Stream2P: Uma Arquitetura Distribuída para Transmissão de Mídia Contínua

Ítalo Cunha, Cristiano Costa e Jussara Almeida[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Brasil

{cunha, krusty, jussara}@dcc.ufmg.br

Abstract. *Among many challenges in streaming media distribution, the high bandwidth requirement is a critical point for the scalability of transmission protocols. In this paper we propose Stream2P, a new streaming media transmission protocol based on hierarchical stream merging which uses spare client resources to minimize server bandwidth consumption. We present a performance analysis of many of the current state-of-the-art streaming protocols for realistic scenarios. This analysis shows that in networks with multicast Stream2P achieves savings of up to 42% in server bandwidth over Bandwidth Skimming if 1% of the clients have spare resources. In overlay networks without multicast, the protocol is efficient when clients transmit two streams.*

Resumo. *Dentre os desafios existentes na distribuição de mídia contínua, o requisito de banda é um ponto crítico para a criação de protocolos de transmissão escaláveis. Neste trabalho propomos Stream2P, um novo protocolo de transmissão de mídia contínua, baseado em fusão hierárquica de fluxos que utiliza recursos ociosos dos clientes para reduzir o consumo de banda do servidor. Apresentamos uma análise do desempenho de Stream2P, comparando-o com protocolos existentes na literatura. Em redes com multicast, Stream2P mostrou redução de até 42% na banda média de servidor utilizada pelo Bandwidth Skimming, se 1% dos clientes transmite um fluxo. Em redes sobrepostas sem multicast o protocolo é eficiente quando os clientes podem transmitir dois fluxos simultaneamente.*

1. Introdução

O aumento da banda disponível na Internet, principalmente pelo usuário final, possibilitou nos últimos anos o surgimento de vários serviços de mídia contínua (*streaming media*) [Chesire et al. 2001], dentre os quais podemos citar ensino a distância[1] e programas de rádio[2] e televisão[3].

Em se tratando de um assunto de grande interesse, intensos foram os esforços gastos no desenvolvimento de técnicas e métodos que tornassem possível a distribuição de mídia através da Internet. Porém, apesar das técnicas avançadas, a distribuição de mídia contínua ainda é limitada devido às características de tempo real associadas à exibição

[1]http://eteach.cs.wisc.edu/index.html
[2]http://musica.uol.com.br/radiouol/
[3]http://tvuol.uol.com.br/

e à grande quantidade de recursos necessários para o funcionamento dessas aplicações. Dentre os recursos necessários estão o espaço local de armazenamento temporário (*buffers*), capacidade de processamento para decodificação e exibição do conteúdo e a banda de rede para transmitir os dados.

Em particular, banda de rede é um recurso essencial para uma transmissão multimídia de qualidade. Servidores de mídia contínua tradicionais utilizam conexões ponto-a-ponto (*unicast*) para transmitir os dados. Porém, esse método apresenta problemas de escalabilidade, pois como os objetos de mídia contínua exigem uma grande quantidade de banda, mesmo uma pequena quantidade de clientes é suficiente para exaurir a banda disponível no servidor.

Tentando sanar esse problema, foram projetados protocolos [Dan et al. 1994, Eager and Vernon 1998, Aggarwal et al. 1996] utilizando compartilhamento de fluxos (*multicast*) para diminuir a carga imposta ao servidor. Esses protocolos apresentam ganhos expressivos em relação à maneira tradicional de transmissão multimídia, mas não realizam vídeo-sob-demanda verdadeiro. Clientes que chegam no sistema são obrigados a esperar uma janela de tempo até que o servidor inicie o envio de dados para exibição.

Protocolos subseqüentes focaram no serviço imediato ao usuário, utilizando compartilhamento de fluxos para prover economia de banda. Dentre os mais relevantes podemos citar o *Patching* [Hua et al. 1998] e o *Bandwidth Skimming* [Eager et al. 1999, Eager et al. 2000, Eager et al. 2001].

Procurando economizar ainda mais recursos de banda do servidor e atingir maior escalabilidade, protocolos descentralizados foram propostos [Guo et al. 2003, Sheu et al. 1997]. Nesses sistemas cada cliente pode funcionar como um servidor de dados transmitindo conteúdo para outros clientes, reduzindo uso de banda no servidor.

Utilizando a idéia de descentralizar a transmissão de conteúdo, desenvolvemos o Stream2P: uma arquitetura distribuída para transmissão de mídia contínua baseada no protocolo *Bandwidth Skimming*. Utilizando essa abordagem, conseguimos grande economia dos recursos de banda do servidor em redes com fluxos compartilhados. Caso a rede não tenha fluxos compartilhados, os clientes podem formar uma rede sobreposta (*overlay*), possibilitando o funcionamento do protocolo proposto nesse cenário.

Dentre as contribuições deste artigo podemos citar:

- Criação de um novo protocolo para distribuição de mídia continua.
- Avaliação de protocolos centralizados e descentralizados de transmissão de mídia contínua no estado-da-arte.
- Mostramos que quando a rede provê fluxos compartilhados, o Stream2P apresenta redução de até 42% na banda média de servidor utilizada pelo *Bandwidth Skimming* quando 1% dos clientes tem banda disponível para cooperar com o sistema de distribuição. Em redes sobrepostas sem fluxos compartilhados, o protocolo é eficiente quando os clientes transmitem dois fluxos.

O restante deste artigo é organizado da seguinte forma. Na seção 2 é apresentada uma revisão bibliográfica dos protocolos (centralizados e descentralizados) de transmissão encontrados na literatura. Na seção 3 introduzimos o protocolo Stream2P. Na seção 4 apresentamos o ambiente em que os experimentos foram realizados. As seções 5 e 6

apresentam os resultados da avaliação dos protocolos em redes com compartilhamento de fluxo e em redes sobrepostas, respectivamente. Finalizamos a nossa discussão e apresentamos direções futuras na seção 7.

2. Fundamentos

Nas subseções abaixo apresentamos protocolos relevantes na literatura e explicamos o seu funcionamento. Todos estes protocolos, incluindo o Stream2P, serão analisados nas seções 5 e 6. Os exemplos utilizados para explicar o funcionamento dos protocolos utilizam os dados dos clientes mostrados na tabela 1. Assumimos que esses clientes requisitam o mesmo objeto do servidor e assistem a mídia sem interrupções.

Clientes	C_1	C_2	C_3	C_4
Momento da requisição (segundos)	0	50	70	85

Tabela 1. Clientes e respectivos momentos de chegada no servidor

2.1. Batching

O *Batching* [Dan et al. 1994] foi um dos primeiros protocolos propostos que utilizaram a funcionalidade de compartilhamento de fluxos para transmitir mídia contínua. Seu funcionamento é bastante simples. O servidor possuí uma janela de tempo (janela de *batching*) na qual clientes conectam-se e esperam pelo início do serviço. Quando o tempo da janela de *batching* termina, um novo fluxo compartilhado é iniciado para servir os clientes conectados durante a duração da janela. A próxima requisição que chega ao servidor dispara a criação de uma nova janela e o processo se repete.

Por exemplo, suponha um servidor com uma janela de *batching* com duração de 30 segundos. O cliente C_1 entra no sistema e uma janela é criada no servidor, agendando o início do próximo fluxo para o segundo 30. Como nenhum outro cliente chega no decorrer de 30 segundos, o fluxo é criado somente para C_1. C_2 chega no segundo 50 e o servidor abre uma nova janela de *batching*, C_3 chega no segundo 70 e entra na mesma janela que C_2. No segundo 80 um novo fluxo é criado pelo servidor para atender C_2 e C_3. C_4 chega no segundo 85, e é aberta uma terceira janela de *batching* que termina no segundo 115.

O *Batching* é um protocolo simples e com grandes possibilidades de economia de banda pelo servidor. Quanto maior o tempo da janela de *batching* maiores as possibilidades de compartilhamento de fluxos e maior a economia de banda no servidor. Porém esse protocolo não provê vídeo-sob-demanda verdadeiro: ao realizarem requisições, os clientes não são imediatamente atendidos.

2.2. Patching

No *Patching* [Hua et al. 1998] o servidor também funciona com janelas de transmissão. O primeiro cliente a chegar numa janela é servido com um fluxo compartilhado, chamado fluxo principal. Clientes subseqüentes recebem o fluxo principal e um fluxo ponto-a-ponto para serviço imediato. Enquanto o cliente recebe e reproduz o conteúdo do fluxo para serviço imediato, os dados do fluxo principal são temporariamente armazenados localmente. Quando o fluxo para serviço imediato tiver transmitido todos os dados que o cliente havia perdido do fluxo principal, ele é terminado e o cliente continua a exibição

com o conteúdo recebido do fluxo principal armazenado localmente. A duração da janela de transmissão é um parâmetro do protocolo, e define o intervalo de tempo entre a criação de fluxos principais. O valor ótimo para esse parâmetro, que minimiza a banda utilizada pelo servidor, é $(\sqrt{2N+1}-1)/N$, onde N é a quantidade de clientes que chega durante um período igual à duração da mídia [Eager et al. 2001].

Para exemplificar, suponha que um servidor transmitindo mídia com o *Patching* está utilizando uma janela de transmissão de 30 segundos, e é submetido às chegadas mostradas na tabela 1. No instante inicial, C_1 chega no sistema e o servidor inicia uma nova janela e abre um novo fluxo principal. Nenhum fluxo para serviço imediato será criado nesta janela de transmissão. No segundo 50 C_2 chega ao sistema e o servidor inicia uma nova janela, criando um novo fluxo principal. C_3 chega ao sistema no segundo 70 e recebe o fluxo principal criado para C_2, e um fluxo para serviço imediato que irá durar 20 segundos. C_4 chega no segundo 85 e é criada outra janela de transmissão.

2.3. Bandwidth Skimmming

O *Bandwidth Skimming* [Eager et al. 1999, Eager et al. 2000, Eager et al. 2001] é um protocolo que utiliza do conceito de fusão hierárquica de fluxos. Todos os fluxos enviados pelo servidor são compartilhados. A estratégia geral dos protocolos de fusão de fluxos é baseada nas seguintes idéias: um cliente deve receber mais dados do que está exibindo, acumulando mídia em armazenamento local; este acúmulo de dados faz com que o cliente alcance outros clientes com relação à quantidade de dados recebidos; um fluxo A se funde com um fluxo anterior B quando todos os clientes escutando A tiverem armazenado localmente dados para exibir a mídia até o ponto transmitido pelo fluxo B.

A figura 1 mostra o funcionamento do protocolo. O eixo x mostra o tempo e o eixo y mostra a posição da mídia. As linhas mostram os fluxos transmitidos pelo servidor. Este exemplo mostra o funcionamento do *Bandwidth Skimming* com a política de seleção do alvo mais próximo (*Closest Target*) [Eager et al. 1999]. Nessa política um novo cliente escuta dois fluxos: o fluxo alvo, o fluxo anterior mais próximo com o qual tenta se fundir, e um fluxo principal, criado para prover serviço imediato ao cliente.

No instante inicial o cliente C_1 chega no servidor. Como não existe nenhum fluxo no sistema, C_1 escuta somente o fluxo principal. No segundo 50 o cliente C_2 entra no sistema. C_2 recebe um novo fluxo, o qual ele reproduz, enquanto recebe e armazena o fluxo principal de C_1 em armazenamento local temporário. No tempo 70, C_3 entra no sistema. Da mesma forma que ocorreu com C_2, é criado um novo fluxo principal para C_3 enquanto ele recebe e armazena dados do fluxo de C_2 localmente. No tempo 85, C_4 entra no sistema. É criado um novo fluxo principal para C_4, que escuta também o fluxo de C_3. No tempo 90 o fluxo principal de C_3 se funde com o de C_2, porque C_3 já terá acumulado dados em armazenamento local para reproduzir a mídia até o ponto enviado pelo fluxo de C_2. Neste instante a tentativa de fusão do fluxo de C_4 com o fluxo de C_3 é cancelada e C_4 escolhe o fluxo compartilhado por C_2 e C_3 como novo alvo. Essa fusão acontece no segundo 125. Finalmente, no segundo 175 o fluxo compartilhado por C_2, C_3 e C_4 funde-se ao fluxo de C_1.

2.4. P2Cast

O *P2Cast* [Guo et al. 2003] é uma versão distribuída do *Patching*. A diferença deste para o *Patching* é que o fluxo para serviço imediato pode ser transmitido por um cliente que

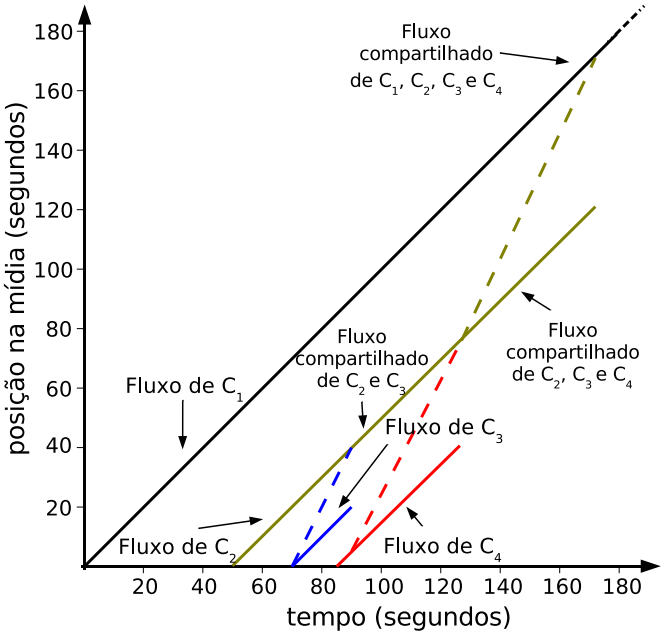


Figura 1. Bandwith Skimming: Funcionamento

possua os dados necessários armazenados localmente. O algoritmo que faz a seleção do cliente que irá enviar o fluxo para serviço imediato leva em consideração a banda e o atraso de rede entre os clientes. Assim como o *Patching*, o *P2Cast* possui o mesmo conceito de janela de transmissão para a criação de novos fluxos principais. Porém, no *P2Cast*, quanto maior o tamanho desta janela maior a economia de banda no servidor, pois aumenta a quantidade de clientes colaborando dentro de uma mesma janela.

Para exemplificar o protocolo vamos utilizar os dados da tabela 1. Suponha que C_1 tenha banda para envio de 2 fluxos, e C_2 e C_3 não possuam banda disponível para envio de fluxos. Desconsidere restrições de latência e suponha que a janela de transmissão seja de 100 segundos. No instante inicial C_1 chega e o servidor abre um fluxo principal compartilhado para servi-lo. C_2 entra no sistema no segundo 50 e começa a escutar o mesmo fluxo principal. C_1 envia o fluxo de serviço imediato para C_2 pois possui banda disponível. C_3 entra no sistema no segundo 70 e recebe o fluxo principal compartilhado. C_2 não possui banda disponível e C_3 recebe o fluxo de serviço imediato de C_1. No tempo 85 C_4 entra no sistema e escuta o fluxo principal. Neste momento, C_1, C_2 e C_3 não possuem banda para iniciar um fluxo de serviço imediato e C_4 recebe esse fluxo do servidor.

3. Stream2P

O Stream2P é uma extensão ao protocolo de distribuição de mídia contínua Bandwidth Skimming, onde o servidor utiliza recursos ociosos nos clientes para minimizar seus requisitos de banda. A relação do Stream2P com o *Bandwidth Skimming* é similar à relação do protocolo *P2Cast* com o *Patching*.

Todos os fluxos criados pelo Stream2P são fluxos compartilhados. Estes fluxos compartilhados podem ser obtidos através da infra-estrutura de rede (*multicast* IP ou *broadcast*) ou redes sobrepostas. Da mesma forma que no *Bandwidth Skimming*, explicado na seção 2, cada cliente recebe dados numa taxa maior que a taxa de exibição.

Ao herdar o comportamento básico do *Bandwidth Skimming*, o Stream2P pode utilizar qualquer uma das variações do *Bandwidth Skimming* para realizar a hierarquia de fusões. Podem ser usadas estratégias de particionamento para casos onde os clientes possuem banda insuficiente para receber dois fluxos [Eager et al. 2000], ou qualquer uma das políticas de seleção de alvos. Neste trabalho assumimos que o cliente possui banda suficiente para receber dois fluxos e utilizamos a política de seleção do alvo mais próximo, que apresenta desempenho próximo do ótimo [Eager et al. 1999].

A principal diferença entre o Stream2P e o *Bandwidth Skimming* é que clientes no Stream2P podem enviar dados para outros clientes na rede, se tiverem o conteúdo armazenado localmente e banda disponível para transmitir o fluxo de dados. A não ser pela sua origem, um fluxo enviado por um cliente é idêntico a qualquer fluxo enviado pelo servidor no que diz respeito à dinâmica do protocolo: fluxos enviados por clientes podem ser alvo ou se fundir a qualquer outro fluxo.

Quando um novo cliente faz uma requisição ao servidor, este inicia o serviço escolhendo o fluxo alvo para o cliente. Se existir fluxo transmitindo posição anterior à posição requisitada pelo cliente, o mais próximo desses fluxos será escolhido como fluxo alvo. Um fluxo principal precisa ser enviado ao cliente para serviço imediato. O servidor procura dentre os fluxos ativos se já existe algum fluxo transmitindo a posição requisitada. Se tal fluxo existir, então este fluxo é usado como fluxo principal do novo cliente e nenhum fluxo é criado. Caso contrário, o servidor procura dentre o conjunto de clientes ativos algum cliente que possa transmitir o fluxo principal para o novo cliente. Para tanto, um cliente precisa ter o dado requisitado em armazenamento local e possuir banda disponível para transmitir um fluxo de mídia. Caso mais de um cliente possa transmitir dados para o novo cliente, o servidor pode usar de um critério de desempate qualquer; neste trabalho escolhemos aleatóriamente o cliente, procurando obter um melhor balanceamento de carga através da distribuição uniforme da banda nos clientes. Se nenhum cliente puder transmitir os dados, o servidor se encarrega de criar o fluxo principal. O algoritmo 1 mostra o processo executado pelo servidor quando um novo cliente faz uma requisição.

Para que o algoritmo 1 seja implementado em sistemas reais, o servidor precisa manter informações a respeito dos clientes no sistema. A lista de todos os fluxos atuais,

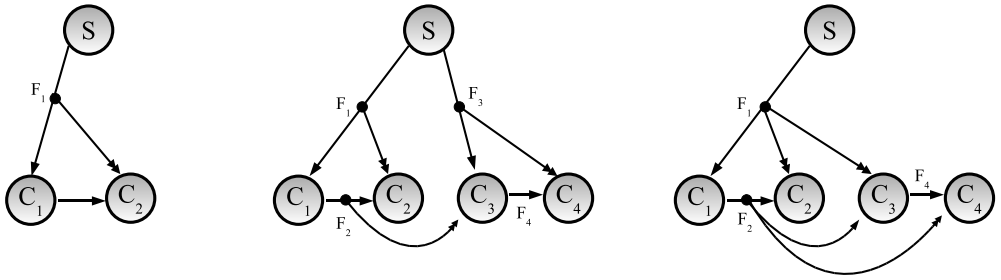
Algoritmo 1 Serviço de nova requisição do cliente C_n

se existe fluxo anterior à posição requisitada **então**
    $A \leftarrow$ fluxo anterior mais próximo à posição requisitada {fluxo alvo}
    adicionar C_n na lista de destinatários de A
se já existe fluxo F transmitindo a posição requisitada **então**
    adicionar C_n na lista de destinatários de F
senão
    $S \leftarrow$ conjunto de clientes capazes de servir a requisição
    se $S \neq \emptyset$ **então**
        selecionar aleatóriamente $C_s \in S$
        C_s transmite fluxo principal para C_n
    senão
        servidor transmite fluxo principal para C_n

sejam eles enviados pelo servidor ou por algum cliente, é necessária para que o servidor possa encontrar o fluxo alvo de um novo cliente. A banda de transmissão e o conteúdo armazenado localmente em cada cliente também precisam ser conhecidos pelo servidor, de forma que ele possa calcular o conjunto de clientes capazes de servir uma requisição. Para que estas informações sejam mantidas no servidor, o cliente precisa apenas informá-lo a respeito da sua banda de rede disponível para transmissão, e o tamanho da área de armazenamento local. Como todas as requisições passam pelo servidor, este pode calcular qual conteúdo cada cliente tem armazenado localmente a partir dos dados requisitados pelo cliente no passado. Toda comunicação de controle entre os clientes e o servidor pode utilizar métodos com garantia de entrega de dados, que não são de mídia contínua.

Suponhamos que os clientes C_1, C_2, C_3 e C_4, chegando ao sistema nos instantes mostrados na tabela 1, acessam um servidor utilizando o protocolo Stream2P. Suponhamos que os clientes C_1 e C_3 tenham banda suficiente para transmitir um fluxo de dados, enquanto os clientes C_2 e C_4 não possuem banda suficiente para transmitir fluxos. Todos os clientes possuem área de armazenamento local suficiente para armazenar toda a mídia.

Quando o cliente C_1 chega ao sistema, o servidor irá criar o fluxo F_1 para servir a requisição. Na ocasião da chegada de C_2, o servidor selecionará F_1 como fluxo alvo de C_2 e o cliente C_1 criará F_2, que será o fluxo principal do cliente C_2. Este cenário é mostrado na figura 2-a onde os círculos representam o servidor e clientes, e as arestas representam os fluxos. Setas simples designam fluxos principais e setas duplas designam fluxos alvos. Quando o cliente C_3 chegar ao sistema, o servidor selecionará F_2 como seu fluxo alvo. O fluxo principal, F_3, para C_3 será enviado pelo servidor, pois nenhum dos clientes presentes no sistema (C_1 e C_2) possuem banda disponível para transmitir-lhe a mídia. Por último, quando o cliente C_4 requisitar a mídia, o servidor selecionará F_3 como o fluxo alvo para C_4. C_3 irá criar o fluxo F_4, que servirá de fluxo principal para C_4. A topologia de transmissão resultante está mostrada na figura 2-b.



(a) Chegadas de C_1 e C_2 (b) Chegadas de C_3 e C_4 (c) Fusão de F_3 a F_2

Figura 2. Exemplo de Topologia de Transmissão no STREAM2P

Uma fusão de fluxos no Stream2P é resolvida da mesma forma que no *Bandwidth Skimming*. Quando F_3 se fundir a F_2 no exemplo anterior, C_3, o único cliente que escutava F_3 como fluxo principal, passará a escutar F_2 como fluxo principal e o fluxo F_3 será fechado. O novo alvo de C_3 será F_1 e o novo alvo de C_4 será F_2. A topologia resultante da fusão de F_3 a F_2 é mostrada na figura 2-c.

Quando um cliente que está transmitindo um fluxo deixa o sistema, um novo fluxo de dados terá de ser criado para substituir o fluxo cancelado. Este fluxo é criado segundo

os passos descritos no algoritmo 1, como se tivesse chegado no sistema uma requisição para a posição que estava sendo transmitida pelo fluxo cancelado. O Stream2P, como qualquer abordagem par-a-par, está sujeito a problemas de disponibilidade devido à transitoriedade dos clientes no sistema. Isto pode ser um problema, pois a saída de um cliente que esteja transmitindo um fluxo pode comprometer o serviço de vários clientes. Este problema pode ser contornado através do uso de armazenamento local, onde alguns segundos da mídia podem ser acumulados para garantir uma exibição sem falhas durante períodos de reconexão.

A banda de rede necessária para operação do Stream2P é da mesma ordem de grandeza da banda necessária pelo *Bandwidth Skimming*, salvo os dados adicionais que precisam transitar entre os clientes e o servidor para manutenção do controle do protocolo. Como estes dados adicionais causam uma carga muito menor à rede do que os fluxos de mídia, e como o *Bandwidth Skimming* utiliza quantidade de banda de rede próxima à mínima possível [Zhao et al. 2002], esperamos que o Stream2P seja eficiente com relação à utilização de banda de rede. Além disso, a política de seleção de clientes para enviar fluxos a novos clientes pode levar em consideração características da rede, como proximidade, objetivando minimizar o consumo de banda de rede. Por último, como o *Bandwidth Skimming*, o Stream2P pode ser diretamente aplicado a cargas interativas com mínimas modificações. Otimizações para interatividade já existentes para o *Bandwidth Skimming* [Rocha et al. 2005] podem ser adaptadas para esse novo protocolo.

4. Metodologia

Avaliar protocolos de transmissão de mídia contínua em um ambiente real é um experimento complexo, e difícil de ser realizado por vários motivos. Um ambiente distribuído com recursos suficientes para os testes é custoso de se obter e manter. Coletar, agregar e sumarizar dados coletados num ambiente distribuído real pode ser impossível, ou talvez possa ser feito apenas parcialmente.[4] A implementação é complexa, pois além da lógica do próprio protocolo é necessário lidar com os problemas de um ambiente real, por exemplo perda de pacotes e latência.

Devido aos fatores apresentados acima, decidimos utilizar simulação para fazer nossas análises. O uso de um simulador é conveniente neste caso porque permite encontrar resultados relevantes de forma simples. Criamos um simulador orientado a eventos, que é dividido em duas grandes partes. A primeira delas é um arcabouço de programação que representa de forma abstrata a rede, os clientes no sistema e os fluxos ativos. A segunda parte do simulador é composta pela implementação dos vários protocolos analisados neste trabalho. Esta abordagem propicia a comparação dos vários protocolos dentro de um mesmo ambiente com as mesmas características.

Para realizar nossas simulações geramos cargas de trabalho com as características apresentadas a seguir. A taxa de chegada de clientes segue um processo de Poisson com taxa λ. A duração da mídia, T, é fixa em 1000 segundos. Cada cliente faz uma requisição ao servidor e deixa o sistema após a requisição ter sido completada. As requisições são do início até o final da mídia. Nossas cargas possuem apenas um objeto de mídia, já que o funcionamento dos protocolos é independente da quantidade de mídias armazenadas no servidor. Assumimos que a mídia possui codificação com taxa de *bits* constante. Por

[4]Por exemplo, devido a política de privacidade dos usuários envolvidos nos testes.

último, cada cliente possui uma quantidade finita de banda disponível para a transmissão de fluxos e espaço local de armazenamento ilimitado.

Nossa análise é feita em função da taxa normalizada de chegada de clientes, N, definida como o produto da taxa de chegada de clientes pela duração da mídia ($\lambda \times T$). N pode ser interpretada como a quantidade média de clientes assistindo a mídia. Variando N conseguimos analisar a escalabilidade dos protocolos. Geramos 30 cargas de trabalho, variando λ para obter N entre 1 e 500. Há 5 versões de cada uma de nossas cargas, cada uma delas geradas com uma semente diferente. Essas versões são usadas para calcular o intervalo de confiança dos resultados.

Para compararmos o desempenho dos protocolos de transmissão usamos a banda média utilizada pelo servidor para atender uma dada quantidade de clientes como métrica de interesse. Expressamos a banda média em quantidade de fluxos de mídia.

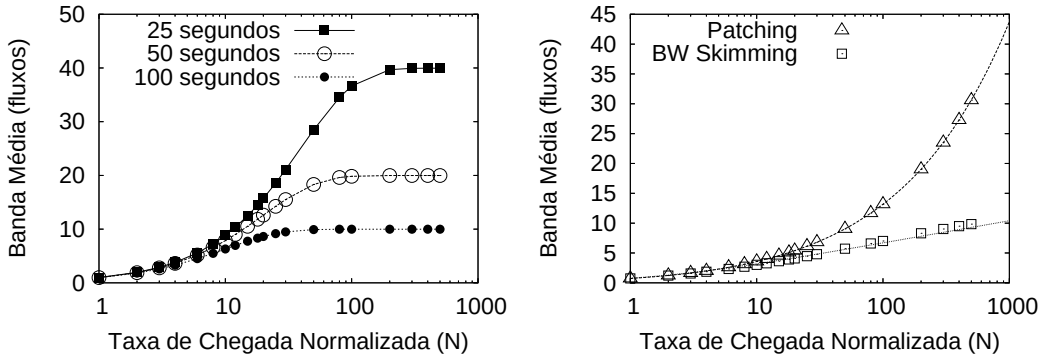
4.1. Validação

Validamos nosso simulador comparando seus resultados com resultados analíticos já conhecidos para os protocolos implementados. Para o protocolo *Batching* a banda utilizada pelo servidor é no máximo a divisão do tamanho do objeto pelo tamanho da janela, $\frac{|objeto|}{|janela|}$. Para o protocolo *Patching* utilizando o valor ótimo para o tamanho da janela de transmissão e chegadas de cliente Poisson, a banda média utilizada pelo servidor é dada por $\sqrt{2 \times N + 1} - 1$ [Eager et al. 2001]. Para protocolos de fusão hierárquica que escutam dois fluxos e taxa de chegada de clientes Poisson, o limite inferior da banda média utilizada pelo servidor é $1.62 \times log(\frac{N}{1.62} + 1)$ [Eager et al. 2001]. Resultados analíticos do desempenho do *P2Cast* não são conhecidos. Desta forma, validamos este protocolo a partir do protocolo *Patching*, no qual ele é baseado.

A figura 3 mostra no eixo y a quantidade de fluxos necessários no servidor para atender a uma dada taxa de chegada de clientes, mostrada no eixo x. A figura 3-a mostra a banda necessária no servidor para o protocolo *Batching* para vários tamanhos de janela. Sabendo que o objeto usado em nossas simulações possui 1000 segundos, o resultado encontrado é o esperado. A figura 3-b mostra a banda analítica (linha) necessária pelo protocolo *Patching* e a banda simulada (triângulos), o R^2 da regressão é 0.9993. A figura 3-b mostra ainda a banda mínima necessária pelos protocolos de fusão hierárquica para transmissão de mídia (linha) e a banda encontrada na simulação do *Bandwidth Skimming* implementado (quadrados). A banda simulada se situa acima da banda ótima até no máximo 6%, o que foi encontrado também em [Eager et al. 1999] para a política de seleção do alvo mais próximo. Todos os dados apresentados estão distantes da média em no máximo 2% com 99% de confiança.

5. Redes com compartilhamento de fluxos

Neste trabalho damos enfoque nas redes com compartilhamento de fluxo, que são redes onde algum nó transmissor envia apenas um fluxo de dados, e estes dados são transmitidos a vários receptores pela própria infraestrutura de rede, consumindo apenas banda equivalente a um fluxo de dados no transmissor e nos receptores. Este é o caso das redes IP que implementam *multicast* ou redes onde a transmissão é feita através de *broadcast*, que é o cenário de redes corporativas ou controladas.



(a) Batching (b) Patching e Bandwidth Skimming

Figura 3. Validação do Simulador

Os protocolos que utilizam recursos dos clientes, como o *P2Cast* e Stream2P, foram analisados para vários cenários, variando a quantidade de recursos disponíveis na rede. Variamos a quantidade de clientes com capacidade para transmitir fluxos de 1% a 100%, assumindo que cada cliente com capacidade de transmissão consegue enviar apenas um fluxo. Todos os dados apresentados estão distantes da média em no máximo 10% com 90% de confiança.

A figura 4 mostra a banda de servidor gasta pelos diversos protocolos de distribuição de mídia. No eixo x está a taxa de chegada de clientes, e no eixo y é mostrada a banda necessária pelo servidor para atender aquela taxa de chegada. As curvas dos protocolos *Patching* e *P2Cast* foram obtidas usando a janela ótima de distribuição [Eager et al. 2001]. As curvas do *Bandwidth Skimming* e do Stream2P foram obtidas para clientes que escutam dois fluxos simultaneamente. A curva do protocolo *P2Cast* foi obtida para 100% dos clientes com recursos para transmitir dados, enquanto a curva do Stream2P foi obtida quando 1% dos clientes podem transmitir dados.

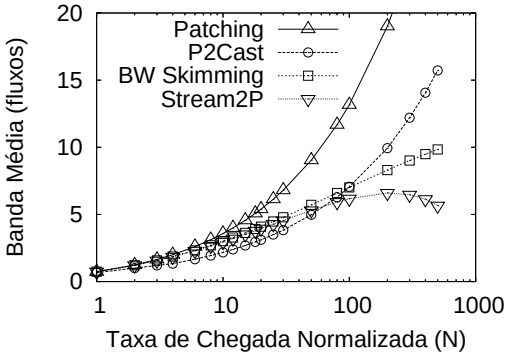


Figura 4. Comparação dos Protocolos

Os dados para o protocolo *Batching* são mostrados na figura 3-a e são conhecidos na literatura. O protocolo *Batching*, apesar de competitivo com os protocolos apresentados na figura 4 para altas taxas de chegada, não é um protocolo de serviço imediato e não suporta interatividade, o que limita sua aplicabilidade.

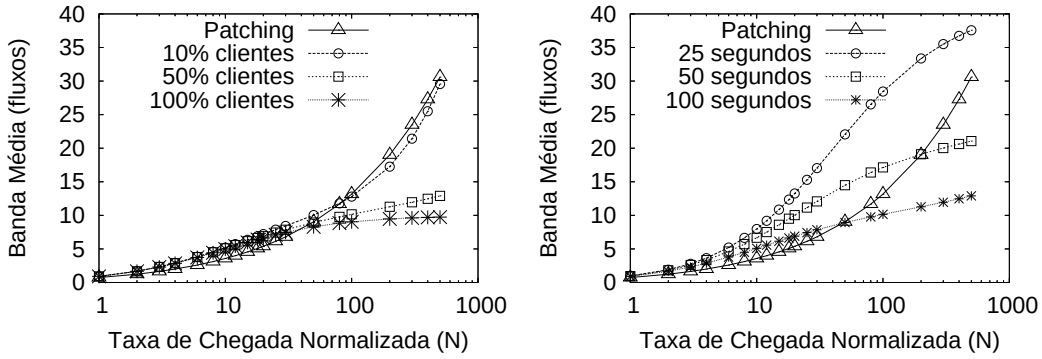
O protocolo *Patching* utiliza mais banda que todos os protocolos mostrados na

figura 4. O *Bandwidth Skimming*, mesmo sendo um protocolo centralizado, precisa de menos banda (até 37%) que o *P2Cast* utilizando janela ótima para o *Patching*. Isso mostra que abordagens distribuídas podem ser menos eficientes do que abordagens centralizadas. Os detalhes de desempenho do protocolo *P2Cast* são abordados na seção 5.1.

O protocolo Stream2P necessita da menor quantidade de banda dentre os protocolos testados (no cenário mostrado, até 42% menos banda que o *Bandwidth Skimming* e até 64% menos banda que o *P2Cast*). A economia de banda vem com o custo de uma maior complexidade do sistema de gerência da rede e dos clientes, que devem ser capazes de enviar fluxos *multicast*. Os desempenho do Stream2P é avaliado na seção 5.2.

5.1. P2Cast

A figura 5-a mostra a banda necessária num servidor usando *P2Cast* para atender uma dada carga quando 10, 50 e 100% dos clientes possuem recursos para transmitir um fluxo de dados para outros clientes, fixando o tamanho da janela de transmissão em 100 segundos. Mostramos a curva de desempenho do *Patching* para comparação. O desempenho do *P2Cast* é muito limitado quando a quantidade de recursos disponíveis na rede é baixo, e os ganhos em banda utilizada no servidor sobre o *Patching* não ultrapassam 9%, quando apenas 10% dos clientes transmitem dados. A medida que a quantidade de recursos disponíveis aumenta, todos os fluxos para serviço imediato passam a ser enviados por clientes, e o servidor fica responsável por enviar apenas os fluxos principais. Por isso, a linha relativa ao cenário onde 100% dos clientes têm banda aproxima-se de 10 fluxos, que é a quantidade de fluxos principais necessários ($\frac{|objeto|}{|janela|}$).



(a) Janela de 100 segundos (b) 50% dos clientes com recursos

Figura 5. Desempenho do P2Cast

O impacto do tamanho da janela no *P2Cast* é mostrado na figura 5-b, em um cenário onde 50% dos clientes possuem banda para transmitir fluxos. O resultado para outras quantidades de clientes com capacidade de transmitir fluxos é qualitativamente similar. Vemos que o aumento da janela diminui a quantidade de banda gasta pelo servidor, devido à maior quantidade de clientes dentro de uma janela e, conseqüentemente, maior cooperação entre os clientes no envio de fluxos para serviço imediato. Com uma janela de 100 segundos e 50% dos clientes enviando dados, o *P2Cast* obtém uma economia de até 57% na banda necessária para o *Patching*. O *Bandwidth Skimming* possui desempenho superior ao *P2Cast* nesse cenário, economizando até 23% mais banda do servidor. Utilizando uma janela de 100 segundos e 100% dos clientes enviando dados, o *P2Cast*

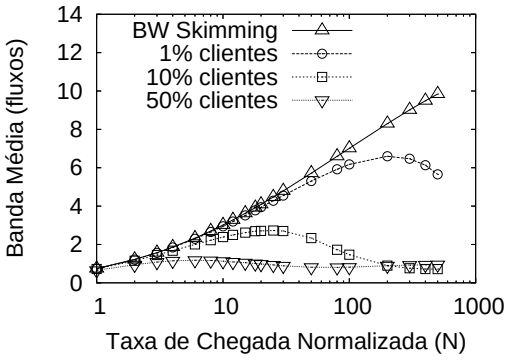


Figura 6. Desempenho do Stream2P

tem desempenho superior ao *Patching* e *Bandwidth Skimming*, economizando 67% e 1% de banda sobre cada um deles, respectivamente.

O aumento do tamanho da janela impacta o desempenho do servidor para cargas intermediárias. Para uma janela de 100 segundos e 50% dos clientes com banda para transmitir dados, o protocolo *Patching* obtém desempenho superior (economizando até 28% de banda do servidor) ao *P2Cast* para valores de N entre 2 e 40, através da utilização de uma janela mais adequada. A utilização de janelas pequenas também impacta o desempenho do *P2Cast* de forma considerável, como mostrado na figura 5-b, para janelas de 25 e 50 segundos. Nestes cenários, o *Patching* economiza até 53% de banda sobre o *P2Cast*. O aumento do tamanho da janela aumenta também a cooperação entre os pares e, conseqüentemente, aumenta as chances de um cliente ter seu fluxo interrompido por causa de seu par deixar a rede [Guo et al. 2003].

O *P2Cast*, apesar de ser mais simples que o Stream2P, precisa de muitos recursos da rede para obter ganhos significativos, e é sensível ao tamanho da janela.

5.2. Stream2P

A figura 6 mostra a banda necessária pelo servidor para atender os clientes quando 1, 10 e 50% dos clientes dispõem de recursos para transmitir um fluxo de mídia. Vemos que para 1% dos clientes com recursos para enviar um fluxo de dados, a banda média do servidor cresce até 6,6 fluxos para 200 clientes simultâneos e depois disso começa a decrescer. Isto acontece porque com mais clientes no sistema a quantidade de clientes com disponibilidade de enviar fluxos também aumenta, pois a quantidade deles que envia fluxo é 1% de todos os clientes. Estes clientes com disponibilidade passam a transmitir fluxos, diminuindo a banda necessária no servidor. Notamos porém que a banda total utilizada pelos clientes e servidor é maior que a banda utilizada pelo *Bandwidth Skimming*. Por exemplo, para $N = 500$, temos em média 5 clientes enviando fluxos e 5,6 fluxos do servidor no Stream2P, o que totaliza uma banda média maior do que a necessária pelo *Bandwidth Skimming* centralizado: 9,8 fluxos.

Quando 50% dos clientes têm recursos para enviar dados, essa quantidade ultrapassa a necessária para a distribuição da mídia. Desta forma, o servidor não precisa despender de quase nenhuma banda. O servidor utiliza aproximadamente um fluxo para distribuir o conteúdo, enquanto clientes se encarregam de repassar o conteúdo para outros clientes. A economia de banda do servidor reflete em gasto banda adicional nos clientes.

A grande economia de banda obtida pelo Stream2P vem ao custo de uma maior complexidade do protocolo, maiores requisitos de banda e armazenamento local no cliente. Mostramos que o Stream2P consegue ganhos muito mais expressivos que o *P2Cast* mesmo se a quantidade de clientes com recursos for menor.

6. Redes Sobrepostas

Nesta seção analisamos o desempenho da nossa proposta e de outros protocolos quando a rede não provê compartilhamento de fluxos. Neste cenário, onde existem apenas conexões ponto-a-ponto, é impossível que o servidor envie um fluxo que sirva vários clientes.

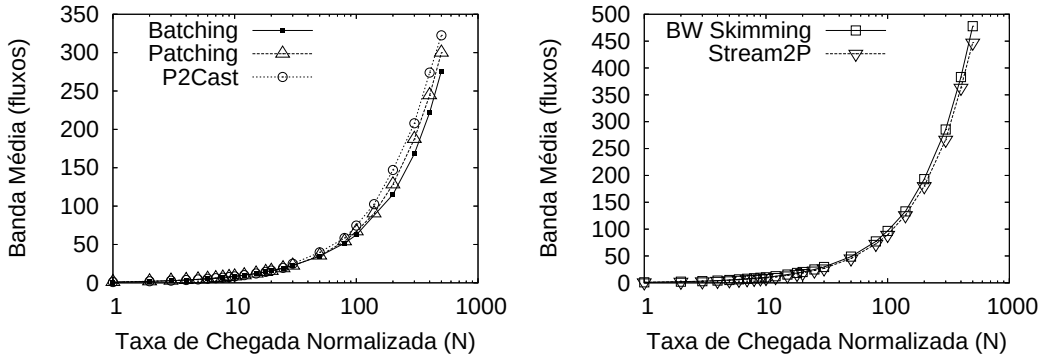
Para suprir esta deficiência, várias propostas de redes a nível de aplicação foram criadas, algumas delas com o propósito específico de distribuir mídia contínua [Rejaie and Ortega 2003, Hefeeda et al. 2003, Castro et al. 2003, Tran et al. 2003]. O objetivo destas redes é fazer que, após o servidor ter enviado um fluxo de dados através da rede de aplicação, esta rede se encarregue de distribuir os dados entre os clientes, sem nenhum encargo adicional de banda para o servidor.

Neste novo cenário, os protocolos continuam funcionando da mesma forma de quando a rede tinha fluxos compartilhados. O controle sobre os clientes e os fluxos a serem transmitidos são os mesmos, a diferença é que quando o servidor precisa criar um fluxo para ser recebido por vários clientes, ele transmite o fluxo através de uma rede sobreposta. Clientes que precisem escutar este fluxo usam da rede sobreposta para recebê-lo. Neste cenário, os clientes precisam ter mais banda disponível do que no cenário de redes com compartilhamento de fluxos para formar a rede sobreposta. Em contrapartida, todos os fluxos neste cenário são simples fluxos ponto-a-ponto.

Em nosso simulador, implementamos uma rede a nível de aplicação simples, na qual os clientes podem enviar dados a outros clientes se tiverem banda disponível. Desconsideramos qualquer efeito de latência. Caso a banda disponível nos clientes seja insuficiente para que a rede sobreposta consiga entregar o fluxo a todos os clientes que o desejam, o servidor será responsável por arcar com esta insuficiência enviando diretamente os dados aos clientes, garantindo assim a qualidade do serviço a todos os clientes.

A figura 7 apresenta resultados para os protocolos analisados, quando 50% dos clientes possuem banda suficiente para transmitir um fluxo. Neste cenário, onde a banda média disponível nos clientes é menor do que um, a rede sobreposta não pode ser montada de forma a substituir os fluxos compartilhados necessários para a operação dos protocolos. Qualquer fluxo necessário para a construção da rede sobreposta, que não possa ser fornecido por um cliente, é fornecido pelo servidor e por isso nenhum dos protocolos atinge os níveis esperados de desempenho. Os protocolos *Batching*, *Patching* e *P2Cast*, mostrados na figura 7-a apresentam comportamento muito similar, pois cada cliente precisa receber um fluxo compartilhado na operação destes protocolos. Alguns destes fluxos são fornecidos por clientes através da rede sobreposta e os fluxos restantes são fornecidos pelo servidor. Os protocolos *Bandwidth Skimming* e Stream2P, mostrados na figura 7-b utilizam mais banda do servidor, pois nestes protocolos os clientes recebem dois fluxos compartilhados. Como a quantidade de fluxos que pode ser transmitida por clientes é a mesma, mais fluxos precisam ser fornecidos pelo servidor.

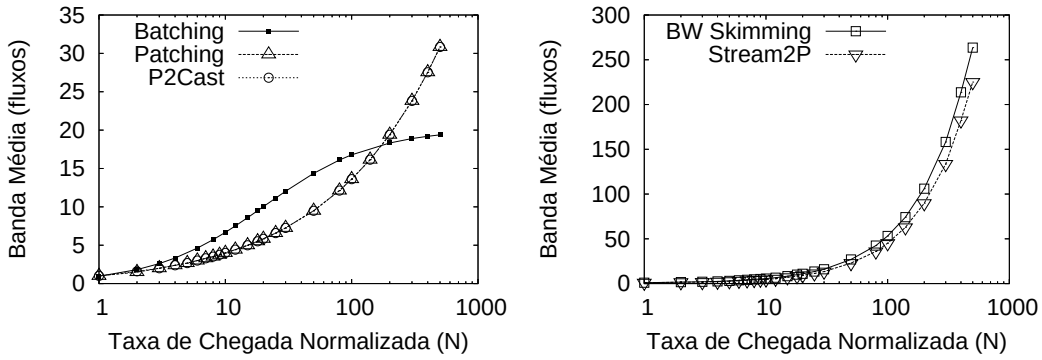
A figura 8 mostra a banda média utilizada pelo servidor para os protocolos analisados, quando todos os clientes possuem banda para enviar um fluxo de dados. Este



(a) Batching, Patching e P2Cast (b) BW Skimming e Stream2P

Figura 7. Comparação dos protocolos para redes sobrepostas (banda < 5)

fluxo disponível em cada cliente pode ser utilizado para formar uma rede sobreposta que substitui um fluxo compartilhado. Os protocolos *Batching*, *Patching* e *P2Cast*, mostrados na figura 8-a, apresentam o desempenho esperado, pois utilizam apenas um fluxo compartilhado para cada cliente na rede. Os protocolos *Batching* e *Patching* atingem o mesmo desempenho apresentado na seção 5. O protocolo *P2Cast* tem desempenho igual ao protocolo *Patching* pois a banda dos clientes é utilizada para a construção da rede sobreposta, não sobrando nenhuma banda disponível para colaboração entre clientes. A figura 8-b mostra o desempenho dos protocolos *Bandwidth Skimming* e Stream2P. Novamente, como estes protocolos utilizam dois fluxos compartilhados para cada cliente, e a banda dos clientes é suficiente para substituir apenas um fluxo compartilhado através de redes sobrepostas, o servidor precisa transmitir vários fluxos para os clientes. Neste cenário a banda do Stream2P é mais de 7 vezes maior que a banda do *Patching*.



(a) Batching, Patching e P2Cast (b) BW Skimming e Stream2P

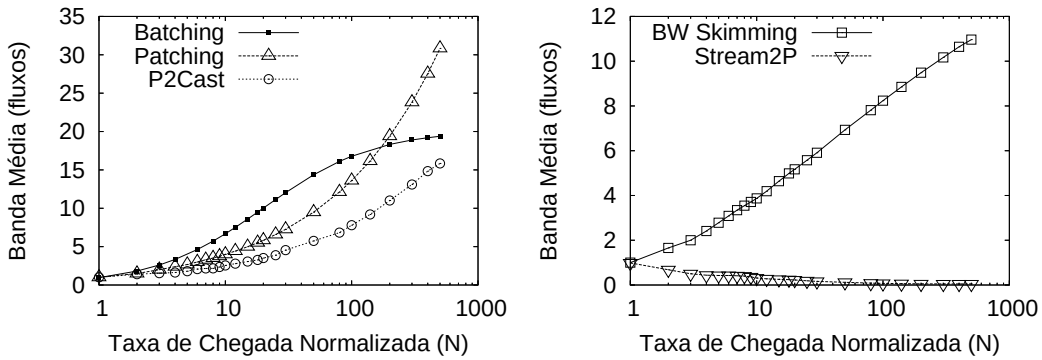
Figura 8. Comparação dos protocolos para redes sobrepostas (banda = 1)

Apesar do *Bandwidth Skimming* e Stream2P não serem eficientes neste cenário, isto deve-se a estarmos usando a versão destes protocolos onde os clientes escutam dois fluxos. Variações do *Bandwidth Skimming* propostas em [Eager et al. 2000] podem ser usadas para diminuir este requisito de banda para 1,2 fluxos. Se os clientes tiverem banda para transmitir em média 1,2 fluxos, sabemos que o *Bandwidth Skimming* é mais eficiente que o *Patching* e o *P2Cast*, que utilizam apenas um fluxo dos clientes. Nesse cenário

esperamos resultado de desempenho do Stream2P qualitativamente superior ao conhecido na literatura para o *Bandwidth Skimming*.

A figura 9 mostra o desempenho dos protocolos quando os clientes possuem banda suficiente para transmitir dois fluxos de mídia. Os protocolos *Batching* e *Patching* apresentam o mesmo desempenho do cenário anterior, pois não fazem nenhum uso da banda adicional disponível nos clientes. O desempenho do protocolo *Bandwidth Skimming* é equivalente ao desempenho apresentado na seção 5. O desempenho do protocolo *P2Cast* melhora em 48% em relação ao cenário anterior, ganho decorrente da cooperação entre clientes possibilitada pela quantidade adicional de banda nos clientes. O protocolo Stream2P também obtém desempenho similar ao obtido em redes com compartilhamento de fluxos, mostrados na seção 5. A banda disponível nos clientes é utilizada para que distribuam o vídeo entre si enquanto o servidor apenas provê o conteúdo, um efeito similar ao visto em redes *BitTorrent*.

A alta economia de banda apresentada pelo Stream2P decorre da utilização dos recursos disponíveis nos clientes de forma mais eficiente que nos outros protocolos. Os protocolos *Batching*, *Patching* e *P2Cast* usam apenas um fluxo compartilhado. Nesses protocolos, os clientes precisam receber e transmitir um fluxo de dados durante a maior parte do período de exibição da mídia. Para os protocolos *Bandwidth Skimming* e Stream2P, que utilizam dois fluxos compartilhados, os clientes precisam receber e transmitir dois fluxos de dados. Isto significa que nesse cenário, os protocolos *Bandwidth Skimming* e Stream2P causam maior carga na rede e maiores requisitos de banda nos clientes.



(a) Batching, Patching e P2Cast (b) BW Skimming e Stream2P

Figura 9. Comparação dos protocolos para redes sobrepostas (banda = 2)

7. Conclusões e Trabalhos Futuros

Este trabalho apresentou o Stream2P, um protocolo distribuído para transmissão de mídia contínua baseado no *Bandwidth Skimming*. Para redes com compartilhamento de fluxos, mostramos que o Stream2P consegue uma economia de banda do servidor de até 42% em relação ao protocolo centralizado *Bandwidth Skimming* e até 64% com relação ao protocolo distribuído *P2Cast*, mesmo quando apenas 1% dos clientes possui banda para transmitir fluxos. Para redes sem compartilhamento de fluxos mostramos que o Stream2P é eficiente se os clientes puderem transmitir dois fluxos de mídia.

O próximo passo neste trabalho é avaliar o impacto causado na qualidade do serviço por clientes deixando o sistema, e a utilização de diferentes mecanismos para man-

ter a exibição ininterrupta da mídia. Pretendemos também avaliar a complexidade do Stream2P, comparando-a com a dos outros protocolos existentes. Por fim iremos incluir extensões ao Stream2P, adequadas para cenários onde os clientes não possuem banda para transmitir dois fluxos.

Referências

Aggarwal, C., Wolf, J., and Yu, P. (1996). A Permutation-Based Pyramid Broadcasting Scheme for Video-on-Demand System. In *IEEE ICMS*, Washington, DC.

Castro, M., Druschel, P., Kermarrec, A., Nandi, A., Rowstron, A., and Singh, A. (2003). SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *ACM Symposium on Operating Systems Principles (SOSP)*, Lake Bolton, New York.

Chesire, M., Wolman, A., Voelker, G., and Levy, H. (2001). Measurement and Analysis of a Streaming Media Workload. In *USENIX Symp. on Internet Technologies and Systems*, San Francisco, California.

Dan, A., Sitaram, D., and Shahabuddin, P. (1994). Scheduling Policies for an On-Demand Video Server with Batching. In *ACM Multimedia*, San Francisco, California.

Eager, D. and Vernon, M. (1998). Dynamic Skyscraper Broadcasts for Video-on-Demand. In *Multimedia Information Systems*, Istanbul, Turkey.

Eager, D., Vernon, M., and Zahorjan, J. (1999). Optimal and Efficient Merging Schedules for Video-on-Demand Servers. In *ACM Multimedia*, Orlando, Florida.

Eager, D., Vernon, M., and Zahorjan, J. (2000). Bandwidth Skimming: A Technique for Cost-Effective Video-on-Demand. In *IS&T/SPIE Conf. on Multimedia Computing and Networking (MMCN)*, San Jose, California.

Eager, D., Vernon, M., and Zahorjan, J. (2001). Minimizing Bandwidth Requirements for On-Demand Data Delivery. *Knowledge and Data Engineering*, 13(5):742–757.

Guo, Y., Suh, K., Kurose, J., and Towsley, D. (2003). P2Cast: Peer-to-Peer Patching Scheme for VoD Service. In *ACM WWW*, Budapest, Hungary.

Hefeeda, M., Habib, A., Botev, B., Xu, D., and Bhargava, B. (2003). PROMISE: Peer-to-Peer Media Streaming Using CollectCast. In *ACM Multimedia*, Berkeley, California.

Hua, K., Cai, Y., and Sheu, S. (1998). Patching: A Multicast Technique for True Video-on-Demand Services. In *ACM Multimedia*, Bristol, United Kingdom.

Rejaie, R. and Ortega, A. (2003). PALS: Peer-to-Peer Adaptive Layered Streaming. In *ACM NOSSDAV*, Monterey, California.

Rocha, M., Maia, M., Cunha, Í., Almeida, J., and Campos, S. (2005). Scalable Media Streaming to Interactive Users. In *ACM Multimedia*, Singapore, Singapore.

Sheu, S., Hua, K., and Tavanapong, W. (1997). Chaining: A Generalized Batching Technique for Video-On-Demand. In *IEEE ICMCS*, Ottawa, Canada.

Tran, D., Hua, K., and Do, T. (2003). ZigZag: An Efficient Peer-to-Peer Scheme for Media Streaming. In *IEEE INFOCOM*, San Francisco, California.

Zhao, Y., Eager, D., and Vernon, M. (2002). Network Bandwidth Requirements for Scalable On-Demand Streaming. In *IEEE INFOCOM*, New York, New York.