Marisa Affonso Vasconcelos

Análise da Distribuição de Streaming Media em Arquiteturas do Tipo Peer-to-Peer

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte

Abril de 2003

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

Análise da Distribuição de Streaming Media em Arquiteturas do Tipo Peer-to-Peer

MARISA AFFONSO VASCONCELOS

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. VIRGÍLIO AUGUSTO FERNANDES ALMEIDA - Orientador
Departamento de Ciência da Computação - ICEx - UFMG

Prof. EDMUNDO ALBUQUERQUE SOUZA E SILVA
Departamento de Engenharia - COPPE/UFRJ

Prof. SÉRGIO VALE AGUIAR CAMPOS
Departamento de Ciência da Computação - ICEx - UFMG

Prof. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - ICEx - UFMG

Belo Horizonte, 14 de abril de 2003.

Resumo

Arquiteturas cliente/servidor têm se mostrado ineficientes em serviços de distribuição de *streaming media*. O número máximo de usuários é limitado pelos recursos computacionais que tem uma demanda maior que nas aplicações *web* tradicionais. Uma das propostas para uma distribuição eficiente é a utilização de redes *Peer-to-Peer* (P2P). Nessa proposta, a distribuição do conteúdo é feita utilizando os próprios clientes, que são chamados de *servents*, já que podem atuar como cliente e servidor ao mesmo tempo. A maioria dos trabalhos na literatura, avalia esse tipo de cenário através de simulações, há uma ausência de trabalhos que fornecem uma análise quantitativa a respeito da utilização dos recursos do servidor e da rede em sistemas P2P. O objetivo desse trabalho é apresentar resultados experimentais que quantifiquem a diminuição da utilização dos recursos do servidor e da rede se uma abordagem P2P for utilizada. Para obter esses resultados, foi construído um ambiente experimental para avaliar as arquieturas cliente/servidor e P2P variando o número de clientes durante os experimentos. Esse ambiente é composto por uma nova aplicação escalável e eficiente chamada *streaming servent*, por um servidor comercial (Darwin) e por uma carga de trabalho que consiste em três arquivos de mídias com características diversas. A escalabilidade do *streaming servent* foi avaliada através de fórmulas analíticas. Os resultados mostram que a utilização de uma arquitetura simples de dois níveis reduziu a carga no servidor de mais de 60%, provendo uma qualidade melhor do *stream* e escalabilidade.

Abstract

Client/server architectures have shown to be inefficient in distributing streaming media. The maximum number of users is limited due to the computing resources, which are more demanding than in traditional web applications. Onde of the proposals for an efficient distribution is the use of a Peer-to-Peer networks (P2P). In this proposal, the content distribution is made by the clients themselves, which are called *servents*, since they act as client and server at the same time. Most of the studies in the literature evaluate this scenario through simulation, so there is a lack of quantitative analysis of the requirements for server and network resources in actual P2P systems. The goal of this work is to fill this gap by providing experimental results that quantify the savings in server and network resources if a P2P approach is adoted. Towards this goal, we have built an experimental testbed to evaluate both architectures, client/server e P2P, with varying the number of clients during the experiments. This experimental testbed is composed of one new efficient and scalable application called *streaming servent*, one streaming server application (Darwin) and a workload consisting three media files. In this work, we use simple analytical formulas to evaluate the scalability of our servent application. The results show that the use of a simple two-level architecture reduced the load on the server by more than 60%, providing better stream quality and scalability.

Agradecimentos

Primeiramente, agradeço a Deus pelo privilégio de realizar esse mestrado em um dos melhores departamentos de Ciência da Computação do Brasil. Orgulho-me muito desse departamento, da qualidade dos professores e da infra-estrutura que não poderia ser melhor, considerando um país como o nosso.

Aos meus pais, Maria Aparecida e Antônio, pelo amor e pelo apoio que sempre me deram em todas as minhas decisões, sobretudo nos momentos difíceis.

Aos meus irmãos Mariana e Daniel pelo carinho. A minha avó Hercília pelo amor e pela preocupação que sempre demonstra por mim.

Aos meus "meninos do E-speed": Juliano, Leonardo Rocha e Ismael pela ajuda que foi indispensável para realização desse trabalho. Eu agradeço muito pelas horas que foram gastas nos experimentos e nas questões "burocráticas" para a realização destes.

Aos amigos do Synergia, Daniela, Valdo, Marcelo, Melissa pelos almoços alegres e descontraídos que me faziam esquecer um pouco das dificuldades enfrentadas no decorrer dessa dissertação.

Ao pessoal do VoD, Alex, Matheus, Lamarque, Guilherme e do laboratório ATM, Isabela, Leonardo, Márcio, Emanuel pela amizade e pela agradável convivência.

Aos amigos da Pós, Rainer e Fernanda Paixão pela oportunidade de trabalharmos juntos, o que foi muito bem reconhecido.

Ao professor Virgílio pela paciência e pelo carinho quando me atendia em sua sala, sempre me estimulando e orientando.

Ao professor Wagner Meira Jr que também acreditou nesse trabalho e não nos deixou desanimar com as dificuldades surgidas durante o mesmo. À Jussara pelas valiosas dicas e revisões e pela grande contribuição que gerou o modelo analítico desse trabalho. Ao professor Antônio Alfredo, por quem tenho muito admiração, pela dedicação que dá ao todos os alunos. Aos professores Berthier e Sérgio Campos pela amizade e pela chance de poder trabalhar durante a graduação e a pós no grupo de Vídeo sob Demanda.

Ao CNPq, pelo apoio financeiro.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nos últimos anos, tem se observado um aumento no tráfego da Internet de dados do tipo *streaming media*. Algumas estimativas feitas para a *web* prevêem que a utilização de serviços de *streaming media* crescerá exponencialmente nos próximos 4 anos, devido a expansão da infra-estrutura de banda larga no setor residencial [29]. Esse crescimento é devido, principalmente, a transmissão de grandes eventos (final da Copa do Mundo, Olimpíadas, etc.), a utilização de serviços como rádios *online*, aplicações de educação à distância e outros. Espera-se que em alguns anos a Internet possa desempenhar o papel da televisão atual em termos de qualidade de áudio e vídeo [28].

No entanto, muitos desafios ainda devem ser superados, antes que o *streaming* de alta qualidade, se torne altamente difundido. O modelo utilizado nos serviços tradicionais da *web*, não pode ser totalmente aplicado aos serviços de *streaming media*. A distribuição do *streaming* possui restrições de tempo-real, o que impõe uma carga significativa sobre os recursos do servidor e da rede. Assim, para prover essa distribuição, o servidor deve alocar, para cada cliente, recursos como armazenamento e largura de banda suficiente para garantir a qualidade da exibição da mídia. Os dispositivos de armazenamento devem disponibilizar largura de banda suficiente para transferência contínua de dados à interface de rede, que por sua vez necessita de banda suficiente para distribuir o *stream* aos clientes. Devido à grande quantidade de largura de banda exigida pelos *streams* multimídia, a largura de banda do servidor determina o número de clientes atendidos simultaneamente [25]. Portanto, se ocorrer um aumento súbito do número de requisições (*flash crowds*), esse tipo de serviço não será escalável, recusando conexões ou deteriorando a qualidade do *stream* recebido pelos clientes.

Atualmente, a maioria dos sistemas de *streaming media* são baseados na arquitetura cliente/servidor, onde o servidor envia uma cópia do *stream* para cada cliente (*unicast*). Essa arquitetura não tem se mostrado satisfatória, por causa do alto consumo de banda e da degradação brusca da qualidade do *stream*. Os estudos, que visam aumentar a escala-

bilidade do serviço, se concentram basicamente em: esquemas otimizados de alocação dos dados nos discos, que tentam diminuir o tempo de *seek* e conseqüentemente o tempo de latência e atrasos na transmissão dos pacotes; distribuição do processamento, utilizando servidores distribuídos com ou sem replicação de conteúdo e técnicas de transmissão como *broadcast* e *multicast* que diminuem a utilização da largura de banda do servidor.

No *multicast*, um único *stream* é consumido por um grupo de clientes. Dessa forma, o servidor só envia o dado para os clientes que o requisitaram e uma única cópia desse dado para o grupo. A dificuldade do *multicast* está em sua implantação, já que a maioria dos roteadores não está habilitada para suportá-lo e que os ISP's (*Internet Service Provider*) ainda não estão capacitados a controlar e restringir o abuso conseqüentes do uso dessa técnica [11, 44]. Por causa disso, foram propostas soluções para implementação do *multicast* em nível de aplicação. Essa implementação pode ser dividida em dois grupos: *overlay-router networks* e redes *peer-to-peer*.

As *overlay-router networks* promovem o *multicast* utilizando vários servidores que atuam como roteadores em nível de aplicação, com funcionalidades *multicast*. Esses servidores recebem o *stream* do servidor central e o repassam para os clientes. Algumas CDNs (Rede de Distribuição de Conteúdo) promovem esse tipo de técnica, redirecionando as requisições dos clientes a servidores próximos geograficamente deles. O problema dessas soluções é o custo dessa infra-estrutura, sendo adequado a grandes *sites* comerciais.

As redes *peer-to-peer* (P2P) se tornaram populares na *web*, especialmente, por causa do baixo custo de sua infra-estrutura e de sua inerente escalabilidade. Redes P2P são redes virtuais construídas em nível de aplicação, que empregam recursos distribuídos para realizar alguma tarefa de forma descentralizada. Esses recursos podem ser: processamento, armazenamento, largura de banda, entre outros. Cada rede desse tipo possui protocolos de roteamento próprios que permitem dispositivos computacionais compartilhar informação e recursos diretamente sem a necessidade de servidores dedicados. Os componentes de uma rede P2P são chamados *servents*, já que tem a função de atuar como servidores e clientes [40].

Dado o grande potencial desse tipo de sistema, que é capaz de atender a milhares de usuários concorrentemente, foi proposto o modelo *peer-to-peer* para distribuição de *streaming media*. Nessa solução, o serviço de distribuição de *streaming media* é modelado através de uma árvore de *multicast*, onde a raiz é formada pelo servidor e os nodos dessa árvore são os *servents*. A distribuição do *stream* é feita em cascata, onde a raiz distribui o *stream* para seus nodos filhos, que por sua vez o repassam para seus descendentes.

Aliada a distribuição de *streaming media*, a tecnologia P2P poderia trazer maior disponibilidade de recursos a um baixo custo se comparado a soluções como IP *multicast* e CDNs. Outra vantagem é o aproveitamento da localidade de referência entre os usuários, que podem estar conectados no mesmo ISP. Isso, leva a uma redução do número de conexões com

o servidor e da quantidade de trafégo replicado. Apesar desse apelo intuitivo, pouco tem sido feito para quantificar a escalabilidade de sistemas desse tipo. Trabalhos relacionados com a distribuição de *streaming media* em redes P2P estão focados, principalmente, em simulações de políticas para a construção dinâmica da árvore de distribuição. Dessa forma, há uma falta de análises que quantifiquem a utilização dos recursos do servidor e da rede em sistemas P2P quando comparado a sistemas tradicionais cliente/servidor.

A motivação desse trabalho é fornecer resultados experimentais para uma análise quantitativa da utilização dos recursos do servidor e da rede para as arquiteturas P2P e cliente/servidor no contexto de distribuição de *live streaming media.*

A principal contribuição desse trabalho foi a construção do ambiente de experimentação para a avaliação dessas arquiteturas. Os experimentos foram realizados em rede local de 100Mbps *switched Ethernet* conectando um servidor de vídeo a um conjunto de estações clientes. Essas estações executavam um conjunto de *streaming servents*, aplicação desenvolvida nesse trabalho, que redistribuiam o *streaming* recebido para outros *servents* do sistema. A carga de trabalho utilizada nos experimentos consistia em três arquivos de mídias com diferente durações, taxa média de transmissão e variabilidade na taxa que estressava diferentes aspectos do sistema. Outras contribuições foram as medidas quantitativas, obtidas experimentalemente que mostram a escalabilidade das arquiteturas cliente/servidor e P2P. Em particular foi quantificada a redução da utilização dos recursos do servidor e da rede, além da melhoria na taxa média de perda e na duração da perda (*bursts*) se a abordagem P2P for utilizada. Finalmente, esses resultados foram validados através de fórmulas analiticas, como a Lei de Little [33]. A partir dessa lei, foram derivadas fórmulas que podem ser utilizadas para definir a árvore de distribuição P2P dada a quantidade máxima de recursos (CPU e largura de banda de rede) que um *peer* queira disponibilizar para a redistribuição do conteúdo.

Todas essas métricas visam mostrar quantitativamente como a escalabilidade de um serviço de distribuição de *streaming media* aliado à tecnologia *peer-to-peer* é mais fácil e mais barata de ser atingida que outras soluções.

1.1 Organização da Dissertação

O restante da dissertação está organizado da seguinte maneira. No Capítulo 2 são apresentados os trabalhos relacionados. O Capítulo 3, mostra alguns conceitos sobre *streaming media*. No Capítulo 4 são apresentados os conceitos relativos a arquitetura *peer-to-peer* e sobre a arquitetura para *live streaming media* utilizada neste trabalho. No Capítulo 5 são detalhadas as configurações utilizadas nos experimentos e as métricas de desempenho escolhidas. No Capítulo 6 são mostradas as análises feitas para a avaliação dos resultados e as fórmulas analíticas que determinam a escalabilidade do *servent*. Finalmente, no Capítulo 7 são apresentadas as conclusões dessa dissertação e algumas sugestões de trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Nessa seção são apresentados resumos dos principais trabalhos que cobrem os seguintes temas relacionados a nosso experimento.

2.1 Serviço Distribuído para *Streaming Media*

Em [17] foi feita uma avaliação através de simulação de um serviço distribuído para *streaming media* sob demanda. O sistema era composto por um ou mais servidores de vídeo, estações clientes e uma rede ATM que cobria uma área do tamanho de um bairro. Foram propostos e analisados esquemas de otimização, como entrega antecipada e recuperação de blocos de um cliente vizinho. No esquema de entrega antecipada, o cliente agenda com o sistema, quando ele irá assistir ao vídeo. O sistema então, começa o envio de blocos de vídeo para o cliente que os armazenará em disco. Se houver largura de banda disponível no sistema, o sistema irá enviar blocos de vídeo em uma taxa maior, visando uma economia de banda para o uso futuro. O esquema de recuperação de blocos do vizinho se assemelha a um esquema P2P. No entanto, o trabalho não levou em consideração o *overhead* imposto ao cliente quando esse passa a servir vídeo ao seu vizinho.

2.2 Multicast em nível de aplicação

Vários trabalhos foram propostos para estudar o problema da distribuição de *streaming media* na Internet. Para tentar resolver o problema da largura de banda do servidor, foram propostas, várias soluções que empregam a utilização de IP *multicast*. O maior problema dessa tecnologia é que nem todos os roteadores têm suporte à *multicast*. Além disso, os ISPs ainda não encontraram maneiras eficientes de controlar e restringir abusos que possam aparecer com sua utilização [44].

Para superar a ausência de IP *multicast*, propõe-se à implementação do *multicast* em nível aplicação, baseado em serviços IP *unicast*. Esse tipo de solução pode ser dividido em dois tipos de abordagens [46]: *overlay netwoks* e peer-to-peer.

2.2.1 *Overlay-Router Networks*

Essa abordagem [30, 26] consiste em espalhar vários servidores pela rede, que se comportam com roteadores em nível de aplicação, com funcionalidades *multicast*. Esses "roteadores" são interconectados formando uma árvore de distribuição, que visam fornecer uma utilização mais eficiente da largura de banda para execução dos serviços. O conteúdo é transmitido da fonte para um conjunto de "roteadores" dessa rede, formando assim, uma árvore *multicast*. Se um novo cliente deseja assistir algum *stream* que já esteja em transmissão, ele deverá se conectar a algum "roteador" que fique no caminho da distribuição de algum cliente anterior. Esse tipo de abordagem aumenta a escalabilidade do sistema, já que os clientes podem receber o *stream* de outros pontos da rede, não somente da fonte. No entanto, essa solução apresenta custos extras devido à utilização de servidores dedicados.

2.2.2 *Peer-to-Peer Streaming Media*

Um dos primeiros trabalhos descrevendo esse tipo de sistema é o da arquitetura *SpreadIt* [16]. Essa arquitetura foi proposta para distribuição de *streams* ao vivo, através de *multicast* em nível de aplicação. A distribuição por *multicast* é feita de forma incremental, à medida que os clientes entram no sistema em forma de árvore, onde a raiz é a fonte do *streaming* e os nodos são os clientes. Nessa árvore cada nodo possui uma camada de *peering* responsável pelas operações de manutenção, como entrada no sistema, redirecionamento para outro nodo, no caso de saída ou falha de um nodo intermediário. Foram analisadas através de simulação, várias políticas de adição e exclusão de nodos nesse sistema. Foi observado que alguns parâmetros de QoS (latência, perda de pacotes e tempo para descoberta de um nodo apto a atender outros clientes) variam linearmente com a profundidade da árvore de *multicast*. Concluiu-se que como a profundidade da árvore é exponencialmente menor que o número de clientes servidos, a QoS é degradada suavemente com o número de clientes.

Um outro esquema de distribuição *streaming media* é o *CoopNet* (*Cooperative Networking* [38]. *CoopNet*, também apresenta um esquema de distribuição utilizando *multicast* em nível de aplicação. A principal diferença em relação a arquitetura *SpreadIt* é a utilização de várias árvores de distribuição, ao invés de uma única árvore. Nessa arquitetura o *stream* original é dividido em *sub-streams* que são codificados separadamente. Cada árvore de distribuição transmite um tipo de *sub-stream* sendo que no melhor caso, o cliente recebe

todas os *sub-streams*. A qualidade do *stream* recebido depende do número de *sub-streams* recebidos pelo cliente. No caso de saída ou falha de algum nodo intermediário, o cliente deixará de receber o *sub-stream* do nodo que saiu do sistema, mas continuará a receber outros *sub-streams* provenientes de outras árvores, de forma que, não haja interrupção na exibição do *stream*. Ou seja, o cliente continuará assistindo a mídia, mesmo com perda de qualidade. Outra diferença em relação ao *SpreadIt* é a distribuição de *streams* sob demanda. Nesse caso, os clientes "cacheiam" a mídia requisitada a fim de prover novos clientes que requisitarem a mesma mídia. Nesse trabalho foram feitas simulações utilizando *traces* de um portal de notícias no dia 11 de setembro de 2001, período que ocasionou *flash crowds* em vários *sites* desse tipo. Mostrou-se que o *CoopNet* conseguiu reduzir a carga no servidor, sem aumentar a carga nos clientes e na distribuição de *streams* ao vivo, o sistema se mostrou robusto.

Em [22] é descrito um modelo P2P para distribuição de *streams* sob demanda. Esse estudo propõe algoritmos para disseminação dos arquivos das mídias e para localização de *peers* que armazenem segmentos do arquivo requisitado. A disseminação de dados é feita através do armazenamento de cópias dos segmentos recebidos pelo cliente. Em troca, esse cliente recebe algum incentivo, que é proporcional à taxa de distribuição e ao número de segmentos "cacheados" por ele. O algoritmo, também visa diminuir o tráfego nos *links* do *backbone*. Para isso, os clientes do mesmo domínio de rede são agrupados, a fim de que a maioria das requisições por arquivos seja atendida dentro do mesmo grupo de que foram originadas. Para a localização de conteúdo, o servidor mantém um índice com informações sobre todos os *peers* do sistema. Essas informações são armazenadas em forma de tabela que contém campos como: endereço IP dos *peers*, taxa de distribuição do *streaming* e um histórico sobre quanto tempo o *peer* permaneceu conectado em sessões anteriores. Para validação dessa arquitetura e de seus algoritmos foram feitas simulações que consideram diferentes taxas de chegada de clientes (constante, *flash crowds* e Poisson) e diferentes porcentagens de *caching* em cada *peer*. Concluiu-se que o modelo foi capaz de suportar os vários tipos de taxas de chegada e que o algoritmo para disseminação de dados manteve a maioria do tráfego dentro do mesmo domínio de rede.

Através de uma análise de custos para identificar arquiteturas, algoritmos e protocolos que fossem viáveis para um sistema P2P em alta escala, foi proposto em [21] um modelo de custos, no qual os *peers* compartilham seus recursos com outros *peers* em troca de incentivos ou recompensas. Foi feita uma comparação entre as arquiteturas P2P e cliente/servidor e mostrou-se que com um pequeno investimento inicial em uma infra-estrutura básica, a arquitetura P2P é capaz de oferecer um serviço em larga escala. Além disso, foi verificado que a arquitetura P2P é mais lucrativa e eficiente que a arquitetura cliente/servidor.

Em [47] foram estudados dois problemas, encontrados em sistemas P2P para *streaming media*. O primeiro problema diz respeito à atribuição de múltiplos *streams* a um dado

peer. Como a largura de banda de um *peer* é menor que a taxa de exibição da mídia, são necessários vários *peers* fornecendo o *stream* para cada cliente. Para isso, foi proposto um algoritmo ótimo para atribuição de dados através de múltiplos *peers* fornecedores por sessão. O algoritmo leva em consideração que os *peers* do sistema possuem larguras de banda heterogêneas. O outro problema é a rápida ampliação da capacidade do sistema. Foi sugerido um esquema de admissão diferenciado, que favorece os *peers* que podem oferecer maior largura de banda a outros *peers*. Dando prioridade de atendimento a esses *peers*, espera-se que o sistema fique mais capacitado, quando estes se tornarem fornecedores do *stream*.

Existem soluções comerciais de *peer-to-peer* para *streaming media* ao vivo como Allcast [9] e BlueFalcon [8], mas não publicações descrevendo quais são as técnicas utilizadas.

Uma parte da análise da arquitetura *SpreadIt* foi feita através de uma análise experimental, mas somente em relação aos atrasos e perda de pacotes. Não foram feitas análises do impacto da arquitetura na qualidade do *stream* recebido pelos clientes e nem no consumo dos recursos do servidor. O diferencial do trabalho de dissertação em relação aos trabalhos descritos acima pode ser diferenciado em três aspectos: (1) os artigos aqui analisados fizeram uso de simulação, enquanto que nesse trabalho as arquiteturas avaliadas foram implementadas e experimentadas; (2) houve a preocupação de se avaliar o consumo dos recursos do servidor (CPU e largura de banda de rede) e da rede (perda de pacotes) em função do número de clientes; (3) o sistema foi avaliado durante períodos quando o conjunto de clientes e a árvore de distribuição eram fixos (ou seja, não foi considerada entrada e saída de clientes do sistema) a fim de se avaliar a escalabilidade máxima que podia ser alcançada.

Capítulo 3

Serviços de Distribuição de *Streaming Media*

Tem-se observado um aumento do consumo de *streaming media* que inclui áudio, vídeo, animação e conteúdo multimídia. Segundo [28], o número de usuários domésticos que acessaram algum tipo de conteúdo do tipo *streaming media* cresceu de 21 milhões em 1999 para aproximadamente 35 milhões em 2000 nos Estados Unidos. Esse crescimento na demanda acontece, principalmente, por causa de grandes eventos como *Super Bowl*, jogos olímpicos, eleições e serviços de comércio eletrônico que disponibilizam *streams* promocionais de seus produtos. A previsão é que em alguns anos, qualidade do áudio e vídeo transmitidas seja bem próxima a qualidade da televisão atual [37].

Nesse capítulo são apresentados alguns conceitos sobre *streaming media*, são descritas os tipos de arquiteturas para esse serviço e são discutidos os problemas que devem ser levados em consideração no projeto desse serviço.

3.1 Definições

Um serviço de distribuição de *streaming media* permite que usuários remotos possam acessar objetos multimídia armazenados em um ou mais servidores, a qualquer momento. Esses objetos são distribuídos pelos servidores na forma de *streams*, que são sequências de pacotes temporalmente ordenadas [16], através de uma infra-estrutura de rede aos clientes dispersos geograficamente. O termo *streaming media* se refere à transferência de dados multimídia ao vivo ou sob demanda, onde o usuário os pode exibir tão logo sejam recebidos, sem a necessidade de um *download* completo [13].

3.1.1 Tipos de Comunicação e Transmissão

Em geral, na maioria dos serviços de *streaming media*, a forma de comunicação mais utilizada é *unicast*, onde o servidor envia uma cópia do dado para cada cliente que requisitou. Outra forma de comunicação é o *broadcast*, em que uma única cópia do dado é enviada para todos os clientes da rede. A desvantagem do *unicast* é o envio de múltiplas cópias do mesmo dado e a do *broadcast* é o desperdiço na largura de banda da rede, já que todos os clientes da rede recebem o dado, mesmo os que não o tenham requisitado. O *broadcast*, também pode gerar um aumento do processamento na máquina cliente, pois se deve descobrir, se o dado recebido é de interesse ou não [27].

O *multicast*, outra forma de comunicação, combina as vantagens das duas formas anteriores e minimiza suas desvantagens. Essa técnica consiste em enviar uma única cópia do dado para os clientes que o requisitaram. Assim, a largura de banda do servidor é menos utilizada, já que, uma cópia do dado (*broadcast*) é enviada somente para o grupo de clientes que o requisitaram (*unicast*).

Várias técnicas de *multicast* para *streaming media* foram propostas na literatura e são basicamente divididas nas seguintes categorias [12]:

- *Batching*: Nessa técnica, a requisição do cliente deve esperar que outras requisições sejam feitas, para que se inicie o *multicast*. A técnica de *batching* só é utilizada para *Near Video On Demand* (um mesmo filme é transmitido em vários canais, separados por intervalos de tempo iguais) [15].

- *Multicast Dinâmico*: Os clientes são organizados na forma de uma árvore de *multicast*, que vai se expandindo à medida que chegam novas requisições. Assim, se um cliente requisitar um *stream*, que já que sendo transmitido, ele se junta àquele *multicast*. Se o serviço oferecer a execução do *stream* desde o início, o cliente precisará de dois *buffers*: um para armazenar os blocos do *multicast* corrente e outro para armazenar o *patching stream*. O *patching* é composto pelos blocos anteriores aos armazenados no *buffer*. Quando a execução do *patching stream* é completada, o cliente começa a executar os blocos do *multicast*, armazenados do buffer [24].

- *Piggybacking*: Nessa técnica, a taxa de envio dos *streams* pelo servidor é alterada para se possa combinar vários *streams* em um só [19].

- *Catching*: Nessa técnica, o sistema possui vários canais: um de banda larga, que conecta todos os clientes (*broadcast*) e outros de bandas menores, que ligam o servidor ao cliente, individualmente. No canal de banda larga, são transmitidos os filmes mais acessados. Dessa forma, quando o cliente requisita um filme, ele o recebe pelo canal de banda larga armazenando esse trecho. Simultaneamente, o cliente recebe os trechos

que faltam da mídia pelo canal de banda menor. Essa técnica consegue fazer com que o tempo de latência se aproxime de zero [18].

Existem implementações de *multicast* em várias camadas da pilha de protocolos, sendo a da camada de aplicação mais utilizada [16]. Nesse tipo de implementação a rede é composta por vários servidores que atuariam como roteadores dividindo e redirecionando o *stream* para os clientes.

Existem dois tipos de transmissão para um *stream*: ao vivo e sob demanda. As requisições para *streams* sob demanda são feitas para arquivos pré-armazenados com duração bem definida. São requisições assíncronas, já que os clientes são independentes uns dos outros, podendo assistir diferentes partes do objeto requisitado ao mesmo tempo. No caso dos *streams* ao vivo, não há a noção de duração total e não há interação do usuário através de funções VCR (*rewind*, *pause*, *fast forward*). São chamados síncronos, pois todos os usuários assistem exatamente o mesmo conteúdo durante toda a sessão (tempo que ficam conectados). Fazendo uma analogia a aplicações cotidianas, o *stream* sob demanda possui uma execução similar a de um vídeo cassete enquanto o *stream* ao vivo é análogo as transmissões de TV [32].

3.1.2 Protocolos para *Streaming Media* na Internet

Diversas especificações de padrões de protocolos para distribuição e controle da transmissão de dados multimídia na Internet têm sido propostos pela IETF (*Internet Engineering Task Force*).

Os protocolos RTP (*Real-time Transport Protocol*) e RTCP (*Real-time Control Protocol*) [41] foram propostos para dar suporte a distribuição de *streaming media*. O RTP foi projetado para transferência de dados e o RTCP para mensagens de controle. A grande maioria dos sistemas, que implementam esses protocolos, utiliza RTP/UDP para entrega dos dados e RTCP/TCP ou RTCP/UDP para controle.

O RTP não garante QoS ou entrega confiável dos dados, mas monitora e auxilia o controle de fluxo do transmissor para o receptor, através de um conjunto de funcionalidades. Informações como origem dos dados, tipo de identificação e sequenciamento dos pacotes é utilizado para determinar o conteúdo dos pacotes e confirmar se estão ordenados corretamente no receptor. Os pacotes RTP fornecem *timestamp*, para detectar atrasos dos pacotes (*jitter*), que podem ser prejudiciais à reprodução da mídia. Outra funcionalidade é a integração de tráfego heterogêneo que permite a união de múltiplas fontes em um fluxo simples.

O protocolo RTCP provê um *feedback* do recebimento dos dados pelos clientes, em termos do número de pacotes perdidos, *jitter*, atrasos, etc. As mensagens de *feedback* são periódicas (enviadas a cada 5 segundos) e não utilizam mais que 5% da banda total da

sessão. A fonte do *streaming* utiliza essa informação para ajustar suas operações como, por exemplo, adaptar a taxa de envio. Outras funcionalidades do RTCP são a sincronização de várias mídias, como por exemplo, áudio e vídeo e medidas sobre o *round-trip time.*

Para estabelecimento da sessão foi proposto o protocolo RTSP (*Real-time Streaming Protocol*) [42]. Ele dá suporte a funcionalidades do tipo VCR tais como: *play, pause, seek* e *record.* Juntamente com o RTSP, o protocolo SDP (*Session Description Protocol*) provê informações descritivas da sessão, por exemplo, se o conteúdo transmitido é áudio ou vídeo, qual o tipo de compressão utilizada, etc.

3.2 Arquitetura de um Serviço de *Streaming Media*

Um sistema de distribuição de *streaming media* é constituído de: um ou mais servidores, que são responsáveis por atender as requisições; de um sistema de armazenamento, onde são gravadas as mídias; de um sistema de rede, encarregado de fazer a comunicação entre servidor e clientes e de estações clientes que enviam requisições e provêem a visualização do filme [17].

Um conceito fundamental é o conceito de *canal* do servidor. Um canal é definido como a quantidade de recursos do servidor (largura de banda, CPU, memória, etc.) exigidos por cada cliente, para garantir a entrega do *stream* de forma contínua. Ao receber uma requisição de algum cliente, o servidor deverá verificar se existe largura de banda suficiente para a transferência de dados entre os dispositivos de armazenamento do *streaming* e a interface de rede, que por sua vez deverá ter largura de banda suficiente para distribuir o *stream* para os clientes. Por causa da grande quantidade de banda requisitada pelos *streams* multimídia, a largura de banda do servidor determina o número de canais ou clientes que são atendidos ao mesmo tempo.

3.2.1 Componentes do sistema

O servidor, que é a parte central do sistema de *streaming media*, tem a função de atender as requisições dos clientes. O funcionamento básico do servidor é recuperar os blocos de dados do sistema de armazenamento e enviá-los pela rede aos clientes. Os clientes, por sua vez, ao receberem blocos do objeto requisitado, o armazenam em um *buffer* e o decodificam através de alguma aplicação que permita a exibição de seu conteúdo.

Para garantir a exibição contínua do *stream* nas aplicações clientes, o servidor deve manter dados suficientes nos *buffers* dos clientes. Isso só é conseguido, porque a taxa que o servidor lê os dados do disco é maior que a taxa que o cliente decodifica e exibe o bloco de vídeo. Assim, a cada envio de dados aos clientes, o servidor envia rajadas de blocos.

Esses blocos são armazenados no *buffer* do cliente e são decodificados, ao mesmo tempo, em que o servidor pode continuar a atender outros clientes.

3.2.2 Topologia Tradicional

A topologia mais utilizada na distribuição de *streaming media* é baseada na arquitetura cliente/servidor. Como mostrado na figura 3.1, ela é composta pela fonte do *streaming* e pelos clientes que requisitam os objetos disponibilizados. A fonte pode ser um servidor, um conjunto de servidores, um conjunto de servidores e *caches* ou um conjunto de servidores e *proxies*.

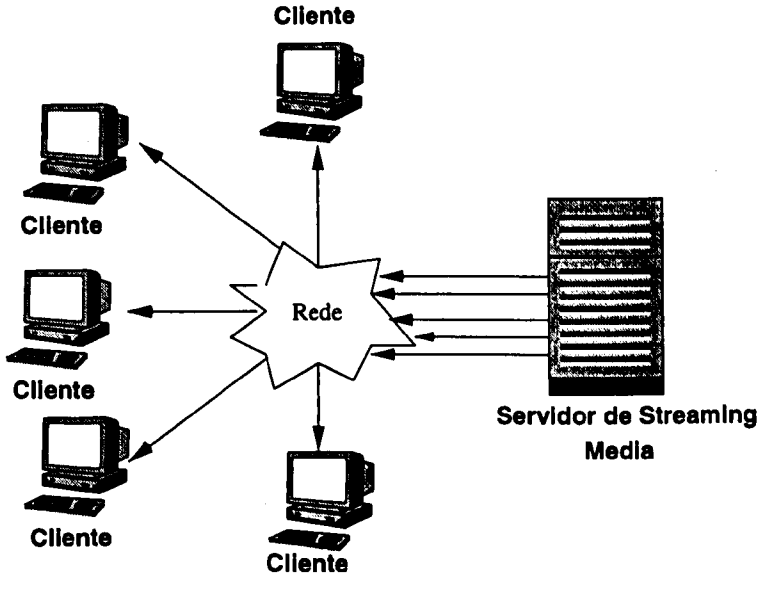


Figura 3.1: Arquitetura cliente/servidor para distribuição de streaming media

O maior problema dessa topologia é que a capacidade máxima (número de clientes simultâneos) do sistema é limitada pelos recursos da fonte do *streaming*. Essa limitação ocorre, principalmente, por causa da largura de banda do servidor, mas pode ser devida a outros recursos da máquina servidora como poder de processamento, tamanho da memória ou velocidade dos dispositivos de I/O. Por exemplo, um servidor de *streaming media* com um *link* T3 (45Mbits/s) poderá servir apenas 45 clientes simultâneos requisitando objetos a uma taxa de 1 Mbit/s [16]. A escalabilidade torna-se mais difícil de ser alcançada, se o serviço tiver os usuários da Internet como público alvo. Situações como *flash crowds*, onde o número de usuários aumenta de forma inesperada, não foram previstas por esse tipo de arquitetura.

3.2.3 Topologia Distribuída

Um sistema distribuído de *streaming media* é composto por um conjunto de servidores, que têm o objetivo de distribuir o maior número possível de *streams* concorrentes. As requisições dos clientes são redirecionadas para um dos servidores do conjunto. A escolha do servidor que irá atender à uma requisição, visa o balanceamento de carga no sistema.

Exemplos desse tipo de sistema são as redes de distribuição de conteúdo (CDN). As CDNs são redes cuja infra-estrutura é composta por um conjunto de servidores dedicados, espalhados geograficamente e interligados por uma rede proprietária. Quando uma requisição é feita, elas são redirecionadas do servidor central para o servidor da CDN que contém a réplica do conteúdo, mais próximo da origem da requisição. O objetivo principal é proporcionar aos usuários, um tempo de resposta menor do que o que seria experimentado se esses utilizassem diretamente o servidor central. Algumas empresas oferecem esse tipo de serviço, como a Akamai [45]. No entanto, esse tipo de infra-estrutura possui um alto custo, sendo apropriado para grandes *sites* comerciais, além de ser desconhecido seu comportamento em situações de *flash crowds* em vários ou todos os *sites* servidos pela CDN [38]. Além disso, as CDNs atuais foram projetadas, a princípio, para atender requisições por conteúdo Web, mas aquelas que dão suporte a conteúdo do tipo *streaming media* são limitadas à distribuição de *streams* ao vivo de baixa qualidade [14].

Outro exemplo de sistema distribuído é a utilização de redes *peer-to-peer* para distribuição de *streaming media.* O funcionamento dessas redes será descrito no capítulo 4.

3.3 Sumário

Nesse capítulo foram apresentados alguns conceitos sobre serviços de distribuição de *streaming media.* Foram apresentados dois tipos de arquiteturas utilizadas nesse tipo de serviço. A arquitetura cliente/servidor que é a mais utilizada, mas possui problemas de escalabilidade e a arquitetura distribuída com vários servidores dedicados, como as CDNs, adequada a servir clientes de grandes portais comerciais, devido ao custo da infra-estrutura. Ainda, considerando a distribuição do processamento e da carga, foi proposta a arquitetura *peer-to-peer* para distribuição de *streaming media*, que será descrita no capítulo seguinte.

Capítulo 4

Serviços de *Streaming Media* em Arquiteturas *Peer-to-Peer*

4.1 Peer-to-Peer

As redes *peer-to-peer* (P2P) são redes virtuais construídas no nível de aplicação. Elas possuem mecanismos próprios de roteamento que permitem dispositivos computacionais compartilhar informação e recursos diretamente, sem servidores dedicados. Assim, os custos de armazenar o conteúdo e a largura de banda utilizada para transmiti-lo são divididos entre os membros da rede (*peers*), que são chamados de *servents* [40].

Nos últimos anos, esse modelo de computação distribuída descentralizada, vem atraído interesse. Sua popularidade se deve principalmente a aplicações de compartilhamentos de arquivos como o Napster [5], Gnutella [3], Kazaa [4] e outros. A grande contribuição desse tipo de sistema é a escalabilidade que permite que milhões de usuários estejam *online* mesmo em períodos de pico [31]. Isso é obtido graças ao comportamento híbrido (cliente + servidor) dos usuários, o que permite a descentralização do processamento.

4.1.1 Características dos sistemas P2P

Essa subseção enumera algumas características importantes segundo [35, 23, 34], que têm impacto na eficiência de sistemas P2P e suas aplicações.

Descentralização

É a principal vantagem que os modelos P2P apresentam em relação à arquitetura tradicional cliente/servidor. A descentralização afeta como os usuários estarão conectados ao sistema e como é a interação entre eles.

No modelo cliente/servidor, a informação fica concentrada em servidores e distribuída através da rede aos clientes. A centralização é ideal em alguns tipos de aplicações em que segurança e acessibilidade são vitais. No entanto, esse tipo de topologia é ineficiente, possui gargalos e desperdício de recursos. Embora, o desempenho e custos do *hardware* estejam melhorando, repositórios centralizados são caros e difíceis de se manter.

A principal vantagem de um sistema totalmente descentralizado, onde todos os componentes possuem o mesmo papel, é a transferência do controle de acesso e dos recursos para os usuários. No entanto, a ausência de um servidor central, com o conhecimento da rede e do conteúdo disponibilizado, dificulta a implementação desse tipo de sistema. Por causa disso, muitos sistemas P2P centralizam algumas de suas funções. Esse é o caso do Napster, onde existe a centralização do índice dos arquivos disponibilizados e a descentralização do *download*, que é feito diretamente entre os *peers*.

Em sistemas descentralizados como o Gnutella e o Freenet [2], para se fazer parte do sistema, novos nodos devem se conectar a nodos que já estejam na rede ou utilizar uma lista de IPs conhecidos de outros *peers*.

Escalabilidade

É um dos benefícios imediatos da descentralização. A escalabilidade leva a questões como: quanto o sistema pode crescer? Se for utilizado *broadcasts*, isso irá saturar a rede? O repositório do índice dos dados pode sofrer *overflow*?

A escalabilidade é vantajosa quando não há detrimento de outras características como o desempenho e o determinismo. Para tentar diminuir esse problema, Napster mantém intencionalmente algumas de suas operações e arquivos centralizados.

Aplicações como o Gnutella e o Freenet possuem um comportamento não-determinístico devido à sua natureza *ad-hoc*. O mecanismo de busca dessas aplicações é feito "às cegas", já que o nodo envia suas consultas através de *broadcasts* na rede, a fim de que outros nodos repassem sua consulta. Isso faz com que o tempo para receber as respostas da consulta seja ilimitado. Além disso, esse mecanismo de busca não garante que o objeto seja encontrado mesmo que ele exista.

Sistemas P2P recentes como o CAN [39] e o Chord [43] possuem um mapeamento consistente entre a chave do objeto e sua localização. Cada nodo mantém somente a informação sobre um subconjunto de nodos do sistema. Isso limita o volume de estados que são mantidos, aumenta a escalabilidade e garante que se o nodo estiver na rede, seus objetos serão sempre alcançados.

Desempenho

O desempenho, no contexto de P2P, não está focando em atrasos da ordem de milisegundos, e sim em questões como: quanto tempo demora para um arquivo ser transferido e quanto de largura de banda uma consulta irá consumir. Por causa de sua natureza descentralizada, essas questões devem ser tratadas, considerando-se um conjunto de fatores combinados.

A comunicação, por exemplo, é influenciada pelo processador e pela velocidade dos dispositivos de I/O da máquina do usuário. No entanto, mesmo que a velocidade de conexão seja rápida, o sistema pode não ser escalável se um número maior de usuários estiver tentando se conectar simultaneamente.

Outra questão é os algoritmos de descoberta de recursos utilizados pelos sistemas P2P. Sistemas centralmente coordenados, como o Napster, sofrem de gargalos em seus recursos e todas as consultas devem passar por um servidor central. Para superar essa limitação, foram propostas arquiteturas híbridas [48]. Essas arquiteturas distribuem algumas das funcionalidades do coordenador a vários servidores, similarmente ao que ocorre com o sistema de DNS, que possuem uma árvore de coordenadores que cooperam entre si. Em sistemas baseados em *flooding*, como o Gnutella, cada requisição de um *peer* é repassada para todos os *peers* conectados a ele, que por sua vez repassam para seus vizinhos. Esse modelo de descoberta de recursos exige muita largura de banda da rede e não é escalável. Para tentar diminuir o número de mensagens que trafegam pela rede, alguns sistema utilizam o conceito de "super-peers". Esses *peers* concentram algumas das requisições à custa de um alto consumo de CPU.

Tolerância a falhas

Os sistemas P2P operam em um ambiente inerentemente não-confiável, já que dependem dos recursos dos usuários. Os recursos podem ser tornar indisponíveis a qualquer momento devido a várias razões, desde falhas na rede ou na máquina dos usuários à vontade dos próprios usuários de não quererem mais compartilhar. O maior desafio dos sistemas P2P é que a responsabilidade da manutenção do sistema é completamente distribuída, ao contrário dos sistemas cliente/servidor onde a manutenção é de responsabilidade somente do servidor.

Algumas soluções têm sido estudadas como a instalação de nodos especiais *relays*, que armazenam temporariamente qualquer modificação ou comunicação até que o destinatário de restabeleça. Outra solução é adotada por sistemas como o Napster e o Gnutella, é a replicação dos arquivos mais populares.

4.1.2 Topologias

Os sistemas P2P são classificados em três principais categorias, de acordo com [23, 35]:

- **Centralizado:** a coordenação dos *peers* é feita por um servidor central. Após receber a informação desse servidor, a comunicação entre os *peers* passa a ser direta. Algumas vantagens desse tipo de topologia são: a facilidade de gerência e segurança. Como desvantagens tem-se a baixa tolerância à falhas, uma vez que alguns dados são mantidos somente pelo servidor central, e limitação da escalabilidade pela capacidade do servidor. Entretanto, a escalabilidade ainda pode ser conseguida graças ao crescente poder de processamento das máquinas, que possibilita ao servidor atender a um grande número de usuários. Alguns exemplos desse tipo de sistema são a arquitetura de busca do Napster e o sistema SETI@Home, que possui um *dispatcher* central de tarefas.

- **Descentralizado:** não existe coordenador, todos os *peers* possuem as mesmas funções. A comunicação é feita através de múltiplos *multicasts*, onde os próprios *peers* funcionam como roteadores, repassando as mensagens para outros *peers*. Algumas vantagens importantes são a extensibilidade e tolerância a falhas. A escalabilidade nesse tipo de sistema é difícil de ser medida, pois a adição de novos usuários, ao mesmo tempo em que torna a rede mais capacitada, aumenta o custo de se manter a consistência dos dados. Exemplo: Freenet e Gnutella.

- **Híbrido (centralizado + descentralizado):** nessa topologia, os *peers* repassam suas consultas aos *super-peers*, os quais comunicam entre si de maneira descentralizada. As vantagens dessa assemelham-se às de topologia descentralizada com uma melhoria na questão da consistência dos dados, uma vez que parte deles é mantida somente pelos *super-peers*. Exemplos de sistemas de topologia híbrida são o KazaA e o Morpheus [36].

A Tabela 4.1 derivada de [35], mostra as vantagens e desvantagens das topologias distribuídas listadas.

	Centralizado	Descentralizada	Híbrida
Gerenciável	sim	não	não
Extensível	não	sim	sim
Tolerante a falhas	não	sim	sim
Segura	sim	não	não
Susceptível a processos judiciais	sim	não	não
Escalável	depende	talvez	aparentemente

Tabela 4.1: Topologias e suas características

4.2 *Streaming Media* e Peer-to-Peer

Como já foi dito, os sistemas de distribuição de *streaming media* baseados em cliente/servidor não são capazes de atender a um grande número de usuários, devido à limitação dos recursos do servidor. Visando aumentar o número de usuários atendidos, foram propostas algumas soluções que vão desde replicação de dados a técnicas de transmissão de dados como *multicast*. Replicação do conteúdo, através da utilização de CDNs, *proxies* e *caches* exige gastos extras com infra-estrutura, sendo adequadas apenas a grandes portais comerciais. Soluções que empregam IP *multicast*, ainda não são viáveis, por causa do número reduzido de roteadores na Internet habilitados a fornecer esse tipo de serviço. Além disso, os ISPs desconhecem os problemas que podem surgir com utilização dessa tecnologia.

Uma solução que vem se tornando popular principalmente por causa da escalabilidade e ausência de custos adicionais com infra-estrutura, são as arquiteturas P2P para *streaming media*. Nesse tipo de arquitetura, é realizado o chamado *multicast* virtual, que é o *multicast* implementado em nível de aplicação. O sistema, então é modelado na forma de árvore de distribuição, que tem como raiz, a fonte do *streaming* e como nodos, os clientes. Cada nodo dessa árvore tem a capacidade de repassar o conteúdo recebido para outros nodos. Assim, um subconjunto de clientes obtém o conteúdo diretamente da fonte e o repassam para o restante dos clientes. Com isso, a capacidade do sistema é ampliada sem a introdução de custos adicionais de infra-estrutura, graças à contribuição dos recursos, principalmente largura de banda dos clientes do sistema.

A grande diferença entre os sistemas tradicionais de P2P e os sistemas de peer-to-peer para *streaming media* é em relação ao modo de compartilhamento entre os *peers*. Os sistemas tradicionais utilizam o modo *open-after-downloading*, ou seja, o arquivo só pode ser exibido, após o cliente recebê-lo por inteiro. Já os sistemas para *streaming media* utilizam o modo *play-while-downloading*, onde os *peers* podem exibir o conteúdo durante a transmissão do arquivo e ainda disponibilizá-lo para outros *peers* [47].

4.2.1 Considerações

Ao se projetar um sistema P2P para *streaming media*, algumas considerações deve ser feitas de acordo com [46, 47]:

- A árvore deve ter uma altura pequena para que os atrasos e perdas acumulativas de cada nodo intermediário não deteriorem a exibição da mídia. Isso, também favorece a eficiência dos mecanismos para descoberta de nodos que irão servir novos nodos.

- Para a utilização eficiente dos recursos da rede e devido à limitação dos recursos de cada receptor, o número de nodos servidos por cada nodo deve ser limitado. Ademais,

um número muito grande de clientes servidos por um nodo provoca atrasos devido à contenção de banda do nodo. Esse tipo de gargalo, geralmente ocorre, quando a árvore tem a forma de estrela, onde a fonte é a raiz.

- O comportamento dos *peers* que recebem o *stream* é imprevisível, portanto o sistema deve ter mecanismos rápidos e eficientes para se recuperar de saídas ou falhas de nodos com descendentes.

- A topologia do sistema deve levar em conta a heterogeneidade de largura de banda dos *peers*, que pode ser devido aos diferentes tipos de redes de acesso ou devido ao desejo de cada peer de contribuir com o sistema.

4.2.2 Arquitetura Implementada

Nesse trabalho, a arquitetura (Figura 4.1) implementada e validada foi baseada em uma topologia hierárquica, que é organizada em níveis. O nível 0 representa o servidor de *streaming media* e o nível 1 corresponde aos clientes servidos diretamente pelo servidor. Assim que os clientes do nível 1 começam a receber o *stream*, eles se tornam servidores potenciais dos clientes do nível 2. Isso ocorre sucessivamente formando uma hierarquia de n níveis. Todos os *servents* envolvidos nos experimentos eram homogêneos, pois estavam na mesma rede local e utilizaram a mesma configuração de *hardware*.

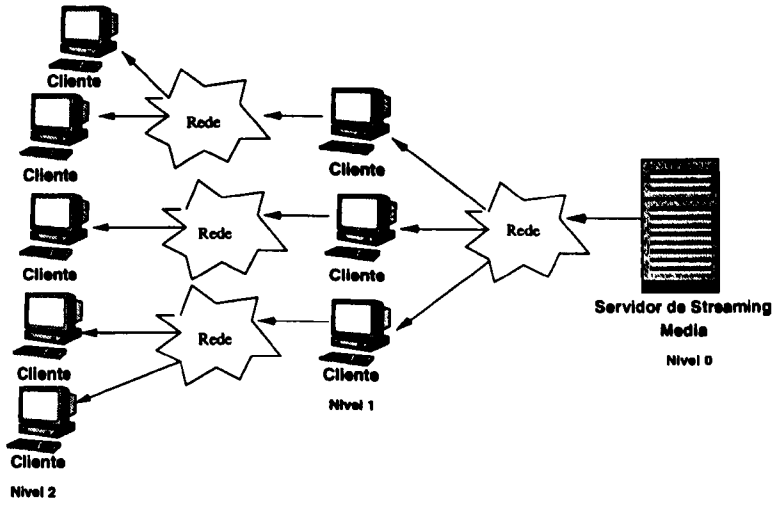


Figura 4.1: Arquitetura P2P para distribuição de streaming media

A arquitetura proposta é baseada em uma arquitetura P2P centralizada, onde o servidor central é a fonte do *stream* e os *servents* utilizam seus próprios recursos para redistribuir o *stream*. Um dos benefícios evidentes é a utilização dessa arquitetura para promoção do compartilhamento de *streams* dentro de um mesmo AS (*Autonomous System*). A arquitetura P2P aliada a distribuição de *streaming media* é um arcabouço para a realização do

multicast em nível de aplicação, dentro de AS que não possuem suporte a *multicast*. A arquitetura experimentada será descrita com mais detalhes no Capítulo 5.

Um elemento importante nessa arquitetura é a forma de conexão entre os *servents*. A descoberta de onde o *servent* deve se conectar para receber o *stream* tem grande impacto na eficiência desta proposta. Entanto, este fato não foi levado em consideração devido ao seu grau de complexidade. A localização dos *servents* será um dos trabalhos futuros a serem desenvolvidos. Será assumido que o *servent* "conhece" a rede e sabe a quem se conectar. Como não há nenhum algoritmo para a distribuição dos *servents*, a árvore de distribuição é sempre balanceada.

4.3 Sumário

Nesse capítulo foi apresentada uma breve definição sobre o que são redes *peer-to-peer*, quais são suas vantagens e suas topologias mais comuns. Foi discutido o funcionamento de uma arquitetura P2P para distribuição de *streaming media* e quais são suas limitações.

A arquitetura P2P é um arcabouço para a realização do *multicast* em nível de aplicação, sem a necessidade de roteadores habilitados e sem a necessidade de máquinas dedicadas a esse propósito. Essa arquitetura utiliza os próprios clientes para redistribuir o *stream*, sempre com a preocupação de não torná-lo um servidor real, ou seja, não saturando seus recursos quando compartilhados. A arquitetura, validada nesse trabalho, é hierárquica, onde os clientes do primeiro nível eram servidos diretamente pelo servidor e o restante era servido por outros *servents*. Para validação da arquitetura foram selecionadas algumas métricas que levaram em consideração o consumo dos recursos do servidor e dos *servents* e a qualidade do *stream* recebido. A metodologia e os resultados desse trabalho serão descritos nos Capítulos 5 e 6.

Capítulo 5

Avaliação da Arquitetura

Este capítulo descreve a metodologia de avaliação do desempenho para verificação da escalabilidade de serviços de distribuição de *live streaming media* em arquiteturas do tipo cliente/servidor e P2P. A escalabilidade nesse tipo de serviço significa aumentar o número de clientes atendidos sem que a qualidade do *stream* recebido por eles seja degradada. Esse problema não é trivial, já que o servidor de *streaming media* possui restrições de tempo-real e requer altas taxas de transferência de dados.

Nesse capítulo, também são apresentados: as aplicações utilizadas nos experimentos: o servidor de *streaming media*, o *streaming servent* implementado e o coletor de métricas; os cenários ou configurações experimentados e as métricas de desempenho escolhidas.

5.1 Aplicações

Uma grande parte do processo de experimentação foi gasta entendendo o funcionamento do servidor Darwin, seus protocolos e no desenvolvimento do *streaming servent*. A criação do *servent* foi necessária, pois não existia nenhum software de recebimento de *stream* com as características necessárias e que fosse de código aberto para que pudessem ser inseridas monitorações. A implementação do *servent* foi feita através de programação *multithread* em linguagem C.

5.1.1 *Darwin Streaming Server*

Para distribuir o *stream inicial*, foi utilizado o servidor *Darwin Streaming Server* [1], versão 4.0 para Linux. Darwin é um projeto *Open Source* desenvolvido pela Apple que também inclui um conjunto de ferramentas e bibliotecas de código aberto. Uma dessas ferramentas é o *Darwin Streaming Server*, que é um servidor de distribuição de *streaming* orientado a eventos e que utiliza os protocolos padrão de distribuição de *streaming* RTSP,

RTP e RTCP (descritos no Capítulo 3). O motivo da escolha desse servidor foi a ausência de outras alternativas de servidores de *streaming media* comerciais de código aberto, durante o período de realização dos experimentos e por ter sido utilizado nos experimentos da arquitetura *SpreadIt* [16]. Não serão tratados neste trabalho, detalhes sobre a implementação e funcionamento do Darwin.

Darwin possui dois modos de operação, a distribuição do conteúdo através de *broadcasts* e sob demanda. Quando o conteúdo é transmitido através de *broadcasts*, o número de pacotes enviados para cada cliente é dependente do momento em que foi feita a conexão. O cliente recebe o arquivo a partir do momento em que se conectou. No modo de operação sob demanda, todos os clientes são servidos com a transmissão completa do arquivo. Ou seja, a cada nova conexão é iniciado um novo *stream*.

Nos experimentos, todos os processos clientes são iniciados aproximadamente ao mesmo tempo. No entanto, o momento em que cada cliente realimente começa a receber dados depende de quão rápido o Darwin é capaz de processar todas as requisições. Assim, foi observado experimentalmente que o número de pacotes enviados para cad cliente podia variar significativamente. Para determinar esse número, que é necessário para o cálculo do número de pacotes perdidos para cada cliente, uma das métricas importantes para escalabilidade do sistema seria necessário instrumentar o código do servidor. Para evitar que a inclusão dessa instrumentação no servidor alterasse seu desempenho, foi escolhido o modo sob demanda para a realização dos experimentos. Apesar da utilização do modo sob demanda, os clientes que não estão conectados ao Darwin recebem o *stream* a partir do momento em que se conectaram. A escolha por esse modo de operação é somente para se ter controle do número de pacotes enviados para cad cliente.

5.1.2 Arquitetura e Desenvolvimento do *Streaming Servent*

Para implementar e experimentar com as arquiteturas cliente/servidor e P2P havia a necessidade de uma aplicação que atuasse com o cliente e servidor ao mesmo tempo, redirecionando os pacotes para os outros clientes. Inicialmente os experimentos seriam realizados com o aplicativo criado para a arquitetura SpreadIt. Mas, alguns experimentos com esse aplicativo desenvolvido em Python, revelaram uma grande perda de pacotes quando o cliente requisitava um vídeo do Darwin. Também, foi cogitada a utilização de servidores de *proxies* já desenvolvidos como o Quicktime Streaming Proxy [7]. No entanto, a alta complexidade desses sistemas, desnecessária a uma aplicação cliente P2P e alguns problemas de desempenho descritos [20] foram motivações para a construção de uma nova aplicação, o *streaming servent*. O objetivo do desenvolvimento desse *servent* era torná-lo o mais eficiente e simples possível. Em particular, como o objetivo desse trabalho é experimentar a distribuição de *live streaming media*, o *servent* construído, atua

apenas como redistribuidor de pacotes e não armazenando o conteúdo recebido localmente. No entanto, essa aplicação é altamente modular facilitando futuras extensões, como por exemplo, módulos para *caching*.

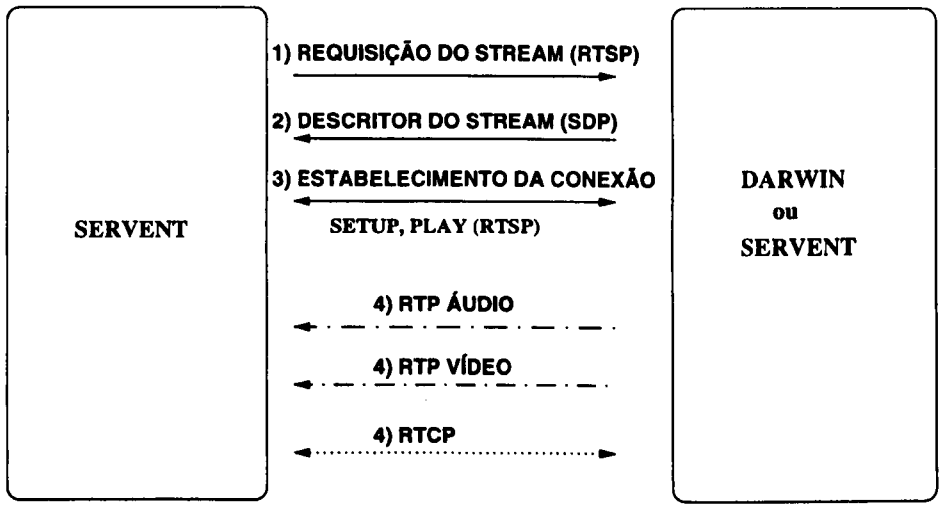


Figura 5.1: Conexão do *servent*

Os *servents* comunicam entre si e com o servidor Darwin através dos protocolos RTSP, RTP e RTCP. O *servent* recebe como parâmetros de entrada o endereço IP, a porta RTSP do servidor ou *servent* ao qual deve ser conectar, o nome do arquivo da mídia requisitada e as portas RTP/RTCP locais que serão utilizadas para receber o *stream*. Durante a execução, todos os pacotes RTP recebidos ou enviados são registrados em um *log*. Nesse *log* são encontradas informações como: o número de seqüência, o tamanho em bytes e o *timestamp* de recebimento ou envio de cada pacote. Tais informações são utilizadas na análise dos experimentos, para calcular o número de pacotes perdidos e a distribuição da duração da perda de pacotes observada por cada *servent* em cada experimento.

A figura 5.1 mostra as mensagens trocadas entre o *servent* e o Darwin para estabelecimento de uma conexão para a transmissão de um novo *stream*. As mesmas mensagens são trocadas entre dois *servents*. O processo se inicia com a interação de mensagens RTSP: o *servent* envia uma mensagem RTSP de requisição do *stream* ao servidor Darwin (1). Este responde enviando um arquivo descritor (SDP), que contem informações sobre o *stream* a ser transmitido (2). De posse do descritor do *stream*, o *servent* e o Darwin trocam de mensagens de SETUP para alocação das portas RTP/RTCP que serão utilizadas nas duas entidades (*servent* e servidor). No caso da requisição ser por um vídeo, o servidor Darwin transmite os *streams* de áudio e vídeo separadamente, alocando assim, dois pares de portas. Após o recebimento das mensagens de SETUP, o *servent* envia a mensagem de PLAY que indica que a transmissão pode iniciar (3). Depois de recebidas as mensagens de PLAY, são criadas duas *threads* para o recebimento e envio simultâneo de cada um dos *streams*. A *thread* do programa principal se mantém ativa, aguardando mensagens do servidor ou

de outros *servents* na porta RTSP. Caso algum outro *servent* tente se conectar a ele, o processo de troca de mensagens é repetido, porém este *servent* está no papel de servidor do *stream*. Os *streams* de dados e controle são trocados entre as entidades utilizando os protocolos RTP e RTCP (4).

Os *servents* conectados a um outro *servent* o consideram como se este fosse a fonte do *stream*. Toda a comunicação entre *servents* ou entre o Darwin e um *servent* é feita da mesma maneira. Um *servent* conectado a outro está sujeito a todos os problemas de atraso e perda de pacotes sofridos pelo *servent* do nível acima. Quando um *servent* perde um pacote, todos os *servents* descendentes dele não irão recebê-lo.

Para verificar a qualidade do *stream* recebido de forma visual, foram feitos alguns testes durante o experimento, conectando aos diversos níveis, clientes QuickTime. O QuickTime [6] foi utilizado para exibir o conteúdo do *stream* transmitido. Esse tipo de teste foi feito para observar se as métricas de qualidade refletiam o que estava sendo exibido, se a perda de pacotes relatada nos *logs* de cada cliente, estaria sendo fator determinante para a queda da qualidade do vídeo.

Não houve nesse trabalho a preocupação com o problema de descoberta de recursos. Para o estabelecimento da sessão, os *servents* recebiam a localização dos *servents* a que iriam se conectar. Foi assumida, uma configuração estática sem entrada e saída de clientes do sistema durante o experimento.

5.1.3 Coletor de Métricas

Esse módulo é responsável por coletar dados sobre a utilização dos recursos do servidor e dos *servents* a cada intervalo de tempo (1 segundo). Os dados sobre CPU foram coletados do arquivo /proc/stat do Linux, onde foi extraído a utilização da CPU no modo usuário, *nice* e sistema. Para coleta dos dados de largura de banda, foi utilizado o arquivo de *log* mantido em /proc/net/dev do Linux que contém informações sobre o número de pacotes e bytes recebidos e enviados em cada interface (a análise foi feita na interface *eth0*).

5.2 Descrição dos Experimentos

A validação da arquitetura proposta foi feita através de experimentos com o *servent* criado. Durante esses experimentos eram coletadas informações sobre a utilização das máquinas e a qualidade dos *streams* recebidos pelos *servents*. Todos os experimentos foram realizados, utilizando-se sete PCs(em uma dessas maquinas estava o servidor Darwin) com a mesma configuração: processador AMD Duron 750MHz, 256MB de memória RAM, 512MB de swap com sistema operacional Linux Red Hat 7.3 (kernel 2.4.18-3). Todas as máquinas eram conectadas em rede local através de um *switch Fast Ethernet*. É importante ressaltar

que a rede estava isolada não havendo outro tipo de tráfego a não ser o dos *streams*. A configuração do experimento é mostrada na Figura 5.2 abaixo.

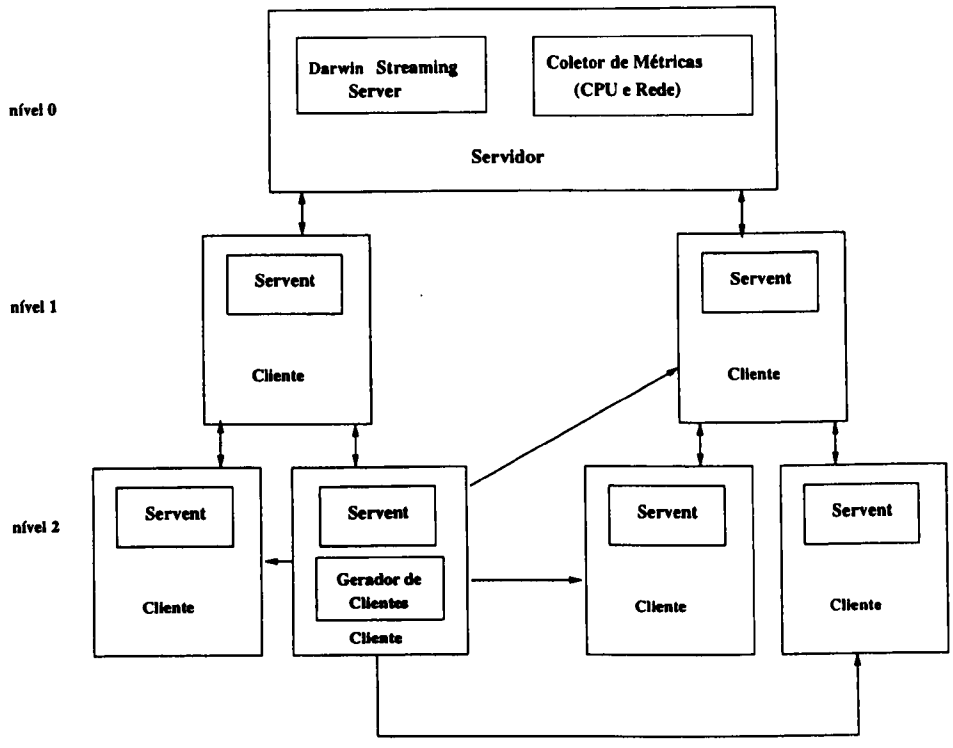


Figura 5.2: Diagrama de módulos dos experimentos

Em uma das máquinas foi instalado o servidor Darwin que era executado juntamente com o coletor de métricas, que fazia a medição da utilização de CPU e rede. Nas outras seis máquinas foram executados os *servents* implementados nesse trabalho. Uma máquina do nível 2 era responsável por disparar os *scripts* de criação de *servents* em cada uma das seis máquinas. No começo de cada experimento, era construída uma árvore estática de distribuição de um nível para as configurações cliente/servidor e de dois níveis para as configurações P2P.

A Figura 5.3 apresenta a distribuição do *stream* para 12 clientes na arquitetura P2P utilizada nos experimentos. Em todas as máquinas eram executados o mesmo número de clientes. Todos os clientes do nível 1 redistribuíam o *stream* recebido do servidor Darwin, para 2 clientes do nível 2, qualquer que fosse o número de clientes. Por exemplo o cliente C_1 da máquina M_1 redistribuía o *stream* para os clientes C_5 e C_6 da máquina M_3. A maior limitação de se colocar um número de clientes muito grande nas máquinas, foi a limitação das portas que eram alocadas para a transferência do *stream*.

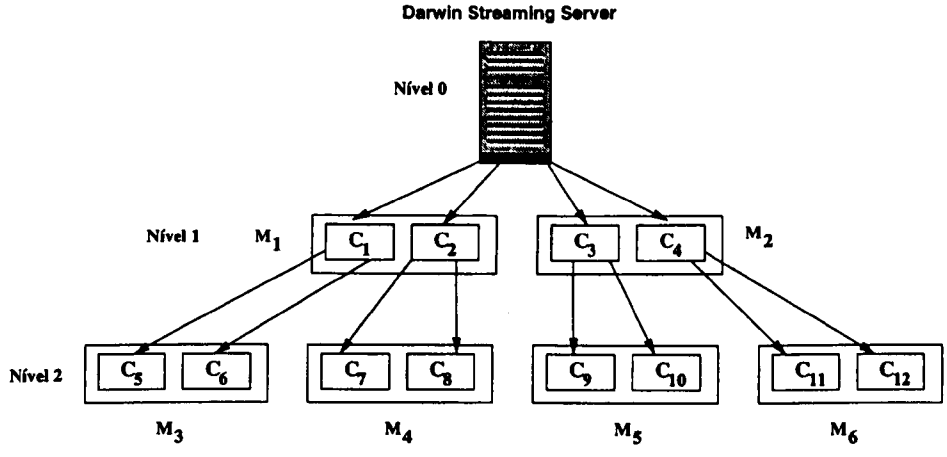
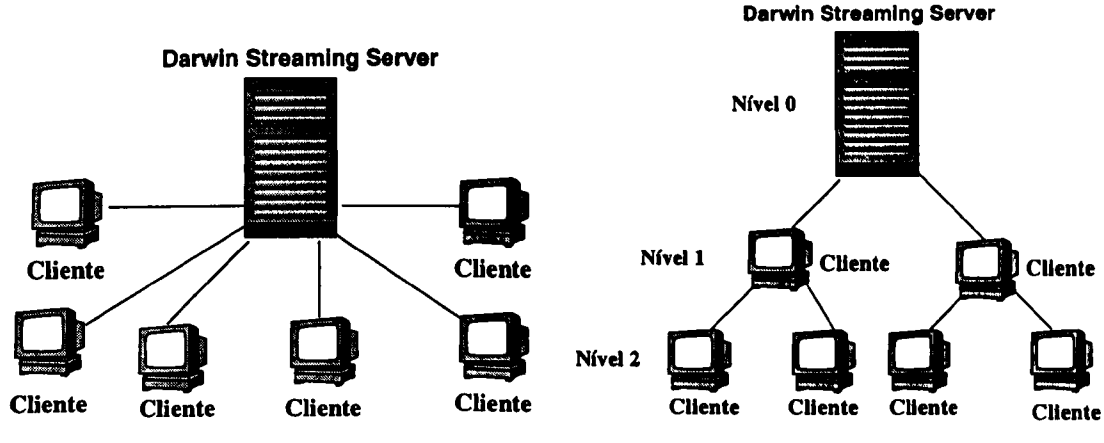


Figura 5.3: Diagrama de distribuição dos *streams* na arquietura P2P

5.2.1 Cenários Experimentados

Para validar a arquitetura proposta foram feitos experimentos em dois tipos de cenários: um utilizando a arquitetura tradicional cliente/servidor, onde todos os clientes recebiam o vídeo diretamente do servidor de *streaming media* (Figura 5.4(a)) e outro utilizando a arquitetura P2P proposta (Figura 5.4(b)). Foi feita também a avaliação dos recursos do *servent* quando esse está servindo a outros *servents*.



(a) Topologia cliente/servidor

(b) Topologia Peer-to-Peer

Figura 5.4: Topologias Experimentadas

Em ambos cenários foi adotada uma topologia hierárquica, que é organizada em níveis. O nível 0 representa o servidor de *streaming media* e o nível 1 corresponde aos clientes atendidos diretamente pelo servidor. Na arquitetura cliente/servidor, todos os clientes se encontram no nível 1. Na arquitetura P2P, foi considerado que todos os clientes do nível 1 são servidores potenciais e que os clientes servidos por esse nível correspondem ao nível 2.

A partir do momento que os clientes do nível 2 começarem a receber o *stream*, eles também se tornam servidores potenciais, e assim por diante. Para avaliação dessas arquiteturas o número de *servents* era aumentado gradativamente em cada máquina. Para cada grupo de n *servents* era analisada a carga do servidor e a qualidade do *stream* recebido.

Para avaliar o impacto no *servent* da redistribuição do *stream* foram feitos experimentos para medição da utilização dos recursos de uma das máquinas clientes. Essa máquina executava um único *servent* que se conectava ao servidor e a n outros *servents*, como pode ser visto na Figura 5.5. Para cada número n de *servents* foram feitas leitura de utilização de CPU e rede no *servent* do nível 1. O objetivo desses experimentos é calcular experimentalmente o *fan-out* do *servent*, ou seja, calcular o número máximo de clientes, que um *servent* pode servir sem saturar seus recursos.

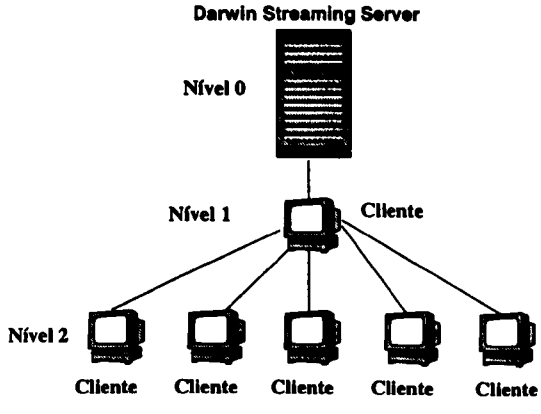


Figura 5.5: Topologia para avaliação do *servent*

5.2.2 Geração de Carga

A cada experimento era executado um número n de *servents*, homogeneamente distribuídos entre 6 máquinas, que recebiam a mídia simultaneamente. Esses *servents* ficavam conectados até que recebessem todo o *stream* e durante esse tempo, eram coletados os dados sobre a utilização dos recursos. Terminado o recebimento da mídia, o experimento era repetido para um número maior de clientes e novamente os dados eram coletados.

Nesse estudo, também foi investigado o comportamento do sistema sobre três mídias diferentes (dois vídeos VBR (*variable bitrate*) e um arquivo de áudio) em cada uma das arquiteturas. Esses arquivos possuem características que cobrem uma variedade de tamanhos, duração, (*bitrate*) e como será mostrado na seção 6, variabilidade no (*bitrate*) instantâneo. É importante ressaltar que as mídias foram escolhidas cuidadosamente para representar diferentes classes de objetos de mídia que estão na Internet atualmente. Além disso, a diversidade das características dos arquivos selecionados estressa diferentes aspectos da operação do sistema. Assim, a experimentação com esses arquivos permite explorar

diferentes cenários que ocorrem em sistemas reais. Essas mídias são caracterizadas de acordo com a Tabela 5.1.

Tipo	Formato	Frame rate	Tamanho (MB)	Tamanho (min)	Bitrate (kbytes/s)	# de pacotes RTP
video clip	mov	15	60	04:38,16	217,6	44772
trailer	mov	30	22	03:31,17	101,9	18311
música	mov	-	8	09:54,13	12,5	6193

Tabela 5.1: Vídeos

5.3 Métricas

Para comparação quantitativa entre os dois cenários, foram escolhidas algumas métricas consideradas relevantes no desempenho dos dois tipos de arquitetura.

5.3.1 Métricas do lado servidor

As métricas de desempenho escolhidas na avaliação do servidor e do *servent* foram:

- **Utilização da CPU**: Foi medida através da porcentagem do tempo de execução gasto pelos estados user + nice + system durante o tempo do experimento. Essa medição foi realizada através da leitura do /proc das máquinas. Foram consideradas a utilização média de CPU (medida para todo o experimento) e a utilização instantânea da CPU (medida em intervalos de 1 segundo).

- **Largura de banda de rede utilizada pelo servidor ou pelo *servent***: Foi medida como sendo a média do número de bytes e pacotes enviados e recebidos, enquanto o *stream* estava sendo distribuído. Essa medição foi realizada através da leitura do /proc das máquinas.

5.3.2 Métricas do lado cliente

No lado do *servents* foram consideradas as seguintes métricas:

- **Porcentagem média de pacotes recebidos**: A porcentagem da mídia recebida foi calculada como sendo a fração do número de pacotes recebidos e o número de pacotes que ele deveria ter recebido. Por exemplo, se um cliente deveria ter recebido 100 pacotes e recebeu somente 90, sua qualidade foi de 90%. No caso do P2P, para o cálculo da qualidade foram utilizados os *logs* dos pacotes RTP recebidos e enviados dos níveis em questão e do nível anterior. Assim, para o cenário P2P foi levada em

conta a perda acumulada nos dois níveis. Isso é feito para cada cliente e depois é feita uma média pelo número de clientes simultâneos.

- **Taxa de bytes recebida**: foi avaliada a quantidade média de bytes recebida por segundo. Essa métrica visa mostrar se o vídeo consegue manter a taxa durante a transmissão.

- **Duração dos eventos de perda de pacotes**: é a distribuição da "duração" dos eventos de perda, medida como sendo o número de pacotes consecutivos perdidos em cada evento.

- **Atraso médio por nível**: Foi definido como sendo o tempo médio de envio de um pacote RTP e o recebimento desse mesmo pacote por outro nodo. Esse experimento foi realizado em uma topologia em cadeia e para que não houvesse problemas com os diferentes *clocks* das máquinas testadas, os pacotes RTP eram repassados por todos nodos da cadeia, sendo que o último nodo o reenviava para o servidor.

Essas métricas foram utilizadas para estimar indiretamente a qualidade do *stream* recebido por cada cliente. A qualidade da mídia percebida por um *servent* é diretamente relacionada ao número de pacotes recebidos do *stream*, comparado ao número total de pacotes. Reciprocamente, a qualidade da mídia, também está diretamente relacionada ao número de pacotes perdidos durante uma transmissão e também ao número de pacotes consecutivos que são perdidos em cada evento de perda em outras palavras a "duração" do evento de perda.

5.4 Sumário

Nesse capítulo foram descritas as aplicações utilizadas nos experimentos realizados: o servidor de *streaming media* Darwin, o *streaming servent* implementado para a redistribuição do *stream* e o coletor de métricas que era responsável, por extrair informações das máquinas envolvidas nos experimentos a respeito da utilização de seus recursos e da qualidade do *stream* recebido.

A fim de validar a arquitetura proposta, foram realizados experimentos com a arquitetura cliente/servidor, para que se pudesse comparar com a arquitetura P2P. As métricas selecionadas levaram em consideração os dois cenários e três tipos de mídias com taxas de transmissão diferentes.

Capítulo 6

Resultados Experimentais

Esse Capítulo provê a avaliação quantitativa da escalabilidade das arquiteturas cliente/servidor e P2P para a distribuição de *live streaming media*. Essa escalabilidade é medida em termos da utilização dos recursos do servidor e em termos da perda de pacotes, a qual tem impacto direto na qualidade do *stream* recebido por cada cliente. Além disso, é apresentada uma avaliação da utilização dos recursos do *servent*, em função do número de clientes, que são servidos por ele (*fan-out* do *servent*). No final desse Capítulo são mostradas fórmulas analíticas que podem ser utilizadas para estimar a escalabilidade do *servent* e determinar o *fan-out* máximo do *servent*, dado a quantidade de recursos que ele deseja dedicar para servir outros clientes.

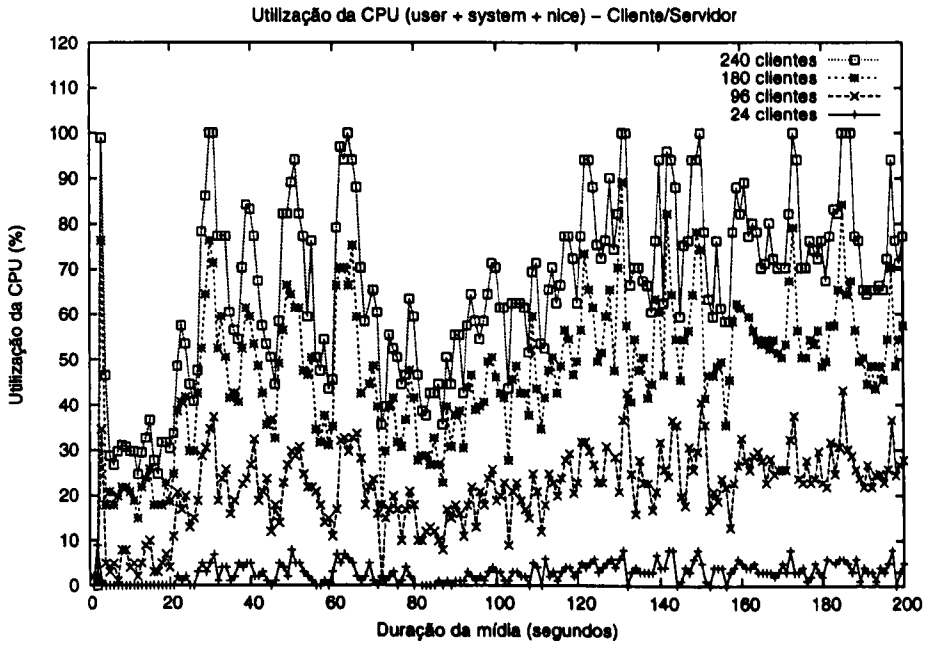
6.1 Avaliação de Carga no Servidor

A avaliação da carga no servidor foi feita considerando a utilização de CPU e o consumo de largura de banda de rede, em função do número de clientes simultâneos que requisitavam a mídia. A preocupação foi observar como é o crescimento dessa utilização e se algum desses recursos poderia saturar. A monitorações foram feitas na máquina onde estava o servidor Darwin. A fim de verificar diferentes comportamentos no servidor, foram utilizados os três tipos de mídias descritas no capítulo anterior.

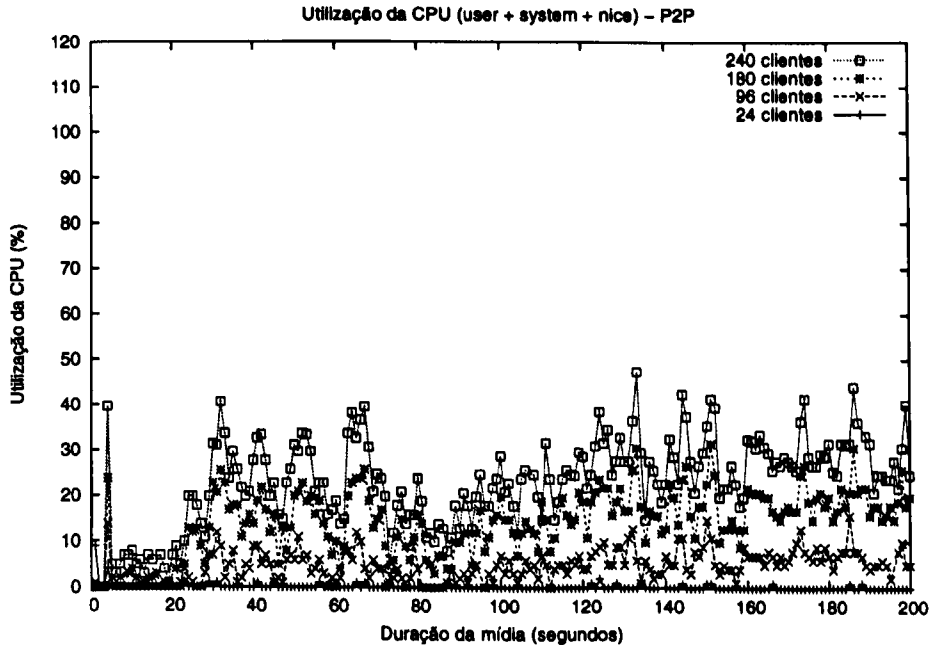
6.1.1 Utilização de CPU

Os gráficos das Figuras 6.1, 6.2, e 6.3 apresentam a utilização instantânea da CPU em função do número de clientes durante a sessão de transmissão do *stream*.

Para o *trailer* mostrado no gráfico 6.1(a) e 6.1(b), observa-se um grande número de picos e vales que correspondem a troca de cenas, com cenas de ação seguidas por cenas compostas quase que totalmente por telas escuras. Os picos e vales acontecem nos mesmos instantes,

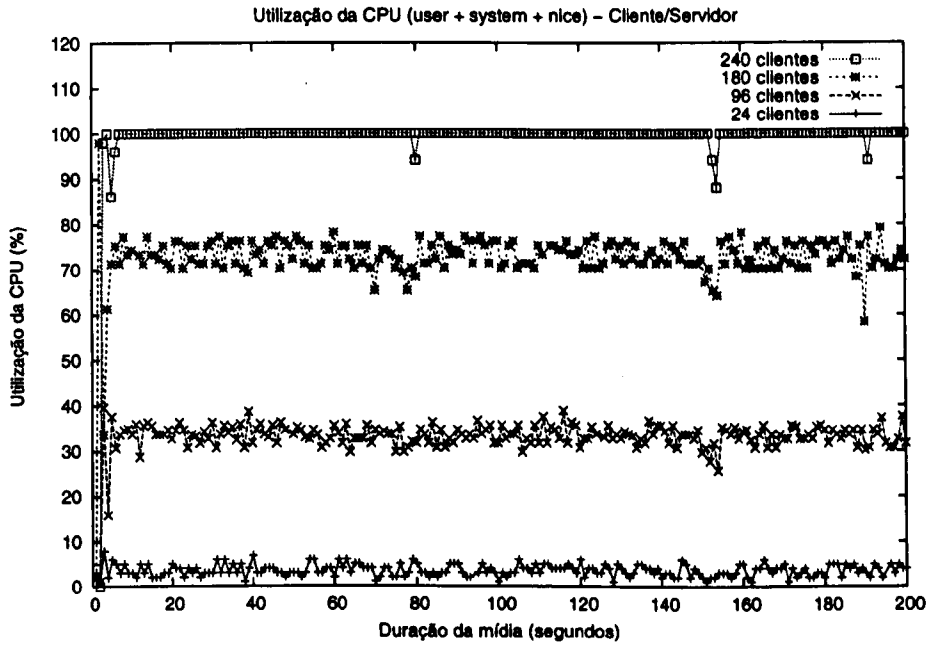


(a) Cliente/Servidor

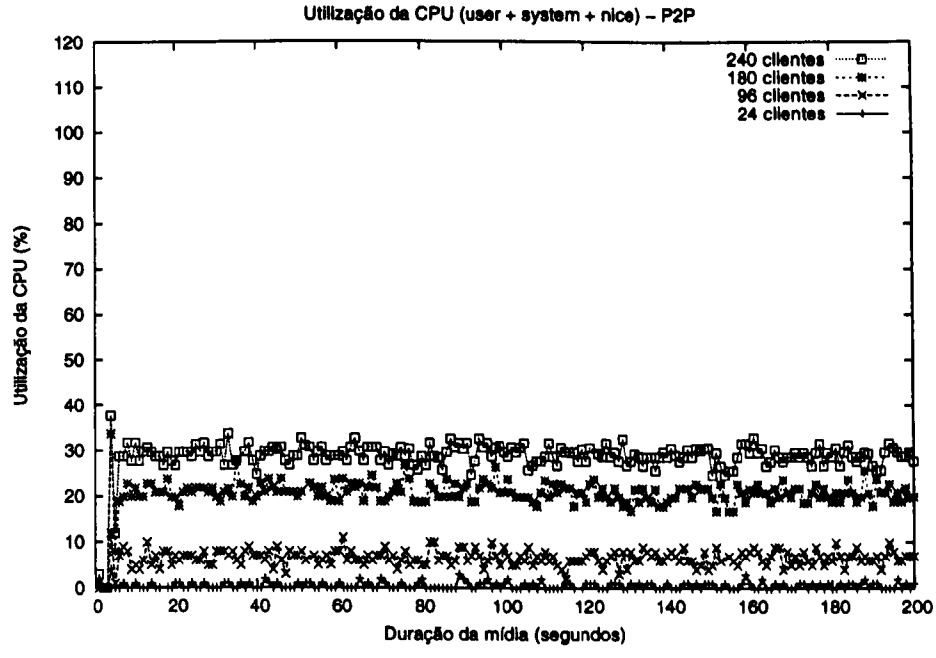


(b) P2P

Figura 6.1: Utilização Instantânea de CPU do servidor para o trailer

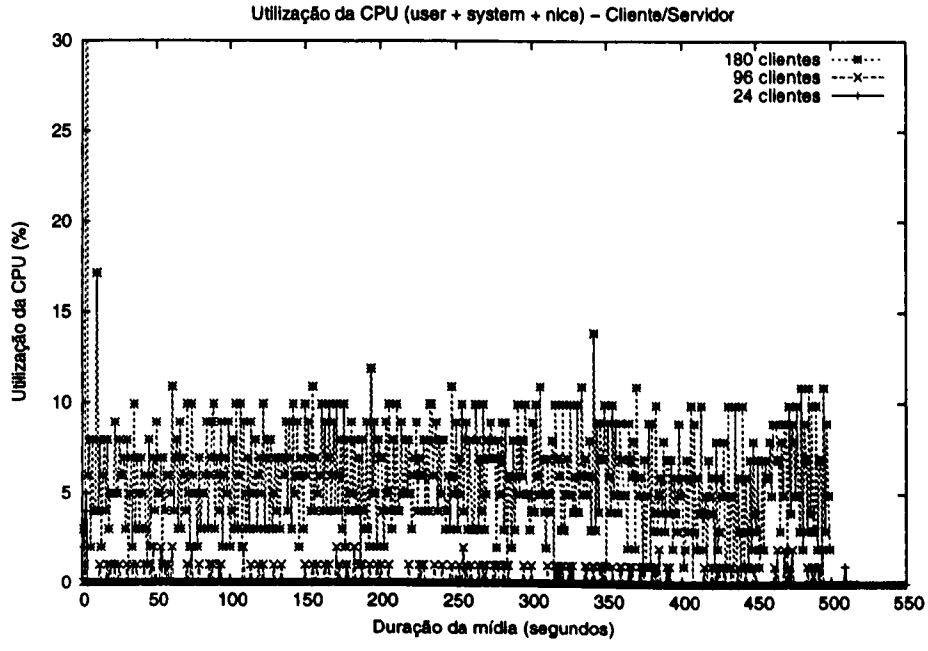


(a) Cliente/Servidor

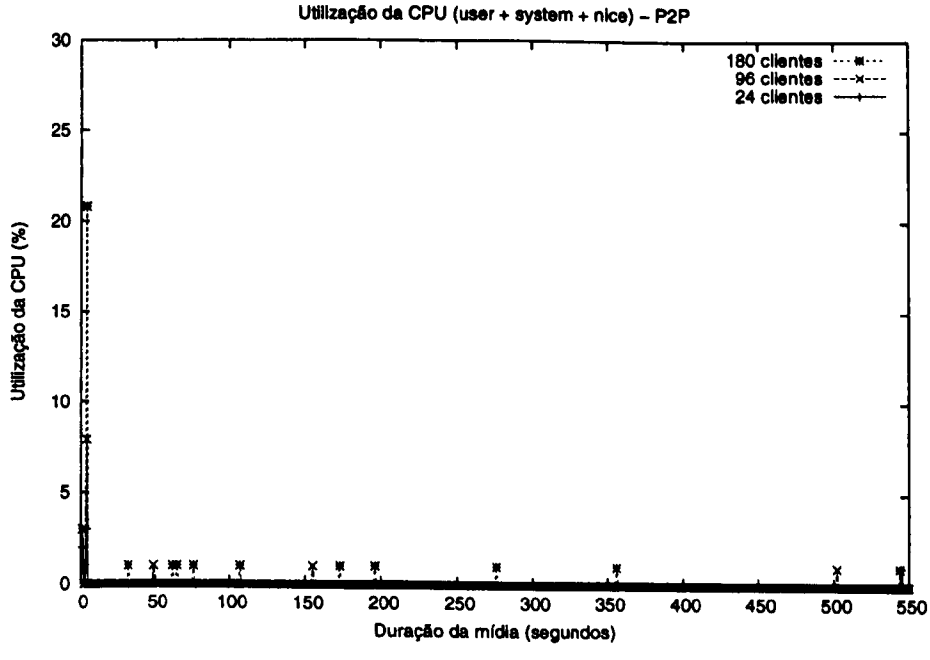


(b) P2P

Figura 6.2: Utilização Instantânea de CPU do servidor para o video clip



(a) Cliente/Servidor



(b) P2P

Figura 6.3: Utilização Instantânea de CPU do servidor para a música

mas sua intensidade depende do número de clientes. Nota-se, que para a arquitetura cliente/servidor, há alguns picos de 100% de utilização, o que sugere que se houver um número maior de clientes para esse vídeo, a saturação desse recurso poderá ocorrer. Para o *video clip* (gráficos 6.2(a) e 6.2(b)), a utilização instantânea da CPU sofre menores variações que o *trailer*, por se tratar de um vídeo com menor troca de cenas. No entanto, por possuir um taxa média elevada (200 kbytes/s), a utilização de CPU do servidor fica acima de 90% para 240 clientes durante quase toda a transmissão, na arquitetura cliente/servidor. No caso da música (gráfico 6.3(a) e 6.3(b)), por não haver grandes variações na taxa de transmissão, a utilização instantânea de CPU é quase constante. A utilização pode ser considerada baixa, já que fica em torno de 10% quando está servindo 180 clientes.

Esses resultados mostram a diversidade das cargas escolhidas para os experimentos e a diminuição significativa da utilização de CPU se for adotada uma arquitetura P2P. A redução da utilização da CPU do servidor no caso do *video clip*, para 240 clientes foi de 100% para 30%. Isso foi obtido, utilizando 80 *servents* acessando o servidor diretamente e redistribuindo o *stream* para mais dois *servents*. Com 240 clientes, podemos observar que a CPU nunca atinge picos de mais 50% de utilização na arquitetura P2P, ou seja, os picos de saturação acontecerão com um número maior de clientes nessa arquitetura do que na arquitetura cliente/servidor. A média da utilização da CPU em função do número de clientes é mostrada na Figura 6.4. A utilização média da CPU para a música foi omitida por ser próxima a zero. A utilização média da CPU nos *servents* do primeiro nível é avaliada na Seção 6.3 desse Capítulo.

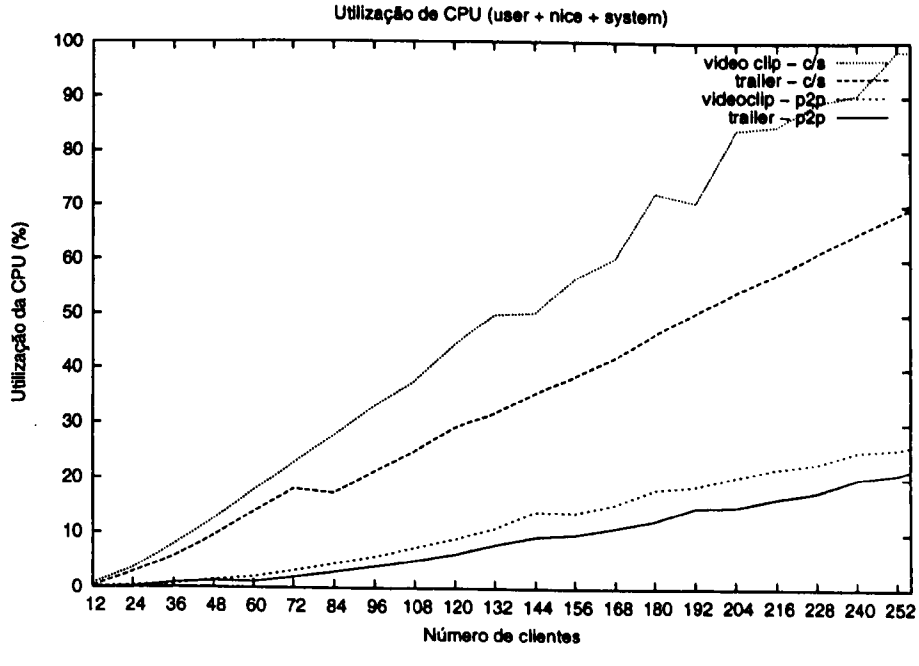


Figura 6.4: Utilização média de CPU do servidor

6.1.2 Largura de Banda de Rede Utilizada pelo Servidor

O gráfico na Figura 6.5 mostra a intensidade do tráfego em Mbits/s, para as três mídias, em função do número de clientes que estão conectados no sistema.

Todas as curvas mostram um crescimento aproximadamente linear como o aumento do número de clientes. No caso do *vídeo clip*, que foi a mídia que utilizou maior largura de banda, a economia ao se utilizar uma arquitetura P2P foi de quase 50%.

As inclinações das retas P2P e cliente/servidor em todas as mídias sugerem que a saturação irá ocorrer com um número maior de clientes no primeiro nível que o número de clientes na arquitetura P2P. Além disso, as inclinações da curva cliente/servidor são maiores que as inclinações das curvas P2P correspondentes. Isso demonstra que um aumento no número de clientes na arquitetura cliente/servidor causa um aumento relativo maior no consumo da largura de banda do servidor do que na arquitetura P2P.

Podemos observar que a demanda por banda de rede cresce linearmente com o número de clientes e que o uso da arquitetura P2P reduziu o consumo de banda por um fator maior que 2. Apesar de não ter sido experimentado até a saturação da banda do servidor, podemos observar que a arquitetura P2P possibilita uma grande economia de banda.

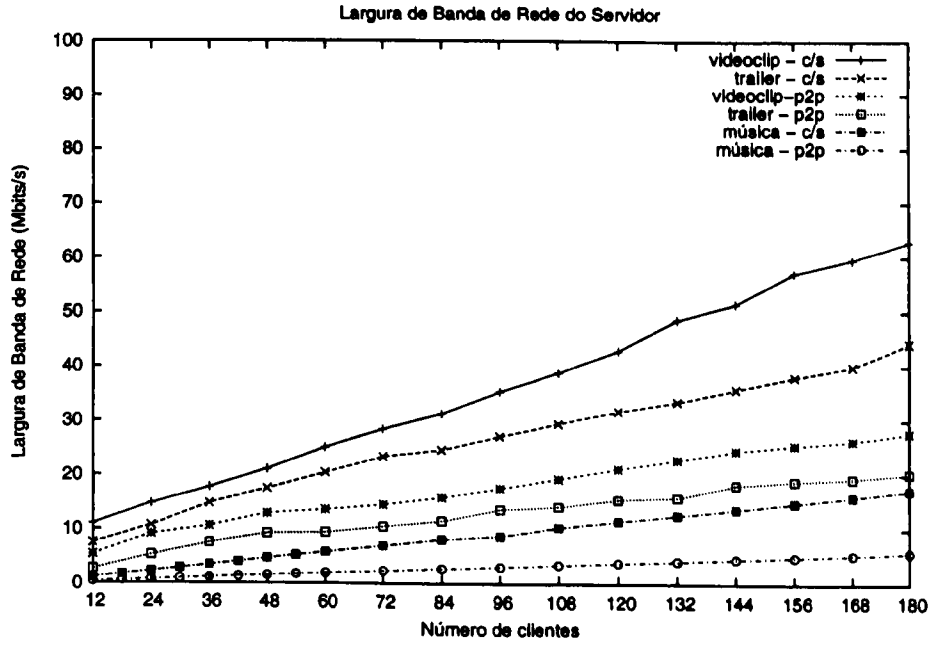


Figura 6.5: Largura de Banda de Rede do Servidor

6.2 Avaliação do Desempenho do Cliente

Essa seção apresenta os resultados que mostram a escalabilidade das arquiteturas cliente/servidor e P2P em termos da qualidade da mídia recebida pelos clientes. Como foi

discutido no Capítulo5, a qualidade da mídia não foi medida diretamente. A medição da qualidade foi feita utilizando o número de pacotes recebidos (ou perdidos) e a distribuição da duração dos eventos de perda de pacotes (número de pacotes consecutivos perdidos em cada evento de perda). A duração dos eventos de perda é importante, já que o impacto de perdas em rajadas pode influenciar na qualidade do *stream*.

6.2.1 Avaliação da porcentagem média de pacotes recebidos

Os gráficos da Figuras 6.6, 6.7 e 6.8 mostram a porcentagem média de pacotes recebidos por cada cliente para cada mídia, em função do número de clientes. Os *streams* estão divididos em áudio e vídeo, porque o servidor Darwin os transmite separadamente.

Esses resultados mostram que a porcentagem do número de pacotes recebidos, e conseqüentemente a qualidade da mídia observada pelos clientes decrescem com o número de clientes. Ou seja, o número de pacotes perdidos cresce com a carga no servidor e na rede.

A distribuição com P2P obteve melhores resultados do que a distribuição com cliente/servidor em todos os momentos como na avaliação do servidor. No caso da música, o ganho não foi grande por causa da baixa taxa de transmissão da mídia. Para os vídeos experimentados, a curva dos pacotes recebidos da arquitetura P2P mostra uma queda suave e gradativa com o aumento do número de clientes, enquanto que a curva da arquitetura cliente/servidor mostra uma queda rápida tendendo a se estabilizar. A arquitetura P2P provê uma qualidade de áudio e vídeo, significantemente melhor, em torno de 70% na média no caso dos vídeos.

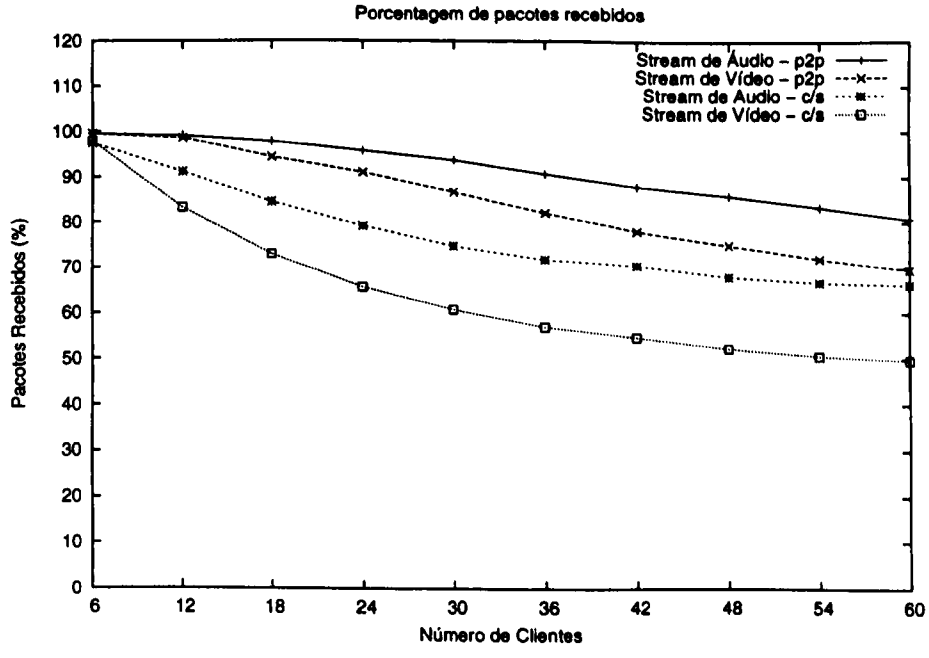


Figura 6.6: Porcentagem de pacotes recebidos, na média, por cada cliente em função do número de clientes - Trailer

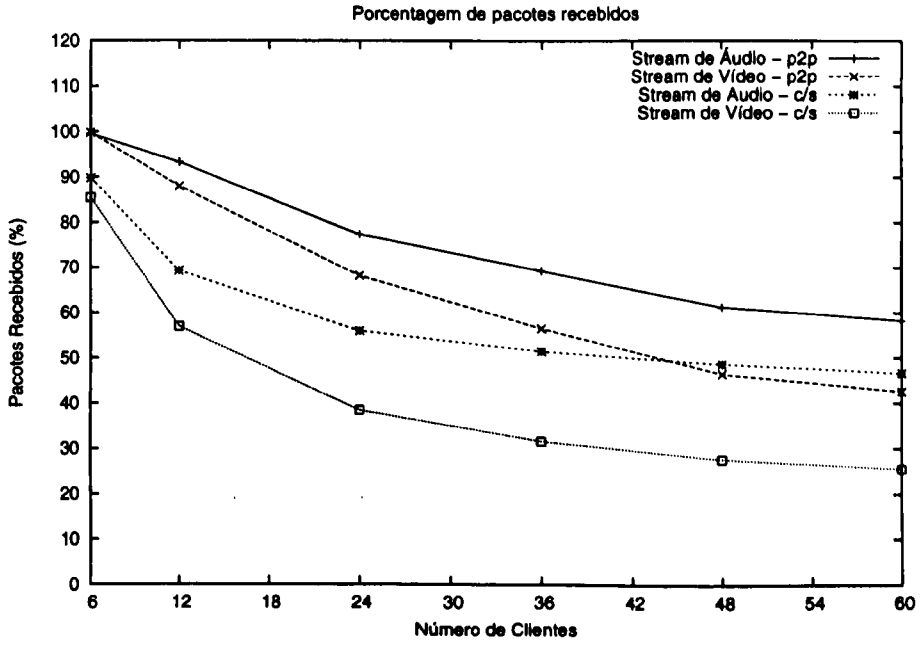


Figura 6.7: Porcentagem de pacotes recebidos, na média, por cada cliente em função do número de clientes - Video clip

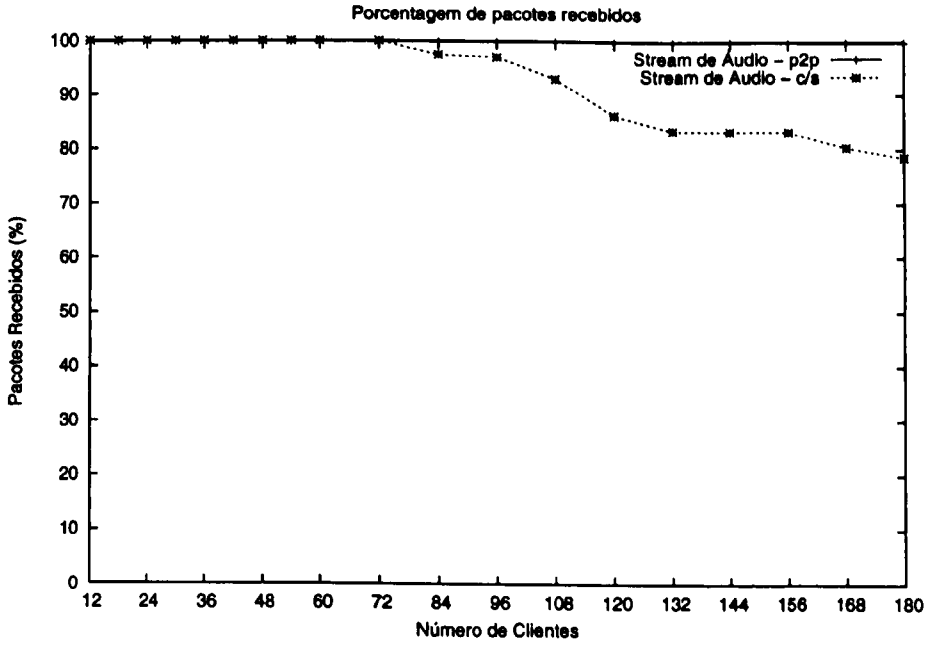


Figura 6.8: Porcentagem de pacotes recebidos, na média, por cada cliente em função do número de clientes - Música

É interessante observar, que devido à alta taxa do *video clip*, a porcentagem de pacotes recebidos é maior que a do *trailer*, para um número fixo de clientes. Além disso, é importante ressaltar a perda de pacotes significante, especialmente para a arquitetura cliente/servidor com um número pequeno de clientes. Foram feitos uma série de experi-

mentos com *tcpdump* [10] e verificou-se que em muitos casos, muitos pacotes não eram nem mesmo transmitidos pelo servidor. Presume-se que algoritmos internos do Darwin, como os algoritmos de *thinning*, que cancelam a transmissão de pacotes que estejam fora de seu *deadline*, sejam responsáveis pelo grande número de perda de pacotes, mesmo com um número reduzido de clientes.

6.2.2 Avaliação da Taxa Recebida

Os gráficos da Figura 6.9, 6.10 e 6.11 mostram o comportamento das taxas em kbytes/s para cada cliente, das três mídias analisadas durante a transmissão do *stream*. Em cada gráfico foi plotado, também, a taxa instantânea de recepção de um único cliente, que corresponde à taxa instantânea da mídia, já que nenhum pacote foi perdido durante experimentos com 1 cliente. Esses gráficos mostram uma outra visão da melhoria na escalabilidade quando a arquitetura P2P é adotada.

Para o *trailer* (Figura 6.9) observa-se que a arquitetura P2P consegue sustentar uma taxa muito próxima da taxa do arquivo original (1 cliente), não somente na média, como mostrado na Figura 6.6, mas durante toda a transmissão do arquivo. Mesmo com um número maior de clientes, a taxa tem uma degradação menor que com a arquitetura cliente/servidor. Pode-se observar também a alta variabilidade na taxa instantânea do *trailer*, consistentemente ao que foi mostrado na Figura 6.1. Em particular, nota-se que aproximadamente depois de 75 segundos do começo da transmissão, a taxa do arquivo cai significativamente. Visualmente, isso corresponde a uma tela escura com poucos elementos no vídeo e quase nenhum som. Ambas as arquiteturas cliente/servidor e P2P foram capazes de distribuir quase todos os pacotes correspondentes a essa cena, mesmo com um número maior de clientes.

No caso da música por ter uma taxa de codificação menor que o dos vídeos não houve grandes ganhos na arquitetura P2P. Acredita-se que com um número maior de clientes, onde a perda for mais pronunciada, o ganho irá ser maior.

As métricas da taxa de recebimento e da porcentagem de pacotes são complementares, como podemos observar nos gráficos correspondentes de cada mídia. No caso do trailer na arquitetura cliente/servidor, para 60 clientes, a taxa de recebimento é quase a metade da taxa de recebimento do vídeo com 1 cliente. Isso se deve, à perda de pacotes observada no gráfico 6.6.

6.2.3 Duração da Perda de Pacotes

A Figura 6.12 mostra os histogramas da "duração" de cada evento de perda de pacotes, observado, na média, por cada cliente. A "duração" de um evento de perda é medida

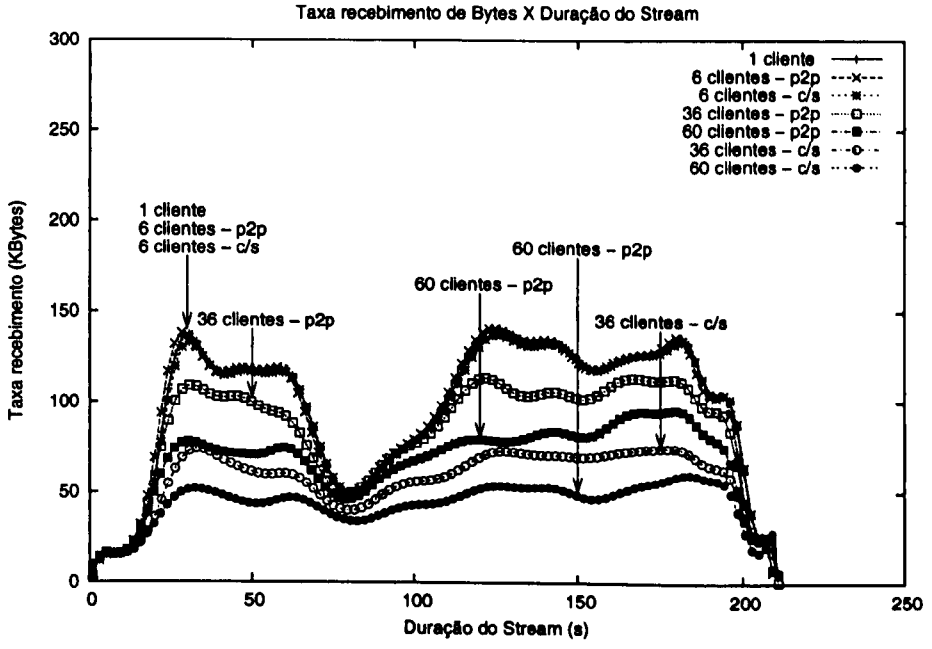


Figura 6.9: Taxa de recebimento por cada cliente (Kbytes/s) - Trailer

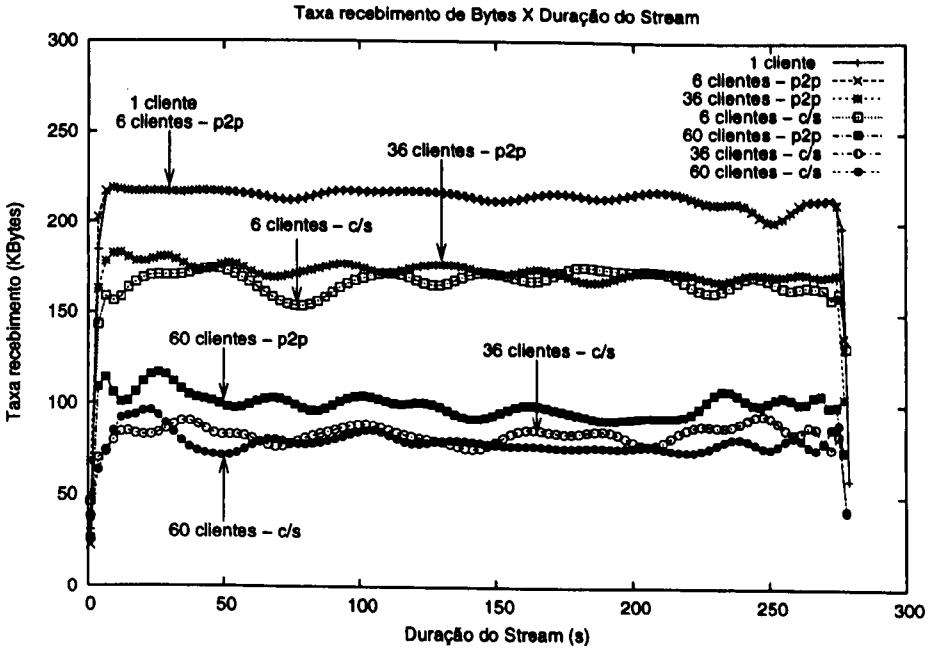


Figura 6.10: Taxa de recebimento por cada cliente (Kbytes/s) - Videoclip

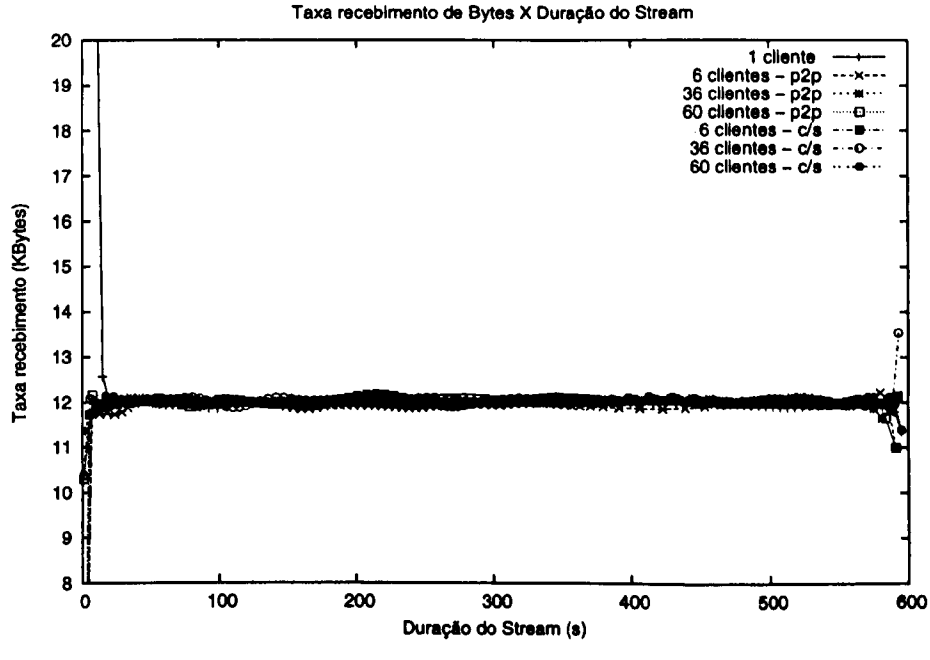


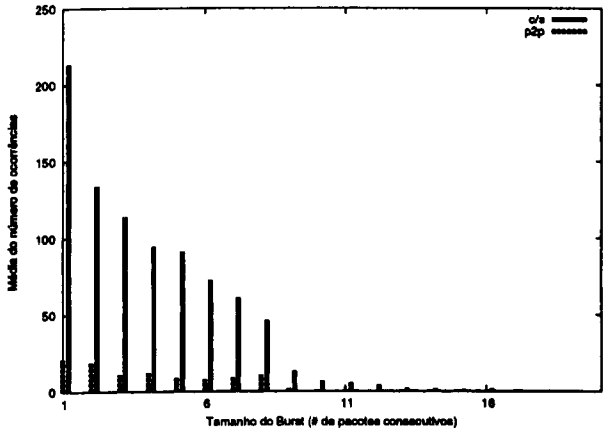
Figura 6.11: Taxa de recebimento por cada cliente (Kbytes/s) - Música

como sendo o número de pacotes consecutivos perdidos em cada evento. A Figura 6.12 mostra a distribuição dessas perdas para os dois vídeos para 12 e 60 clientes para as duas arquiteturas. O arquivo de música foi omitido por não ter apresentado perdas significativas.

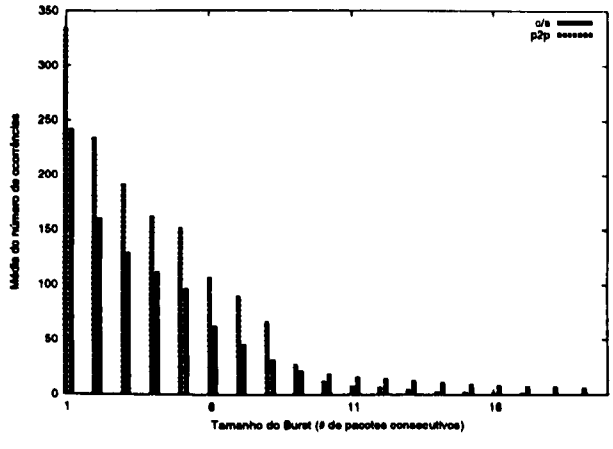
Pode-se observar, que a distribuição da duração das perdas para o *trailer* (Figuras 6.12(a) e 6.12(b)), o qual possui menor *bitrate*, se concentra em um número menor de pacotes consecutivos. Esses gráficos mostram que o número de ocorrências dos eventos de perda de pacotes, para qualquer duração, cresce com o número de clientes. Além disso, em muitos casos, a distribuição da duração de perda para arquitetura P2P é mais inclinada em torno de um número menor de pacotes consecutivos, o que sugere uma qualidade melhor da mídia observada pelos clientes. Conforme mostrado na Figura 6.12(b), embora o número de ocorrências dos eventos de perdas com até 9 pacotes consecutivos seja maior para P2P do que para cliente/servidor, a média da duração da perda na arquitetura P2P é de 3.8 pacotes consecutivos, enquanto que na arquitetura cliente/servidor é de 5.8 pacotes consecutivos.

A Tabela 6.1 sumariza os dados dos histogramas da Figura 6.12, mostra os números totais de eventos de perdas e a duração média desses eventos para cada configuração. É importante notar que comparado às abordagens cliente/servidor, tanto as médias de duração de perdas e os números totais de eventos de perda são menores para a arquitetura P2P nas cargas correspondentes. Com isso é esperado que a arquitetura P2P consiga distribuir os *streams* com uma qualidade melhor para seus clientes do que a arquitetura cliente/servidor.

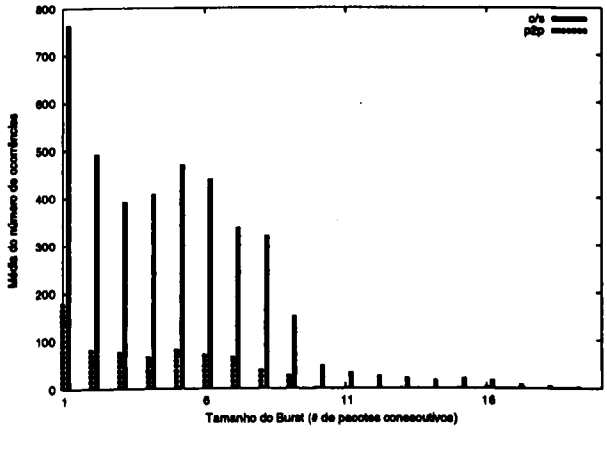
Para verificar de forma visual, a qualidade do *stream* recebido, foram feitos testes com o *player Quicktime* para ambas arquiteturas. Para cada curva dos gráficos foram assistidas as



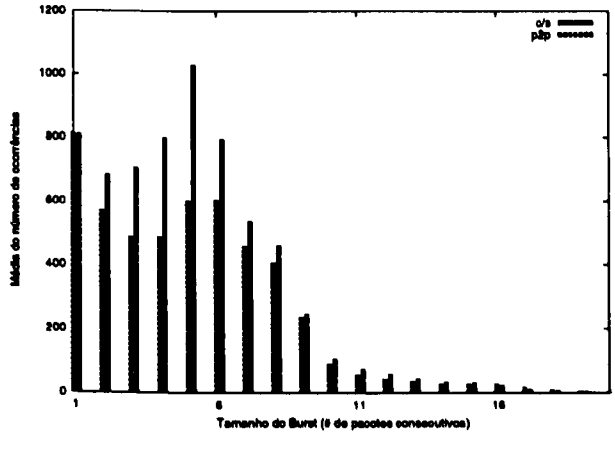
(a) Trailer, 12 clientes



(b) Trailer, 60 clientes



(c) Videoclip, 12 clientes



(d) Videoclip, 60 clientes

Figura 6.12: Histograma da Duração da Perda

Mídia	Cliente/Servidor				Peer-to-Peer			
	12 clientes		60 clientes		12 clientes		60 clientes	
	# de eventos	Duração média	# de eventos	Duração média	# de eventos	Duração média	# de eventos	Duração média
trailer	21	4.2	1413	3.8	172	3.8	1076	5.8
video clip	144	4.1	5025	5	795	4.6	6473	4.9

Tabela 6.1: Duração Média dos Eventos de Perda

mídias e observou-se que as transmissões na arquitetura cliente/servidor apresentavam interrupções e congelamento da transmissão do vídeo, o que não foi o caso para a arquitetura P2P.

Baseado nos resultados de CPU e rede do servidor pode-se observar uma grande economia na utilização desses recursos. Mas, é importante ressaltar que essa economia não afetou a qualidade do *stream*, pelo contrário, ela foi melhorada. Esses resultados demonstram o potencial de arquiteturas do tipo P2P para distribuição de *streaming media* em termos da escalabilidade e economia de recursos.

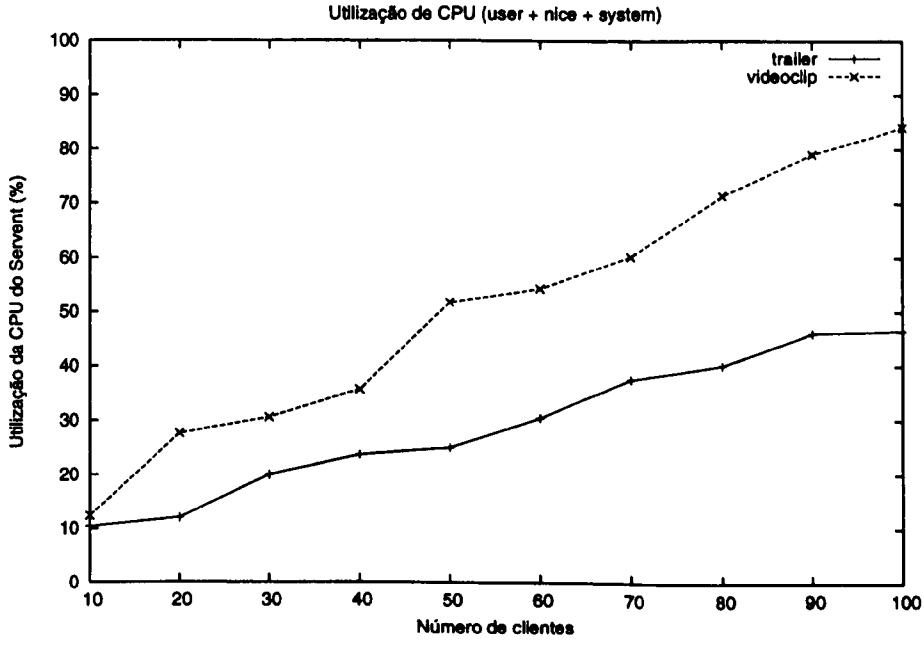
6.3 Avaliação de carga no *servent*

Nessa Seção é avaliada a carga imposta ao *servent* em função do número de clientes para os quais são redirecionados o *stream*, ou seja, o número de clientes diretamente conectados ao *servent* em uma árvore de distribuição P2P. Para essa análise, foram executados uma série de experimentos com um único *servent* rodando em uma máquina, recebendo pacotes diretamente do Darwin e os redirecionado para um número variável de clientes, distribuídos em outras cinco máquinas.

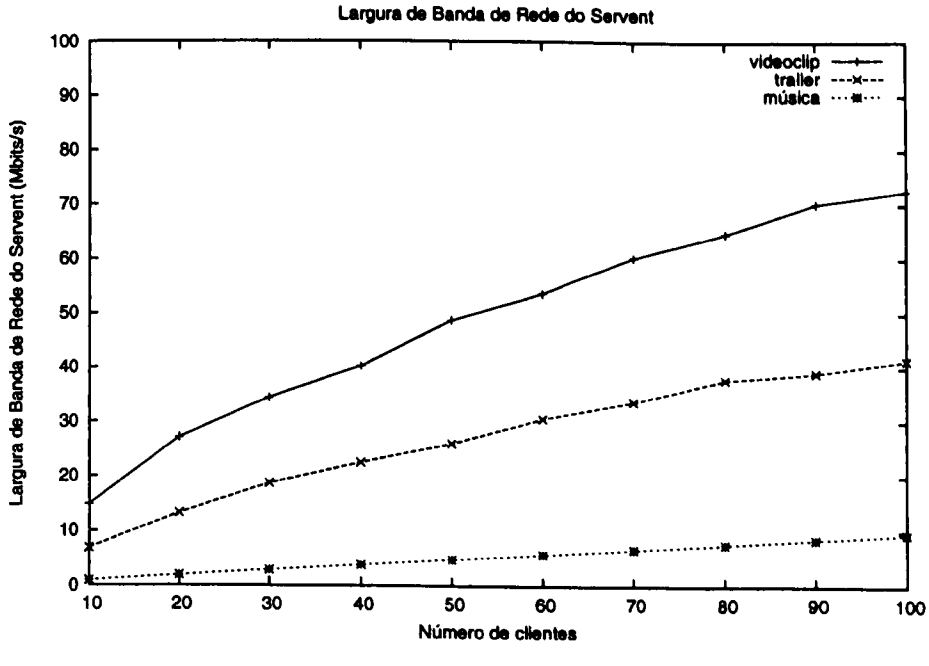
Os resultados mostram que o *servent* ao prover o *stream* a outros *servents* tem a utilização de CPU com esse serviço praticamente desprezível e a utilização de rede cresce com o número de clientes. Para que se verificassem variações maiores na utilização dos dois recursos foram realizados testes com número superior a 10 *servents*.

Os gráficos 6.13(a) e 6.13(b) mostram a utilização percentual de CPU e a banda de rede de saída do *servent*, respectivamente, em função do número de *servents* conectados ao *servent* de avaliação. A curva de utilização de CPU para a música foi omitida, porque ela foi inferior a 5% com todas as quantidades de *servents*. As figuras mostram que, tanto a utilização de CPU quando a banda de rede de saída do *servent* crescem aproximadamente de forma linear com o número de clientes e dependem fortemente da taxa (*bitrate*) do arquivo transmitido.

A utilização de CPU fica próxima a 10%, se o *servent* estiver servindo 10 outros *servents*, redistribuindo qualquer um dos dois vídeos. Conclui-se assim que uma pequena parcela do tempo de CPU é utilizada para redistribuir os pacotes para um número menor de clientes, por exemplo, de 2 a 5 clientes, que seria o *fan-out* esperado em um sistema P2P real. Esses resultados combinados com os resultados de perda de pacotes da seção anterior mostram que se um cliente dedicar largura de banda de rede suficiente para servir a pelo menos mais dois clientes, os benefícios esperados (menos perda de pacotes e melhora na qualidade do *stream*) devem servir como um forte incentivo para participar de um sistema de distribuição P2P, porque o impacto na CPU do cliente é mínimo.



(a) Utilização de CPU do *servent* em função do número de clientes



(b) Largura de Banda de Rede em função do número de clientes

Figura 6.13: Utilização dos recursos do *servent*

6.4 Avaliação do atraso por níveis

A avaliação dos atrasos por níveis foi feita utilizando-se uma topologia em cadeia. Assumiu-se que o servidor S envia um pacote A para o cliente C_1. O cliente C_1 envia A para C_2 e C_2 envia A para C_3. O cliente C_3 envia A de volta para S, mas para uma porta diferente da inicial. Como S gravou o tempo de saída de A e recepção por C_3, a diferença entre esses tempos dá o atraso que o pacote A sofreu atravessando 4 níveis. Assim, foram medidos os atrasos para todos os pacotes transmitidos por S e obtido a média para um determinado nível.

Número de níveis	Atraso em microsegundos	Topologia
2	1753	$S \dashrightarrow C_1 \dashrightarrow S$
3	1825	$S \dashrightarrow C_1 \dashrightarrow C_2 \dashrightarrow S$
4	5522	$S \dashrightarrow C_1 \dashrightarrow C_2 \dashrightarrow C_3 \dashrightarrow S$
5	6834	$S \dashrightarrow C_1 \dashrightarrow C_2 \dashrightarrow C_3 \dashrightarrow C_4 \dashrightarrow S$

Tabela 6.2: Atraso por níveis

A tabela 6.2 mostra o atrasos por níveis para o vídeo *trailer*. O servidor e duas máquinas da mesma sub-rede são representados pelo número 1, 2 e 3 respectivamente. As máquinas representadas pelo número 4 e 5 estão em sub-redes diferentes entre si e diferentes do servidor.

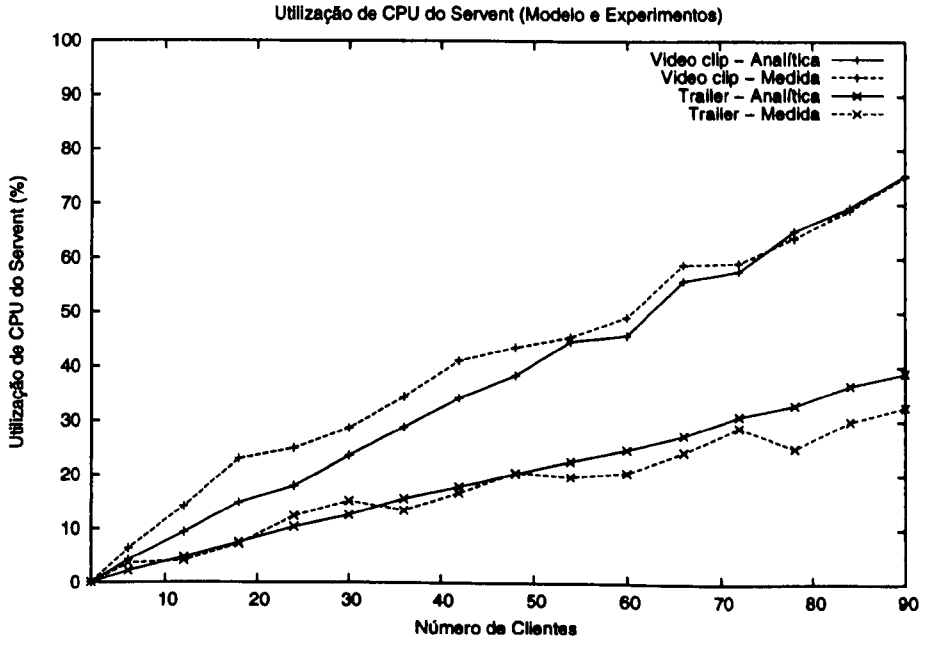
Podemos observar que o atraso que é da ordem de milisegundos cresce quase linearmente na mesma sub-rede, mas chega a ficar 80% maior entre a sub-redes das máquinas 4 e 5. Foi feita, também uma avaliação do *overhead* do *servent* implementado. A forma de avaliação foi a mesma que a descrita acima, mas ao invés dos clientes estarem em máquinas diferentes, eles estavam na mesma máquina que o servidor.

6.5 Cálculo do *fan-out* do *Streaming Servent*

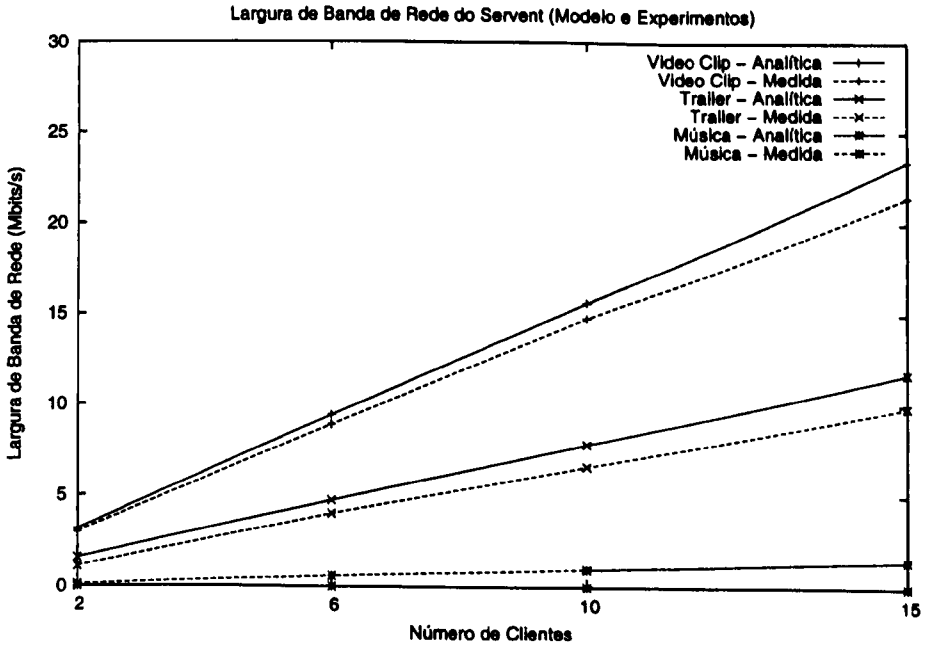
As curvas "analíticas" das Figuras 6.14(a) and 6.14(b) mostram a utilização esperada de CPU e a largura de banda de saída do *servent*, calculada através de fórmulas analíticas[1]. Essas fórmulas foram derivadas da Lei de *Little* [33], que foi utilizada para calcular a média da utilização de CPU do *servent*. Como cada arquivo de mídia é transmitido através de dois *streams* separados (áudio e vídeo) com características diversas, em particular, diferentes tamanhos e taxa de pacotes, a lei de *Little* foi aplicada separadamente para calcular a média de utilização de CPU para processar cada *stream*, como é mostrado a seguir:

$$U_{CPU,vídeo} = \lambda_{vídeo} * S_{vídeo}(n),$$

[1] As fórmulas analíticas foram desenvolvidas em conjunto com a professora Jussara Marques de Almeida



(a) Utilização de CPU do *servent* em função do número de clientes



(b) Largura de Banda de Rede em função do número de clientes

Figura 6.14: Utilização dos recursos do *servent*

$$U_{CPU,áudio} = \lambda_{áudio} * S_{áudio}(n), \tag{6.1}$$

onde $\lambda_{vídeo}$ é a taxa média de pacotes de vídeo, $S_{vídeo}(n)$ é a média de tempo de CPU do *servent* gasta para receber cada pacote de vídeo da fonte e redistribui-lo a n clientes e $U_{CPU,vídeo}$ é a média de utilização de CPU do *servent* para o *stream* de video. Os valores $\lambda_{áudio}$, $S_{áudio}(n)$ e $U_{CPU,áudio}$ correspondem as medidas para o *stream* de áudio. As taxas médias de pacotes $\lambda_{vídeo}$ e $\lambda_{áudio}$, específicas para cada arquivo e os valores $S_{vídeo}(n)$ e $S_{áudio}(n)$ são obtidos através dos *logs* do *servent* ao final de cada experimento. Os valores $\lambda_{vídeo}$ e $\lambda_{áudio}$ são 81.2 e 5.4 pacotes/segundo, respectivamente, para o *trailer* e 150.62 e 10.14 pacotes/segundo para o *vídeo clip*.

A utilização média total da CPU do *servent* pode ser calculada através de:

$$U_{CPU} = \frac{(\lambda_{vídeo} * U_{CPU,vídeo} + \lambda_{áudio} * U_{CPU,áudio})}{(\lambda_{vídeo} + \lambda_{áudio})} \tag{6.2}$$

É importante observar que a utilização de CPU calculada pela equação 6.2 segue o comportamento dos valores medidos. Além disso, as curvas analíticas são quase retas, especialmente na região típica de um *servent* (servindo um número menor de clientes). Assim, as equações 6.1 podem ser reescritas:

$$\begin{aligned} U_{CPU,vídeo} &\approx \lambda_{vídeo} * S_{vídeo} * n, \\ U_{CPU,áudio} &\approx \lambda_{áudio} * S_{áudio} * n, \end{aligned} \tag{6.3}$$

onde $S_{vídeo}$ ($S_{áudio}$) é o tempo de processamento para redistribuir um pacote de vídeo (áudio) para um único cliente.

A banda de rede de saída do *servent* pode ser analiticamente estimada como:

$$BW = b * n \tag{6.4}$$

onde b é o *bitrate* do arquivo. Como mostrado na Figura 6.14(b), a largura de banda do *servent* pode ser estimada como uma precisão razoável.

As equações 6.2 e 6.4 podem ser utilizadas para estimar o *fan-out* máximo, F, de um *servent*, ou seja, o número máximo de clientes conectados a um *servent* em um sistema P2P , dado uma quantidade de recursos que se queira disponibilizar para redistribuição para outros clientes. Dado λ e b, característicos do arquivo, S, o tempo de CPU da máquina *servent* gasto para processar cada pacote e dado que o servent pode dedicar somente até S_{CPU} de seu tempo de CPU e S_{bw} de sua largura de banda de rede de saída, F pode ser estimado como sendo:

$$F = min(S_{CPU}/(\lambda * S), S_{bw}/b) \tag{6.5}$$

A equação 6.5 pode ser utilizada para auxiliar na construção e na avaliação de características de uma árvore de distribuição P2P. Por exemplo, seja uma árvore de distribuição P2P com n clientes homogêneos, com o mesmo *fan-out* máximo F, calculado pela equação 6.5, o número mínimo de níveis dessa árvore pode ser calculado como:

$$\begin{aligned} n &\leq \sum_{i=1}^{L} F^i \\ n &\leq (F^{L+1} - F)/(F-1) \\ &= F/(F-1) * (F^L - 1) \\ F^L &\geq n * (F-1)/F + 1 \\ L &\geq \log_F (n * (F-1)/F + 1) \end{aligned}$$

O número de níveis em uma árvore de distribuição é a chave para avaliar o atraso esperado e a perda observada pelos clientes, especialmente nos níveis inferiores. Por exemplo, se a equação 6.6, for aplicada para determinar o número mínimo de níveis de uma árvore P2P com 1000 clientes, para a distribuição do arquivo de música e assumindo-se que cada cliente possa dedicar até 1 Mbps de sua banda para distribuição do conteúdo. Pode-se observar, na Figura 6.14(b) que o cliente tem largura de banda suficiente para redistribuir os pacotes para aproximadamente 10 clientes. A utilização de CPU para a música é mínima. Assim, o *fan-out* máximo F é igual a 10. Aplicando $F = 10$ e $n = 1000$ à equação 6.6 , o número máximo de níveis que a árvore P2P poderia ter é de 3. Dessa maneira, o atraso e a perda de pacotes que alguns clientes irão observar será equivalente a pelo menos 3 *hops* na árvore.

6.6 Sumário

A implementação de uma arquitetura do tipo P2P mostra a escalabilidade do serviço de *streaming media*. Os resultados obtidos mostram que a distribuição de *streaming media* em redes *peer-to-peer* melhora esse serviço. Na avaliação dos experimentos pode ser observado um claro aumento da escalabilidade do servidor e uma melhora na qualidade do *stream* recebido por cada cliente. Para um mesmo número de *servents* na rede, a utilização do servidor sempre está abaixo, utilizando P2P. A porcentagem de pacotes e taxa de recepção do *stream* é uma métrica válida para a medição da qualidade do *stream*. Esses experimentos

foram validados através de fórmulas analíticas, que também possibilitam o cálculo do *fan-out* do *servents* e o número de níveis que uma árvore P2P deverá ter, dada as características da mídia e os recursos (CPU e rede) dos clientes.

Capítulo 7

Conclusões

7.1 Sumário

É conhecido que um dos gargalos dos servidores de *streaming media* é a largura de banda, que limita o número de clientes atendidos. Técnicas de *multicast* minimizam esse problema, enviando uma única cópia do *stream* para vários clientes. No entanto, a maioria dos roteadores ainda não tem suporte a essa técnica, o que leva a sua implementação em nível de aplicação. Uma maneira de implementar esse paradigma é através de redes *peer-to-peer*, em que os próprios clientes realizam o *multicast*, redistribuindo o stream para outros clientes.

Nesse trabalho foi realizada experimentalmente, a análise de desempenho de uma arquitetura *peer-to-peer* para distribuição de *streaming media*. O sistema experimental é composto de um servidor de *streaming media* e vários clientes, que nessa arquitetura são chamados de *servents*, já que podem receber e redistribuir um *streaming*. A arquitetura foi implementada em dois níveis, sendo que os *servents* do primeiro nível recebem o *stream* diretamente do servidor e os do segundo nível recebem o *stream* dos do primeiro nível.

Os experimentos foram validados através da comparação com a arquitetura cliente/servidor. Em ambas arquiteturas, foram analisadas como recursos do servidor e a largura de banda disponível são afetados com aumento do número de clientes e com mídias a diferentes taxas. Foi analisado também, o comportamento dos recursos do *servent*, quando este está redistribuindo o *stream*. A avaliação foi baseada em três métricas: consumo de largura de banda da rede, utilização da CPU e qualidade do *stream* recebido pelos clientes.

7.2 Resultados

A arquitetura P2P se mostrou ser uma boa estratégia para todas essas métricas, não somente reduzindo a demanda computacional e assim permitindo um número maior de clientes sendo servidos, mas também melhorando a qualidade do *stream* distribuído.

Um resultado experimental fornece um argumento muito concreto para a validação desta arquitetura de distribuição de *stream*. Resultados favoráveis em um ambiente com todas as características reais como rede, protocolos, interfaces, colisões, sistemas operacionais mostram que os resultados são pertinentes.

Este trabalho é de grande relevância, pois apresenta uma proposta e consegue através de experimentações validar a mesma. A continuidade dos estudos, tal como a inclusão de novas funcionalidades ao *servent* estão entre os trabalhos futuros. Experimentos em redes maiores e em maior escala também poderão reforçar ainda mais os resultados obtidos.

7.3 Trabalhos Futuros

Uma extensão desse trabalho seria a experimentação em redes maiores e redes heterogêneas em termos de largura de banda. Além disso, poderiam ser avaliados o comportamento sos sistema com diferentes tipos de vídeo ou áudio tentando estabelecer as correlações entre as características dos vídeos e os ganhos providos.

Outra questão seria a implementação de políticas para balanceamento da árvore de distribuição do *stream* para que sua altura não provoque atrasos que comprometam a qualidade de recebimento do *stream*. Outro ponto a ser pesquisado é a entrada e saída de clientes do sistema.

Bibliografia

[1] Darwin Streaming Server. http://developer.apple.com/darwin/projects/streaming/.

[2] Freenet Home Page. http://freenet.sourceforge.com/.

[3] Gnutella Development Home Page. http://www.gnutella.wego.com/.

[4] Kazaa HomePage. http://www.kazaa.com/.

[5] Napster Home Page. http://www.napster.com/.

[6] Quicktime Player. http://www.apple.com/quicktime/.

[7] Quicktime Streaming Proxy. http://www.apple.com/quicktime/resources/qt4/us/proxy/.

[8] Radio service preaches peer-to-peer: Tech news cnet.com. http://news.com.com/2100-1023-276659.html?legacy=cnet&tag=lh.

[9] Streaming Media World: Allcast: Peer-to-Peer Broadcasting. http://www.streamingmediaworld.com/video /docs/allcast/index.html.

[10] Tcpdump. http://www.tcpdump.org/.

[11] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proc. of ACM SIGCOMM 2002.*, 2002.

[12] Y. Cai and K. A. Hua. An Efficient Bandwidth-Sharing for True Video on Demand Systems. In *Proc of the 9th ACM International Multimedia Conference*, 1999.

[13] M. Cherise, A. Wolman, G. M. Woelker, and H. M. Levy. Measurement and Analysis of a Streaming-Media Workload. In *Proc. of the 3rd USENIX Symp. on Internet Tecnologies and Systems*, 2001.

[14] C. Cranor, Green M., C. Kalmanek, D. Shur, S. Sibal, Sreenan C., and K. van der Merwe. Enhanced Streaming Services in a Content Distribution Network. *IEEE Internet Computing*, pages 66–75, Jul./Aug. 2001.

[15] D. Dan, A.and Sitaram and Shahabuddin P. Dynamic Batching Policies for an On-demand Video Server. In *Multimedia Systems,4(3)*, 1996.

[16] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming Live Media over a Peer-to-Peer Network. Technical report, Computer Science Department, Stanford University, Abril 2001.

[17] D. A. S. dos Santos. Análise de Desempenho de um Serviço Distribuído de Vídeo Sob Demanda. Master's thesis, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, 2001.

[18] L. Gao, Z. Zhang, and D. Towsley. Catching and Selective Catching: Efficient Latency Reduction Techniques for Delivering Continuous Multimedia Streams. In *Proc. of the 7th ACM International Multimedia Conference*, 1999.

[19] L. Golubchik, J. Lui, and Muntz R. Adaptive Piggybacking: a Noval Technique for Data Sharing Video-On-Demand Storage Servers. In *ACM Multimedia Systems,4(3)*, 1996.

[20] D. Grimm. Quicktime Streaming Proxy Performance Improvements. http://www.telecommunications.crc.org.au/content/ConfPapers/gr imm.pdf.

[21] M. Hefeeda, Habib A., and B. Bhargava. Cost-Profit Analysis of a Peer-to-Peer Media Architecture. submitted for review.

[22] M. Hefeeda and B. Bhargava. On-demand Media Streaming Over the Internet. Technical report, Purdue University, 2002.

[23] T. Hong. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter Performance. O'Reilly and Associates, 2001.

[24] K. A. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True Video-on-Demand Services. In *Proc. of the 6th ACM International Multimedia Conference*, 1998.

[25] K. A. Hua and M. Tantaoui. Cost Effective and Scalable Video Streaming Techniques. Class Notes, http://www.cs.ucf.edu/ kienhua/classes/COP6731/Papers/streamming.pdf.

[26] K. A. Hua, D. A. Tran, and R. Villafane. Overlay Multicast for Video On Demand on the Internet. In *Proc. of ACM Symposium on Applied Computing*, 2003.

[27] Microsoft Inc. Multicast Streaming Media Applications and Deployment. White Paper, 1999.

[28] Internet.com. Streaming Media May Drive Future Net Growth. http://cyberatlas.internet.com/big_picture/hardware/article/0,1323,5931_495161,00.html.

[29] Internet.com. Study: Streaming Media Marketing to Rake in $ 3.1 Billion in 2005. http://www.internetnews.com/IAR/article.php/12_786611, 2001.

[30] J. Jannotti, D. K. Gifford, and K. L. Johnson. Overcast: Reliable Multicasting with an Overlay Network. In *USENIX Symposium on Operating System Design and Implementations*, 2000.

[31] R. Lienhart, M. Holliman, Y. Chen, I. Kozintsev, and M. Yeung. Improving Media Services on P2P Networks. *IEEE Internet Computing*, pages 73–77, Jan./Feb. 2002.

[32] D. Luparello, S. Mukherjee, and S. Paul. Streaming Media Traffic: an Empirical Study. In *Proc. of Web Caching Workshop*, 2001.

[33] D. Menasce and V. Almeida. *Capacity Planning for Web Services: metrics, models and methods*, chapter Basic Performance Concepts. Prentice Hall, 2001.

[34] D. Milojicic, V. Kalageraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical report, Hewlett-Packard Company, 2002.

[35] N. Minar. Distributed Systems Topologies. In *The O'Reilly Peer-to-Peer and Web Services Conference*, 2001.

[36] Morpheus. http://www.morpheus.com/.

[37] Wireless NewsFactor. Net Set for Streaming Media. http://www.wirelessnewsfactor.com/perl/story/4492.html.

[38] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing Streaming Media Content using Cooperative Networking. In *Proc. of the 12th Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2002)*, 2002.

[39] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of the SIGCOMM*, 2001.

[40] M. Ripeanu, I Foster, and A. Iamnitchi. Mapping the Gnutella Network. *IEEE Internet Computing Journal*, 6(1), 2002.

[41] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, IETF, Janeiro 1996. http://www.ietf.org/rfc/rfc1889.txt.

[42] H. Schulzrinne, A. Rao, and R. Lanphier. RTSP: Real-Time Streaming Protocol. RFC 2326, IETF, Abril 1998. http://www.ietf.org/rfc/rfc2326.txt.

[43] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the SIGCOMM*, 2001.

[44] D. Stolarz. Peer-to-Peer Streaming Media Delivery. In *First International Conference on Peer-to-Peer Computing(P2P'01)*, 2002.

[45] Akamai Technologies. Free Flow Content Delivery System. http://www.akamai.com.

[46] D. Tran, K. Hua, and T. Do. ZIGZAG: An Efficient Peer-to-Peer Schema for Media Stre aming. In *To appear at IEEE INFOCOM 2003*, 2003.

[47] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On Peer-to-Peer Media Streaming. In *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS 2002)*, 2002.

[48] B. Yang and H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. In *Proc. of the 27th Very Large Data Base (VLDB'01)*, 2001.