Abordagem para Obter uma Rede de Sobreposição Coesa Visando Aplicações de *Live Streaming* em Sistemas P2P

Vânia R. Sávio Rodenas[1], **Ingrid Jansch-Pôrto**[1], **Marinho P. Barcellos** [1,2]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre, RS – Brasil

[2]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681, Prédio 32 – Porto Alegre, RS – Brasil

{vrsrodenas,ingrid}@inf.ufrgs.br, marinho@acm.org

Resumo. *Aplicações peer-to-peer (P2P) estão se tornando cada vez mais populares na Internet. Exemplos são compartilhamento de arquivos, vídeo sob-demanda, VoIP e live streaming. Os dois últimos possuem requisitos temporais rígidos. Como o modelo P2P não assegura o cumprimento dos requisitos nos serviços de distribuição de conteúdos "ao vivo", isso fica a cargo das aplicações. Nesse contexto, um dos maiores desafios é manter o serviço perante a entrada e saída de nodos participantes da rede P2P. A busca de arquiteturas de redes adaptativas e auto-organizáveis visa contribuir na obtenção dos requisitos funcionais e de qualidade de serviço desejados. Este artigo reflete resultados preliminares de uma pesquisa cujo objetivo é projetar protocolos para construção e manutenção de redes de sobreposição dinâmicas mais eficientes e escaláveis, visando aplicações de live streaming em ambientes P2P.*

1. Introdução

No paradigma P2P, todos os nodos participantes formam uma rede virtual sobre uma rede física. Uma das principais características desse tipo de rede é que cada nodo contribui com seus recursos disponíveis (por exemplo, largura de banda, espaço de armazenamento e poder computacional) [Li e Yin 2007]. Consequentemente, essa abordagem apresenta um potencial para escalar junto com o tamanho do grupo, pois o crescimento da quantidade de nodos vem acompanhada do aumento potencial da quantidade de recursos disponíveis.

Um grande número de aplicações atuais, incluindo TV pela Internet, transmissão de eventos esportivos, jogos *online* e educação à distância, requer suporte para transmissões em *live streaming*. Esta categoria de aplicações caracteriza-se pela entrega de conteúdo multimídia que está sendo codificado e transmitido quase simultaneamente a uma quantidade de usuários potencialmente grande.

A popularização de aplicações de *live streaming* dependerá, dentre outros fatores, do funcionamento eficiente e robusto em redes de larga escala. Para tal, são vários os desafios a serem enfrentados, dentre os quais pode-se destacar: *(i)* o número de usuários recebendo *streams* pode ser potencialmente grande; *(ii)* os atrasos na difusão devem manter-se dentro de limites aceitáveis; *(iii)* comportamento robusto do sistema mesmo na ocorrência de *churn* (alta taxa de entrada e saída de nodos na rede); *(iv)* manutenção da qualidade adequada na reprodução (ou seja, a taxa de pacotes perdidos deve ser mínima)

[Fodor e Dan 2007]. Esses desafios diferem dos impostos pelas aplicações de transferência de arquivos (ex., BitTorrent e Gnutella) onde, em geral, o interesse dos usuários é disperso no tempo, há bastante variação na latência da difusão, mas nenhuma perda ou corrupção de dados é tolerada.

Dentre os desafios citados no parágrafo anterior, o foco deste trabalho reside na criação e na manutenção da rede de sobreposição (ou *overlay*). Os mecanismos usados na operação do sistema de *streaming* P2P devem voltar-se à busca de uma estrutura coesa e robusta, com baixa latência, baixas taxas de perdas e escalável. Parte significativa dos sistemas ou algoritmos propostos na literatura para a difusão de conteúdos "ao vivo" consideram, em sua descrição ou experimentos, uma rede de sobreposição estática, adiando a análise do problema. Atualmente, encontram-se sistemas de *live streaming* em redes P2P desenvolvidos em ambientes acadêmicos, tais como Anysee [Liao et al. 2006], CoolStreaming [Xie et al. 2007] e Prime [Magharei e Rejaie 2007]. No entanto, os dois primeiros fazem análises limitadas sobre o impacto que a entrada e saída de nodos possui sobre o funcionamento das aplicações, e não disponibilizam código que permita avaliações independentes das ações de difusão de conteúdos. O Prime, por sua vez, emprega uma rede estática. Mecanismos mencionados nesses sistemas serviram de inspiração na proposta de mecanismos usados em algoritmos aqui descritos.

O restante deste artigo está organizado como segue. Na Seção 2, são apresentados o conceito de *live streaming* e a caracterização básica das redes de sobreposição utilizadas por aplicações de *live streaming* em redes P2P. A Seção 3 reúne os trabalhos diretamente relacionados a este. Na Seção 4, é apresentada a proposta de abordagem da dinâmica comportamental dos nodos, com ênfase nos procedimentos de controle de entrada destes e manutenção da rede, com a chegada e saída de nodos. As conclusões e considerações sobre a continuação do trabalho são apresentadas na Seção 5, que encerra o artigo.

2. *Live Streaming* e Redes *Peer-to-Peer*

As aplicações *live streaming* possuem tipicamente uma única origem (também conhecida como *source*), responsável por gerar e iniciar a disseminação do conteúdo. A origem pode ser assumida como não falha, permanecendo presente durante toda uma sessão de transmissão. O endereço da origem é conhecido previamente pelos interessados, servindo como um ponto de encontro (*rendezvous*) para novos usuários entrarem na sessão, os quais serão considerados os nodos membros do sistema [Liu et al. 2008a].

As aplicações de *live streaming* utilizam arquiteturas de redes adaptativas e auto-organizáveis para conseguir alcançar seus requisitos de qualidade [Jurca et al. 2007]. As estruturas básicas que são geralmente consideradas para prover a organização necessária para essas aplicações são: rede de sobreposição com topologia em árvore, e rede de sobreposição com topologia em malha (*mesh*), também conhecida como *data-driven*. Variações dessas estruturas, tal como árvores múltiplas, e combinações de árvore com malha também têm sido empregadas.

Nas redes de sobreposição em árvore, os nodos são organizados dentro de estruturas de árvore que realizam a entrega de dados explorando propriedades básicas dessas. Os nodos nessa estrutura possuem relações bem definidas, como pai-filho. Como todos os pacotes são enviados através da árvore, uma limitação ou problema em um nodo próximo à raiz pode afetar negativamente um grande número de receptores. Torna-se crítico ga-

rantir que a árvore forneça bom desempenho para todos os receptores [Liu et al. 2008a]. Adicionalmente, os nodos-folha não atuam como disseminadores, observando-se um desequilíbrio nos papéis dos componentes da rede. As arquiteturas de árvores múltiplas visam tratar os problemas apresentados pelas árvores simples, provendo redundância nos caminhos da rede e buscando justiça na participação dos nodos. Entretanto, tal abordagem tem como limitação sua excessiva complexidade. Ela conduz à solução de problemas contraditórios, como minimizar a profundidade da árvore, enquanto simultaneamente provê a diversidade de caminhos de rede. A topologia física subjacente deve ser cuidadosamente estudada para obter uma disseminação eficiente de conteúdo [Jurca et al. 2007].

As redes de sobreposição em malha diferenciam-se das organizadas em árvore, pois nas primeiras não se constrói nem se mantêm uma estrutura explícita para a entrega de dados [Liu et al. 2008a]. Comparativamente, os nodos estabelecem e desfazem seus vizinhos de forma mais autônoma. Cada nodo busca manter vários vizinhos com os quais ele possa estabelecer relações de troca, recebendo e transmitindo conteúdo. Se um dos seus vizinhos sai do sistema, o nodo continua recebendo conteúdo dos vizinhos que ainda permanecem. Como os nodos mantêm vários nodos vizinhos simultaneamente, a saída de um deles não afeta severamente os que estão ligados a ele; logo, as redes em malha podem ser consideradas mais robustas contra o *churn* [Liu et al. 2008b]. Embora exista uma sobrecarga de mensagens associada à gerência da malha e à disseminação de conteúdo, ela parece ser independente do tamanho da rede, mas tende a aumentar com o dinamismo da rede (ou seja, as entradas e saídas dos nodos) [Fodor e Dan 2007].

3. Trabalhos Relacionados

Chainsaw [Pai et al. 2005] é um sistema utilizado para disseminar conteúdo em aplicações *live streaming*. Seus autores não tratam da criação e da manutenção da rede de sobreposição. Entretanto, a forma por eles empregada para disseminação é base para a que será empregada na rede composta pelos procedimentos aqui apresentados. Os próximos trabalhos, diretamente relacionados ao presente artigo, são exemplos de sistemas para a disseminação de conteúdos gerados "ao vivo" em redes P2P que incluem mecanismos para criação e/ou manutenção de uma rede em malha. Todos os sistemas aqui apresentados possuem como característica comum a presença de um nodo de *bootstrap*, que serve como ponto de entrada para novos usuários do sistema.

No Anysee [Liao et al. 2006], os nodos entram no sistema contactando um nodo central (*bootstrap*), e selecionam um ou mais nodos com os quais se conectar. Cada nodo mantém um conjunto de vizinhos lógicos. Nesse sistema é adotado um esquema de otimização considerando múltiplas redes (*inter-overlay*), em que recursos podem constar de múltiplas redes de sobreposição. É utilizada uma técnica que considera a localização física dos nodos para otimizar essa rede, encontrar os vizinhos mais próximos e eliminar as conexões lentas. Em cada nodo, é mantido um conjunto de caminhos ativos para a distribuição de pacotes e um outro conjunto de caminhos *backup*, utilizado para substituir os caminhos ativos. Quando o número de caminhos contidos no conjunto *backup* é menor que um limiar pré-estabelecido, um algoritmo de otimização é executado visando encontrar outros caminhos apropriados. Quando um caminho ativo é eliminado por desempenho insatisfatório, ou quando um nodo sai do sistema, um novo caminho para a redistribuição dos pacotes é selecionado a partir do conjunto *backup*.

No CoolStreaming [Xie et al. 2007], cada nodo mantém uma lista parcial dos identificadores dos nodos ativos, denominada *membership cache* (mCache). Um nodo usa a informação da mCache para estabelecer uma conexão TCP com algum outro nodo ativo no sistema. Um nodo que foi inserido recentemente contacta o nodo *bootstrap* para obter uma lista com um subconjunto de nodos no sistema e os armazena em sua mCache. O nodo de *bootstrap* seleciona aleatoriamente um conjunto de nodos dentre o conjunto completo desses. Baseado na lista de nodos presentes na sua mCache, o nodo randomicamente seleciona alguns deles para estabelecer conexões TCP (ditas parcerias). Uma vez que essa parceria é estabelecida, os nodos trocam as informações disponíveis em suas mCaches, de forma que um nodo aprenda sobre os parceiros do outro. Por várias razões, as parcerias podem ser desfeitas gradualmente; neste caso, os nodos devem executar uma re-seleção de parcerias (*partner reselection*) para garantir o serviço.

No Prime [Magharei e Rejaie 2007], os nodos participantes formam uma malha randômica direcionada, numa relação pai-filho. Cada nodo tenta manter um número suficiente de nodos-pai que possam coletivamente preencher toda a largura de banda disponível em seu enlace de entrada. Quando um nodo necessita de um (ou mais) novos nodos-pai, ele requisita a um nodo central um subconjunto randômico de nodos participantes, e então solicita que sirvam como seus nodos-pais. O Prime emprega um mecanismo dinâmico de entrega de conteúdo; seu principal objetivo é maximizar a qualidade percebida em cada nodo. Esse mecanismo possui relativa flexibilidade e pode ser controlado por nodos individualmente no sistema, porém é considerado complexo pelos próprios autores. No escopo do sistema Prime não estão definidos, ou previstos, métodos para a manutenção da rede de sobreposição.

4. Modelo de Funcionamento Proposto para o Sistema

Nesta seção será apresentada uma proposta de mecanismo para construção e manutenção da topologia em aplicações de *live streaming* em redes P2P em malha. Primeiramente, define-se o modelo proposto do sistema (Subseção 4.1). A seguir, apresenta-se os procedimentos de entrada de novos nodos (Subseção 4.2) e de manutenção realizada pelos nodos devido à característica dinâmica dessa rede (Subseção 4.3) .

4.1. Modelo do sistema

Assume-se que os nodos são corretos e executam o mecanismo proposto, cujo objetivo é organizar os nodos em uma topologia em malha. Existem três tipos de nodos no sistema: *bootstrap server*, origem (ou *source*) e nodos interessados em receber o conteúdo disseminado. O papel do *bootstrap* é controlar a entrada de nodos que não estavam executando a aplicação de difusão; possui uma visão global dos nodos componentes deste sistema. O nodo origem é responsável por gerar e disseminar o conteúdo, que se encontra organizado em blocos de tamanho fixo. Os demais participantes são homogêneos em relação à largura de banda disponibilizada para a aplicação; portanto a quantidade de vizinhos de cada nodo deve aproximar-se do valor médio observado na topologia.

A cada nodo presente no sistema está associado um identificador único. Um nodo mantém as seguintes estruturas: mCache (*membership cache*), que contém uma visão parcial dos nodos do sistema, informando quais desses são vizinhos, e se eles estão ativos, inativos ou em estado desconhecido (quando não há informação suficiente sobre o estado

do nodo); bMap (*block map*), indica a presença (ou não) de um determinado bloco; pendReq (*pending requests*), que registra requisições de blocos pendentes e a quais vizinhos foram solicitados, na forma de tuplas (vizinho, bloco).

4.2. Entrada de Nodos no Sistema

O Algoritmo 1 mostra o procedimento realizado por um nodo solicitante (n_s) que deseja ingresso no sistema. O nodo n_s envia uma mensagem BOOTSTRAP ao *bootstrap server* (BS), pedindo sua inclusão (linha 3). Se n_s recebe uma mensagem NEW_NODE de BS com uma mCache criada e inicializada com os endereços de um subconjunto de nodos presentes no sistema (linha 7), n_s cria um mapa de blocos bMap (linha 8) e uma lista de requisições pendentes pendReq (linha 9), ambos vazios. Para evitar espera indefinida e decidir quando deve lidar com condições de erro, n_s limita o número de tentativas e o prazo para recebimento de resposta, retransmitindo quando necessário.

Algoritmo 1 Nodo n_s solicita entrada no sistema

```
 1: tent_count ← 0;  msg ← null
 2: while tent_count ≤ tent_max ∧ msg ≠ NEW_NODE do
 3:     send (BS, BOOTSTRAP) {envia msg inicial ao bootstrap server}
 4:     msg ← RECEIVE (timeout) {msg = null se ocorre timeout}
 5:     tent_count ← tent_count + 1
 6: end while
 7: if msg = NEW_NODE then
 8:     bMap ← {} {inicializa o mapa de blocos}
 9:     pendReq ← {} {cria lista de requisições pendentes}
10:     neighb_count ← 0
11: else
12:     {falha na entrada de n_s na rede}
13: end if
```

O Algoritmo 2 apresenta o comportamento do *bootstrap server*. Quando BS recebe de n_s uma solicitação de entrada no sistema (linha 4), BS inicializa uma mCache destinada a n_s (linha 5), que fornece a ele a visão de um pequeno subconjunto de nodos. Essa mCache contém os endereços de nodos randomicamente selecionados a partir da mCache de BS, e portanto presentes no sistema (linha 14). Então, BS insere o endereço de n_s em mCacheBootstrap (linha 17) e envia uma mensagem a n_s contendo a mCache recém-criada (linha 18). Caso ocorra algum erro durante o processamento da mensagem em BS, n_s é notificado do erro (linha 22). A verificação da quantidade de nodos inseridos em sua mCache (linha 7) é necessária apenas na fase inicial de composição do sistema: se o único nodo for o origem, ou se a quantidade de vizinhos deste não tiver atingido ainda seu limite superior, então o endereço da origem é inserido na mCache de n_s.

4.3. Manutenção da Rede de Sobreposição e Solicitação de Novos Vizinhos

O Algoritmo 3 mostra o comportamento de um nodo n_p à procura de novos vizinhos. O nodo n_p seleciona de sua mCache o próximo nodo na sequência, aqui identificado como n_v (linha 3), e envia uma mensagem a n_v solicitando a conexão (linha 5). Caso n_p não receba resposta, receba uma mensagem inválida ou uma notificação de que sua requisição foi negada (linha 7), n_p deve procurar um novo nodo para ser seu vizinho. Se n_p receber uma resposta positiva de n_v com a mCache deste (linha 10), n_p deve enviar a sua mCache (linha 11).

Algoritmo 2 Funcionamento do nodo *bootstrap* (BS)

```
1:  msg ← null
2:  while true do
3:      msg ← receive ()
4:      if msg = BOOTSTRAP then
5:          mCache_n_s ← new (mCache) {cria uma mCache para o n_s}
6:          i ← 1
7:          if mCacheBootstrap.size = 1 ∨ srcNeighbs < srcNeighbs_max then
8:              mCache_n_s[i] ← Source
9:              srcNeighbs ← srcNeighbs + 1
10:             i ← i + 1
11:         else
12:             if srcNeighbs ≥ srcNeighbs_max then
13:                 while i ≤ neighb_max do
14:                     mCache_n_s[i] ← random (mCacheBootstrap.id)
15:                     i ← i + 1
16:                 end while
17:                 mCacheBootstrap.id[length] ← n_s
18:                 send (n_s, NEW_NODE) {envia msg para n_s confirmando a entrada deste}
19:             end if
20:         end if
21:         if Error then
22:             send (n_s, BOOTSTRAP_ERROR) {envia msg para n_s em caso de erro}
23:         end if
24:     end if
25: end while
```

Algoritmo 3 Nodo n_p solicita um novo vizinho n_v

```
1:  msg ← null;  j ← 1
2:  while neighb_count < neighb_max do
3:      n_v ← mCache_n_p.id[j]
4:      j ← j + 1
5:      send (n_v, REQ_NEIGHBORS) {n_p envia msg para estabelecer vizinhança com n_v}
6:      msg ← receive (timeout) {msg = null se ocorre timeout}
7:      if msg = null ∨ msg = DENIED_REQUEST ∨ msg inválida then
8:          {procura outro nodo}
9:      else
10:         if msg = MCACHE then
11:             send (n_v, MCACHE) {n_p envia sua mCache para n_v}
12:             msg ← receive (timeout)
13:             if msg = null ∨ msg inválida then
14:                 {n_p descarta a mCache recebida de n_v e procura novo vizinho}
15:             else
16:                 if msg = NEIGHB_ACCEPTED then
17:                     mCache_n_p.neighb[j − 1] ← true
18:                     pendReq[length] ← n_v
19:                     neighb_count ← neighb_count + 1
20:                     Procedimento Atualiza_mCache (mCache_n_p, mCache_n_v)
21:                 end if
22:             end if
23:         end if
24:     end if
25: end while
```

O nodo n_p então deve aguardar uma resposta do futuro vizinho (n_v) confirmando o recebimento de sua mCache. Quando receber essa resposta (linha 16), n_p insere o nodo n_v como seu vizinho, insere n_v em sua pendReq, incrementa seu número de vizinhos (linhas 17-19) e realiza o procedimento para a atualização de sua mCache (linha 20).

Ao ser executado o procedimento Atualiza_mCache (mCache_n_p, mCache_n_v) (linha 20), n_p revisa as informações da sua mCache, com base na mCache de n_v recebida de n_v. Com isso, n_p pode conhecer outros nodos da rede, além de confrontar informações sobre estado de nodos já constantes da sua mCache e com os quais n_p não tem contato direto. Estas informações são úteis para a busca de novos vizinhos. Assim, quando o nodo atinge o limiar pré-estabelecido como a quantidade mínima de vizinhos, ele executa o algoritmo para busca de novos vizinhos (Algoritmo 3) a partir de nodos ali constantes. A confirmação de estados inativos de nodos do sistema, reportados em múltiplas mCaches pode ser repassada a BS para providências. Ações de remoção nas mCaches de nodos que saíram do sistema resultam em agilidade nas atividades de manutenção da rede e da difusão de conteúdos; assim, os nodos devem revisar as informações constantes em suas mCaches para exclusão de inativos, liberando espaço para nodos adicionais.

O Algoritmo 4 apresenta as ações no estabelecimento de vizinhança do ponto de vista de (n_v), o qual é convidado por (n_p).

Algoritmo 4 Nodo n_v recebe solicitação de n_p para estabelecer vizinhança

```
1:  msg ← null;    id ← null
2:  msg ← receive ()
3:  if msg = REQ_NEIGHBORS then
4:      if neighb_count < neighb_max then
5:          send (n_p, MCACHE) {n_v envia sua mCache para nodo solicitante n_p}
6:          msg ← receive (timeout) {msg = null se ocorre timeout}
7:          if msg = null ∨ msg inválida then
8:              {descarta a requisição recebida}
9:          else
10:             if msg = MCACHE then
11:                 id ← Busca (mCache_n_v.id, n_p) {n_v busca o endereço do nodo n_p em sua mCache}
12:                 if id = null then
13:                     mCache_n_v.id[length] ← n_p
14:                     mCache_n_v.neighb[length] ← true
15:                 else
16:                     mCache_n_v.neighb[id] ← true
17:                 end if
18:                 pendReq[length] ← n_p
19:                 neighb_count ← neighb_count + 1
20:                 Procedimento Atualiza_mCache (mCache_n_v, mCache_n_p)
21:                 send (n_p, NEIGHB_ACCEPTED) {n_v envia msg confirmando que aceita vizinho}
22:             end if
23:         end if
24:     else
25:         send (n_p, DENIED_REQUEST) {n_v envia msg recusando pedido de vizinhança}
26:     end if
27: end if
```

Quando n_v recebe o pedido (linha 3), ele deve verificar quantos vizinhos possui (linha 4). Se esse valor for inferior a um limite superior de vizinhos pré-estabelecido, ele

envia mensagem a n_p com a sua mCache (linha 5) e aguarda resposta de n_p. Se n_p responder com a sua mCache (linha 10), n_v deve realizar uma busca em sua mCache (linha 11) para verificar a existência de n_p. Se não for encontrado, n_v deve inserir n_p em sua mCache (linha 13). O estado do nodo inserido (ou pré-existente) passa a ser indicado como vizinho de n_v (linhas 14 ou 16). Além disso, n_v insere n_p em pendReq (linha 18), incrementa o número de vizinhos (linha 19), realiza a atualização da sua mCache (linha 20) e envia mensagem confirmando o estabelecimento da vizinhança (linha 21). Se n_v não receber a mCache de n_p, ou receber uma mensagem distinta (linha 7), n_v descarta a solicitação recebida sem estabelecer a conexão com n_p. Caso n_v já possua a quantidade máxima de vizinhos, nega a requisição a n_p (linha 25).

5. Conclusões

Nesse artigo, foram apresentados os procedimentos relacionados à dinâmica da rede de sobreposição de um sistema que será utilizado como base para aplicações *live streaming* em redes P2P dinâmicas. Foram descritos os procedimentos de inserção de nodos no sistema e de estabelecimento de parcerias através das quais os mecanismos de difusão fazem a distribuição de conteúdos. Os próximos passos compreendem a implementação e avaliação dos algoritmos apresentados. Como o objetivo do trabalho é contribuir na viabilidade experimental em ambientes P2P voltados para aplicações *live streaming*, a disponibilização dos módulos desenvolvidos será extremamente útil na obtenção de ambientes onde a dinâmica comportamental dos nodos é uma premissa, facilitando o estudo das propriedades dos algoritmos usados na gerência dessa flutuação característica das redes P2P.

Referências

Fodor, V. and Dan, G. (2007). Resilience in live peer-to-peer streaming. *IEEE Communications Magazine*, 45(6):116 – 123.

Jurca, D., Chakareski, J., Wagner, J.-P., and Frossard, P. (2007). Enabling adaptive video streaming in p2p systems. *IEEE Communications Magazine*, 45(6):108 – 114.

Li, B. and Yin, H. (2007). Peer-to-peer live video streaming on the internet: Issues, existing approaches, and challenges. *IEEE Communications Magazine*, 45(6):94 – 99.

Liao, X., Jin, H., Liu, Y., Ni, L. M., and Deng, D. (2006). Anysee: Peer-to-peer live streaming. pages 1 – 10. 25th IEEE Intl. Conf. Computer Comm., INFOCOM.

Liu, J., Rao, S. G., Li, B., and Zhang, H. (2008a). Opportunities and challenges of peer-to-peer internet video broadcast. volume 96, pages 11 – 24. Proceedings of the IEEE.

Liu, Y., Guo, Y., and Liang, C. (2008b). A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 1(1):18 – 28.

Magharei, N. and Rejaie, R. (2007). Prime: Peer-to-peer receiver-driven mesh-based streaming. pages 1415 – 1423. IEEE Intl. Conf. Computer Comm., INFOCOM.

Pai, V., Kumar, K., Tamilmani, K., Sambamurthy, V., and Mohr, A. E. (2005). Chainsaw: Eliminating trees from overlay multicast. In *Proc. The 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, Ithaca, NY, USA.

Xie, S., Li, B., Keung, G. Y., and Zhang, X. (2007). Coolstreaming: Design, theory, and practice. *IEEE Transactions on Multimedia*, 9(8):1661 – 1671.