissão. Utilizaremos, então, a métrica RTT, definida neste trabalho como o tempo decorrido no cliente entre o instante de envio da requisição de um bloco e a recepção completa deste bloco — como ilustrado na Figura 4(a).

O funcionamento do algoritmo ocorre da seguinte forma: para cada bloco que o cliente receber, será calculado um valor de RTT. O algoritmo irá utilizar a distribuição empírica dos n RTTs mais recentes e calcular um percentil q superior nesta distribuição. Este percentil corresponde a um valor de RTT, que representa um RTT máximo tolerável. Este valor é, então, descontado do menor RTT observado e utilizado como atraso de *playout*. Assim, além do parâmetro q, que regula o *trade-off* entre qualidade de vídeo e interatividade, o algoritmo utiliza um parâmetro adicional n, que indica o grau de adaptação do algoritmo. Quanto menor for o valor de n, mais rapidamente o algoritmo irá se adaptar a mudanças na rede.

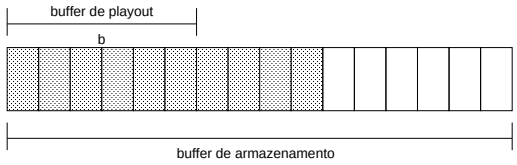
Para pôr este atraso de *playout* em prática, nosso mecanismo utiliza dois *buffers* no cliente. O primeiro deles, denominado *buffer de armazenamento*, é o *buffer* total utilizado pelo cliente para armazenar os blocos recebidos. Além disso, definimos como um subconjunto do *buffer de armazenamento* um segundo *buffer*, denominado *buffer de playout*, conforme ilustrado na Figura 4(b). O *buffer de playout* deve estar completamente preenchido para que a reprodução da mídia seja iniciada. Desta forma, o objetivo do *buffer de playout* é inserir na transmissão o atraso de *playout* calculado pelo mecanismo de adaptação. O tamanho b, em blocos, do buffer de *playout*, pode ser calculado pela equação 1:

$$b = \left\lceil \frac{p_d \, \lambda}{s} \right\rceil \tag{1}$$

onde p_d é a duração do atraso de *playout*, em segundos, λ é a taxa de transmissão do vídeo (definida pelo mecanismo de controle de fluxo e conhecida pelo cliente), em bps; e s é o tamanho do bloco, em bits.



(a) Cálculo do RTT para um bloco de vídeo. As setas pretas representam a requisição, e as setas azuis representam os pacotes que compõem o bloco requisitado.



(b) *Buffer de armazenamento* e *buffer de playout*.

Figura 4. Elementos do mecanismo adaptativo.

Ao contrário de transmissão de voz, cuja reprodução possui períodos intermitentes de silêncio, a transmissão de vídeo pré-armazenado é contínua. Logo, não há momentos durante a reprodução do vídeo em que seja possível adaptar o *buffer de playout* sem que isso impacte na qualidade do vídeo reproduzido. Entretanto, há três situações em que podemos e iremos fazer esta adaptação: *a*) quando o vídeo começa a ser reproduzido (no início da reprodução ou no retorno de pausa); *b*) quando o usuário realiza um salto para outro ponto do vídeo; e *c*) quando há completa ausência de blocos para reprodução.

Para o envio de requisições, utilizaremos um sistema de duas taxas. A primeira taxa, chamada taxa normal de requisições, guiará o processo normal de envio de requisições. Utilizaremos a taxa de transmissão do vídeo λ imposta pelo mecanismo de controle de fluxo. Esta taxa é superior à taxa de reprodução do vídeo, o que faz com que o *buffer de armazenamento* seja preenchido gradualmente. Após o *buffer de armazenamento* ser totalmente preenchido, as requisições serão feitas somente quando algum bloco for reproduzido e uma posição do *buffer* ficar vazia.

A segunda taxa, chamada taxa de *prefetch* e representada por λ_p, será utilizada para realizar a operação de *prefetch*. Esta operação é realizada para preencher apenas o *buffer de playout*, o que ocorre somente nas três situações em que o tamanho do *buffer de playout* é adaptado. O envio destas requisições será controlado, utilizando a taxa de *prefetch* como taxa de requisição de blocos. A taxa de *prefetch* deve ser superior à taxa de transmissão do vídeo, para garantir que utilizaremos a taxa máxima de transmissão de blocos disponível no servidor. Neste período, ocorrerá um acúmulo de requisições pendentes no servidor e, como estas pendências influenciam o RTT associado a estes blocos, devemos descartar estes valores de RTT (exceto a primeira requisição).

Após requisitar todos os blocos do modo *prefetch*, o cliente irá aguardar um período de tempo Δ, antes de retornar ao modo normal de envio de requisições. O objetivo é garantir que o servidor possa servir todas as requisições do modo *prefetch* antes de receber uma nova requisição, como ilustrado na Figura 5. Note que o cliente não aguarda o *buffer de playout* ser preenchido para iniciar o modo normal de envio de requisições. Isto é necessário pois, para retardos de transmissão grandes, o *buffer de playout* será completamente consumido antes que o primeiro bloco do modo normal seja recebido pelo cliente. Esta antecipação impede que este problema ocorra. O antigo cliente do sistema

RIO estava sujeito a este problema, que pode degradar a qualidade de vídeo, como iremos mostrar na seção 6.

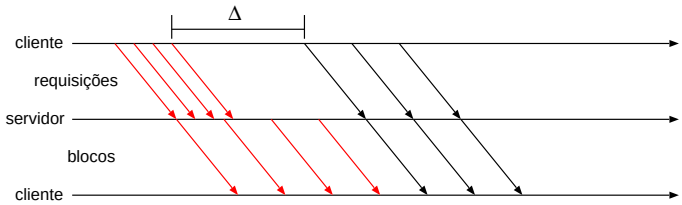


Figura 5. Transição entre modo *prefetch* e modo normal de envio de requisições.

O valor do tempo Δ é calculado da seguinte forma. Seja λ a taxa de transmissão do vídeo, λ_p a taxa de *prefetch* e b o tamanho do *buffer de playout*. No modo *prefetch*, o tempo entre requisições é igual a $1/\lambda_p$. Assim, supondo que o primeiro destes blocos seja requisitado no tempo t, o último será requisitado no tempo $t + (b-1)/\lambda_p$. No entanto, o servidor está limitado a servir requisições à taxa λ, o que significa que ele terminará de servi-las após um tempo b/λ. Assim, queremos iniciar o modo normal b/λ após o início do modo *prefetch*. Com estas informações, podemos calcular o valor de Δ:

$$\Delta = \frac{1}{\lambda_p} + b \cdot \left(\frac{1}{\lambda} - \frac{1}{\lambda_p} \right)$$

Devemos ressaltar que este mecanismo foi projetado de forma a manter a fila de requisições pendentes no servidor com a menor quantidade de requisições possível. Com isto, minimizamos a quantidade de requisições servidas desnecessariamente (como as realizadas pouco antes de um salto, por exemplo) e evitamos que o tamanho destas filas influencie as medidas de retardo. Além disso, assumimos que o servidor é sempre capaz de transmitir pacotes a uma taxa mínima igual à taxa de reprodução do vídeo.

6. Avaliação

O mecanismo proposto foi implementado e incorporado ao cliente de visualização do sistema RIO. Nesta seção, faremos uma avaliação do novo cliente com o objetivo de ilustrar o desempenho deste mecanismo. Optamos por realizar os experimentos em uma rede emulada em vez de uma rede real. Isto nos permite ter mais controle sobre o retardo introduzido pela rede e melhor avaliar o mecanismo proposto. Todos os experimentos foram realizados em uma *workstation*, com CPU Intel Pentium 4, com 2.8 GHz de processamento e 1.75 GB de memória RAM, executando o sistema operacional Ubuntu 8.04 LTS. A rede foi emulada utilizando o módulo netem [netem 2010], controlado pela ferramenta Tc [iproute 2010], que opera o módulo de controle de tráfego presente no *kernel* do sistema operacional. O cliente e o servidor foram executados na mesma máquina e a rede foi emulada na interface *loopback*. O retardo inserido nos pacotes corresponde apenas ao retardo causado pela transmissão e não inclui retardos ocasionados por *buffers* na interface de rede ou retardos de processamento. A ordem de envio dos pacotes é preservada.

Os experimentos foram realizados considerando o seguinte cenário: um cliente se conecta ao servidor RIO e começa a reproduzir o vídeo. A cada 30 segundos de vídeo, ele realiza um salto para um ponto aleatório do vídeo. Além disso, a cada três saltos,

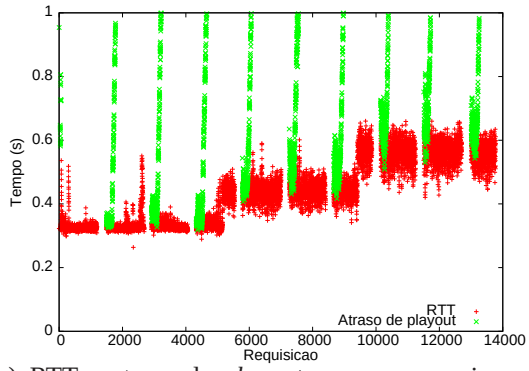
após 15 segundos de vídeo, mantivemos o retardo médio $\bar{d}$ inserido nos pacotes, mas modificamos a variação deste retardo. Esta variação não é uma medida estatística, mas um parâmetro do emulador. Estes valores (15 e 30 segundos) foram escolhidos para permitir que a transmissão de dados atinja a estabilidade antes do salto seguinte ou da mudança de variação de retardo. Foram utilizadas três variações de retardo, que chamaremos v_1, v_2 e v_3. Após um total de nove saltos, em três condições de rede distintas, a reprodução é interrompida.

Foram realizados experimentos com retardo médio $\bar{d} = 300$ ms e variações $v_1 = 0$ ms, $v_2 = 140$ ms e $v_3 = 280$ ms, e com retardo médio $\bar{d} = 1500$ ms e variações $v_1 = 0$ ms, $v_2 = 740$ ms e $v_3 = 1480$ ms. Este retardo é uniformemente distribuído no intervalo $[\bar{d}-v, \bar{d}+v]$. O retardo de 300 ms e as variações associadas foram escolhidos por serem representativos dos atrasos observados na Internet, enquanto o atraso de 1500 ms e as variações associadas foram escolhidos para permitir uma análise do mecanismo em condições extremas. Outros parâmetros que variaram entre experimentos foram a taxa de transmissão do vídeo no servidor λ, com os valores de 1741 kbps e 1400 kbps, e o tamanho de bloco s, com valores de 4096 bytes e 32768 bytes. A taxa de codificação do vídeo utilizado é de 1228 kbps. O tamanho do pacote utilizado pelo sistema RIO é fixo e igual a 1500 bytes . Assim, blocos de 4096 bytes são fragmentados em 3 pacotes e blocos de 32768 bytes são fragmentados em 22 pacotes. O valor de percentil q utilizado foi de 2%. As métricas utilizadas para avaliação do mecanismo foram as seguintes:

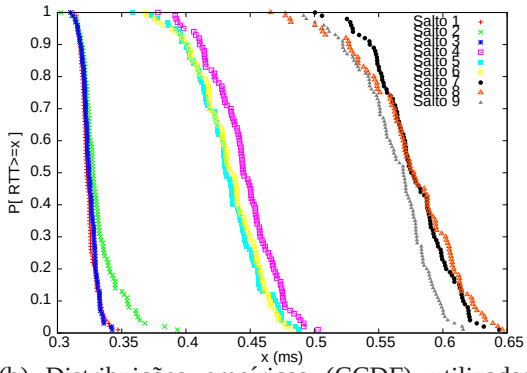
RTT: Tempo decorrido entre o envio da requisição de um bloco e a sua recepção;
Atraso de *playout*: Tempo decorrido entre o envio da requisição e a sua reprodução;
***Buffer de playout*:** Tamanho, em blocos, do *buffer de playout* previsto pelo mecanismo.

Apresentamos na Figura 6 os resultados para a seguinte combinação de parâmetros: λ = 1741 kbps, s = 4096 bytes e $\bar{d}$ = 300 ms. O primeiro gráfico, na Figura 6(a), apresenta os valores de RTT e de atraso de *playout* para cada requisição de bloco. Percebemos que os RTTs atravessam três etapas. A primeira etapa, até a requisição 5150, corresponde à primeira condição da rede emulada, com variação de retardo $v_1 = 0$ ms. A segunda etapa, entre as requisições 5151 e 9414, corresponde à segunda condição, com variação $v_2 = 140$ ms, e a terceira etapa, a partir da requisição 9415, corresponde à última condição, com variação $v_3 = 280$ ms. Apresentamos, na tabela 6(d), a média e o desvio padrão das amostras de RTT coletadas em cada uma destas etapas. Note que, quanto maior é a variação do retardo inserido nos pacotes, maior é o RTT médio sofrido pelos blocos. Isto vem do fato de que um bloco só chega completo no cliente quando todos os pacotes que o compõem chegarem. Logo, quanto maior for o retardo máximo destes pacotes, maior é o RTT do bloco. Além disso, sabemos que, a cada estágio, o retardo dos pacotes irá variar mais em torno de um mesmo valor médio. Portanto, a cada estágio, o retardo máximo dentre estes pacotes, em média, atinge valores maiores. Com isto, o RTT dos blocos também irá, em média, atingir valores maiores.

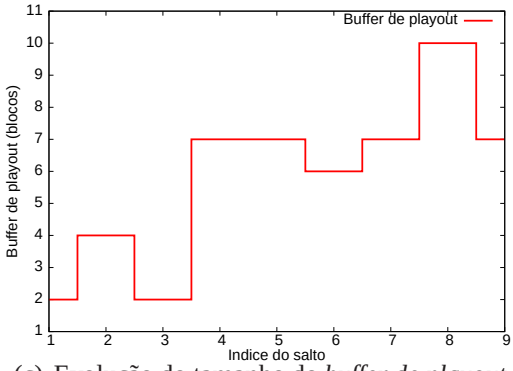
Este mesmo comportamento pode ser observado se analisarmos as distribuições empíricas de RTT utilizadas em cada salto pelo algoritmo de adaptação. Estas distribuições são apresentadas na Figura 6(b). Para os três primeiros saltos, que ocorreram durante a primeira condição da rede emulada, foram obtidas amostras de retardo com menor variação, e com média inferior aos outros saltos. Para os três saltos seguintes, na condição de variação intermediária de retardo, obtivemos amostras com variação



(a) RTT e atraso de *playout*, com o mecanismo proposto.



(b) Distribuições empíricas (CCDF) utilizadas pelo mecanismo em cada salto.



(c) Evolução do tamanho do *buffer de playout*.

Estágio	μ (s)	σ (s)
1	0.3326	0.0224
2	0.4417	0.0289
3	0.5600	0.0352

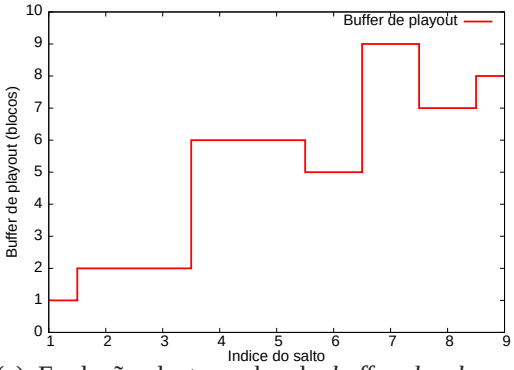
(d) Média e desvio padrão das amostras de RTT coletadas em cada etapa.

Figura 6. Resultados para λ = 1741 kbps, s = 4096 bytes e $\bar{d}$ = 300 ms.

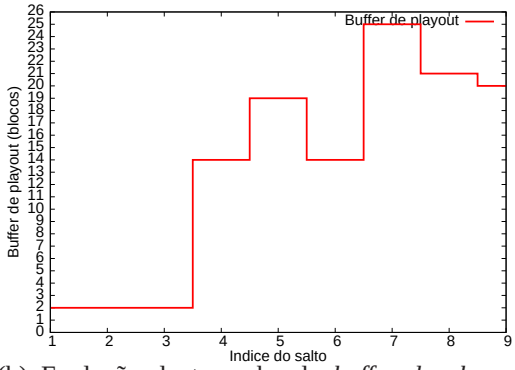
e média pouco superiores. Da mesma forma, as amostras obtidas nos três últimos saltos possuem variação e média ainda maiores. Isto reafirma nossa conclusão anterior, de que maiores variações no retardo inserido nos pacotes implicam em maiores RTTs sofridos pelos blocos. Devemos ressaltar que o valor absoluto do retardo observado não interfere nestas conclusões, mas somente a faixa de valores na qual este retardo varia.

Observamos também na Figura 6(a) que o atraso de *playout* dos blocos é crescente com o tempo, mas periodicamente retorna ao patamar do RTT. O aumento gradual no atraso de *playout* é consequência de a taxa de transmissão ser substancialmente superior à taxa de codificação do vídeo. Este aumento é interrompido em cada salto por causa da interrupção da reprodução. É importante ressaltar que isto não é um problema, já que o *buffer de armazenamento* do cliente é capaz de armazenar estes blocos. Este comportamento não foi observado nos experimentos com taxa de transmissão inferior (1400 kbps).

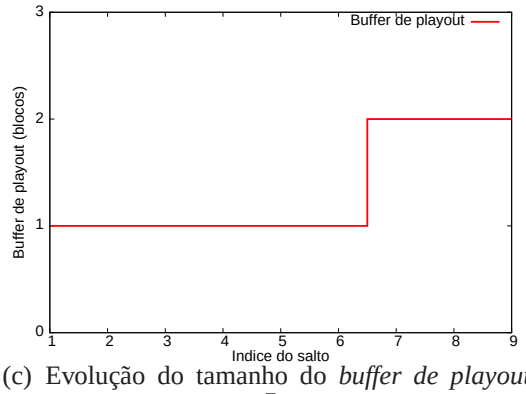
A Figura 6(c) mostra a evolução do tamanho do *buffer de playout* a cada salto realizado no cliente. Percebemos que o mecanismo se adapta como esperado a essas mudanças — nos primeiros saltos, o tamanho de *buffer* previsto pelo mecanismo é muito baixo, compatível com a baixa variação do retardo. Nos saltos seguintes, o mecanismo detecta que o retardo varia mais e ajusta o *buffer de playout* para valores maiores. Da mesma forma, na situação de alta variação de retardo na rede, o mecanismo ajusta no-



(a) Evolução do tamanho do *buffer de playout*, para $\lambda = 1400$ kbps, $\bar{d} = 300$ ms, $s = 4096$ bytes.



(b) Evolução do tamanho do *buffer de playout*, para $\lambda = 1741$ kbps, $\bar{d} = 1500$ ms, $s = 4096$ bytes.



(c) Evolução do tamanho do *buffer de playout*, para $\lambda = 1741$ kbps e $\bar{d} = 300$ ms, $s = 32768$ bytes.

Figura 7. Resultados para outros experimentos.

vamente o *buffer* para valores ainda maiores. Este comportamento está de acordo com o esperado para o mecanismo.

Experimentos com taxas de transmissão inferior apresentaram comportamento quantitativamente semelhante, conforme ilustrado na Figura 7(a). Já nos experimentos com retardo médio superior (1500 ms), o mecanismo previu tamanhos maiores de *buffer*, como apresentado na Figura 7(b), mas foi igualmente observada uma diferença entre as três etapas dos experimentos. Ressaltamos que, nos experimentos com atraso médio superior, utilizamos variações de retardo também superiores (0 ms, 740 ms e 1480 ms).

Nos experimentos com blocos maiores (32768 bytes), observamos um comportamento claramente distinto, como apresentado na Figura 7(c). Notamos que o *buffer* previsto pelo mecanismo é muito menor e varia muito menos do que nos experimentos com blocos de 4096 bytes, se mantendo praticamente constante. Isto ocorre por dois motivos:

Blocos maiores implicam em menor variação de RTT

Sabemos que o tamanho de pacote utilizado pelo sistema RIO é fixo. Por causa disto, blocos de tamanho maior serão fragmentados em uma quantidade maior de pacotes para serem transmitidos, gerando mais amostras de retardo de transmissão. Podemos, então, modelar o retardo de transmissão de cada pacote utili-

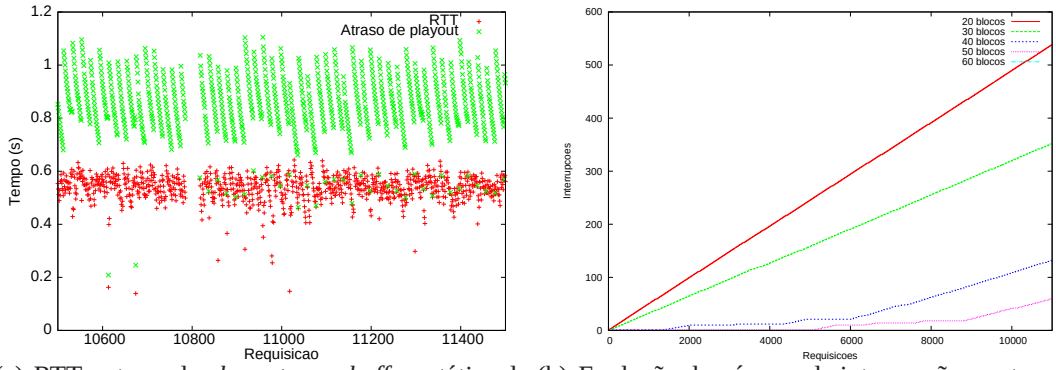
zando uma variável aleatória X, com distribuição $f_X(x)$, e o RTT de cada bloco utilizando uma variável aleatória Y definida como o máximo de um conjunto de amostras de X contendo um total de n amostras [Trivedi 2001]. Sabemos que, quanto maior for a quantidade de amostras n, maior é o valor esperado de Y, e intuitivamente podemos esperar que a variância de Y irá diminuir. Assim, se utilizarmos blocos maiores, a fragmentação de cada bloco em uma quantidade maior de pacotes implica em menores variações do RTT.

Blocos maiores implicam em *buffers* menores

Como descrito na seção 5, podemos facilmente calcular o tamanho do *buffer de playout* b necessário para um dado atraso de *playout* p_d, através da equação 1: $b = \lceil p_d \, \lambda / s \rceil$. Recapitulando, λ é a taxa de transmissão do vídeo e s é o tamanho do bloco utilizado na transmissão. Esta equação nos diz que o tamanho do *buffer de playout* que deve ser utilizado é inversamente proporcional ao tamanho do bloco. Assim, blocos maiores resultam em *buffers* menores e em variações menores de *buffer*.

Para efeito de comparação, realizamos alguns experimentos com o mecanismo estático do sistema RIO. Para o primeiro destes experimentos, utilizamos *buffer* com tamanho igual a 20 blocos. A Figura 8(a) apresenta os valores de RTT e de atraso de *playout* para cada requisição de bloco. Podemos observar um atraso de *playout* periodicamente decrescente. Para explicar este fenômeno, considere um período que começa com um valor alto de *playout* (por exemplo, 1.1 s, aproximadamente) e termina com um valor inferior ao inicial (por exemplo, aproximadamente 0.8 s). Cada um dos atrasos do período representa o atraso de um bloco. No início do período o *buffer* está cheio, portanto o primeiro atraso é o atraso do primeiro bloco armazenado no *buffer*, o segundo atraso é o atraso do segundo bloco armazenado e assim sucessivamente. Como os primeiros blocos do *buffer* são os primeiros a serem requisitados e, portanto, chegam primeiro no cliente, eles ficarão mais tempo armazenados e terão um atraso de *playout* maior que os últimos blocos do *buffer*. Isto explica o atraso de *playout* decrescente. Além disso, como o *buffer* está cheio, o cliente somente irá requisitar novos blocos quando a reprodução for iniciada. No entanto, antes que o cliente receba estes novos blocos, todos os blocos acumulados terão sido reproduzidos, e o *buffer* irá esvaziar. Com isto, a transmissão será interrompida e o cliente irá novamente aguardar o *buffer* ser completamente preenchido, reiniciando o período. O resultado deste fenômeno é uma transmissão intermitente, mesmo com um tamanho de *buffer* estático superior ao que foi previsto pelo mecanismo adaptativo como sendo suficiente. Estas interrupções não ocorrem com o mecanismo adaptativo pois ele realiza a requisição de novos blocos com antecipação.

Nos experimentos seguintes, para avaliar a relação entre estas interrupções e o tamanho do *buffer* utilizado, variamos o tamanho de *buffer* utilizado no mecanismo estático. Utilizamos *buffers* de 20, 30, 40, 50 e 60 blocos. Para estes experimentos, a Figura 8(b) apresenta, no eixo horizontal, a quantidade de blocos requisitados e, no eixo vertical, a quantidade de vezes que o *buffer* estático foi completamente esvaziado — ou seja, a quantidade de interrupções na reprodução do vídeo causadas por falta de dados. Para o *buffer* estático de 20 blocos, percebemos pelo gráfico que, para cada 20 requisições de blocos, ocorre uma interrupção. Isto significa que o *buffer* é consumido completamente e, logo em seguida, a transmissão é interrompida por falta de blocos. Este comportamento também pôde ser observado para o *buffer* estático de 30 blocos. Já para os experimentos



(a) RTT e atraso de *playout*, com *buffer* estático de 20 blocos.

(b) Evolução do número de interrupções na transmissão, para diversos tamanhos de *buffer* estático.

Figura 8. Resultados para experimentos com o mecanismo de *buffer* estático, para λ = 1741 kbps, s = 4096 bytes e $\bar{d}$ = 300 ms.

com *buffer* de 40 e 50 blocos, observamos uma quantidade muito menor de interrupções. Além disto, observamos que a frequência das interrupções aumenta conforme o experimento avança e as condições da rede emulada são modificadas. Isto indica que, quanto maior a variação do retardo inserido nos pacotes, maior é a probabilidade de o *buffer* esvaziar durante a transmissão. Para o último experimento, com *buffer* de 60 blocos, não foi observada nenhuma interrupção. Note que, para os mesmos parâmetros, o mecanismo adaptativo sempre previu a utilização de um *buffer* menor ou igual a 10 blocos — um ganho de 83% em interatividade com relação ao mecanismo estático. Ressaltamos que, nos experimentos com o mecanismo adaptativo, não foi observada a ocorrência de nenhuma interrupção por esvaziamento de *buffer*, ou seja, para que obtivéssemos a mesma qualidade do vídeo reproduzido, pudemos utilizar um atraso de *playout* seis vezes menor com o mecanismo adaptativo do que com o mecanismo estático.

7. Conclusão

A principal contribuição deste trabalho é o projeto e implementação de um mecanismo adaptativo, baseado em atraso de *playout*, para contornar a variabilidade de retardo de transmissão observada na Internet. Este mecanismo foi projetado para o sistema RIO, que possui arquitetura "*pull*", e embora esta característica não tenha impedido a reutilização das idéias e conceitos que guiam os algoritmos existentes, ela impediu que fosse realizada a simples adaptação destes algoritmos, exigindo um projeto original. A avaliação da implementação do mecanismo proposto evidenciou a importância de se realizar a transmissão de dados em blocos pequenos, pois isto permite que o mecanismo se adapte às condições da rede. Possíveis trabalhos futuros incluem realizar testes com escassez de recursos (taxa de transmissão igual à taxa de codificação do vídeo) e com a utilização de dados reais, como *traces* de retardo de transmissão e *logs* de comportamento de usuários reais [Tomimura et al. 2006] [Alves et al. 2007].

Referências

Alves, B. C. B., Leão, R. M. M., and de Souza e Silva, E. (2007). Caracterizando variáveis de interatividade dos alunos do curso de computação do CEDERJ baseado no servidor multimídia RIO. In *XXVII Congresso da Sociedade Brasileira de Computação, VI*

Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance).

iproute (2010). iproute2 — The Linux Foundation. http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2. (Acesso em 4 de março de 2010).

Kurose, J. F. and Ross, K. W. (2009). *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, USA, 5th edition.

LAND (2010). Land. http://www.land.ufrj.br/. (Acesso em 22 de fevereiro de 2010).

Moon, S. B., Kurose, J. F., and Towsley, D. F. (1998). Packet audio playout delay adjustment: Performance bounds and algorithms. *Multimedia Syst.*, 6(1):17–28.

Muntz, R. R., Santos, J. R., and Berson, S. (1997). Rio: A real-time multimedia object server. *SIGMETRICS Performance Evaluation Review*, 25(2):29–35.

netem (2010). netem — The Linux Foundation. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem. (Acesso em 4 de março de 2010).

Netto, B. C. M., Azevedo, J. A., de Souza e Silva, E. A., and Leão, R. M. M. (2005). Servidor Multimídia RIO em Ensino a Distância. In *6th International Free Software Forum*.

Pinto, J. and Christensen, K. J. (1999). An algorithm for playout of packet voice based on adaptive adjustment of talkspurt silence periods. In *LCN*, pages 224–231.

Ramjee, R., Kurose, J. F., Towsley, D. F., and Schulzrinne, H. (1994). Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *INFOCOM*.

RIO (2010). Home - RIO Multimedia System. http://www.land.ufrj.br/rio. (Acesso em 22 de fevereiro de 2010).

Tomimura, D., Leão, R. M. M., de Souza e Silva, E. A., and Filho, F. S. (2006). Caracterização e Modelagem do Comportamento de Usuários Acessando um Vídeo de Ensino a Distância. In *XXVI Congresso da Sociedade Brasileira de Computação, V Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)*.

Trivedi, K. (2001). *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Wiley, Chichester.