SAMUEL LUCAS VAZ DE MELLO

UMA ABORDAGEM PARA DISTRIBUIÇÃO DE FLUXOS DE CONTEÚDO MULTIMÍDIA AUXILIADA POR REDES *PEER-TO-PEER*

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná. Orientador: Prof. Elias Procópio Duarte Jr.

CURITIBA
2007

Sumário

Resumo

A distribuição de conteúdo é uma aplicação típica de redes de computadores. Para uma quantidade grande de usuários, a concentração do tráfego no ponto fornecedor do conteúdo pode tornar-se um fator limitante para a escalabilidade do sistema. Diversos sistemas utilizam redes *peer-to-peer* para distribuição de arquivos na Internet, evitando a concentração do tráfego em um único ponto e melhorando assim a escalabilidade do sistema. Este trabalho propõe um sistema que utiliza redes *peer-to-peer* para auxiliar a distribuição de fluxos de dados, como transmissões de áudio e vídeo ao vivo. Os diversos usuários estabelecem periodicamente acordos de encaminhamento de partes do fluxo entre si. O sistema foi implementado e experimentos demonstram significativa redução no tráfego de dados na origem do conteúdo, bem como o comportamento do sistema na presença de falhas.

Abstract

Content distribution is one of the key applications of computer networks. For a large number of users, the amount of bandwith required at the content source can become a bottleneck for the system's scalability. An approach to deal with this problem is to use peer-to-peer networks for distributing files on the Internet, avoiding network traffic concentration and improving the system's scalability. This paper presents a system that uses peer-to-peer technology to assist the distribution of streaming data, such as live audio and video. Peers periodically negotiate deals for forwarding parts of the streaming data. The system has been implemented and experimental results show an expressive reduction in the amount of network traffic at content source as well as the system behavior in the presence of peer crashes.

Capítulo 1

Introdução

A distribuição de um determinado conteúdo, como um arquivo ou fluxo de dados, para uma quantidade possivelmente grande de clientes é uma aplicação típica de redes de longa distância como WANs (*Wide Area Network*) corporativas ou a Internet. Como exemplo podemos citar a distribuição de arquivos e as transmissões de áudio e vídeo on-line. O modelo cliente-servidor é tradicionalmente a solução mais comum para este tipo de aplicação. Porém com o advento das redes *peer-to-peer* [11] novas abordagens têm sido propostas para aumentar a eficiência e flexibilidade destes sistemas.

No modelo cliente-servidor, o conteúdo é disponibilizado em um servidor de onde é transferido para os clientes. Todas estas transferências passam necessariamente pelo servidor, o que causa a concentração do custo de *upload* e é um possível fator limitante de desempenho. Esses fatores são especial-

mente indesejados quando o conteúdo compreende uma grande quantidade de dados.

Exemplos típicos incluem a Web, arquivos disponibilizados por FTP (*File Transfer Protocol*) e os sistemas comuns de transmissão de áudio e vídeo on-line [6, 7]. O modelo cliente-servidor é exemplificado na figura 1.1 que destaca que um único servidor é responsável por atender individualmente a cada um dos clientes.

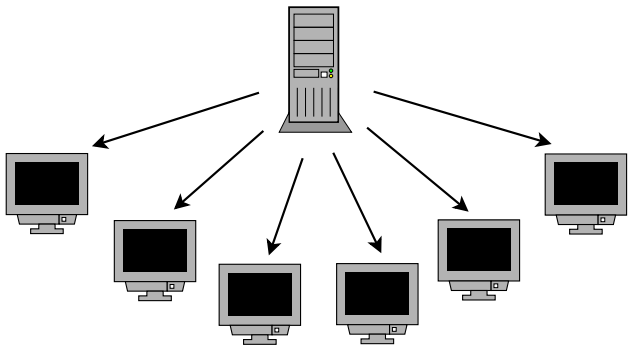

Figura 1.1: Modelo cliente-servidor de distribuição de conteúdo.

Como cada cliente é atendido individualmente pelo servidor, existe um paralelismo que pode ser utilizado para aumentar a eficiência do processo como um todo. Por exemplo, se todos os clientes estão obtendo um arquivo grande a uma taxa relativamente pequena, a transferência demora um tempo relativamente longo. Neste período, o número de clientes que já obtiveram pelo menos uma parte do conteúdo aumenta mais rapidamente do que o número de clientes que já obtiveram todo o conteúdo. Então, se os clientes

receberem partes diferentes do arquivo eles podem formar uma rede *peer-to-peer* e trocar partes recebidas entre si diminuindo o custo de *upload* no servidor e aumentando o grau de paralelismo na transferência para os clientes. Exemplos de aplicações que utilizam este tipo de abordagem, como o BitTorrent [1] e o SplitStream [2], são apresentadas no capítulo 2. O modelo é ilustrado na figura 1.2.

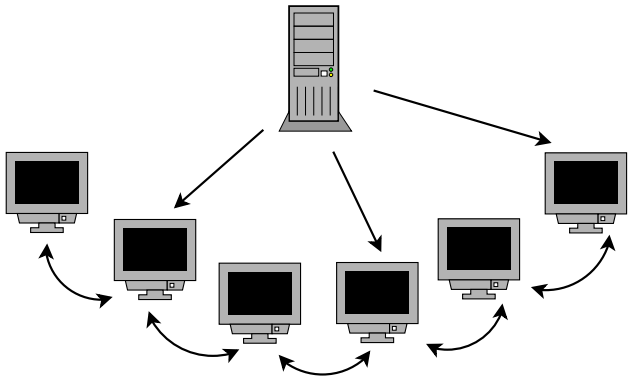

Figura 1.2: Estrutura de distribuição de conteúdo utilizando redes *peer-to-peer*.

As aplicações que mais se beneficiam da estrutura que utiliza redes *peer-to-peer* são aquelas em que o conteúdo representa um grande volume de dados, especialmente a distribuição de grandes arquivos e fluxos multimídia, como transmissão de áudio e vídeo. Entretanto, estes dois tipos de aplicações apresentam diferenças significativas no processo de distribuição do conteúdo. Na distribuição de arquivos todo o conteúdo está disponível no início do processo e o seu tamanho é fixo e conhecido. Por outro lado, na distribuição

de fluxos multimídia o conteúdo é produzido no decorrer do tempo por uma aplicação servidora e armazenado em buffers que são transmitidos para os clientes. É necessário que os dados desse buffer estejam disponíveis no cliente de forma ordenada dentro de um certo limite de tempo, porque a aplicação que utiliza esses dados consome o buffer de forma ordenada e a uma taxa constante.

Na distribuição de arquivos, como o conteúdo está disponível em sua totalidade no início da distribuição, clientes que começam a obter um arquivo ao mesmo tempo podem receber partes diferentes do arquivo e posteriormente trocar as diferentes partes recebidas entre si de forma a recompor o arquivo original. Na transmissão de fluxos multimídia, como o conteúdo é gerado no decorrer do tempo, a diversidade de partes obtidas por clientes que iniciam a transferência ao mesmo tempo é muito menor, limitando-se às partes de um buffer relativamente pequeno.

Outra diferença entre fluxos multimídia e transferência de arquivos é que os dados dos fluxos multimídia não têm um final conhecido previamente, o que faz com que a origem dos dados continue necessariamente fornecendo dados durante todo o período em que o fluxo existir.

Para a distribuição de arquivos, a origem pode transmitir o arquivo completo uma única vez e já pode inclusive encerrar seu processamento, uma vez que os clientes que já obtiveram partes do arquivo continuam trocando par-

tes entre si, atendendo também a clientes novos que venham a se conectar. Como os fluxos multimídia não possuem um final conhecido previamente, a origem do fluxo precisa estar continuamente gerando e transmitindo novos dados para o clientes.

Este trabalho apresenta um sistema de transmissão de fluxos multimídia que utiliza uma estrutura *peer-to-peer* para realizar a distribuição do conteúdo. Um servidor é responsável por criar o conteúdo, armazená-lo em um buffer, dividir o buffer em partes e transmitir estas partes a um grupo inicial de nós. Estes nós interagem entre si para estabelecer acordos para trocar as partes do buffer recebidas. Os acordos de encaminhamento são renegociados de tempos em tempos, adaptando-se a mudanças no estado da rede. Como os clientes precisam obter o buffer completo dentro de um limite de tempo e os nós que encaminham partes podem falhar, um mecanismo é responsável por solicitar diretamente ao servidor as partes que não forem recebidas até um determinado limiar de tempo. Foram realizados experimentos que mostram o impacto do uso do sistema sobre a quantidade de dados enviados e o comportamento na presença de falhas.

O restante desta dissertação está organizado da seguinte forma. O capítulo 2 apresenta sistemas para distribuição de conteúdo em redes de longa distância. O capítulo 3 descreve com detalhes o sistema proposto. O capítulo 4 apresenta a implementação, os experimentos realizados e os resultados obtidos.

O capítulo 5 conclui o trabalho.

Capítulo 2

Distribuição de Conteúdo em Redes de Longa Distância

Este capítulo apresenta soluções para a distribuição de conteúdo utilizando redes *peer-to-peer*. Na seção 2.1 é apresentado o BitTorrent, um sistema para distribuição de arquivos na Internet. Na seção 2.2 é apresentado o SplitStream, um sistema para distribuição de fluxos de dados e a seção 2.3 apresenta outros trabalhos relacionados.

2.1 Distribuição de Arquivos com o BitTorrent

BitTorrent [1] é um sistema para distribuição colaborativa de arquivos bastante popular. Ele é freqüentemente utilizado para a distribuição de arquivos grandes, como imagens de discos de instalação de sistemas operacionais, que costumam utilizar diversos gigabytes e gerariam um custo de *upload* para o servidor bastante alto se disponibilizados no modelo tradicional.

O BitTorrent tornou-se bastante popular e utilizado em larga escala por ser o primeiro sistema a prover uma boa solução para os problemas logísticos e de robustez envolvidos. O arquivo é disponibilizado em uma rede *peer-to-peer* em que os nós trocam partes do arquivo já recebidas entre si até conseguirem recompor o arquivo original. A simples tarefa de determinar qual nó possui qual parte do arquivo e qual deve enviar qual parte para algum outro nó é difícil de realizar sem um grande *overhead*.

O processo de distribuição do arquivo usando BitTorrent inicia com a disponibilização em um servidor HTTP (*Hypertext Transfer Protocol*) de um arquivo com a extensão `.torrent` contendo algumas informações sobre o arquivo de conteúdo a ser distribuído. Essas informações incluem o tamanho, o nome, o código de verificação (*hash*) de cada parte do arquivo e o endereço de um *tracker*, processo que mantém uma lista de clientes atualmente

transferindo um determinado arquivo.

Após obterem o arquivo com extensão `.torrent`, os clientes conectam-se ao *tracker* para obterem uma lista de outros clientes que também estão transferindo o mesmo arquivo. O *tracker* tem como função apenas auxiliar os nós a acharem uns aos outros. Ele utiliza um protocolo bastante simples construído sobre HTTP em que cada nó informa qual arquivo está obtendo e em que porta está ouvindo e o *tracker* responde com uma lista de outros nós que também estão obtendo o mesmo arquivo.

O servidor e os clientes que estão obtendo o arquivo passam a formar uma rede ponto a ponto em que os nós trocam partes do arquivo entre si. Todos os problemas logísticos da transferência do arquivo são resolvidos através da interação entre os nós. Algumas informações sobre as taxas de *upload* e *download* são enviadas para o *tracker* apenas para efeitos estatísticos. Como as conexões entre um determinado nó e outros nós são estabelecidas a partir da lista retornada pelo *tracker*, o BitTorrent determina que o *tracker* retorne uma lista de nós escolhidos aleatoriamente, fazendo com que a rede forme um grafo aleatório que é conhecidamente robusto [1].

O arquivo é dividido em partes de tamanho fixo, tipicamente de 250KB, e o código de verificação de cada parte está contido no arquivo com extensão `.torrent`. Após um nó obter uma parte de um arquivo, ele primeiro calcula o código de verificação, compara com o contido no arquivo .torrent e, caso a

verificação seja positiva, disponibiliza essa parte para os outros nós da rede.

Cada nó é responsável por tentar maximizar sua taxa de *download*. Como as transferências são feitas utilizando TCP (*Transmission Control Protocol*), o BitTorrent procura manter diversas requisições pendentes ao mesmo tempo para evitar atrasos entre as transmissões e o recebimento dos reconhecimentos de entrega do TCP. Para evitar este atraso, o BitTorrent divide cada parte em pedaços menores, tipicamente de 16KB, e mantém um certo número requisições pendentes em paralelo. Sempre que um pedaço é recebido, outro é solicitado.

Na decisão de qual será a próxima parte do arquivo a ser pedida para outro nó, prioriza-se a parte menos freqüente na rede, isto é, a que está sendo disponibilizada pelo menor número de outros nós. Esta técnica procura fazer com que os nós possuam partes que sejam do interesse de seus vizinhos e possam transferi-las quando solicitados. Ela também replica rapidamente partes raras na rede, diminuindo a probabilidade de que uma parte disponibilizada apenas por um pequeno número de nós deixe de estar disponível na rede devido à eventual desconexão destes poucos nós que a disponibilizavam. A única exceção a esta regra ocorre no início da distribuição, em que a escolha é feita aleatoriamente, já que todas as partes são igualmente raras.

Outra característica desse tipo de distribuição de arquivos é que cada nó está mais interessado em obter partes do que em fornecer partes. Porém,

por restrição matemática, a quantidade total de partes obtidas na rede deve ser igual à quantidade total de partes fornecidas na rede. Para incentivar os nós a fornecerem partes, o BitTorrent associa as taxas de *upload* e *download*, como descrito abaixo.

Os nós obtém paralelamente partes de tantos nós quanto for possível e sempre fornecem partes para um número limitado de nós. A decisão sobre para quais nós fornecer é baseada na taxa de *download* atual, ou seja, os nós de quem se obtém as maiores taxas de *download* são priorizados na decisão. Para o cálculo da taxa de *download* atual são considerados os dados recebidos nos últimos 20 segundos. O BitTorrent reavalia os nós para os quais fornece partes a cada dez segundos. Este período é suficientemente pequeno para refletir rapidamente mudanças ocorridas na situação da rede e grande o suficiente para que o TCP utilize toda a capacidade disponível entre os dois nós envolvidos na transferência. A cada três períodos de dez segundos cada nó fornece uma parte para um nó escolhido aleatoriamente, independente da taxa de *download* obtida a partir dele, com o objetivo de tentar encontrar nós dos quais possa-se obter taxas mais vantajosas tanto de *upload* quanto de *download*.

Os nós continuam trocando partes até que o arquivo todo tenha sido transferido. Após um nó terminar de obter o arquivo completo, ele continua fornecendo partes até que o BitTorrent seja fechado.

2.2 Distribuição de Fluxos de Dados com o SplitStream

A abordagem proposta pelo SplitStream [2] procura dividir a tarefa da distribuição de conteúdo entre os nós de forma equalitária. Ele utiliza o *framework* de redes ponto a ponto Pastry [3] e o sistema de comunicação coletiva Scribe [4], construído sobre o Pastry. Estes sistemas são descritos a seguir.

O Pastry é um *framework* que cria uma rede *peer-to-peer* escalável, estruturada e auto-organizada. Nele, cada nó ou objeto recebe um identificador chamado *nodeId* ou chave, de 128 bits que pode ser visto como uma seqüencia de dígitos em base 2^b, onde b é um parâmetro configurável, tipicamente 4. Dada uma mensagem e uma chave de destino, o Pastry roteia a mensagem para o nó cujo *nodeId* é numericamente mais próximo da chave.

Para ser capaz de rotear as mensagens, cada nó mantém uma tabela de roteamento e um conjunto de vizinhos. A tabela de roteamento tem tipicamente $log_{2^b}N$ linhas e 2^b colunas. A entrada na i-ésima linha da tabela se refere a nós que têm os i primeiros dígitos do *nodeId* igual ao nó local. O $(i+1)$-ésimo dígito do *nodeId* de um nó na linha i e coluna c é c. Assim, para cada posição da tabela de roteamento podem existir vários nós que poderiam ocupar a posição. É escolhido o nó com menor tempo de resposta entre eles.

Para rotear uma mensagem, cada nó inicialmente verifica se o destino

pertence ao seu conjunto de vizinhos. Em caso negativo, a mensagem é encaminhada para o nó numericamente mais próximo do destino da mensagem na tabela de roteamento. O número de passos esperado para a entrega da mensagem é de menos de $log_{2^b} N$.

A figura 2.1 ilustra o roteamento de uma mensagem do nó com *nodeId 65a1fc* para a chave *d46a1c*. Os pontos representam os *nodeId* de nós ativos no espaço de identificadores do Pastry. A mensagem é enviada pelo nó *65a1fc* para o nó mais próximo da chave de destino contido em sua tabela de roteamento, *d13da3*. De forma semelhante o nó *d13da3* encaminha a mensagem para o nó *d4213f* que entrega a mensagem ao destino, o nó *d462ba* que é o nó numericamente mais próximo da chave de destino em toda a rede. Note que a cada passo do roteamento uma quantidade crescente de dígitos é igual entre o identificador do nó e a chave de destino.

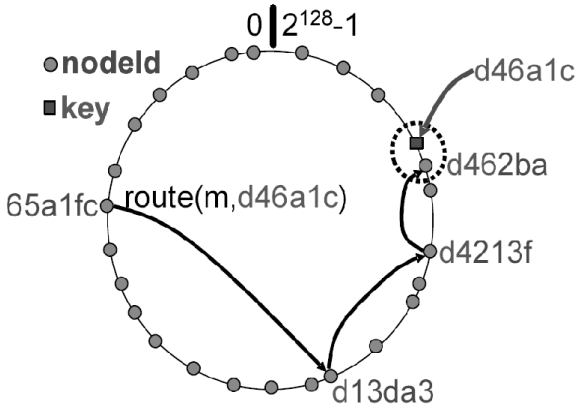

Figura 2.1: Roteamento de uma mensagem no Pastry [3].

O Scribe [4] é um sistema que cria uma estrutura para comunicação coletiva em uma rede *peer-to-peer*. utilizando o roteamento de mensagens provido pelo Pastry. Ele permite que os nós criem grupos, se associem a grupos existentes ou deixem grupos a que pertençam. Quando um nó envia uma mensagem para um grupo, ela é entregue para todos os nós pertencentes ao grupo.

Cada grupo possui uma chave, que é um identificador válido no espaço de identificação do Pastry. O grupo é estruturado como uma árvore formada pelas rotas de todos os nós pertencentes ao grupo até a chave do grupo. Isso cria uma árvore de encaminhamento de mensagens com raiz no nó com identificador numericamente mais próximo da chave do grupo. Os nós pertencentes a esta árvore de encaminhamento de mensagens são chamados de encaminhadores e podem ou não pertencerem ao grupo. Cada nó encaminhador mantém uma lista de seus filhos na árvore de encaminhamento de mensagens. Quando algum nó deseja enviar uma mensagem para todo o grupo, ele envia a mensagem para a raiz da árvore, que inicia o processo de encaminhamento.

Quando algum nó deseja associar-se a um grupo, ele envia uma mensagem para a chave do grupo. O Pastry roteia a mensagem até o próximo nó no caminho até a chave, onde é entregue para função de encaminhamento do Scribe. Esta função verifica se o nó é um encaminhador e em caso positivo

adiciona o novo nó na sua tabela de filhos. Caso o nó não seja encaminhador ele passa a ser, e envia uma mensagem para o próximo nó no caminho até a raiz da árvore.

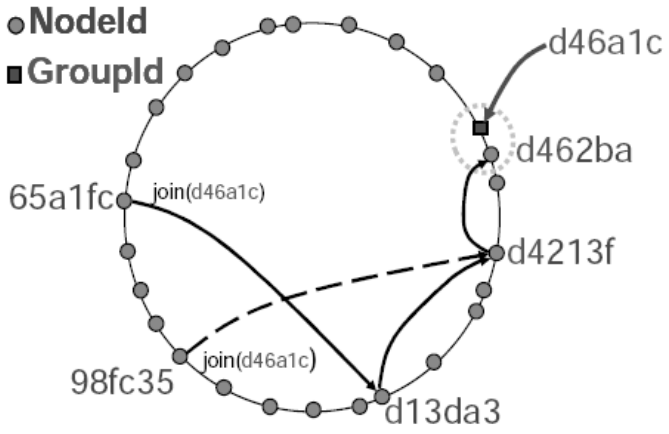

Figura 2.2: Exemplo de formação de árvore Scribe [5].

A figura 2.2 demonstra como as árvores são construídas usando como exemplo uma árvore com identificador *d46a1c*. O nó com *nodeId d46ba* é a raiz da árvore porque ele possui o *nodeId* numericamente mais próximo ao identificador da árvore. A linha contínua representa a solicitação para entrada no grupo do nó *65a1fc*. O Pastry roteia a solicitação para o nó *d13da3* que torna-se um encaminhador, adiciona *65a1fc* com seu filho na árvore e reenvia a solicitação de entrada no grupo. O nó *d4213f* recebe a solicitação, torna-se encaminhador, adiciona *d13da3* como seu filho e encaminha a solicitação de entrada no grupo para *d462ba*. Ao receber a solicitação *d462ba*,

que é a raiz da árvore, adiciona *d13da3* como filho. Após isto, o nó *98fc35* envia também uma solicitação de entrada no grupo, que está representada pela linha pontilhada. O Pastry roteia a solicitação para o nó *d4213f* que, por já ser um encaminhador, apenas adiciona *98fc35* como filho. Uma descrição completa do algoritmo de gerenciamento de grupo do Scribe pode ser encontrado em [4].

O SplitStream [2] utiliza a estrutura das árvores de encaminhamento de mensagens fornecida pelo Scribe para realizar distribuição eficiente de conteúdo. A quantidade de nós que são folha em uma árvore é significativamente maior do que o número de nós internos na árvore. Essa proporção é de mais de 50% de folhas para árvores binárias e mais de 90% em árvores em que cada nó tem 16 filhos. Como nas árvores de encaminhamento de mensagens apenas os nós internos encaminham mensagens, isso causa a concentração do custo de encaminhamento em poucos nós.

O SplitStream procura distribuir este custo de encaminhamento de forma mais equilibrada entre os nós dividindo o conteúdo em partes que são encaminhadas através de árvores que não possuam nós internos em comum, ou seja, cada nó só é um nó interno em uma das árvores sendo folha em todas as outras.

A figura 2.3 exemplifica a estrutura criada pelo SplitStream para a distribuição de conteúdo com origem no nó com número 1. A figura apresenta

duas árvores, uma representada pelas linhas contínuas e a outra pelas linhas pontilhadas. O conteúdo original é divido em duas partes e cada parte transmitida por uma árvore.

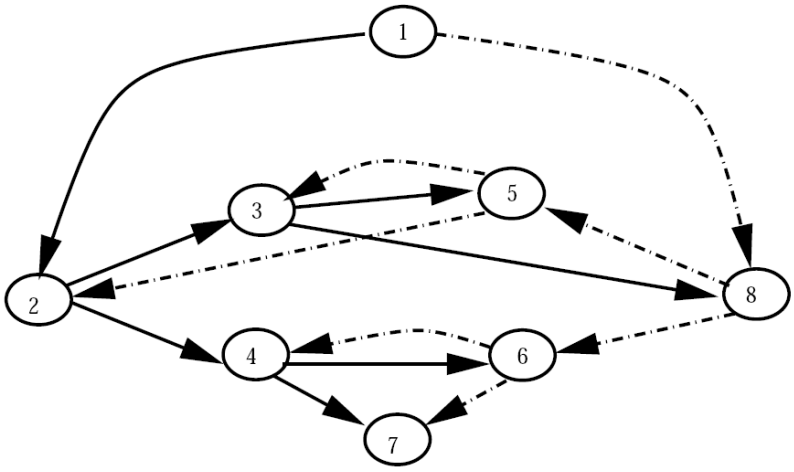

Figura 2.3: Árvores de encaminhamento de mensagens criada pelo SplitStream [2].

A construção das árvores sem nós interiores em comum é feita aproveitando-se as propriedades de roteamento do Pastry. Normalmente o Pastry encaminha uma mensagem através de nós cujos identificadores possuem progressivamente mais dígitos em comum com a chave da mensagem. As árvores criadas pelo Scribe são construídas utilizando-se as rotas até a chave do grupo, assim os nós internos da árvore possuem um certo número de dígitos em comum. Desde modo, escolhendo-se chaves para os grupos com uma quantidade su-

ficiente de dígitos mais significativos diferentes obtém-se árvores sem nós internos comuns.

Embora o SplitStream apresente uma solução eficiente para a distribuição de fluxos de dados em redes *peer-to-peer* notamos que o desempenho desta solução na Internet pode não ser satisfatório. A abordagem do SplitStream está baseada na distribuição equalitária do custo de transmissão entre os nós. Porém isso muitas vezes não é desejado na Internet, já que a capacidade de transmissão de cada nó é diferente. Considere o caso de um grupo formado por diversos nós que possuam conexão rápida e um único nó que possua conexão significativamente lenta, por exemplo devido à restrição de velocidade da linha com que o nó está conectado à Internet. Alguma das árvores de distribuição do SplitStream pode conter o nó com conexão lenta como nó interno. Por sua conexão ser lenta, o tempo necessário para realizar a transmissão dos dados para seus filhos é significativamente maior do que o tempo utilizado por um nó com conexão rápida. Deste modo, a distribuição do conteúdo das outras árvores, que possuem apenas nós internos com conexão rápida, terminará antes desta e todos os nós descendentes do nó lento precisam esperar até que ele termine de encaminhar os dados. Assim, para uma quantidade possivelmente grande de nós, o tempo total necessário para a transmissão aumenta significativamente devido a um único nó com conexão lenta.

2.3 Outros Trabalhos Relacionados

Entre outros trabalhos relacionados com a distribuição de conteúdo na Internet podemos citar o próprio IP (*Internet Protocol*) Multicast [8]; trabalhos mais próximos incluem Avalanche [10], Overcast [9], PeerCast [18], ZIGZAG [19] e sistemas de compartilhamento de arquivos em redes *peer-to-peer*, descritos abaixo.

IP Multicast [8] foi desenvolvido com o objetivo de prover comunicação coletiva de forma eficiente como um recurso primitivo de uma rede. Ele permite que qualquer nó comunique-se com todos os outros nós do grupo enviando os dados apenas uma vez. Ele utiliza uma faixa especial de endereços IP para os grupos e necessita de suporte dos equipamentos envolvidos na comunicação. A limitação do espaço de endereçamento, a falta de suporte a Multicast nos equipamentos de rede e a ausência de um mecanismo para controle de acesso e faturamento dificultaram a popularização do Multicast, especialmente na Internet.

Avalanche [10] utiliza uma estratégia bastante parecida com a do BitTorrent. A principal diferença é a utilização de *Network Coding* para codificar os dados trocados entre os nós. Ao invés dos nós trocarem partes do arquivo original, cada nó cria um conjunto de dados codificado a partir das partes já obtidas e envia este conjunto de dados junto com uma lista das partes

utilizadas na codificação. Após receber uma certa quantidade de dados codificados o nó consegue reconstituir o arquivo original. A utilização de banda é equivalente à transmissão simples das partes do arquivo original, porém o uso de *Network Coding* pode aumentar a diversidade dos dados disponíveis na rede. Maiores detalhes podem ser obtidos em [10].

Overcast [9] é um sistema de *multicast* em nível de aplicação para distribuição de conteúdo originado em um único ponto. Ele utiliza protocolos próprios para criar e manter árvores de distribuição que disponibilizam dados para os clientes utilizando protocolo HTTP. Esta árvore é formada por nós responsáveis por fazer o encaminhamento dos pacotes, que não são necessariamente clientes que estejam obtendo o conteúdo. Assim, clientes pré-existentes podem acessar o conteúdo sem a necessidade de modificações. Para obter o conteúdo desejado, o cliente faz a solicitação à raiz da árvore, que redireciona a solicitação HTTP para um nó pertencente a árvore que atenderá a solicitação. O fato de não serem necessárias modificações nos clientes facilita a instalação na infraestrutura já existente da Internet.

PeerCast [18] é um programa que utiliza uma uma única árvore *multicast* para distribuir conteúdo entre nós conectados. A abordagem utilizada para determinar o posicionamento de cada nó na árvore é bastante simples. Quando um novo nó deseja integrar-se à arvore, ele envia uma solicitação de inclusão para algum nó que, possuindo disponibilidade de recursos para

atendê-lo, adiciona-o como filho na árvore. Caso o nó não possua recursos disponíveis, ele redireciona a solicitação para um de seus filhos. Esta abordagem pode levar a construção de árvores desbalanceadas.

ZIGZAG [19] é um sistema de distribuição de fluxos de conteúdo que utiliza uma rede *peer-to-peer* organizada hierarquicamente para efetuar a transmissão do fluxo. Propriedades do algoritmo utilizado garantem que a árvore de distribuição possua altura máxima $O(log_k N)$.

Existe uma grande quantidade de outros sistemas *peer-to-peer* [11] relacionados à distribuição de conteúdo. Alguns propõe infraestruturas genéricas para redes *peer-to-peer* fornecendo roteamento de informações na rede, como o Chord [13], CAN can, Pastry [3] e Tapestry [14]. Outros provêm soluções para a troca de arquivos entre os nós juntamente com mecanismos para a publicação e busca de conteúdo, como o Gnutella [15], Kazaa [16] e Napster [17].

Outros sistemas de distribuição de fluxos de conteúdo incluem NICE [21], Bayeux [20], CoopNet [22], CollectCast [23], GnuStream [24] e DONet [25].

Capítulo 3

O Sistema Proposto

A distribuição de um fluxo de conteúdo multimídia para uma quantidade potencialmente grande de clientes requer a transferência de grandes volumes de dados através da rede. A utilização de uma rede *peer-to-peer* para auxiliar na distribuição do fluxo evita que todo o volume de dados precise obrigatoriamente passar pelo servidor, que tornar-se-ia ponto de contensão de desempenho e limitaria a quantidade de clientes atendidos. Ao utilizar a capacidade de transmissão dos clientes que estão recebendo o fluxo para encaminhar os dados para outros clientes, a demanda por capacidade de transmissão no servidor diminui.

Uma grande quantidade de aplicações pode beneficiar-se dessa diminuição da demanda por capacidade de transmissão no servidor. Como exemplos podemos citar a transmissão de um canal de televisão pela Internet, a exibição

de um vídeo em vários computadores de um laboratório ou a transmissão de um pronunciamento da direção para as diversas filiais de uma empresa. As características das redes utilizadas para a transmissão do fluxo podem ser bastante diversas, porém em todos os casos a diminuição da demanda de transmissão no servidor pode proporcionar vantagens na escalabilidade.

O sistema proposto a seguir apresenta como principal vantagem a diminuição da demanda de transmissão no servidor. Ele é um sistema híbrido cliente-servidor e *peer-to-peer* porque possui um servidor ao qual todos os clientes estão conectados. A tarefa de produção do fluxo multimídia cabe exclusivamente ao servidor porém a distribuição é compartilhada entre o servidor e os clientes.

3.1 Descrição do Sistema Proposto

O sistema proposto é composto por um servidor e um conjunto de clientes. O servidor é responsável por produzir o fluxo de dados que será transmitido para os clientes e por ajudar os clientes a encontrarem uns aos outros, além de transmitir uma quantidade suficiente de cópias do fluxo para os clientes de forma que estes possam trocar as partes recebidas entre si e obter o fluxo completo.

Os clientes são responsáveis por obter e executar o fluxo multimídia para

o usuário e servirem partes do fluxo para outros clientes, formando uma rede *peer-to-peer*. Cada cliente conecta-se a um grupo de outros clientes para tentar obter partes do fluxo e fornecer partes que possua. Caso o cliente não consiga, por algum motivo, obter alguma parte do fluxo de outros nós o servidor fornecerá essa parte, evitando que o usuário perceba interrupções na execução do fluxo multimídia.

O fluxo multimídia é dividido em *fatias* de tamanho fixo e numeradas seqüencialmente. As fatias são produzidas e consumidas à uma taxa constante. Após produzir cada fatia do fluxo, o servidor divide esta em *blocos* de mesmo tamanho que são enviados para um determinado numero de clientes. Cada bloco da fatia também é numerado através de um índice que indica sua posição dentro da fatia.

Os clientes encaminham os blocos recebidos para outros nós cumprindo acordos de encaminhamento estabelecidos. Cada acordo de encaminhamento é referente a blocos com um mesmo número de bloco e é válido por uma determinada quantidade de fatias do fluxo, sendo renegociados após este período. Todos os acordos de encaminhamento estabelecidos no sistema possuem a mesma duração e iniciam em fatias cujo número seqüencial é múltiplo da duração. Caso algum cliente não consiga, por qualquer motivo, estabelecer acordos para receber alguma parte do fluxo, ele pode estabelecer acordos diretamente com o servidor.

A figura 3.1 mostra o fluxo dos dados para a transmissão de uma fatia do fluxo. No servidor, o módulo produtor de mídia cria o fluxo e divide-o em fatias (1). Cada fatia recebe uma numeração seqüencial e é então dividida em uma certa quantidade de blocos (2). No exemplo da figura, cada fatia é dividida em quatro blocos. Os blocos são transmitidos pela rede até chegarem ao cliente (3). Esta transmissão pode ocorrer diretamente entre o servidor e o cliente ou utilizando a rede *peer-to-peer* formada pelos demais clientes. O cliente reordena os blocos recebidos reconstruindo a fatia do fluxo (4) que é então executada para o usuário pelo módulo consumidor de mídia (5).

Os mecanismos de cada componente do sistema serão descritos com detalhes a seguir.

3.1.1 Servidor

O servidor realiza tarefas de produção de mídia, gerenciamento de clientes e encaminhamento inicial dos blocos para alguns clientes.

O módulo produtor de mídia cria as fatias do fluxo a serem enviadas para os clientes. A taxa em que as fatias são criadas e o tamanho de cada fatia são parâmetros do algoritmo. As fatias criadas pelo produtor de mídia são divididas em blocos que são enviados aos clientes e em seguida armazenadas em um buffer para atender eventuais requisições de reenvio de blocos recebi-

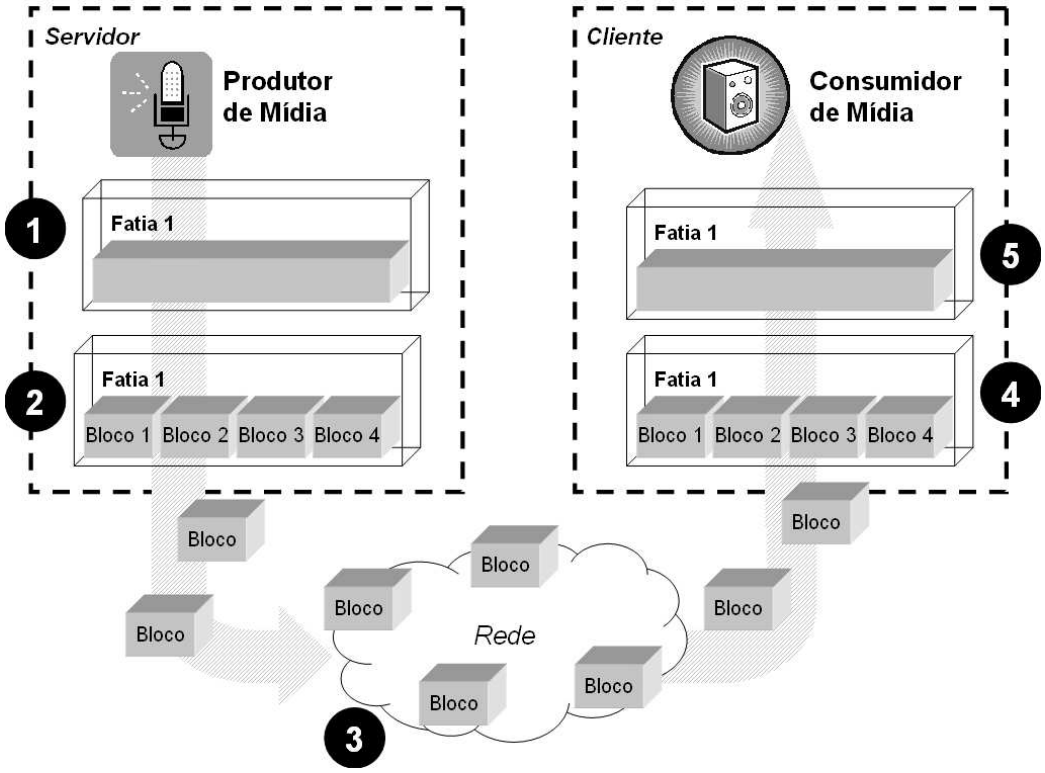

Figura 3.1: Fluxo dos dados

das dos clientes. Este buffer mantém as cópias dos blocos enviados por, pelo menos, a duração de um acordo de encaminhamento.

Quando um cliente inicia, ele conecta-se ao servidor. Ao receber a conexão, o servidor adiciona o novo cliente a uma lista de clientes ativos e envia para o cliente dados sobre o fluxo que é transmitido. Estes dados são usados pelo cliente para configurar seu módulo consumidor de mídia e incluem, por exemplo, a taxa em que as fatias são geradas e o tamanho de cada fatia.

A lista de clientes conectados é usada para ajudar os clientes a encontrarem uns aos outros. Os clientes enviam para o servidor requisições de informações sobre outros clientes na rede e o servidor responde com um determinado número de clientes da lista de clientes conectados escolhidos aleatoriamente.

O servidor mantém uma lista de acordos de encaminhamento de blocos. Após a criação de uma nova fatia do fluxo, o servidor encaminha os blocos para os clientes nesta lista. Os acordos são criados atendendo às solicitações dos clientes ou espontaneamente pelo servidor. Em ambos os casos os clientes recebem notificações de que os acordos foram criados.

Os clientes requisitam acordos de encaminhamento com o servidor quando não conseguem estabelecer acordos com nenhum outro cliente. O servidor cria acordos de encaminhamento espontaneamente em dois casos. O primeiro é quando um cliente acaba de conectar-se e ainda precisará de algum tempo para conseguir conectar-se a outros clientes e estabelecer acordos de encaminhamento. Neste caso, o servidor espontaneamente envia o fluxo para o cliente durante o período de um acordo.

O outro caso está relacionado com a disponibilidade de blocos na rede, em que o servidor precisa enviar cada fatia do fluxo para a rede ao menos uma vez para que os clientes sejam capazes reconstruir a fatia completa para execução. Quanto mais vezes o servidor transmitir espontaneamente cada fatia para a

rede, mais fácil será para os clientes encontrarem outro cliente que possua acordo para receber os blocos desejados e que possa encaminhar. Desta forma, o servidor escolhe uma determinada fração dos clientes conectados para receber estes acordos espontâneos. Esta fração e a quantidade de cópias da fatia completa que são enviadas para estes clientes são parâmetros do algoritmo. Os clientes são escolhidos de acordo com a ordem crescente do tempo de ir-e-vir (*RTT, Round Trip Time*). Após criar espontaneamente acordos para estes nós o servidor notifica todos os nós conectados de que acordos para uma nova série de fatias já estão disponíveis na rede e os clientes podem começar a estabelecer acordos entre si.

3.1.2 Cliente

O cliente é o componente do sistema responsável por obter o fluxo da rede, apresentá-lo ao usuário e servi-lo para outros clientes.

Após iniciar, cada cliente registra-se no servidor e torna-se disponível para receber conexões de outros clientes, atuando como um *peer* da rede de distribuição. Como resposta ao registro do cliente o servidor envia informações sobre o fluxo transmitido, como a taxa em que as fatias devem ser consumidas, o tamanho de cada bloco e a quantidade de blocos que forma cada fatia.

O cliente usa estas informações para configurar o módulo consumidor de mídia. Este módulo é responsável por ordenar os blocos que foram recebidos da rede, reconstruindo as fatias do fluxo para que sejam executadas para o usuário. No início do processo o módulo consumidor permanece ocioso por um período de tempo para que um conjunto inicial de dados seja recebido da rede. Este procedimento cria um buffer que reduz os efeitos de *jitter* na aplicação. Como as fatias são consumidas a uma taxa fixa e assumimos que os clientes possuem capacidade de rede suficiente para receber os blocos a tempo, o consumidor de mídia sempre exibirá os dados para o usuário com uma defasagem aproximadamente constante em relação aos blocos que estão sendo recebidos da rede. Isso faz com que não haja uma defasagem muito grande entre as posições do fluxo que os clientes da rede estão executando em um determinado momento.

Algum tempo antes de consumir cada fatia, o módulo consumidor verifica se todos os blocos desta fatia já foram recebidos da rede. Caso algum bloco ainda não tenha sido recebido, o cliente envia uma requisição emergencial para o servidor solicitando apenas os blocos faltantes. O servidor usará o buffer de reenvio de blocos para atender esta requisição. Este é um procedimento emergencial que não deve ser executado freqüentemente.

Os clientes mantém listas dos acordos estabelecidos para recebimento e encaminhamento de blocos. Todos os acordos estabelecidos no sistema pos-

suem uma mesma duração, medida em número de fatias. A duração dos acordos é parâmetro do algoritmo e todos os acordos iniciam em fatias cuja numeração seqüencial é múltipla da duração. A fatia em que cada acordo inicia é chamada *fatia-base* do acordo. Cada acordo refere-se ao encaminhamento de todos os blocos com mesmo índice de bloco, de fatias com numeração seqüencial entre a fatia-base do acordo e o próximo múltiplo da duração dos acordos.

Por exemplo, num fluxo em que a duração dos acordos é de 10 fatias o primeiro acordo possui fatia-base 0 e refere-se às fatias de 0 a 9, o segundo possui fatia-base 10 e refere-se às fatias 10 a 19 e assim consecutivamente.

A lista de acordos de recebimento indica os acordos onde outros nós encaminham blocos para o cliente. Quando o cliente recebe uma solicitação de acordo para encaminhamento referente a um determinado índice de bloco e fatia base, ele primeiro verifica na lista de acordos de recebimento se já possui um acordo estabelecido para receber esses dados. Os clientes só se comprometem a encaminhar dados para os quais já possuam acordo de recebimento.

Para cada acordo são armazenadas informações sobre o outro nó do acordo, que pode ser outro cliente ou o servidor, o índice de bloco, a fatia-base a que o acordo se refere, uma lista com os clientes por onde os dados passam até chegar no destino e a estimativa do tempo necessário para transferir um

bloco desde o nó anterior na lista. Com os dados dessa lista é possível ter uma estimativa de quanto tempo decorre entre a disponibilização de um novo bloco pelo servidor e o recebimento pelo cliente.

Cada cliente mantém também uma lista de outros clientes conhecidos. Logo após o registro no servidor o cliente solicita informações sobre outros clientes. A quantidade de outros clientes retornada pelo servidor é parâmetro do algoritmo. Periodicamente todos os outros clientes são monitorados e durante esse monitoramento são medidos o tempo necessário para transmitir um bloco e trocadas informações sobre o tempo estimado para recebimento de cada bloco. O tempo em que esse monitoramento ocorre também é parâmetro do algoritmo.

Esta lista de outros clientes é usada durante a negociação dos acordos. Após o servidor enviar espontaneamente acordos para um conjunto de clientes, todos os clientes recebem uma notificação para iniciarem a negociação de acordos para a nova série de fatias iniciada na próxima fatia-base. Ao receber esta notificação o cliente cria uma lista de clientes para cada índice de bloco que precisa negociar acordo, ordenados pelo tempo estimado de recebimento dos blocos com aquele índice nos acordos anteriores. Além disso, é iniciado um temporizador para finalizar a negociação dos acordos.

O cliente inicia a negociação de acordo para cada bloco enviando uma solicitação para o primeiro cliente da lista construída para aquele bloco. Após

enviar a requisição de acordo para o cliente, ele é removido do inicio da lista e inserido novamente no final desta. Caso o outro cliente aceite encaminhar os blocos solicitados é encerrada a negociação para estes blocos e uma nova entrada é inserida na lista de acordos de recebimento. Se o outro cliente rejeitar a solicitação de acordo, é enviada uma requisição para o próximo cliente da lista. Os nós para os quais já se tentou estabelecer acordo são inseridos novamente no final da lista para que, caso não se consiga firmar acordo com nenhum outro cliente conhecido, sejam tentados novamente. Como os nós somente aceitam encaminhar blocos para os quais possuam acordo de recebimento, este tempo entre as tentativas pode ser suficiente para que o outro cliente consiga estabelecer um acordo para o bloco desejado.

Este processo se repete até que o temporizador para finalizar a negociação expire. Neste momento o cliente verifica se foram estabelecidos acordos para todos os blocos. Caso não tenha sido possível estabelecer acordo para algum bloco uma requisição de acordo é enviada para o servidor.

Além de estabelecer o acordo com o servidor, o cliente também solicita informações sobre outros clientes, já que com os clientes aos quais se está conectado não foi possível receber todas as partes do fluxo. Assim, nas próximas negociações existirá um número maior de outros clientes para se tentar estabelecer acordo, aumentando a probabilidade de sucesso. Esta medida também beneficia indiretamente os outros clientes que estão conectados

a este porque terão maior probabilidade de sucesso no estabelecimento de acordos para receber dados a partir deste cliente.

Além da lista de acordos de recebimento, os clientes mantém também uma lista de acordos de encaminhamento, que indica para quais outros clientes cada bloco recebido deve ser encaminhado.

Ao receber uma requisição de acordo, o cliente verifica inicialmente se possui acordo de recebimento para os dados solicitados. Caso possua, o tamanho da lista de acordos de encaminhamento é verificada. Cada cliente deve manter pelo menos um determinado número de acordos de encaminhamento. Se a lista de acordos de encaminhamento for menor do que este número o novo acordo é aceito. Caso contrário é rejeitado. O número de acordos de encaminhamento que cada cliente precisa manter é parâmetro do algoritmo.

3.2 Diagramas de Seqüência

Esta sessão apresenta diagramas de seqüência do sistema proposto com o objetivo de exemplificar o funcionamento deste. São apresentados dois diagramas. O primeiro exemplifica a inicialização de um cliente e o segundo a negociação de acordos.

O diagrama da figura 3.2 exemplifica a interação entre um novo cliente que acaba conectar-se ao sistema e o servidor. São representados o servidor

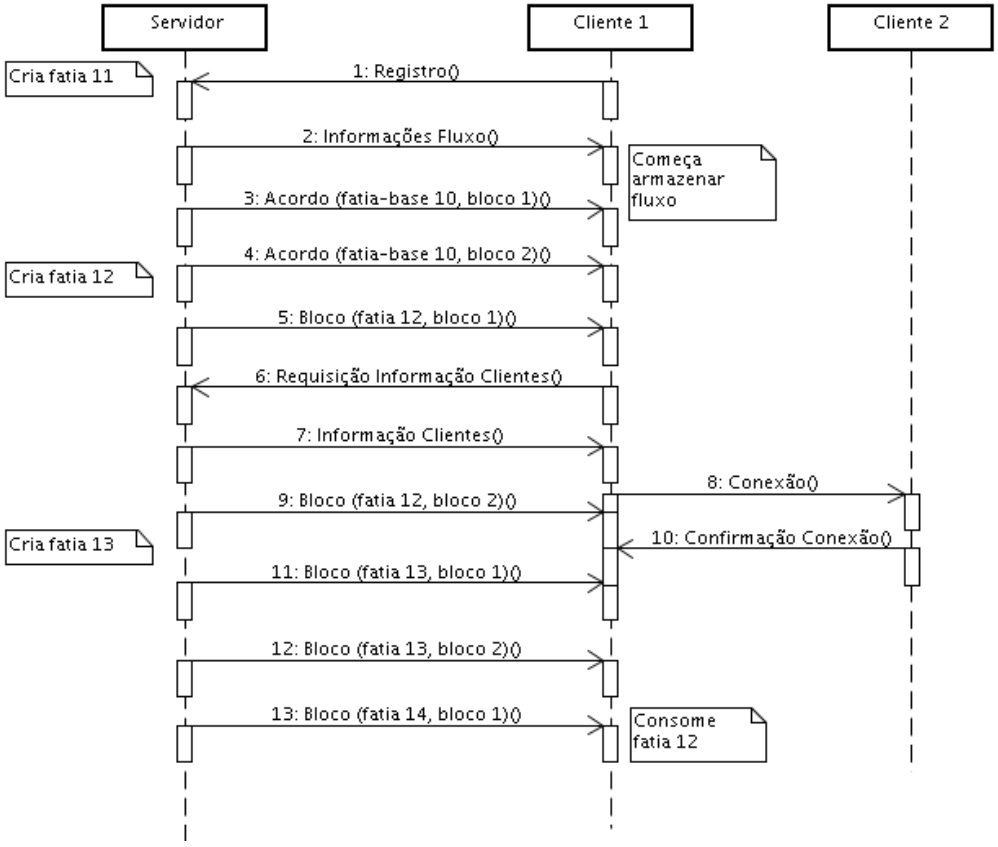

Figura 3.2: Digrama de seqüência: Inicialização

(*Servidor*), o cliente em questão (*Cliente 1*) e um outro cliente (*Cliente 2*) apenas para exemplificar a interação entre Cliente 1 e os outros clientes. As interações entre Cliente 2 e o Servidor não são representadas. No exemplo cada fatia é composta por dois blocos e os acordos tem duração de dez fatias.

No início do diagrama o servidor já está operando e o fluxo sendo produzido. Ao iniciar o cliente envia a mensagem de registro para o servidor (mensagem 1). O servidor responde com informações sobre o fluxo (mensa-

gem 2) e acordos para todos os blocos para a fatia-base atual (mensagens 3 e 4). A última fatia produzida pelo servidor foi a de número 11, assim a fatia-base atual é a 10 e os acordos são válidos até a fatia de número 19. A partir de então, o servidor envia os dados do fluxo para o cliente (mensagens 5, 9, 11, 12 e 13) enquanto os acordos forem válidos.

Ao receber as informações sobre o fluxo (mensagem 2) o cliente configura o módulo consumidor de mídia que começa a armazenar os dados recebidos em um buffer e inicia um temporizador para iniciar a consumir os blocos após uma certa defasagem. O módulo consumidor de mídia apenas começa a consumir os blocos e executar o fluxo para o usuário no ponto marcado com a nota próximo ao final do diagrama.

Após conectar-se ao servidor, o cliente solicita informações sobre outros clientes (mensagem 6) e recebe estas informações do servidor (mensagem 8). Esta interação não possui nenhuma relação de sincronismo com o recebimento do fluxo e por esta razão é representada no diagrama entre o recebimento de dois blocos do fluxo. Após receber as informações o cliente conecta-se aos outros clientes cujas informações foram recebidas (mensagens 8 e 10).

O segundo diagrama está contido na figura 3.3 e apresenta um exemplo de negociação de acordos. São representados o servidor, o cliente em questão (*Cliente 1*) e um segundo cliente para apresentar a interação do cliente em questão com o restante dos clientes da rede. O fluxo de mensagens do *Cliente*

2 não é completamente representado no diagrama.

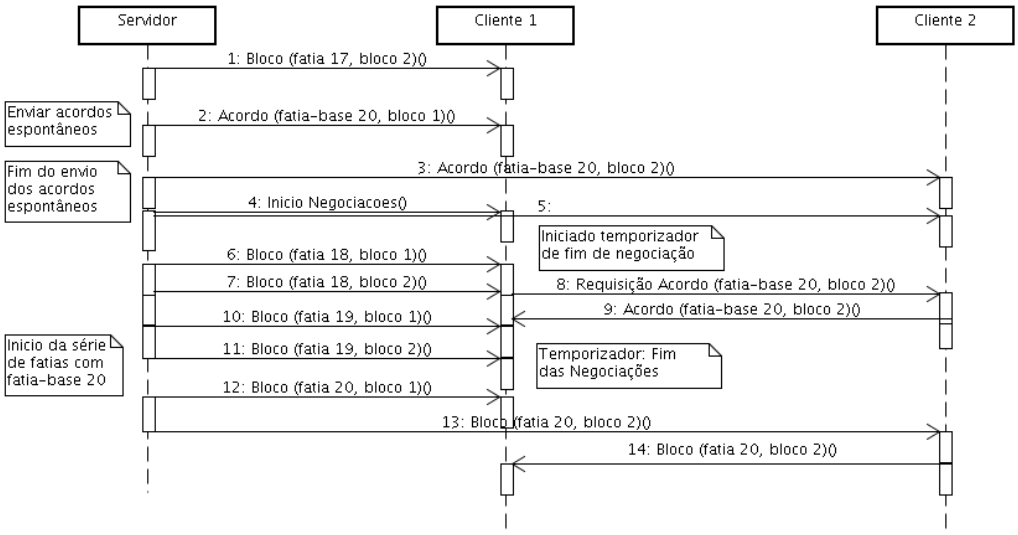

Figura 3.3: Digrama de seqüência: Negociação de Acordos

No início do diagrama assume-se que o *Cliente 1* possui acordos para recebimento de todas partes do fluxo com o servidor e está conectado ao *Cliente 2*. A duração dos acordos utilizada foi de 10 fatias. O servidor está enviando os dados da fatia 17. O diagrama inicia com o *Cliente 1* recebendo o segundo bloco da fatia 17 (mensagem 1).

Ao terminar de enviar a n-ésima fatia após cada fatia-base o servidor envia espontaneamente acordos para um subconjunto dos nós. O número de fatias antes do servidor enviar espontaneamente os acordos é parâmetro do algoritmo e neste exemplo foi utilizada a sétima fatia. Deste modo, ao terminar de enviar a fatia 17, que é a sétima fatia depois da última fatia-base,

o servidor envia espontaneamente acordos para os clientes.

O servidor envia espontaneamente acordos para um determinado número de clientes de forma que estes recebam uma certa quantidade de cópias de cada fatia. O número de clientes escolhidos e a quantidade cópias a ser transmitida são parâmetros do algoritmo. Neste exemplo, serão selecionados 2 clientes que receberão ao todo uma cópia de cada fatia. Desta forma, cada cliente escolhido receberá do servidor um bloco de cada fatia. Os acordos são válidos sempre para a série de fatias iniciada na próxima fatia-base. No exemplo do diagrama, o *Cliente 1* é selecionado para receber acordo para o bloco 1 (mensagem 2) e o *Cliente 2* para receber acordo para o bloco 2 (mensagem 3), ambos com fatia-base 20.

Após enviar espontaneamente os acordos para os clientes escolhidos, o servidor notifica todos os clientes conectados para que comecem a negociar acordos de encaminhamento para a nova fatia-base (mensagens 4 e 5). Ao receber esta mensagem o cliente inicia um temporizador para finalizar as negociações, indicado por uma nota no diagrama. Além disso, os clientes verificam para quais blocos da próxima fatia base necessitam de acordo e iniciam as tentativas de acordo. Como o *Cliente 1* já possui acordo para o bloco 1, recebido diretamente do servidor, precisa negociar apenas um acordo para o bloco 2. Ele envia então uma requisição de acordo para o *Cliente 2* (mensagem 8). Ao receber esta requisição, *Cliente 2* verifica se já possui

acordo para recebimento do bloco 2 e fatia-base 20 e se a quantidade de acordos de encaminhamento ainda não atingiu o seu limite. Neste exemplo, ambas as condições são verdadeiras e o acordo é aceito (mensagem 9). Se alguma destas condições fosse falsa *Cliente 2* enviaria uma mensagem para *Cliente 1* rejeitando a requisição de acordo. Neste caso, *Cliente 1* enviaria então requisição de acordo para outro cliente.

Quando o temporizador para finalizar negociação expira, *Cliente 1* verifica se todos os blocos possuem acordo de recebimento estabelecido para a fatia-base 20. Caso algum bloco não possuísse acordo seria enviada uma requisição de acordo para o servidor. Adicionalmente seriam solicitadas informações sobre outros clientes na rede, uma vez que os clientes aos quais *Cliente 1* está atualmente conectado não foram suficientes para conseguir estabelecer acordos para todos os blocos. Como neste exemplo o cliente estabeleceu acordos para o recebimento de todos os blocos nenhuma ação é tomada quando o temporizador expira.

Durante o período de negociação dos novos acordos os acordos vigentes para a fatia-base 10 continuam a ser cumpridos. Assim o servidor envia para o *Cliente 1* as fatias 18 e 19 (mensagens 6, 7, 10 e 11).

A fatia 20 inicia a nova série de fatias à qual os acordos firmados se referem. O servidor encaminha o bloco 1 para o *Cliente 1* (mensagem 12) e o bloco 2 para o *Cliente 2* (mensagem 13). Ao receber o bloco 2 da

fatia 20 *Cliente 2* encaminha este bloco para *Cliente 1* cumprindo o acordo estabelecido (mensagem 14). Este procedimento se repetirá para todos os blocos da série de fatias iniciada na fatia-base 20. Desta forma, o *Cliente 1* consegue obter todos os blocos do fluxo.

Não foram representados no diagrama por motivo de simplificação as requisições de acordo recebidas pelo *Cliente 1* nem as negociações feitas pelo *Cliente 2*.

Capítulo 4

Resultados Experimentais

Este capítulo descreve a implementação e os experimentos realizados com o sistema proposto.

O sistema foi implementado em linguagem Java. Todas as mensagens trocadas pelos nós foram modeladas como objetos Java que são serializados e transmitidos por conexões TCP/IP. A serialização é realizada utilizando recursos nativos de Java. Embora o recurso de serialização nativo não seja ótimo em termos de quantidade de dados transmitidos devido às informações adicionais de controle, ele é suficiente para demonstrar as propriedades do sistema proposto.

Foi desenvolvida uma classe encapsuladora para as funcionalidades de rede que permite a criação de limites artificiais de capacidade de transmissão e também realiza a coleta de estatísticas de quantidade de dados enviados

e recebidos. Estes limites artificiais de capacidade de transmissão permitem que várias instâncias de clientes sejam executadas em um mesmo computador, facilitando o processo de experimentação.

Os experimentos foram realizados utilizando a transmissão de um fluxo de 128 Kbps. Cada fatia é composta por 8 blocos de 4 KB cada, sendo consumidos a cada 2 segundos. Os acordos duram 10 fatias e o servidor envia espontaneamente acordos para 40% dos clientes após enviar a quinta fatia de cada acordo. A quantidade de cópias do fluxo transmitidas pelo servidor variaram entre os diversos experimentos. Todos os clientes possuíam limite artificial para envio e recebimento de 1024 Kbps. Os experimentos foram realizados diversas vezes e os resultados apresentados são representativos dos valores obtidos.

O restante do capítulo apresenta os três experimentos realizados, medindo o impacto do sistema proposto na quantidade de dados enviados pelo servidor, a influência da quantidade de cópias do fluxo enviadas pelo servidor na necessidade de *upload* dos clientes e o comportamento do sistema na presença de falhas.

4.1 Impacto do Sistema Proposto na Quantidade de Dados Transmitidos pelo Servidor

O primeiro experimento mostra o impacto da variação do número de cópias do fluxo transmitidas espontaneamente para os clientes na quantidade de bytes enviados pelo servidor para a rede. Foram realizadas execuções para 1 a 5 cópias do fluxo, com 8, 16 e 32 clientes. Cada execução durou 180 segundos. A medição foi realizada no servidor e considerou todos os dados enviados por este, incluindo mensagens de controle. Em todos os casos a taxa de interrupção de exibição do fluxo para o usuário foi inferior a 1%, isto é, mais de 99% das fatias de dados estavam disponíveis para serem consumidas no tempo apropriado.

O gráfico da figura 4.1 e a tabela 4.1 apresentam o resultado. Para cada número de clientes estão representadas as quantidades totais de dados transmitidas pelo servidor nas execuções com 1 a 5 cópias, respectivamente. A última coluna de cada grupo representa a quantidade mínima de dados a ser transmitida caso o servidor atenda individualmente todos os clientes, sem considerar mensagens de controle e cabeçalhos.

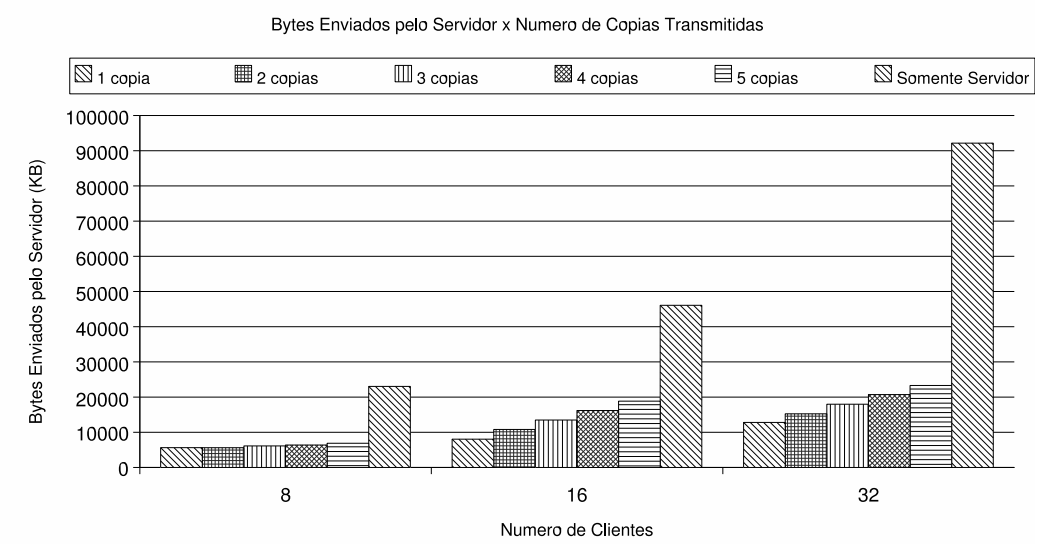

Figura 4.1: Dados Enviados pelo Servidor x Número de Cópias do Fluxo

Clientes	1 Cópia	2 Cópias	3 Cópias	4 Cópias	5 Cópias	Somente Servidor
8	5637 KB	5620 KB	6125 KB	6380 KB	6880 KB	23040 KB
16	8049 KB	10803 KB	13486 KB	16186 KB	18853 KB	46080 KB
32	12809 KB	15249 KB	17982 KB	20710 KB	23297 KB	92160 KB

Tabela 4.1: Dados Enviados pelo Servidor x Número de Cópias do Fluxo

4.2 Influência do Número de Cópias do Fluxo Transmitidas pelo Servidor na Necessidade de Upload dos Clientes

Quanto mais cópias o servidor envia para a rede, menor é a necessidade de encaminhamento de blocos pelos nós. O gráfico da figura 4.2 e a tabela 4.2

mostram a variação da quantidade média enviada de dados pelos clientes de acordo com a quantidade de cópias do fluxo enviadas pelo servidor. Os valores representam a média aritmética da quantidade de dados enviados por cada cliente, incluindo mensagens de controle e monitoramento. Nota-se que para 16 e 32 clientes houve uma ligeira diminuição da quantidade de dados enviados por cliente. Para 8 clientes o número permaneceu praticamente constante, devido ao fato de que as cópias distribuidas pelo servidor são entregues a apenas 40% dos clientes. Utilizando estes parâmetros, um sistema com 8 clientes não é grande o suficiente para tirar proveito das cópias adicionais transmitidas pelo servidor.

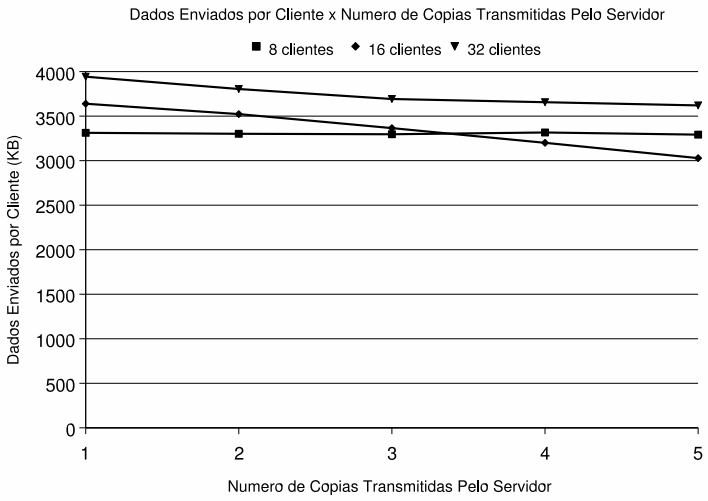

Figura 4.2: Dados Enviados pelo Servidor x Número de Cópias do Fluxo

Para o primeiro experimento os acordos enviados espontaneamente pelo servidor eram distribuídos entre 40% dos clientes. Foram realizadas execuções

Cópias	8 Clientes	16 Clientes	32 Clientes
1	3313 KB	3639 KB	3943 KB
2	3302 KB	3523 KB	3805 KB
3	3297 KB	3366 KB	3692 KB
4	3317 KB	3201 KB	3657 KB
5	3292 KB	3029 KB	3620 KB

Tabela 4.2: Dados Enviados pelo Servidor x Número de Cópias do Fluxo

variando esta quantidade de clientes entre 20%, 40% e 60%, com o servidor transmitindo sempre 3 cópias do fluxo. Como o limite artificial de capacidade de transmissão é o mesmo para todos os nós não houve variação significativa nos valores obtidos.

4.3 Comportamento do Sistema na Presença de Falhas

Outro experimento realizado analisa o comportamento do sistema na presença de falhas em nós. As configurações do fluxo utilizadas foram as mesmas do experimento anterior e o servidor transmitia espontaneamente 3 cópias do fluxo para a rede. Foram utilizados 32 clientes e, na metade do período de observação, 16 clientes escolhidos aleatoriamente falhavam. Os clientes que recebiam dados encaminhados pelos nós falhos deixam de receber os dados

e precisam solicitar retransmissão desses dados ao servidor. Mesmo com a presença da falha, a taxa de interrupção de exibição do fluxo para o usuário foi inferior a 1%, isto é, mais de 99% das fatias de dados necessárias para os clientes não-falhos estavam disponíveis no tempo apropriado. O gráfico da figura 4.3 apresenta o impacto da falha na quantidade de dados enviados pelo servidor, devido a estas retransmissões.

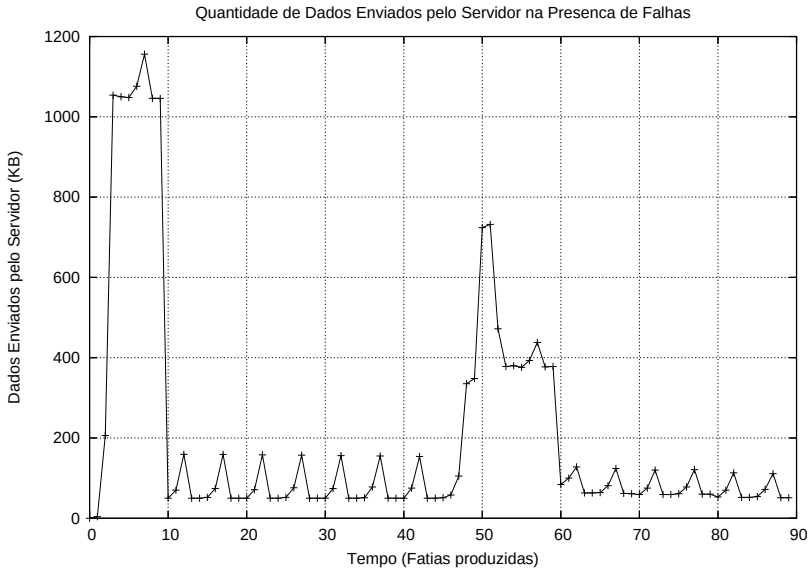

Figura 4.3: Quantidade de Dados Enviados pelo Servidor na Presença de Falhas

Podemos observar no gráfico uma grande quantidade de dados transmitidos do início da transmissão até o instante da criação da fatia número 10. Este pico deve-se aos acordos iniciais que o servidor envia a todos os clientes logo após a conexão. Nos experimentos realizados, todos os clientes conectavam-se ao mesmo tempo no início do experimento, entretanto, em

ambientes reais espera-se que estes clientes cheguem ao sistema conforme outras distribuições, tipicamente Poisson, o que diminuiria este pico inicial.

Os clientes começam a utilizar os acordos de encaminhamento estabelecidos no instante da criação da fatia número 10. A partir deste momento o sistema entra em regime de operação até a falha, que acontece próxima à produção da fatia número 45. Durante este período observam-se pequenos picos causados por mensagens de controle e monitoramento enviadas pelo servidor. Após o período impactado pela falha, o servidor entra novamente em regime de operação, apresentando também pequenos picos, desta vez um pouco menores já que a quantidade de clientes ativos agora é menor.

A falha acontece durante a produção da fatia de número 45 porém os impactos são percebidos no servidor apenas no momento da produção da fatia número 48. Esta defasagem ocorre porque os clientes solicitam a retransmissão dos dados faltantes ao servidor apenas momentos antes de serem necessários para consumo. Após a falha, os clientes consumem os dados armazenados em buffers e quando estes acabam solicitam a retransmissão ao servidor. Os efeitos da falha estendem-se até o momento da criação da fatia número 60, em que uma nova série de acordos entra em vigor.

Capítulo 5

Conclusão e Trabalhos Futuros

Este trabalho apresentou um sistema de distribuição de fluxos de conteúdo multimídia auxiliado por redes *peer-to-peer*. Nele, os usuários que estão recebendo o conteúdo estabelecem acordos para o encaminhamento de partes do fluxo, evitando a concentração do tráfego na origem do fluxo. Experimentos mostraram que o uso do sistema proporciona significativa redução na necessidade de transmissão de dados pelo servidor quando comparado com o atendimento de todos os clientes diretamente pelo servidor. Mostraram também que a transmissão espontâneas de cópias adicionais do fluxo pelo servidor reduzem a necessidade de transmissão de dados pelos clientes e que o sistema é capaz de recuperar-se em caso de falha em nós clientes sem que haja interrupção no fornecimento do fluxo para clientes não-falhos.

Como trabalhos futuros podemos mencionar a implementação de um sis-

tema utilizando um protocolo de rede mais eficiente que a serialização nativa de Java e a experimentação com topologias em que os clientes apresentem diferentes capacidades de transmissão entre si. Além disso, pode-se implementar novas políticas para a escolha dos clientes que receberão os acordos enviados espontaneamente pelo servidor e realizar experimentos para determinar quais políticas atendem melhor a cada tipo de topologia.

Referências Bibliográficas

[1] B. Cohen, "Incentives Build Robustness in BitTorrent," *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[2] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth Multicast in a Cooperative Environment," *SOSP'03*, 2003.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems," *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[4] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A Large-scale and Decentralised Application-level Multicast Infrastructure," *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, 2002.

[5] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "Scalable Application-level Anycast for Highly Dynamic Groups," *Networked Group Communications (NGC 2003)*, 2003.

[6] Windows Media Platform, http://www.microsoft.com/windowsmedia, Acesso em fevereiro de 2006.

[7] Helix Community, http://www.helixcomunity.org, Acesso em fevereiro de 2006.

[8] S. Deering, "Host Extensions for IP Multicasting," RFC 1112, 1989.

[9] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr, "Overcast: Reliable Multicasting with an Overlay Network," *Fourth Symposium on Operating System Design and Implementation (OSDI)*, 2000.

[10] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," *IEEE/INFOCOM 2005*, 2005.

[11] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys*, 2004.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-addressable Network," *ACM SIGCOMM'01*, 2001.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *ACM SIGCOMM'01*, 2001.

[14] B. Zhao, J. Kubiatowicz, and A. Joseph. "Tapestry: An Infrastructure for Fault-resilient Wide-area Location and Routing," Technical Report UCB//CSD-01-1141, 2001.

[15] Gnutella, www.gnutella.com, Acesso em fevereiro de 2006.

[16] Kazaa, www.kazaa.com, Acesso em fevereiro de 2006.

[17] A. Oram, *Peer-to-Peer: O Poder Transformador das Redes Ponto a Ponto*, Editora Berkeley, 2001.

[18] PeerCast, www.peercast.org, Acesso em junho de 2007.

[19] D. A. Tran, K. A. Hua, T. Do, "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming," *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, 2003.

[20] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, "Bayeux: An Architecture for Scalable and Fault-Tolerant Wide-Area Data Dissemination," *NOSSDAV'01*, 2001.

[21] S. Banerjee, B. Bhattacharjee, C. Kommareddy, "Scalable Application Layer Multicast," *ACM SIGCOMM'02*, 2002

[22] V. N. Padmanabhan, H. J. Wang, P. A. Chou, K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," *NOSSDAV'02*, 2002.

[23] M. Heffeeda, A. Habib, B. Botev, D. Xu, B. Bhargava, "PROMISE: Peer-to-Peer Media Streaming Using CollectCast," *ACM Multimedia (MM'03)*, 2003.

[24] X. Jiang, Y. Dong, D. Xu, and B. Bhargava, "A P2P Media Streaming Prototype," *IEEE International Conference on Multimidia and Expo ICME 03*, 2003.

[25] X. Zhang, J. Liu, B. Li, T. -S. P. Yum, "Coolstreaming DONet: A Data-Driven Overlay Network for Live Media Streaming," *IEEE INFOCOM '05*, 2005.