UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

UMA ARQUITETURA DE PROXIES PARA AMBIENTES PAR-A-PAR

CRISTIANO MACIEL DA SILVA

Belo Horizonte

Minas Gerais

Julho de 2004

Cristiano Maciel da Silva

UMA ARQUITETURA DE PROXIES PARA AMBIENTES PAR-A-PAR

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

Minas Gerais

05 de Julho de 2004

RESUMO

A tecnologia par-a-par emerge como um novo paradigma para sistemas distribuídos. A continuidade de seu crescimento depende do desenvolvimento de mecanismos que garantam a escalabilidade da rede. Este trabalho analisa a utilização de sistemas de replicação no ambiente par-a-par de forma a reduzir a utilização da banda de rede e latência percebida pelo usuário, aumentando a disponibilidade dos serviços. Além de prover conteúdo em um ambiente par-a-par específico, tal sistema de replicação pode atuar (também) como um interligador de diversos sistemas par-a-par, tornando possível que clientes de um sistema tenham acesso aos recursos disponíveis em quaisquer dos sistemas conectados ao replicador. A arquitetura de replicação projetada atende a todos estes requisitos, tendo sido aplicada a duas populares redes par-a-par: *DirectConnect* e *eDonkey*. Os resultados experimentais mostram que esta estratégia de replicação permite a redução da latência percebida pelo usuário e o aumento da disponibilidade dos recursos.

ABSTRACT

Peer-to-peer is emerging as a new paradigm for distributed computing. Its continuing growing, however, depends on the development of mechanisms that guarantee the scalability of the networks. This work analyzes the utilization of caching systems in peer-to-peer environments, reducing bandwidth requirements and user-perceived latency, and increasing the availability of the services. Besides serving content of a specific network, such caching system may also act as a gateway among different peer-to-peer systems, making possible that clients of one system have access to the available resources in any of the connected systems. We designed a proxy architecture that fulfills these requirements and applied it to two popular P2P networks: *DirectConnect* and *eDonkey*. Simulation results show that our caching strategy allows the reduction of user latency and the increase of the resource availability.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

A tecnologia par-a-par refere-se, principalmente, ao compartilhamento de recursos presentes em computadores localizados na periferia (ou bordas) da Internet. Estes computadores, também chamados de nodos (ou *peers*), tipicamente não apresentam um uso computacional intenso contínuo, sendo comumente de uso pessoal em ambiente de trabalho, acadêmico ou doméstico. O processamento realizado nestas estações segue um padrão com presença de picos de utilização e existência de longos períodos de baixa demanda computacional. Com o recente avanço da tecnologia, estas estações conseguiram um grande aumento de seu poder de processamento e a possibilidade de utilização destas máquinas em um ambiente distribuído oferece uma imensa capacidade computacional.

Sistemas par-a-par são arquiteturas distribuídas, tipicamente sem controle centralizado ou organização hierárquica, nas quais o *software* executado em cada nodo é equivalente em funcionalidade. Sua natureza distribuída torna este tipo de sistema inerentemente robusto a certos tipos de falhas, tornando-os altamente indicados para aplicações que necessitem de grande capacidade de armazenamento e processamento de longa duração. O modelo par-a-par também apresenta o benefício da escalabilidade para o tratamento de crescimentos não-polinomiais no número de usuários e equipamentos conectados, devido ao nível de descentralização que pode ser obtido nesta arquitetura. Com o recente crescimento do tráfego par-a-par, torna-se natural a avaliação do potencial de utilização de sistemas de replicação neste ambiente. Entretanto, fundamentais diferenças existem entre os serviços Web e par-a-par [49], bem como entre seus tráfegos associados, justificando uma análise criteriosa da tecnologia a ser empregada no sistema de replicação a fim de superar estes desafios.

Como diferenças relevantes entre os tráfegos par-a-par e Web aponta-se o tamanho dos recursos compartilhados, caráter transiente dos nodos de uma rede par-a-par, limitação de banda associada aos nodos par-a-par e complicadores relacionados à identificação de recursos em redes par-a-par. Enquanto na Web cada recurso é identificado de forma única através de um endereço de *host* juntamente com o nome do recurso (arquivo), o mesmo não ocorre para as redes par-a-par. Entretanto, de forma similar ao conteúdo Web, o conteúdo par-a-par possui objetos muito populares criando o potencial para a redução de tráfego redundante, que é o objetivo desta.

1.1 Motivação

Com o crescimento da utilização de aplicações par-a-par é fundamental identificar novos mecanismos que colaborem para a redução do tráfego na rede, aumentando sua escalabilidade e replicando recursos de acordo com as preferências dos usuários. Sistemas de replicação contemplam estas características, principalmente quando analisados os relevantes ganhos de desempenho proporcionados no ambiente Web [38]. Com a adoção deste tipo de sistema (de replicação) no ambiente par-a-par podem ser atingidos outros ganhos como o aumento do potencial de anonimato e segurança dos nodos (afinal, o sistema de replicação acrescenta uma nova camada de abstração na comunicação entre os nodos), além da possibilidade de sua utilização como interligadores (*gateways*) entre diversas redes par-a-par, convertendo consultas, respostas e conteúdo. Agentes do sistema de replicação (*proxies*) também podem realizar atividades de análises estatísticas da rede, redirecionando requisições para nodos que ofereçam uma melhor qualidade de serviço e alterando a qualidade da mídia servida, de forma a atender aos requisitos computacionais e de largura de banda de cada nodo em específico.

1.2 Objetivo

Atualmente, as grandes corporações e os provedores de acesso à Internet registram uma fatia importante de seu canal de comunicação com a rede mundial consumido por tráfego par-a-par. Este trabalho propõe o projeto de um sistema de replicação para utilização na fronteira da rede corporativa com a rede mundial, tendo como objetivo

racionalizar a utilização do referido canal para este tipo de conteúdo, obtendo-se resultados que comprovem a eficácia de tal abordagem.

1.3 Descrição

Com este objetivo, neste trabalho são apresentados os diversos sistemas par-a-par atualmente em uso, as técnicas de replicação que podem ser empregadas, as principais diferenças entre o ambiente par-a-par e Web, além das funcionalidades extras que podem ser agregadas a um sistema de replicação para ambientes par-a-par, como por exemplo: coleta de dados sobre estabilidade de nodos; interligação entre redes par-a-par distintas, dentre outras. Após a caracterização de sistemas par-a-par e sistemas de replicação, procede-se a descrição detalhada dos experimentos realizados, bem como as soluções desenvolvidas, análise de amostras de dados utilizadas, resultados obtidos e conclusão da pesquisa.

1.4 Organização

Este documento encontra-se organizado da seguinte forma. O Capítulo 2 discute os trabalhos relacionados. O Capítulo 3 apresenta a formalização de sistemas de replicação para ambientes par-a-par. O Capítulo 4 apresenta uma avaliação de escalabilidade da utilização sistemas de replicação no ambiente par-a-par, sendo esta análise realizada com base na rede *FreeNet* através do simulador *Aurora.* O capítulo 5 descreve a avaliação de interoperabilidade entre os protocolos *DirectConnect* e *eDonkey* e a caracterização dos ganhos que podem ser obtidos com esta estratégia no ambiente par-a-

par em termos de taxa de acertos do sistema de replicação. Por fim, o Capítulo 6 apresenta as conclusões e a relação de trabalhos futuros que podem complementar esta pesquisa.

CAPÍTULO 2

TRABALHOS RELACIONADOS

Este capítulo apresenta a tecnologia de sistemas par-a-par, classificando seus diversos tipos, aplicações e técnicas atualmente empregadas. A apresentação se dá através dos diversos trabalhos realizados que compuseram a evolução desta tecnologia, caracterizando suas diversas aplicações e áreas de utilização, bem como suas principais técnicas e desafios. Os principais trabalhos em destaque são:

SQUIRREL [13], que pode ser definido como uma tecnologia de replicação Web distribuída, apresenta como contribuição o desenvolvimento de uma arquitetura descentralizada que oferece as mesmas funcionalidades de um sistema de replicação Web centralizado. Entretanto, este trabalho difere de SQUIRREL no sentido de não buscar o tratamento de tráfego Web, mas sim par-a-par.

FARSITE [8] consiste de um sistema de arquivos baseado em uma tecnologia par-a-par que garante grande persistência e disponibilidade de dados, similar a um sistema de arquivos tradicional, utilizando o mesmo número de réplicas para cada objeto. Este trabalho difere de FARSITE no sentido de não apresentar como objetivo o oferecimento de fortes requisitos de persistência de dados, mas sim, uma estratégia de replicação proporcional à popularidade do recurso. Em contraste com um sistema de arquivos, o objetivo de uma comunidade par-a-par não é prover fortes requisitos de persistência, mas sim maximizar a quantidade de conteúdo disponível.

CHORD [28] define um algoritmo que garante a resolução de qualquer consulta com o envio de apenas O(log n) mensagens em um sistema constituído de n nodos, sendo que cada nodo deve manter informação sobre apenas O(log n) outros nodos. Este trabalho difere de CHORD na estratégia utilizada para otimizar o roteamento. Enquanto CHORD implementa um novo algoritmo, neste trabalho insere-se uma nova entidade na rede com o objetivo de replicar arquivos freqüentemente acessados.

PASTRY [24] considera a localidade de rede visando otimizar os trajetos das mensagens. Cada nodo mantém uma pequena lista de antecessores e sucessores para garantir a corretude, mesmo quando nodos falham. Este sistema permite o roteamento de mensagens para o nodo numericamente mais próximo de uma chave em menos que $\log_2{}^b n$ passos (b é um parâmetro de configuração com valor típico quatro). Já a entrega é garantida a menos que $L/2$ (L é um parâmetro de configuração com valor típico 2^b) nós com identificadores adjacentes falhem simultaneamente. De forma similar a CHORD, PASTRY busca otimizar o roteamento da rede, enquanto neste trabalho busca-se replicar arquivos freqüentemente requisitados [35].

2.1 Sistemas par-a-par

A tecnologia par-a-par refere-se a uma classe de sistemas e aplicações que empregam recursos distribuídos de forma a realizar atividades de uma maneira descentralizada [22]. Entre os recursos compartilhados, destacam-se: poder de processamento, capacidade de armazenamento de dados e disponibilização de conteúdo [34].

São encontradas diversas definições para esta tecnologia. O *Intel P2P Working Group* a define como o "compartilhamento de serviços e recursos computacionais através da troca direta entre sistemas" [21]. Alex Weytsel define par-a-par como "o uso de dispositivos na periferia da Internet através de uma abordagem não-cliente" [31]. Ross Lee Graham define par-a-par através de três requisitos chaves:

a) computadores com operação similar aos servidores;
b) presença de um sistema de endereçamento independente do *serviço de nomeação de domínios* (DNS)[1];
c) capacidade de operação com conectividade variável [29].

Clay Shirky da *O'Reilly e Associados* propõe que "par-a-par é uma classe de aplicações que consegue aproveitar recursos (armazenamento, ciclos, conteúdo, presença humana) disponíveis na periferia da Internet". Já Kindberg define par-a-par como "sistemas com tempos-de-vida independentes". O fato é que a atividade ou função realizada por uma aplicação par-a-par pode ser tanto de computação distribuída [26], como de compartilhamento de conteúdo ou colaborativa.

---

[1] *Domain naming service.*

Aplicações par-a-par de computação distribuída têm como objetivo a divisão de grandes tarefas em pequenos sub-conjuntos que podem ser processados em paralelo sobre um número de nodos independentes. Grande parte das implementações deste modelo são direcionadas a aplicações que exigem alto grau de processamento, buscando o aproveitamento de ciclos não utilizados pelos diversos computadores conectados à rede mundial [5]. Usualmente, a mesma tarefa é executada em cada nodo com alteração única dos dados de entrada. Exemplos deste tipo de aplicação incluem a busca por vida extraterrestre [51], quebra de chaves criptográficas, cálculos de risco marginal, avaliação de crédito e análises demográficas.

Aplicações par-a-par de compartilhamento de arquivos têm como foco o armazenamento e a recuperação de dados em diversos nodos componentes da rede [32], permitindo que estes nodos busquem e baixem arquivos disponibilizados em outros nodos. Como regra geral, as implementações atuais não incorporam complexos mecanismos de garantia de confiabilidade, deixando a cargo do usuário a responsabilidade por escolhas inteligentes de localização de arquivos, bem como pela realização de novas tentativas em caso da ocorrência de falhas durante o processo de transferência do arquivo [51]. Na verdade, estas aplicações têm como objetivo o aproveitamento de área de armazenamento não-utilizada em nodos, criando uma espécie de servidor distribuído de arquivos, cuja confiabilidade pode ser incrementada através da utilização de técnicas tradicionais de banco de dados (replicação, por exemplo) [44].

Aplicações par-a-par colaborativas permitem que usuários realizem interação em tempo real sem a necessidade de um servidor central para o recebimento e a distribuição da informação ou conteúdo. Programas que permitem a troca de mensagens instantâneas entre usuários são uma subclasse deste tipo de aplicação [4]. Similarmente,

também emergem aplicações compartilhadas que permitem a interação através da visualização e edição da mesma informação simultaneamente por usuários (ou nodos), possivelmente separados por milhares de quilômetros. Por fim, jogos são um outro tipo de aplicações colaborativas. A FIG. 1 ilustra esta organização.

FIGURA 1 – Classificação de sistemas par-a-par com base na atividade da rede

Embora a tecnologia par-a-par date dos primórdios da Internet, recentemente este tipo de aplicação passou a receber uma atenção especial devido à explosão de popularidade das aplicações de compartilhamento de arquivos, com destaque ao *Napster* [20]. Hoje, par-a-par recebe grande atenção da indústria através de grupos como o *P2P Working Group* [21], liderado por parceiros industriais como *Intel*, *HP*, *Sony*, contando também com a participação de diversas outras empresas. Existe também o *JXTA* que trata da elaboração de um padrão aberto para o desenvolvimento de aplicações (par-a-par), liderado pela *Sun Microsystems*. Em paralelo, esta tecnologia também recebe atenção da academia através de uma grande variedade de eventos com foco na pesquisa de tópicos específicos desta área como o *Internacional Workshop on P2P Computing*, *Global and P2P Computing on Large Scale Distributed Systems*, *International Conference on P2P Computing* e *O'Reilly P2P and Web Services Conference*.

Além da função ou atividade executada pela rede, também é possível classificar sistemas par-a-par com base na arquitetura empregada na sua construção (ou paradigma

de projeto). Os sistemas em uso atualmente são concebidos a partir dos seguintes paradigmas: modelo puro; modelo híbrido; modelo intermediário.

Sistemas par-a-par puros não permitem a existência de uma entidade centralizadora. As redes *Gnutella* [10, 27] e *FreeNet* [9] são exemplos deste tipo de sistema.

Sistemas par-a-par híbridos permitem a existência de uma entidade central que é acionada pelos nodos para o envio ou a obtenção de meta-informações, como a localização de algum dado ou a verificação de credenciais de segurança, dentre outros. A partir daí, segue-se a tradicional comunicação par-a-par. São exemplos destas aplicações: *Napster, Groove, Aimster, Magi, Softwax* e *iMesh.*

Sistemas par-a-par intermediários se utilizam do conceito de super nodos para o armazenamento de grandes quantidades de dados ou mesmo informações que nodos comuns tipicamente não possuem. Neste caso, os nodos comuns buscam por informações nestes nodos especiais sempre que não conseguem encontrá-la de outra maneira. A FIG. 2 exemplifica o modelo híbrido e puro.

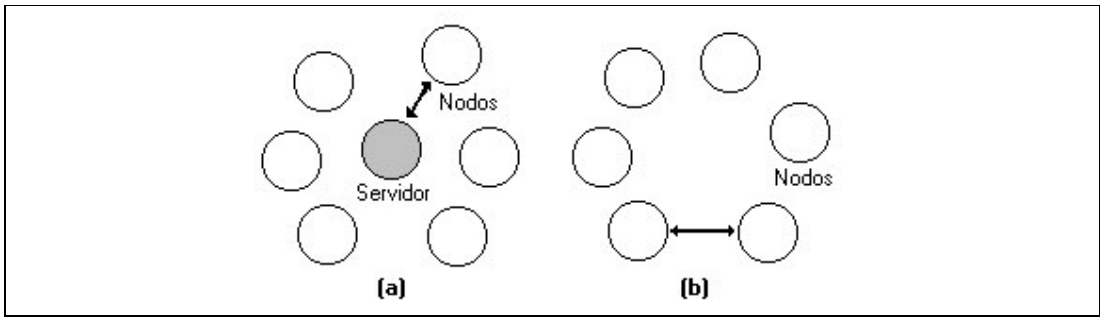

FIGURA 2 – Exemplo de sistemas par-a-par com modelo híbrido (a) e puro (b)

2.2 Sistemas de compartilhamento de arquivos

Através da análise dos diversos tipos de sistemas par-a-par e suas características é possível visualizar que um sistema de replicação somente terá utilidade para redes de compartilhamento de arquivos. Este tipo de rede apresenta duas atividades básicas [46]: a localização de arquivos e o acesso efetivo ao conteúdo do arquivo. A atividade de **localização de arquivos** é complexa, geralmente envolvendo grande utilização de largura de banda e alta latência para a descoberta do arquivo desejado [48]. Esta localização envolve uma consulta aos metadados que descrevem o recurso desejado, possivelmente retornando sua localização. Já o **acesso ao conteúdo do arquivo** deve ser realizado de maneira eficiente e robusta a erros de rede e dos nodos [40].

2.3 Localização de arquivos em redes par-a-par

As principais classes de técnicas utilizadas para a localização ou descoberta de arquivos em redes par-a-par de compartilhamento de arquivos são descritas a seguir:

2.3.1 Técnicas baseadas na propagação de consultas

É uma classe de técnicas que utiliza a propagação de consultas nodo-a-nodo com o intuito de localizar o arquivo. Um dado nodo que deseje pesquisar por um recurso propaga uma consulta para seus vizinhos. Cada vizinho, ao receber a consulta, verifica no seu repositório local por dados equivalentes e realiza novamente a propagação para os vizinhos dos vizinhos, promovendo a difusão da consulta pela rede. Quando alguma resposta é obtida, ela retorna pelo caminho inverso seguido pela consulta. Apresentam-

se como variações desta técnica: inundação; *propagação estilo gossip*; caminhada aleatória; difusão direta.

A **inundação** utiliza um parâmetro de profundidade com o intuito de limitar seu escopo, tendo como resultado sistemas não escaláveis com alto consumo de banda de rede. ***Propagação estilo gossip*** tenta reduzir o consumo de banda de rede através do envio das consultas para apenas um sub-conjunto dos vizinhos do nodo, seguindo um padrão similar ao da propagação de vírus.

Caminhada aleatória *(random walk)* é um caso extremo da *propagação estilo gossip*, onde cada nodo propaga a consulta para apenas um único outro nodo escolhido de forma aleatória.

Difusão direta tenta melhorar o desempenho da *propagação estilo gossip* através da seleção explícita de quais nodos devem ser contatados para a propagação da consulta. Adotando métricas como conectividade e preferência pessoal, tal abordagem permite que um nodo rapidamente localize recursos populares em uma rede com topologias bem realistas.

2.3.2 Técnicas baseadas no armazenamento de índices

As técnicas baseadas em índices promovem um suporte às buscas por palavras-chave, podendo empregar três tipos de abordagens: índices centralizados; índices distribuídos; índices híbridos.

Índices centralizados necessitam de um substancial investimento em banda de comunicação e *hardware*. Além disto, devido às questões judiciais associadas à

distribuição de mídias, juntamente com o exemplo *Napster*, nenhuma tecnologia par-a-par recente se utiliza desta técnica.

Índices distribuídos é uma técnica em que a tabela de índices se encontra distribuída e compartilhada pelos diversos nodos componentes da rede. Apresenta como aspectos positivos a redução de vulnerabilidade legal, bem como um grande potencial de desempenho e escalabilidade. Entretanto, sua dificuldade de manutenção da consistência entre as diversas cópias e sua intrínseca falta de segurança tornam a rede susceptível a sabotagens por nodos maliciosos.

Índices híbridos propõem a utilização de diversas entidades especiais (super nodos) no lugar do servidor centralizado. Estas entidades surgem naturalmente de nodos tradicionais a partir de certas avaliações de banda de comunicação, capacidade de processamento e espaço de armazenamento. No sentido de obter resultados para consultas, cada nodo contata um ou mais super nodos, sendo que estes (super nodos) também podem se comunicar entre si para a resolução de consultas. Integrando as vantagens dos esquemas centralizado e distribuído, este é o método mais popular atualmente. Foi inicialmente implementado pelo *Fasttrack*, sendo também utilizado pelos sistemas *Morpheus* [19], *Kazaa* [15], além das implementações de busca em *JXTA*.

2.3.3 Técnicas baseadas em tabelas *hash* distribuídas

Esta classe de técnicas atribui um *identificador global único* (*GUID*)[2] a cada recurso armazenado no sistema, tipicamente o *algoritmo seguro de hash* (*Sha-1*)[3] [23] do

[2] *Global Unique Identifier.*

conteúdo. A rede é estruturada de forma que uma dada chave possa ser rapidamente localizada independente da quantidade de nodos presentes. Neste modelo, buscas por palavras-chave podem ser facilmente implementadas através da elaboração e disponibilização de catálogos. Como desvantagem, o custo de manutenção da rede pode tornar-se proibitivo na situação em que nodos apresentem-se muito transientes.

2.3.4 Técnicas baseadas em histórico

Esta classe de técnicas não se utiliza de consultas baseadas em requisições do usuário. Ao invés disto, tenta prever a relevância para o usuário de cada recurso encontrado na rede através do armazenamento de informações passadas para a elaboração do perfil do usuário. Baseado em informações históricas sobre o conteúdo mantido e descartado pelo nodo, o sistema é então capaz de determinar elementos comuns interessantes (ao nodo). Embora este tipo de sistema possa ser bastante útil para a aquisição de dados em pano de fundo, sua utilização de forma interativa é comprometida devido a sua alta latência.

2.4 Recuperação de conteúdo em redes par-a-par

As técnicas discutidas no item anterior têm como objetivo realizar a localização de arquivos na rede par-a-par de compartilhamento de arquivos. O passo seguinte à localização é o efetivo acesso ao recurso (conteúdo do arquivo) através de um mecanismo eficiente e apto a lidar com situações de falhas de comunicação, tanto por instabilidade da rede, quanto pelo caráter transiente dos nodos envolvidos na transação

---

[3] *Secure hash algorithm.*

[37]. De forma a superar estes obstáculos e proporcionar que o nodo lide com a heterogeneidade da banda de rede, dois mecanismos podem ser empregados: requisição de intervalos; entrega segmentada.

Requisição de intervalos é uma técnica que permite que nodos requisitem pequenas porções de conteúdo, possibilitando a reinicialização de transferências a partir de seus pontos de falha de conexão, representando uma economia de recursos de rede e de processamento.

Entrega segmentada é um mecanismo complementar que permite que um nodo baixe recursos de diversos outros nodos em paralelo, reduzindo o período de tempo necessário para sua completa obtenção. Enquanto no ambiente Web a localização de recursos se dá através de um identificador de servidor aliado ao nome do arquivo, no ambiente par-a-par, devido ao caráter transiente de seus nodos, este mecanismo de identificação deve ser elaborado de forma independente do nome do nodo. Surge como alternativa viável, a utilização de características intrínsecas ao recurso como nome, tamanho e algum tipo de assinatura digital referente ao conteúdo do arquivo.

2.4.1 Identificação única de arquivos

Para a identificação de forma única e eficiente de arquivos em uma rede par-a-par utiliza-se um mecanismo de cálculo de assinatura digital baseada no conteúdo do arquivo. Através deste mecanismo pode-se retomar a transferência de arquivos que foram transferidos incompletos devido a falhas ocorridas em uma sessão anterior ou mesmo permitir a transferência paralela de pequenos trechos dos arquivos através de diversos nodos. Como principais algoritmos para o cálculo da assinatura digital,

destacam-se: *MD4*; *Sha-1*. O ***MD4*** atribui um identificador de 128 *bits* para cada arquivo com base em seu conteúdo, resultando na probabilidade de $1/2^{64}$ que dois arquivos recebam o mesmo identificador. Já o algoritmo ***Sha-1*** atribui para cada arquivo de tamanho inferior a 2^{64} *bits* um identificador de 160 *bits*.

2.5 Sumário

Este trabalho apresenta interesse no estudo de mecanismos de replicação para redes par-a-par, sendo natural considerar seu escopo como restrito à análise de redes de compartilhamento de arquivos, pela própria natureza deste tipo de aplicação. Foram escolhidas três redes populares para serem utilizadas ao longo deste trabalho: *FreeNet*, *DirectConnect* e *eDonkey*. Conforme os conceitos recém-apresentados, a rede *FreeNet* pode ser caracterizada como um sistema par-a-par puro, sem a existência de entidades centralizadoras na rede, onde é utilizada uma técnica de propagação de consultas com base em um *hash* do conteúdo do arquivo. A rede *DirectConnect* é um sistema par-a-par híbrido, existindo uma entidade central à rede acionada pelos nodos para o envio ou a obtenção de meta-informações. Nesta rede, a busca é realizada através de propagação de consultas das unidades centralizadoras para os nodos detentores de recursos. A rede *eDonkey* é um sistema par-a-par intermediário com a presença de entidades que atuam como supernodos para o armazenamento de grandes quantidades de dados ou mesmo informações que nodos comuns não possuem. Nesta rede, a busca é realizada através da utilização de índices distribuídos que permitem a localização dos recursos. De maneira similar, todas estas redes apresentam a operação de requisição de intervalos, uma

técnica que permite que nodos requisitem pequenas porções de conteúdo, conforme já mencionado.

CAPÍTULO 3

SISTEMAS DE REPLICAÇÃO PARA O AMBIENTE PAR-A-PAR

3.1 Apresentação

Ao longo desta seção define-se um sistema de replicação para o ambiente par-a-par através da caracterização de suas funcionalidades e principais estratégias para seu o projeto. Um sistema de replicação consiste na utilização de uma entidade intermediária ao provedor e requisitante de um recurso, onde ocorre armazenamento dos recursos transacionados no chamado repositório de replicação, permitindo que estes recursos

sejam futuramente disponibilizados a novas requisições, com redução do custo de comunicação com seus detentores originais. Tal mecanismo, quando devidamente aplicado, pode ser responsável por uma importante economia de rede, balanceamento de carga entre provedores de conteúdo, redução de latência de rede percebida pelos usuários e aumento da disponibilidade de recursos. Com o recente crescimento da rede mundial em termos de popularidade e tamanho, aumentam também os requisitos de escalabilidade de sua infra-estrutura. Crescimento exponencial desacompanhado de soluções de escalabilidade, eventualmente resultam em uma carga de rede superior à sua capacidade, com latência para respostas inaceitável e possibilidade de colapso dos serviços da rede [3]. A utilização de sistemas de replicação para ambientes par-a-par tem como promessa a melhora da percepção da rede pelos nodos, mascarando latências da rede de longa distância através da redução do intervalo de tempo necessário para a obtenção de respostas de requisições. Com a inerente replicação de conteúdo, ocorre também o aumento da disponibilidade de recursos, já que falhas da rede não mais implicam na sua indisponibilidade [36]. Acredita-se que a principal utilização de um sistema de replicação de conteúdo par-a-par encontra-se nas grandes corporações e provedores de acesso à Internet, que atualmente registram uma fatia importante de seu canal de comunicação com a rede mundial consumido por este tipo de tráfego. Desta forma, este trabalho propõe o projeto de um sistema de replicação para utilização na fronteira da rede corporativa com a rede mundial, tendo como objetivo racionalizar a utilização deste canal para conteúdo par-a-par, conforme ilustra a FIG. 3.

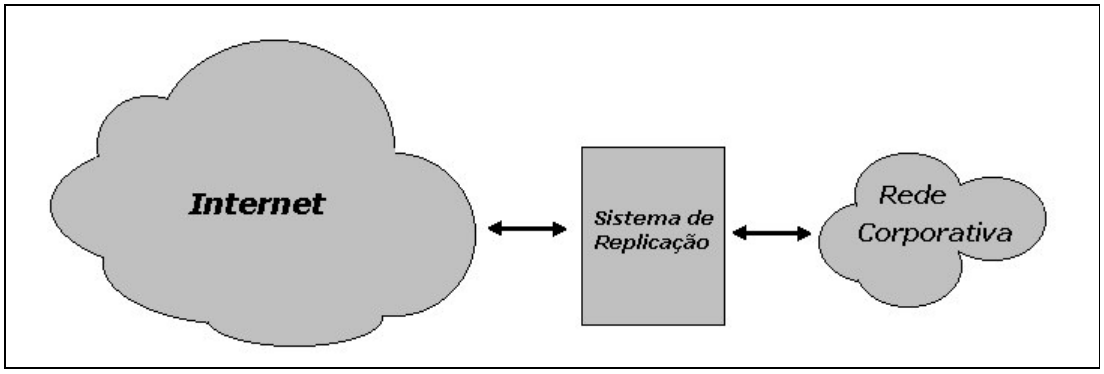

FIGURA 3 – Posicionamento do sistema de replicação

O funcionamento genérico de um mecanismo de replicação é simples. Um nodo que deseja localizar um recurso solicita esta informação ao sistema de replicação. Assim que o sistema de replicação recebe a requisição do nodo solicitante[4], este verifica se possui o recurso solicitado em seu repositório de replicação. Caso afirmativo, o conteúdo é retornado ao cliente. Caso negativo, o sistema de replicação propaga a requisição para outros nodos da rede na tentativa de localizar e servir o conteúdo [31]. Quando o sistema de replicação consegue obter uma cópia do recurso, este realiza seu armazenamento no repositório de replicação e encaminha uma cópia para o nodo requisitante. Destacam-se como operações típicas de uma rede par-a-par: requisições de consulta (ou localização); requisições de obtenção (ou acesso) de recurso.

As **requisições de consulta** são o instrumento pelo qual nodos pesquisam a localização (ou existência) de recursos na rede. Estas consultas podem ser baseadas em palavras-chave presentes no nome do recurso, tamanho ou *hash* do conteúdo binário. Estas consultas são propagadas pela rede e respondidas por nodos que percebem uma situação de equivalência. As **requisições de obtenção de recursos** são utilizadas por

---

[4] As expressões 'nodo solicitante' e 'nodo originador' são utilizadas com o mesmo significado ao longo do texto.

nodos que já localizaram um recurso interessante e desejam realizar uma cópia para o seu repositório local. Estas requisições são enviadas diretamente para o nodo detentor do recurso, que realizará sua transferência pela rede.

Para ser efetivo, um sistema de replicação deve lidar com estas operações da rede, existindo diversos níveis de atuação que podem ser exercidos pelo sistema de replicação. **Sistemas de replicação transparentes** não alteram requisições, respostas ou conteúdo, realizando suas atividades através da interceptação de requisições. Este modelo permite a instauração de diversos controles administrativos (como exemplo, a implementação de políticas de balanceamento de carga). Entretanto, este modelo viola o argumento fim-a-fim [25] não mantendo constantes as terminações de uma conexão [3].

Sistemas de replicação não transparentes podem alterar requisições, respostas ou conteúdos e podem ser utilizados para a realização de conversões ou formatações de conteúdo de forma a servir os nodos com restrições de banda de rede de uma maneira mais apropriada.

Sistemas não replicadores atuam apenas como intermediários na propagação de requisições, respostas e conteúdo formando elos de conexão entre redes par-a-par de mesma natureza e atuando no auxílio de nodos com conexões de baixa qualidade [30].

Sistemas de replicação interligadores atuam na conversão de requisições e respostas entre diversas redes par-a-par, possibilitando que nodos consigam acessar recursos presentes em outras redes, realizando uma tarefa de unificação do sistema par-a-par.

Sistemas de replicação tuneladores atuam tipicamente no envio de conteúdo para nodos cercados por *firewalls*, usualmente não realizando replicação de conteúdo.

Todas estas possibilidades de atuação do sistema de replicação são originais do ambiente Web [43]. Mas, pelas diferenças entre os ambientes Web e par-a-par, e pelo fato de um sistema de replicação estar estreitamente relacionado com seu ambiente de atuação, deve-se realizar uma comparação entre as características dos tráfego Web e par-a-par. Tipicamente, o tamanho de um objeto trocado em uma rede par-a-par é superior a três ordens de grandeza do tamanho de um objeto Web. Apenas esta informação já constitui uma diferença relevante entre estes dois tipos de tráfego. Além disso, em contraposição ao serviço Web, nodos de uma rede par-a-par possuem caráter transiente e momento de conexão indeterminado, ou seja, *a priori* não é possível determinar a quantidade de tempo que um determinado nodo permanecerá ativo. De forma análoga, quando um nodo é percebido inativo, também não se consegue prever o momento em que este nodo retornará ao estado de ativo. Outra diferença fundamental refere-se ao fato dos nodos par-a-par não poderem ser assumidos como dedicados integralmente às atividades executas pela rede. Enquanto os servidores Web são máquinas específicas para servir este tipo de conteúdo, o mesmo já não ocorre com os nodos par-a-par, havendo restrições de recursos de processamento, armazenamento e banda de rede. Além disto, no ambiente par-a-par surge a opção de obtenção de conteúdo de múltiplas fontes, reduzindo-se o tempo de obtenção do recurso através da requisição de pequenos trechos relativos a um mesmo arquivo para diferentes nodos. Entretanto, de forma similar ao conteúdo Web, o conteúdo par-a-par possui objetos muito populares criando o potencial para a redução de tráfego redundante [16].

3.2 Sumário

Esta seção apresentou as atribuições de um sistema de replicação para o ambiente par-a-par, uma entidade intermediária ao provedor e requisitante de um recurso, através da caracterização de suas funcionalidades. Este tipo de sistema pode ser categorizado pela sua atuação na rede, tendo sido abordado neste trabalho o projeto de um sistema **não-transparente** e **interligador**.

CAPÍTULO 4

AVALIAÇÃO DE ESCALABILIDADE - ESTUDO DE CASO DA REDE FREENET

Nesta seção são descritos os experimentos de escalabilidade realizados através de simulação de redes *FreeNet*, onde pretende-se identificar os ganhos que podem ser obtidos utilizando-se sistemas de replicação em uma rede par-a-par. *FreeNet* é uma rede descentralizada, projetada com o objetivo de prover o anonimato necessário para salvaguardar o direito de liberdade de expressão, possibilitando que todos os nodos consigam inserir e recuperar recursos, inexistindo a possibilidade de implementação de algum mecanismo de controle de conteúdo. *FreeNet* define um roteamento inteligente conjugado com replicação e espelhamento automático de dados populares. Desde a

divulgação da idéia proposta por Ian Clarke [9], o projeto *FreeNet Open Source* ganhou visibilidade e popularidade crescentes.

4.1 Rede FreeNet

Cada nodo *FreeNet* possui seu próprio repositório de dados onde documentos são encriptados e armazenados. Após o armazenamento de um documento neste repositório, sua recuperação somente é possível com a utilização da chave de encriptação. Sem esta chave, a localização de qualquer documento é inviabilizada. Cada nodo se comunica com os outros através de um canal encriptado sobre TCP/IP. Quando um nodo recebe uma requisição por algum recurso, primeiramente, ele checa seu repositório de dados. Na eventualidade deste recurso não existir localmente, o nodo propaga a consulta para um vizinho que possua o documento com entrada *hash* mais próxima do *hash* referente ao documento solicitado. A cada propagação desta mensagem, um contador de escopo de inundação é decrementado. A requisição falha quando o contador de inundação atinge o valor zero e nenhum documento foi encontrado. Caso a requisição atinja um nodo que não possui o documento procurado e nem um vizinho para propagação da requisição, estando o valor do contador de escopo de inundação maior que zero, uma mensagem de *Falha na Requisição* é propagada para o nodo anterior, o qual buscará por um novo caminho (ou ramo) para o encaminhamento da requisição, seguindo o formato de um mecanismo de *backtracking*. Esta situação encontra-se ilustrada na FIG. 4 com a seqüência A → B → C→ D → E.

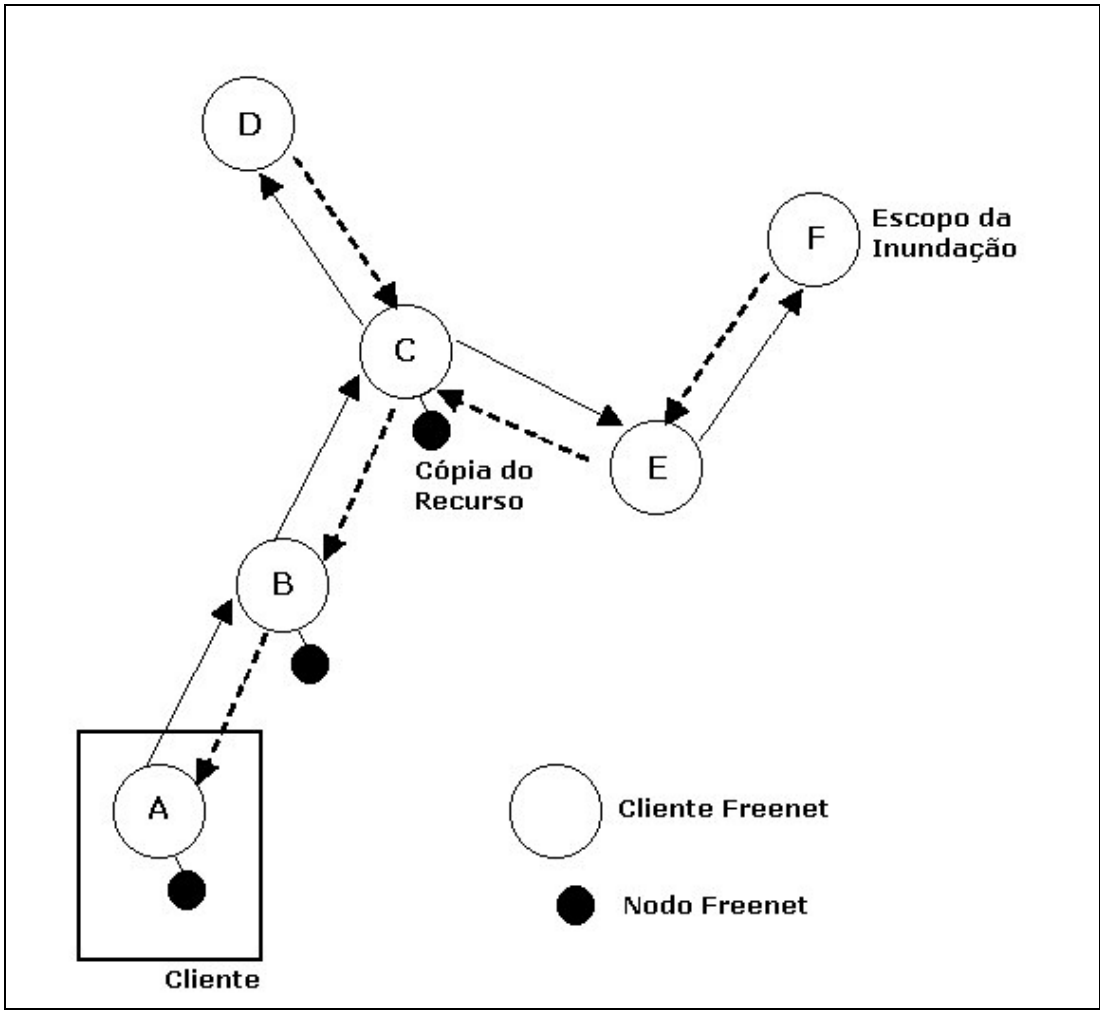

FIGURA 4 – Roteamento em redes FreeNet

No nodo F, caso o documento não tenha sido encontrado no repositório de dados e o contador de escopo de inundação seja maior que zero, uma mensagem de *Falha na Requisição* será propagada por todo o caminho de volta ao nodo A até que seja atingido o cliente que iniciou a requisição. Entretanto, caso o documento ou recurso esteja disponível no nodo F, uma mensagem de *resposta de dados* será propagada pelo fluxo F→ E → C → B→ A (sem passar pelo nodo D). Cada nodo neste caminho irá replicar o documento, e, para aumentar a entropia do tráfego fortalecendo o anonimato, cada nodo pode, aleatoriamente, atribuir qualquer outro nodo conhecido como originador do

conteúdo. Pelo mesmo motivo, quando a requisição atingir o nodo F, mesmo que o contador de escopo de inundação seja zero, este nodo pode propagá-la com uma pequena probabilidade tornando bem complicada a análise de tráfego neste tipo de rede.

Cada nodo da rede *FreeNet* implementa um repositório de dados que armazena tanto conteúdo de arquivos, como referências de arquivos conhecidos (*links*) na forma de uma fila *first-in first-out* (FIFO) com o recurso mais recentemente acessado sendo promovido para o final da fila (similar a uma lista cronológica). Quando um novo documento é inserido no nodo, este é direcionado para o fim da lista. Caso o repositório de dados esteja com sua capacidade esgotada, as entradas presentes no início da lista são removidas até que espaço suficiente seja liberado.

A inserção de documentos é similar ao processo de localização. Uma inserção somente acontece quando uma mensagem *Inserção de Dados* é propagada. Na FIG. 4, embora a requisição do documento tenha seguido A → B → C→ D → E → F, a mensagem de *Inserção de Dados* somente passou por F→ E → C → B→ A. As mesmas considerações são aplicáveis para a atribuição aleatória de originador de documento e para a política de replicação ao longo da rede.

4.2 Aurora, o simulador FreeNet

A avaliação da utilização de sistemas de replicação nas redes *FreeNet* foi realizada através de simulação, tendo sido escolhido o *Aurora* [2] como simulador pela sua ampla utilização pela comunidade acadêmica. A topologia da rede implementada pelo *Aurora* segue o padrão de um anel com cada nodo sendo iniciado com conexões para apenas dois de seus vizinhos (os vizinhos imediatos). O crescimento da rede é simulado através

de uma função linear no tempo. Para cada mensagem, o sistema armazena o número de nodos pelos quais esta mensagem passou até atingir seu destino. Requisições sem sucesso (recurso não encontrado) são automaticamente convertidas para um alto valor de caminho percorrido, inexistindo dados diretos sobre a quantidade de requisições que falharam. Através do *Aurora* pode-se configurar os seguintes parâmetros relativos à estrutura da rede:

a) tamanho inicial da rede;

b) tamanho final da rede;

c) documentos armazenados por nodo (capacidade máxima);

d) referências armazenadas por nodo (capacidade máxima);

e) intervalo de documentos distintos existentes na rede;

f) parâmetro de contador de escopo de inundação para operações de *inserção*;

g) parâmetro de contador de escopo de inundação para operações de *requisição*;

h) parâmetro de contador de escopo de inundação para operações de *anúncio.*

Com foco na simulação, o *Aurora* permite a configuração da quantidade de operações realizadas pela rede durante a simulação. Este processo de simulação é um laço de repetição onde em cada iteração o simulador decide qual operação da rede *FreeNet* será executada. São duas as operações possíveis: requisição de conteúdo; inserção de conteúdo. *Aurora* fará a escolha de uma delas com uma idêntica (50% para cada opção). Quando uma operação de *requisição* é selecionada, um nodo requisitante e uma chave são geradas através das distribuições uniforme e *Zipf* [33], respectivamente (sendo estas distribuições definidas pelo simulador). Um comportamento similar ocorre quando a operação de *inserção* é selecionada, ou seja, *Aurora* gera o nodo que fornecerá

o arquivo, juntamente com a chave que lhe será atribuída. O simulador possui um módulo de rede que implementa as operações de requisição e inserção, além do arranjo de nodos que constitui uma abstração de nodos reais.

Cada nodo é representado por um identificador numérico e uma referência para uma entidade repositório de dados. Esta entidade lida com itens de dados, que são a representação interna do simulador para chaves e referências. O simulador também possui um módulo de estatísticas que consiste de registradores para a contabilização de indicadores como distância média e máxima percorrida pelas mensagens e cálculo do número de caminhos existentes no grafo. Por fim, de forma a garantir a corretude da simulação, *Aurora* realiza intensivas verificações das propriedades do sistema através de um modelo de programação por contrato.

Para implementar o sistema de replicação no *Aurora*, o código fonte original do simulador teve que ser alterado com a eleição de alguns nodos para possuírem um comportamento especial. Estes nodos especiais são referenciados como agentes do sistema de replicação e estão sujeitos às seguintes normas:

a) a quantidade de agentes do sistema de replicação será constante em cada simulação, não sendo contemplado o crescimento dinâmico destes agentes;
b) cada nodo estará associado a somente um agente do sistema de replicação;
c) a capacidade de armazenamento de um agente do sistema de replicação não será considerada, sendo razoável assumir que um agente do sistema de replicação possui uma capacidade de armazenamento muito superior à de um nodo comum;
d) antes do início da simulação, cada agente do sistema de replicação conhecerá todos os seus nodos subordinados, ou seja, os nodos que servem ou solicitam documentos a ele;

e) cada nodo somente solicita arquivos ao seu agente do sistema de replicação. De forma a atingir este requisito sem violar o protocolo *FreeNet*, cada nodo deve conhecer apenas o seu agente do sistema de replicação. Cada documento entregue ao nodo sempre irá incluir o agente do sistema de replicação como o originador do conteúdo;

f) cada agente do sistema de replicação somente conhece seus nodos subordinados e os outros agentes existentes na rede. Não haverá disputa por nodos entre os agentes do sistema de replicação.

Como estratégia de simulação, definiu-se que cada nodo ingressa na rede compartilhando apenas um único documento. Cada operação (seja *requisição* ou *inserção*) executada pelo simulador corresponde a um instante de tempo. Não são executadas operações em paralelo. Quando recursos são recebidos pelos agentes do sistema de replicação, uma cópia do conteúdo é armazenada no repositório de replicação, sendo que agentes do sistema de replicação também podem armazenar referências.

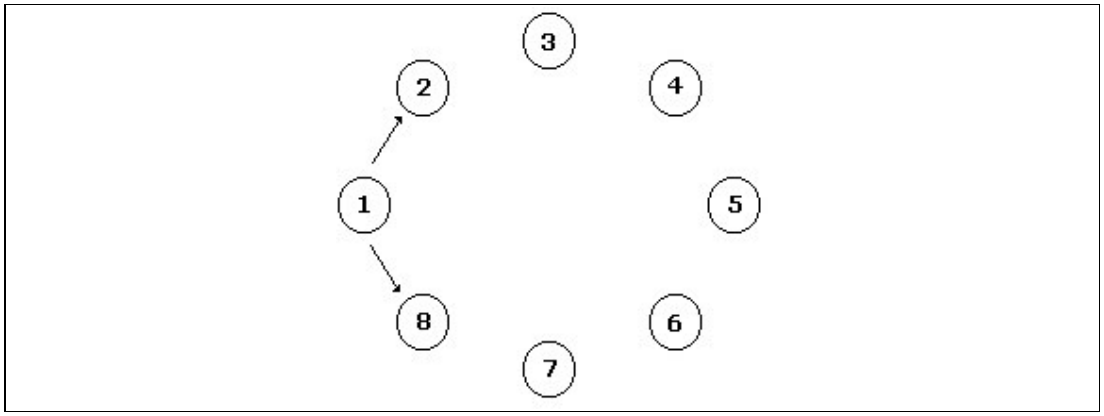

FIGURA 5 – Topologia para uma rede FreeNet contendo 10 nodos

Para ilustrar as modificações realizadas, a FIG. 5 apresenta a topologia de uma rede par-a-par ausente de agentes do sistema de replicação. Esta rede segue o tradicional estilo em anel, onde cada nodo é conectado com dois de seus vizinhos imediatos. Quando são adicionados agentes do sistema de replicação, a topologia desta rede toma a forma da figura seguinte. Os agentes se conectam através de um anel com apenas dois vizinhos imediatos, enquanto cada nodo possui uma conexão direta com seu agente do sistema de replicação.

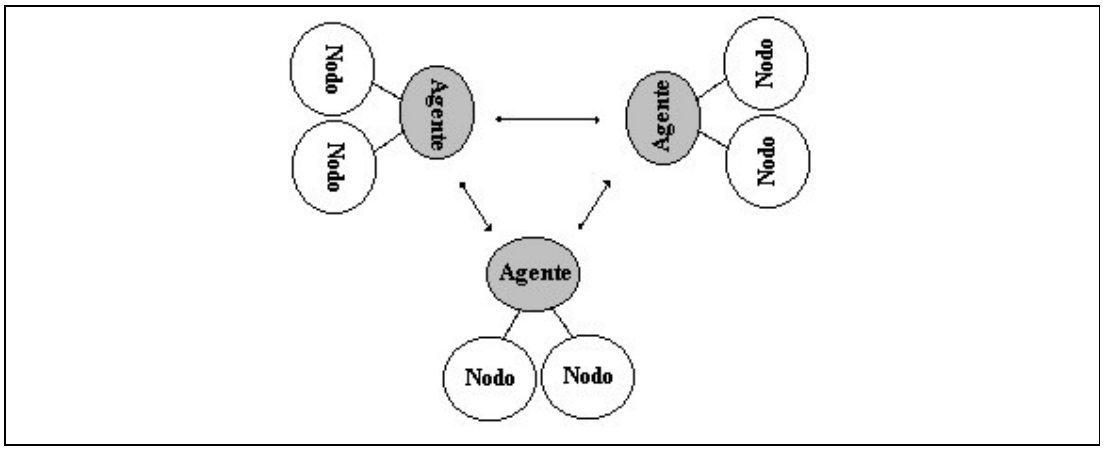

FIGURA 6 – Topologia para uma rede FreeNet com sistema de replicação

Para efeitos de simulação, a entidade agente do sistema de replicação foi considerada como uma extensão da entidade nodo, implicando na possibilidade desta originar requisições de documentos nos moldes de um nodo comum. Em verdade, nota-se que a diferença está na capacidade de armazenamento e na visão da rede que cada entidade possui.

4.3 Simulações

Nesta seção são descritas as simulações realizadas, juntamente com os resultados obtidos. Cada simulação consiste da execução de um determinado número de iterações em uma rede *FreeNet*, onde cada iteração pode ser uma consulta ou inserção de um conteúdo qualquer na rede. Conforme já mencionado, cada nodo inicia na rede com apenas um recurso armazenado em seu repositório local. O resultado dos experimentos é o valor médio e mediana da quantidade de vezes que cada mensagem foi propagada (ou seja, o tamanho do percurso percorrido pela mensagem). Neste contexto, um valor de quinhentos passos significa que o documento ou recurso procurado não foi encontrado, sendo este um valor padrão atribuído pelo simulador.

A primeira simulação apresenta uma rede ausente de agentes do sistema de replicação contendo um mil, dois mil e três mil nodos. O segundo experimento insere cinqüenta agentes do sistema de replicação nestas redes, identificando a relação entre os comportamentos assumidos pelas redes de um mil, dois mil e três mil nodos, comparando estes resultados com o experimento anterior. O terceiro experimento varia o número de agentes do sistema de replicação de forma a manter uma relação constante entre o número de nodos e agentes do sistema de replicação para redes de um mil, dois mil e três mil nodos, objetivando definir se o que realmente importa é a quantidade de agentes ou a relação entre a quantidade de nodos e agentes do sistema de replicação. Para todas as simulações, o contador de escopo de inundação é fixado com o valor nove, que representa um valor típico utilizado por clientes *FreeNet*. Mais relevante que o valor absoluto utilizado é a compreensão de seu impacto nos resultados. Este contador age como um seletor da capacidade de localização de recursos na rede, podendo ser

considerado como um fator de coesão, tendo efeito direto na redução ou aumento das distâncias lógicas da rede. Quanto maior o valor do contador de escopo de inundação para uma mesma distância física, maiores são as chances de localização de recursos, pois, maiores são os nodos consultados.

4.3.1 Rede sem replicação

Esta simulação analisa o comportamento assumido pela rede ausente de um sistema de replicação durante três mil iterações. São apresentados os resultados obtidos para redes com um mil, dois mil e três mil nodos. Como resultados da simulação, apresenta-se a média (e mediana) do tamanho dos caminhos percorridos pelas requisições[5].

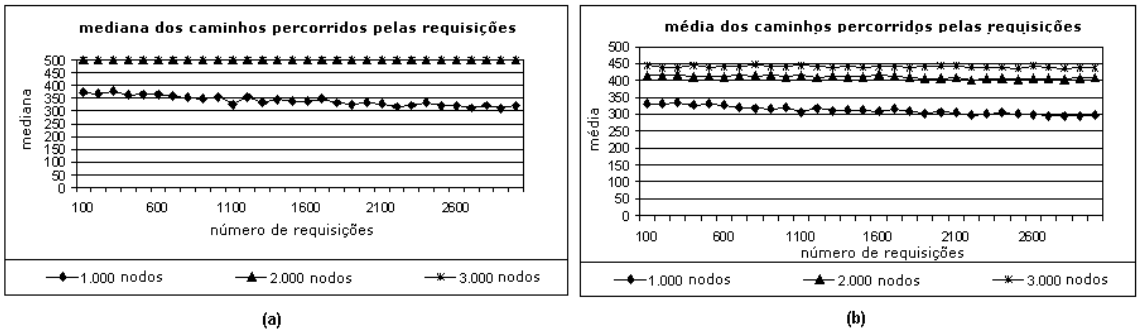

GRÁFICO 1 – Mediana (a) e média (b) para rede sem replicação

Para uma rede com um mil nodos, a quantidade de requisições concluídas com sucesso é baixa. Entretanto, com o passar do tempo, mesmo que de uma maneira lenta, a quantidade de transações concluídas com sucesso cresce, pois, nodos passam a conhecer e referenciar outros nodos, além de melhorar a distribuição dos documentos ao longo da

[5] *Quando um recurso não é encontrado, o simulador atribui um caminho de tamanho quinhentos.*

rede. Para redes com dois mil e três mil nodos, o cenário é mais perverso devido à grande quantidade de nodos.

4.3.2 Rede com replicação

Esta simulação ocorre com a rede populada com cinqüenta agentes do sistema de replicação, também para um mil, dois mil e três mil nodos, sendo executada por três mil iterações.

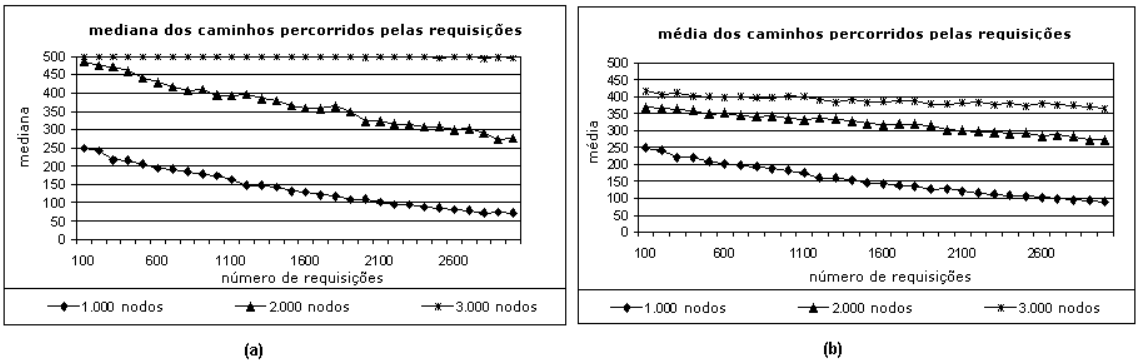

GRÁFICO 2 – Mediana (a) e média (b) para rede com 50 replicadores

Para uma rede contendo apenas mil nodos o caminho percorrido pelas requisições decresce rapidamente para um estado em que grande parte das requisições é completada com sucesso. Quanto maior o tamanho da rede, mais demorada é esta convergência. Com três mil nodos as três mil iterações não são suficientes para tornar a rede operacional. Embora seja possível notar um ganho de desempenho em relação ao cenário ausente de agentes do sistema de replicação, ainda não se têm subsídios sobre o motivo pelo qual o desempenho continua baixo para redes com três mil nodos. Quando inseridos cinqüenta agentes do sistema de replicação numa rede de um mil nodos, coloca-se uma relação de um agente para cada vinte nodos. Já, para uma rede

constituída de três mil nodos, os mesmos cinqüenta agentes implicam numa relação de um agente do sistema de replicação para cada sessenta nodos. De forma a encerrar esta questão, apresenta-se a simulação a seguir que oferece uma medida da elasticidade da rede através da manutenção da relação nodos por agentes do sistema de replicação, sendo sempre uma relação constante de um agente para cada cinqüenta nodos. Em uma rede de um mil nodos são instanciados vinte agentes. Para uma rede de dois mil nodos são instanciados quarenta agentes. Por fim, para três mil nodos são instanciados sessenta agentes do sistema de replicação.

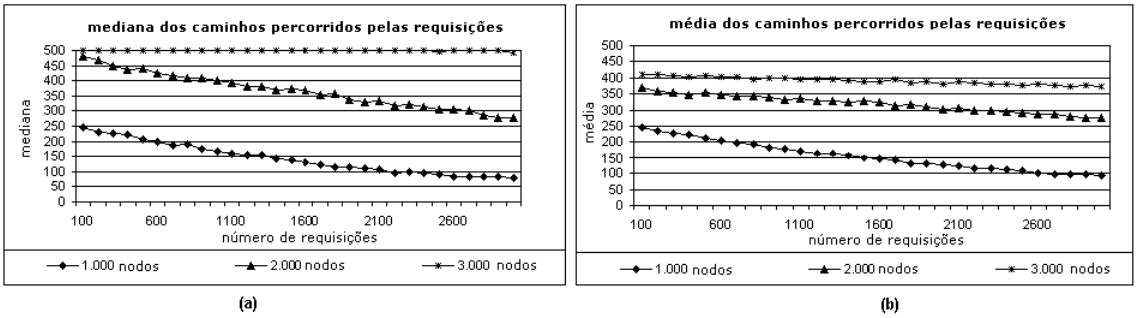

GRÁFICO 3 – Mediana (a) e média (b) para rede com 50 nodos por replicador

Comparando estes resultados com os obtidos na seção anterior conclui-se que o desempenho da rede não é linear com a quantidade de agentes do sistema de replicação. Redes com uma maior quantidade de nodos apresentam uma dispersão maior de recursos, exigindo um maior número de operações para ingresso em um estágio de plena operação[6]. Pelo protocolo *FreeNet*, conforme requisições caminham pela rede nodos passam a se conhecer. No cenário de três mil nodos, estas ligações formadas ainda são

[6] *O termo 'plena operação' denota uma situação em que grande parte das requisições da rede são completadas com sucesso, ou seja, o momento no qual a rede está efetivamente realizando sua função de prover conteúdo.*

insuficientes para permitir que recursos sejam encontrados com o valor de escopo de inundação oferecido (nove passos).

4.4 Sumário

Nesta seção foram apresentados os experimentos de escalabilidade realizados através do simulador *Aurora* referente às redes *FreeNet* com o objetivo de identificar os ganhos potenciais pela utilização de um sistema de replicação. Cada experimento é realizado com e sem a presença dos agentes de replicação, de forma a possibilitar a comparação de resultados. O desempenho da rede é medido através do número de propagações que uma mensagem sofre para ser atendida.

Com a rede integrada com agentes do sistema de replicação, requisições são atendidas de forma mais eficiente, exigindo-se menos propagações entre nodos e melhorando o desempenho do sistema. Quanto maior é a rede, mais esforço é necessário para a localização de um recurso.

As simulações indicam que agentes do sistema de replicação para um sistema par-a-par oferecem ganhos de desempenho através da redução lógica de distâncias da rede. No caso específico das redes *FreeNet*, conforme o sistema opera e os nodos interagem, atalhos são criados e nodos distantes se tornam alcançáveis. Em um instante de tempo futuro, agentes do sistema de replicação possuirão tantas referências quantas forem comportadas em seu repositório de dados. No próximo capítulo será apresentada a implementação real de um sistema de replicação para os ambiente par-a-par *DirectConnect* e *eDonkey*.

CAPÍTULO 5

AVALIAÇÃO DE INTEROPERABILIDADE

Nesta seção é apresentado o sistema computacional desenvolvido para atuar como replicador de conteúdo par-a-par, bem como os experimentos realizados para comprovação de sua eficácia na redução da utilização de banda de rede.

5.1 Protótipo do sistema de replicação

Neste trabalho avaliam-se os ganhos de otimização de transferências de conteúdo par-a-par que podem ser obtidos através do posicionamento do sistema de replicação na fronteira da rede corporativa com a Internet. Como estratégia de interação entre nodos, o sistema pode:

a) permitir que todos os nodos sejam visíveis a qualquer outro nodo da rede, uma estratégia interessante que pode ser aplicada a redes que possuem uma entidade concentradora (*DirectConnect*, por exemplo), tornando-a mais atraente aos usuários pela possibilidade de utilização de eventuais funcionalidades extras relativas à interação de usuários em tempo real (troca de mensagens instantâneas, por exemplo);
b) mascarar a existência de nodos vizinhos, permitindo que cada nodo enxergue apenas o sistema de replicação é um mecanismo interessante quando o sistema de replicação atua em redes par-a-par de acesso completamente descentralizado;

O sistema de replicação deve ser percebido pela rede como um nodo típico, destacando-se apenas por sua capacidade de oferecer conteúdo e alta disponibilidade. Neste sistema, a entidade responsável por replicar arquivos e referências[7] é denominada de **repositório de replicação**. Através do armazenamento de referências para arquivos, em adição ao conteúdo dos arquivos, o sistema torna-se apto a responder solicitações de localização de arquivos sem a necessidade de propagar uma consulta pela rede. Ao receber uma solicitação de localização de arquivo proveniente da rede, o sistema realiza uma busca no seu repositório de replicação. Caso exista referência ao recurso solicitado, esta é retornada ao nodo solicitante. Caso negativo, o sistema dispara uma busca na rede. Caso o recurso seja encontrado, o sistema armazena a referência do arquivo no repositório de replicação e envia uma mensagem de recurso encontrado ao nodo

[7] *Referências são links para arquivos presentes na rede, conhecidos pelo sistema de replicação e ainda não presentes no repositório de replicação.*

solicitante, declarando-se detentor do recurso e disponibilizando-se a entregá-lo. Caso o nodo opte por receber o conteúdo, o sistema realiza a obtenção efetiva do arquivo, replica-o internamente e encaminha uma cópia para o nodo solicitante. Para as próximas solicitações referentes a este mesmo arquivo, o sistema de replicação estará apto a realizar o seu serviço de forma independente do nodo detentor original. Com esta política de atuação, o sistema de replicação caracteriza-se como **não transparente** (já que altera parâmetros da requisição) e **replicador** (pelo fato de armazenar uma cópia do conteúdo localmente). Destacam-se como principais desafios da implementação deste sistema:

a) identificar um comportamento genérico seguido por todas as redes par-a-par de compartilhamento de arquivos;
b) projetar um mecanismo para a correta identificação de recursos nas diversas redes par-a-par, pois, em contraposição ao ambiente Web, a utilização do nome do arquivo como identificador único não é efetiva;
c) reconhecer os diversos tipos de encriptação de conteúdo praticado pelas redes par-a-par e prover mecanismos que permitam sua extensão ou a incorporação de novos mecanismos desenvolvidos *a posteriori*;
d) prover mecanismos eficientes para a alocação de memória, processamento e armazenamento, devido aos grandes montantes de dados manipulados por uma aplicação deste tipo;
e) permitir a expansão do sistema de replicação para tratar novas redes par-a-par de compartilhamento de arquivos;
f) prover políticas inteligentes para a reposição de arquivos no repositório de replicação, visto que seu compartilhamento é uma tarefa susceptível a eventos

freqüentes de mudanças na popularidade (como exemplo, o lançamento de um novo filme ou música);

g) lidar com heterogeneidades das diversas redes;
h) não tornar-se um ponto de contenção de desempenho ou falhas: a motivação para o desenvolvimento de um sistema deste tipo é justamente ampliar a disponibilidade de recursos e otimizar suas transferências;

Desta forma, identificam-se como características necessárias em um sistema de replicação de conteúdo par-a-par:

a) operação com redes par-a-par distintas através da realização do processo de compartilhamento de arquivos entre estas, possibilitando que usuários (ou clientes) de uma determinada rede tenham acesso ao conteúdo mantido pela outra (funcionalidade de *gateway*);
b) suporte à requisição de intervalos, estando apto a receber e entregar arquivos de forma segmentada baseada na requisição de pequenos trechos, visto que diversas redes par-a-par utilizam-se deste artifício;
c) facilidade e praticidade para inclusão de suporte a novas redes par-a-par emergentes, mantendo o sistema de replicação sempre atualizado com as novas tendências dos usuários;
d) disponibilidade independente entre redes, na eventualidade de uma das redes alvo do sistema de replicação tornar-se indisponível, as operações executadas pelos clientes das outras redes devem permanecer operacionais.

5.1.1 Organização do sistema de replicação

Para que o sistema de replicação de conteúdo par-a-par opere de forma adequada devem ser implementados alguns serviços independentes da rede alvo, tais como:

a) gerenciar filas de requisições;
b) distribuir capacidade de processamento entre as diversas requisições;
c) localizar arquivos nas diversas redes par-a-par tratadas;
d) obter arquivos da rede;
e) armazenar arquivos no sistema de replicação;
f) implementar uma política de reposição de páginas em memória e em disco;
g) entregar arquivos a nodos solicitantes.

Embora os serviços citados anteriormente tenham caráter generalista e independente de rede, também é possível identificar serviços específicos como:

a) autenticação de usuários;
b) formatação de pacotes para envio e recebimento;
c) algoritmos de criptografia de arquivos;
d) distribuição de mensagens instantâneas e comandos de administração de usuários;
e) limitação de transferências simultâneas por nodos;
f) outros detalhes referentes a rede par-a-par em questão.

Estes serviços não apresentam relevância sob o aspecto da replicação, porém, são fundamentais para que o sistema consiga interagir com os nodos da rede. A existência

destas duas classes de serviços motiva o projeto do sistema de replicação em duas camadas complementares, conforme a FIG. 7.

FIGURA 7 – Camadas do sistema de replicação

A primeira camada contém, efetivamente, a inteligência do sistema com operações fundamentais de compartilhamento de arquivos (obtenção e disponibilização de arquivos, armazenamento e organização dos arquivos em disco, política de reposição de arquivos, dentre outros) e a implementação de um nodo genérico de uma rede par-a-par. A segunda camada é responsável pela interoperação do sistema com as diversas redes par-a-par alvo da replicação.

Esta modelagem tem o efeito prático de criar um sistema par-a-par genérico de compartilhamento de arquivos. A camada de inteligência funciona como a padronização de operações realizadas pela rede, seguida pela camada de interoperação que trata da tradução do protocolo padrão definido pelo sistema de replicação para os protocolos específicos de cada rede, sendo esta a razão de sua denominação de camada de tradução. Para o sistema de replicação interessam apenas as operações de ‘enviar busca por arquivo’, ‘receber resposta da busca’, ‘receber trecho de arquivo’ e ‘enviar trecho de arquivo’.

Enquanto na camada de inteligência (ou núcleo) existe apenas uma entidade (o núcleo em si), na camada de interoperação existe uma entidade de tradução para cada rede par-a-par gerenciada (também chamada de *plugin*). Esta abordagem apresenta

vantagens como a simplificação do processo de acréscimo de novos protocolos (*plugins*) e a possibilidade de compartilhamento de recursos entre redes par-a-par distintas (funcionalidade de *gateway*). A FIG. 8 detalha a organização interna proposta para o sistema de replicação.

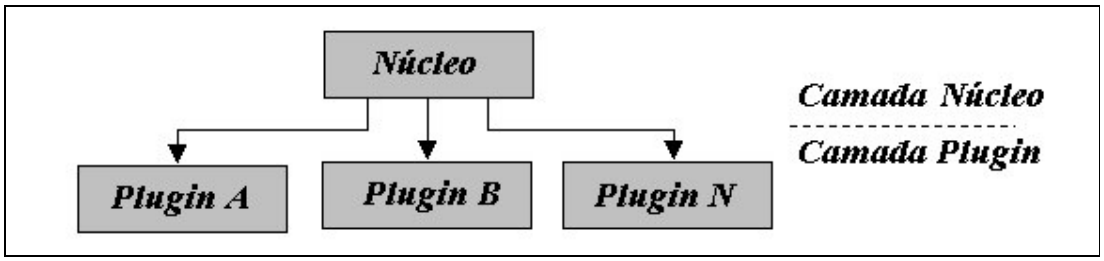

FIGURA 8 – Detalhamento do sistema de replicação

5.1.2 Localização de arquivos

Para tornar mais claro o funcionamento do sistema de replicação, apresenta-se um exemplo ilustrativo do caminho percorrido por uma requisição de localização de arquivo: suponha duas redes par-a-par **Ra** e **Rb** interligadas pelo sistema de replicação **S,** composto pelo núcleo **N** e pelos *plugins* **Pa** e **Pb**. Na rede **Ra** existem os nodos **Na1** e **Na2** (nodos comuns clientes da rede **Ra**), enquanto que na rede **Rb** existem os nodos **Nb1** e **Nb2** (nodos comuns clientes da rede **Rb**), conforme ilustra a FIG. 9.

Suponha que o nodo **Na1** tente localizar o arquivo **A** que encontra-se presente no nodo **Nb1**. Para realizar esta operação, o nodo **Na1** envia uma requisição de busca para o *plugin* **Pa**, que realiza a interface do sistema de replicação com a rede **Ra**. Este *plugin* propaga a requisição para o núcleo **N** que verifica se este arquivo possui localização conhecida. Em caso afirmativo, o núcleo **N** retorna que conhece o arquivo para o *plugin*

Pa. O *plugin* **Pa** declara-se detentor do arquivo e retorna a resposta para o nodo solicitante **Na1**.

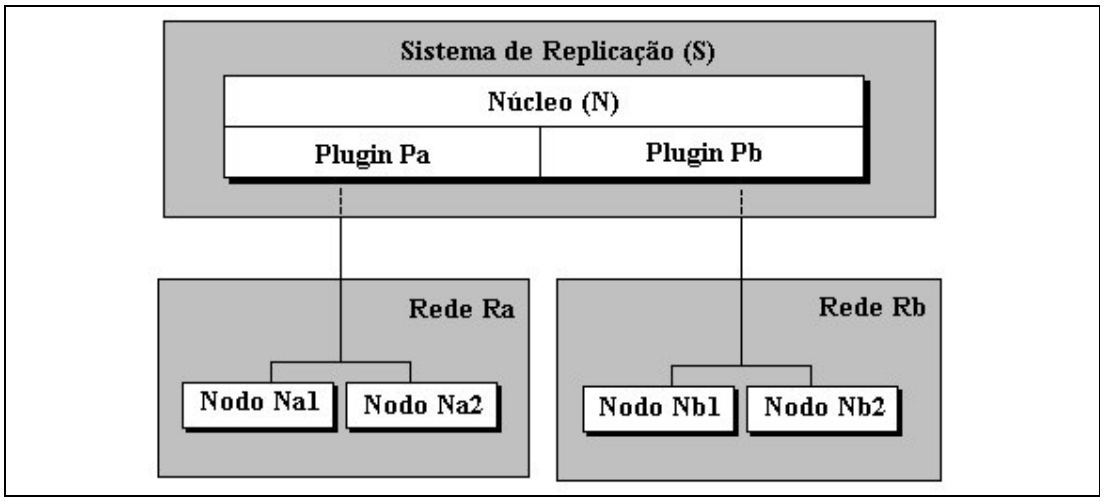

FIGURA 9 – Interação do sistema de replicação com redes par-a-par

No caso do núcleo não ter visto este arquivo antes, uma solicitação de busca pelo arquivo **A** é enviada para os *plugins* **Pa** e **Pb**. Cada *plugin* propaga a consulta para os nodos de sua rede. Quando a consulta atinge o nodo **Nb1** (detentor do recurso), uma resposta é retornada ao *plugin* **Pb**, que trata de repassá-la ao núcleo. O núcleo realiza seu armazenamento na lista de arquivos conhecidos e retorna a resposta ao *plugin* **Pa**, que a converte para o protocolo utilizado na rede **Ra**, declarando-se detentor original do recurso e propagando a resposta para o nodo solicitante **Na1**. A FIG. 10 apresenta os passos seguidos em uma requisição de localização de arquivos.

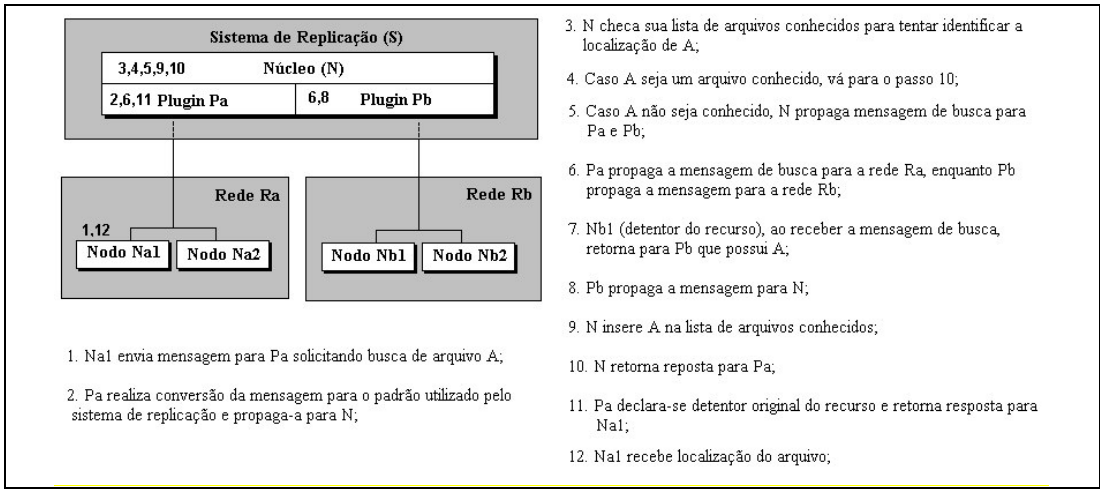

FIGURA 10 – Fluxo para localização de recursos

Generalizando, um nodo deve encaminhar sua solicitação para o sistema de replicação. Esta solicitação é recebida no sistema pelo chamado *plugin*, que é a camada de interface que entende os detalhes da rede par-a-par de origem da requisição, decodificando a mensagem e analisando seu conteúdo. O *plugin* então encaminha esta mensagem em um formato padrão para o núcleo do sistema de replicação. O núcleo tenta identificar se conhece a localização do recurso. Caso afirmativo, uma mensagem é enviada do núcleo para o *plugin* que solicitou a localização. O *plugin*, ao receber a resposta do núcleo, declara-se detentor do recurso e propaga a mensagem para o nodo solicitante. Caso o núcleo não conheça o arquivo, será necessário realizar uma busca para checar se este arquivo existe em algum dos nodos conhecidos pelo sistema de replicação. Tal requisição de localização é enviada para todos os *plugins* de forma paralela. Cada *plugin* converte a requisição para o formato de sua rede e a propaga para seus nodos. Conforme são obtidos resultados positivos para a requisição, o *plugin* os propaga para o núcleo. Por sua vez, o núcleo propaga o resultado de volta ao *plugin* que solicitou o arquivo. Por fim, o *plugin* propaga a resposta para o nodo solicitante.

5.1.3 Obtenção de arquivos

O processo para que um nodo consiga obter um arquivo é similar ao de uma localização. Considere as mesmas redes **Ra** e **Rb** interligadas pelo sistema de replicação **S**, composto pelo núcleo **N** e pelos *plugins* **Pa** e **Pb**, existindo na rede **Ra** os nodos **Na1** e **Na2** e na rede **Rb** os nodos **Nb1** e **Nb2**. Suponha que o nodo **Na1** deseje baixar o arquivo **A** presente no nodo **Nb1**.

Inicialmente, o usuário deve realizar uma pesquisa para encontrar o arquivo de interesse (processo usual em aplicações de compartilhamento de arquivos). Esta localização de arquivo retorna um nome codificado que contém o *plugin* detentor do recurso, nodo detentor do recurso e caminho do arquivo no nodo. Como exemplo: `'\\<plugin>\<nodo>\<caminho do arquivo>'`. Desta forma, o nodo recebe a resposta como `'\\Pb\Nb1\A'`. De posse desta informação, o nodo **Na1** submete a requisição de obtenção de arquivo. O *plugin* **Pa** reconhece a operação e a propaga para o núcleo **N**, que verifica se este arquivo já encontra-se presente no repositório de replicação.

Em caso afirmativo, o núcleo informa ao *plugin* **Pa**, que repassa esta mensagem para **Na1**. **Na1** dá início ao processo de *download* solicitando segmentos do arquivo (ou o arquivo como um todo, dependendo da rede par-a-par em questão). Caso o núcleo não tenha o arquivo no repositório de replicação, este deverá ser obtido. Então, **N** utiliza o nome codificado para determinar o nodo detentor do arquivo solicitado. Neste caso, uma requisição de obtenção de arquivo será enviada para o *plugin* **Pb** informando o caminho do arquivo (**A**) e o nodo detentor (**Nb1**). **Pb** contata o nodo **Nb1** e solicita o recebimento de uma cópia do arquivo **A**. **Nb1** transfere os bytes para **Pb** que os repassa

ao núcleo, onde são armazenados no repositório de replicação. Conforme o arquivo é transferido, pacotes de dados são propagados do núcleo para o *plugin* **Pa**, que os encaminha para o nodo **Na1** (solicitante original). A partir deste momento, as próximas requisições ao arquivo **A** serão servidas pelo sistema de replicação. A FIG. 11 sumariza este processo.

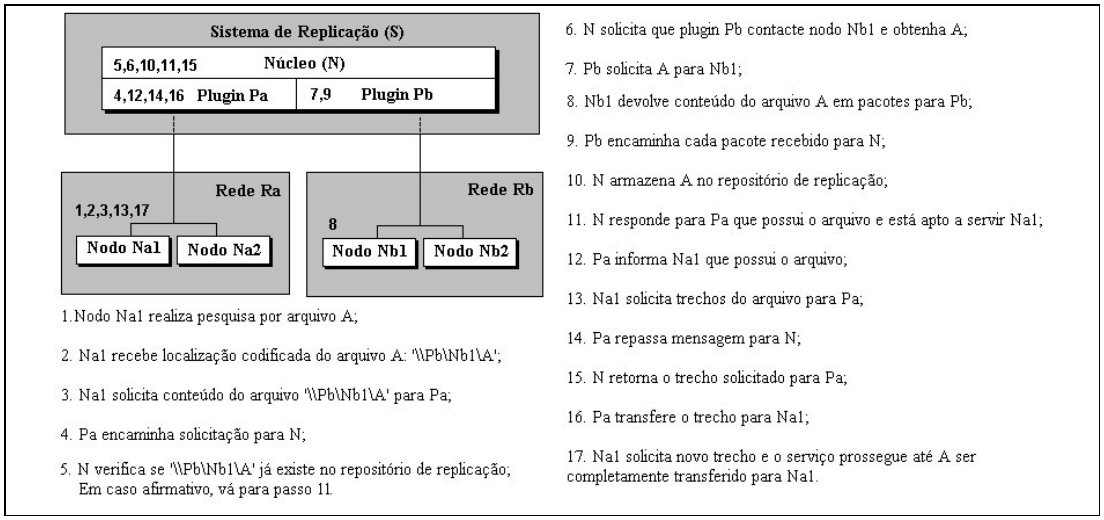

FIGURA 11 – Fluxo para obtenção de recursos

5.1.4 Repositório de replicação

O sistema possui dois repositórios de replicação: repositório de referências; repositório de conteúdo. O **repositório de referências** armazena a localização de arquivos, sendo populado pelo núcleo do sistema de replicação quando são oferecidas respostas de requisições de localização de arquivos, reduzindo-se a necessidade de comunicação do replicador com os nodos da rede para a obtenção de respostas redundantes (já conhecidas).

Devido ao caráter transiente dos nodos, cada registro do repositório de referências deve ser associado a um selo de validade, permitindo-se que estes registros expirem por decurso de prazo, mantendo-se a lista sempre atualizada e em conformidade com a rede. O **repositório de conteúdo** tem como objetivo permitir a replicação efetiva dos arquivos (que são armazenados em segmentos), aumentando sua disponibilidade e reduzindo o tempo necessário ao seu acesso. Para cada arquivo são mantidos, dentre outros, os seguintes dados: *plugin* que o originou, endereço do nodo que o disponibilizou, nome do arquivo, tamanho do arquivo, tamanho do segmento, tipo de encriptação utilizada no arquivo, quantidade de segmentos, tamanho de cada segmento e chave de identificação.

O cálculo da chave de identificação é baseado no conteúdo efetivo do arquivo/segmento, de forma a permitir a identificação de arquivos iguais com nomes distintos, bem como arquivos distintos com mesmo nome. A utilização do armazenamento de arquivos de forma segmentada é interessante pois representa a possibilidade de iniciar o serviço de um arquivo, mesmo antes dele estar todo armazenado no repositório de replicação.

5.1.5 Implementação do sistema de replicação

A implementação do sistema de replicação foi realizada na linguagem C utilizando-se a biblioteca APR da *Apache Software Foundation* [1], um *framework* consolidado para implementação de aplicações com grandes requisitos de desempenho, além de um meio de prover interoperacionalidade entre diversas plataformas.

A comunicação entre núcleo e *plugins* se dá através da chamada de funções pertencentes a uma *interface de programação de aplicações* (API)[8] disponibilizada pelo núcleo, bem como pelo compartilhamento de memória. Durante o processo de compilação, o núcleo e cada *plugin* geram um arquivo objeto específico. O último passo da compilação refere-se à junção destes objetos, gerando-se um executável que implementa o sistema de replicação. Como vantagens desta opção, destaca-se sua simplicidade e possibilidade de compilação individual de módulos, já que o núcleo é um sistema complexo composto de mais de duzentas mil linhas de código. Como desvantagens, por se tratar de um único executável, o sistema não permite ser executado em diversas máquinas como apenas uma instância. No entanto, esta estratégia atende aos requisitos do sistema, pois sua extensão depende apenas da codificação de novos *plugins* que suportem outros tipos de redes.

O programa principal deve iniciar o sistema de replicação através da chamada da função `P4P_Init`, passando como parâmetro uma estrutura do tipo `p4p_t`. Em seguida, para cada *plugin*, o programa deve realizar seu registro através da função `P4P_AddPlugin`, especificando a função de inicialização do *plugin*. No sistema de replicação existem dois sentidos de interação: núcleo solicita serviços do *plugin* e *plugin* solicita serviços do núcleo.

5.1.5.1 Comunicação núcleo - *plugin*

Durante sua inicialização, cada *plugin* aloca a memória necessária para sua operação, bem como define a estrutura `p4p_plug_t` que será freqüentemente acessada pelo núcleo

[8] *Application programming interface.*

em suas tarefas de comunicação com o *plugin*. Nesta estrutura, o *plugin* preenche quais são as funções que o núcleo deve acionar para realizar determinadas operações definidas na tabela abaixo. Finalmente, após preenchida a estrutura `p4p_plug_t`, um *plugin* deve instanciar uma ou mais linhas de execução para realização de atividades como aguardar por requisições provenientes da rede e gerenciar os nodos descobertos. Num momento futuro, algum nodo solicitará operações para o *plugin*. Suponha que esta operação seja o envio de uma mensagem instantânea para algum outro nodo da rede. Neste caso, o *plugin* simplesmente encaminha esta mensagem ao nodo destino, pois não se trata de uma operação referente ao sistema de replicação.

A TAB. 1 apresenta a relação entre as operações solicitadas do núcleo ao *plugin* e funções responsáveis por sua execução. Nesta tabela, as funções *callback* merecem uma atenção maior. Quando o *plugin* solicita a realização de uma consulta ou a obtenção do conteúdo de um arquivo ao núcleo, não existem garantias de quando este resultado retornará, ou mesmo se retornará (o nodo que detém o arquivo pode deixar a rede, por exemplo). Desta forma, quando o núcleo conseguir resultados para as solicitações, estes serão enviados através destas funções, que podem ser chamadas a qualquer momento, exigindo que todos os dados de controle para a entrega da solicitação estejam disponíveis (endereço do nodo, porto de conexão e demais dados referentes ao protocolo).

TABELA 1 – Apresentação das operações solicitadas do núcleo para o plugin

Operação	**Função responsável**
Núcleo solicita que plugin suspenda sua execução	`p4p_plug_t.suspended`
Núcleo solicita que plugin realize a liberação de recursos (usualmente, para finalizar a execução)	`p4p_plug_t.cleanup_fn`
Núcleo solicita que plugin retorne a execução normal	`p4p_plug_t.resume_fn`
Núcleo solicita propriedades de operação do plugin	`p4p_plug_t.getprops_fn`
Núcleo solicita o estado atual do plugin (em execução ou suspenso)	`p4p_plug_t.getstatus_fn`
Núcleo solicita que plugin realize uma busca por arquivo em sua rede	`p4p_plug_t.query_fn`
Núcleo solicita que plugin realize o download de um trecho de arquivo	`p4p_plug_t.download_fn`
Núcleo solicita que plugin obtenha o tamanho de um arquivo que será posteriormente baixado	`p4p_plug_t.getsize_fn`
Função callback através da qual o núcleo informa respostas de uma requisição de busca para o plugin	`p4p_plug_t.addqyreps_fn`
Função callback através da qual o núcleo disponibiliza um trecho de arquivo solicitado para download pelo plugin	`p4p_plug_t.addbytes_fn`

A função `p4p_plug_t.addqyreps_fn` recebe como parâmetros a estrutura representando a consulta realizada (`p4p_needq_t`), originalmente preenchida pelo *plugin* no momento da solicitação e o conjunto de resultados retornados pelo núcleo (`p4p_qyreplist_t`). O corpo interno da função consiste de um laço, onde, para cada

registro retornado, os dados são convertidos da estrutura de resposta e encaminhados para o nodo solicitante.

A função `p4p_plug_t.addbytes_fn` possui um comportamento similar, ou seja, existe uma estrutura que foi preenchida pelo *plugin* indicando a operação de *download* a ser realizada (`p4p_needb_t`), bem como uma estrutura contendo os bytes referentes ao arquivo (`p4p_bytereplist_t`).

5.1.5.2 Comunicação *plugin* - núcleo

No caso do *plugin* receber uma mensagem passível de replicação (isto é, localização de arquivos, consulta ao tamanho de arquivos e solicitações de *download*), o *plugin* deve extrair os dados relevantes (geralmente, nome do arquivo) e chamar funções especiais definidas na interface de comunicação com o núcleo, apresentadas na TAB. 2.

TABELA 2 – Operações solicitadas do plugin para o núcleo

Operação	**Função Responsável**
Plugin solicita resultados para uma busca de arquivos. Quando o núcleo consegue os dados, ele os retorna através de `p4p_plug_t.addqyreps_fn`	`P4P_NeedQyReps()`
Plugin solicita trecho do arquivo para o núcleo. Quando o núcleo consegue os dados, ele os retorna através de `p4p_plug_t.addbytes_fn`	`P4P_NeedBytes()`

Quando o *plugin* recebe uma solicitação de localização de arquivos proveniente da rede, a estrutura `p4p_needq_t`, responsável por descrever uma operação de localização, deve ter os campos preenchidos conforme detalhado abaixo:

a) campo `p4p_needq_t.plug` deve receber a estrutura `p4p_plug_t`, identificadora do *plugin* solicitante;

b) campo `p4p_needq_t.scope` define se a operação deve se restringir a recursos armazenados no sistema de replicação (`P4P_UseCache`), somente recursos externos (`P4P_UseExt`) e se a operação deve sofrer novas tentativas em caso de falha (`P4P_UseRetry`), ou qualquer combinação destas opções (ex. `P4P_UseCache | P4P_UseExt`);

c) campo `p4p_needq_t.canceled` deve ser preenchido como zero. Este campo permite cancelar a operação após ter sido iniciada, bastando apenas alterar seu valor diferente de zero;

d) campo `p4p_needq_t.bw_min` restringe a busca para apenas nodos com taxa de transmissão maior que o valor associado a esta variável;

e) campo `p4p_needq_t.exts` apresenta as extensões de arquivos que interessam à pesquisa;

f) campo `p4p_needq_t.size_max` restringe a busca a arquivos inferiores ao valor associado a esta variável;

g) campo `p4p_needq_t.size_min` restringe a busca a arquivos maiores que o valor associado a esta variável;

h) campo `p4p_needq_t.keywords` apresenta a lista de palavras-chave a serem pesquisadas.

Então, basta chamar a função `P4P_NeedQyReps`, fornecendo como parâmetro as estruturas `p4p_t` e `p4p_needq_t`. Os resultados da busca serão retornados através da função *callback* `p4p_plug_t.addqyreps_fn` (detalhada anteriormente).

Por outro lado, quando o *plugin* recebe uma solicitação de *download* de arquivo proveniente da rede, a estrutura `p4p_needb_t`, responsável por descrever uma operação de *download*, deve ser preenchida, conforme detalhado abaixo:

a) campo `p4p_needb_t.plug` deve receber a estrutura `p4p_plug_t` identificadora do *plugin* solicitante;

b) campo `p4p_needb_t.scope` define se a operação deve se restringir a recursos armazenados no sistema de replicação (`P4P_UseCache`), somente recursos externos (`P4P_UseExt`) e se a operação deve sofrer retentativas em caso de falha (`P4P_UseRetry`), ou qualquer combinação destas opções (ex. `P4P_UseCache | P4P_UseExt`);

c) campo `p4p_needb_t.canceled` deve ser preenchido como zero. Este campo permite cancelar a operação de *download* após ter sido iniciada, bastando apenas igualar seu valor diferente de zero;

d) campo `p4p_needb_t.path` é preenchido com o caminho do arquivo quando o *plugin* deseja obter arquivos de outras redes. Caso o arquivo esteja sendo procurado apenas na rede do *plugin* requisitor, este campo deve estar nulo;

e) campo `p4p_needb_t.uri` é preenchido com o caminho do arquivo quando o *plugin* deseja obter arquivos apenas de sua rede par-a-par. Esta diferenciação entre os campos `path` e `uri` é necessária. O campo `path` é preenchido segundo o padrão de nomes do núcleo, enquanto que o campo `uri` pode ser preenchido com um padrão definido pelo *plugin*;

f) campo `p4p_needb_t.range.lo` é preenchido com o limite inferior do intervalo de bytes a serem obtidos do arquivo (menor valor aceito é zero);

g) campo `p4p_needb_t.range.hi` é preenchido com o limite superior do intervalo de bytes a serem obtidos (maior valor é o tamanho do arquivo menos um);

h) campo `p4p_needb_t.plug_data` é um apontador de livre uso pelo *plugin*, possibilitando que dados de controle sejam recuperados em momentos futuros.

Feito isto, basta chamar a função `P4P_NeedBytes` fornecendo como parâmetro as estruturas `p4p_t` e `p4p_needb_t`. Os bytes solicitados serão retornados através da função *callback* `p4p_plug_t.addbytes_fn` (detalhada anteriormente).

5.2 Estudo de casos

Para a elaboração dos estudos de casos e obtenção de resultados experimentais, optou-se pela implementação dos *plugins* referentes às redes *DirectConnect* e *eDonkey*. A utilização de duas redes deve-se a necessidade de avaliar a funcionalidade de interligação entre sistemas par-a-par distintos. Para guiar os experimentos foram utilizados dados reais cedidos por um provedor nacional de grande porte.

A FIG. 12 apresenta a arquitetura desenvolvida para o protótipo.

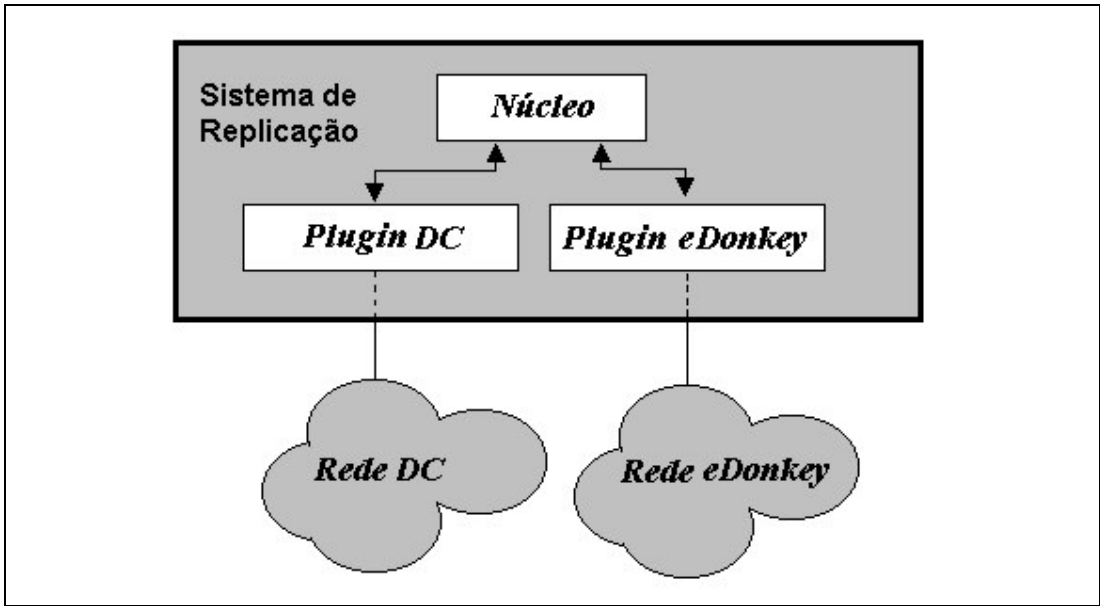

FIGURA 12 – Arquitetura do protótipo desenvolvido

A seguir detalha-se a arquitetura do protótipo desenvolvida e utilizada nos experimentos.

5.2.1 Estudo de caso - DirectConnect

DirectConnect é um protocolo em formato texto desenvolvido pela corporação *Neo-Modus.com*, que permite que nodos independentes compartilhem arquivos de seus repositórios locais. Tal protocolo define entidades centralizadoras denominadas *hubs*, que permitem a disseminação da informação entre os clientes conectados. *Hubs* também comunicam entre si permitindo a troca de mensagens de nodos registrados em *hubs* vizinhos. Este protocolo contempla a comunicação cliente-servidor, cliente-cliente e servidor-servidor. A FIG. 13 apresenta a topologia típica de uma rede *DirectConnect*.

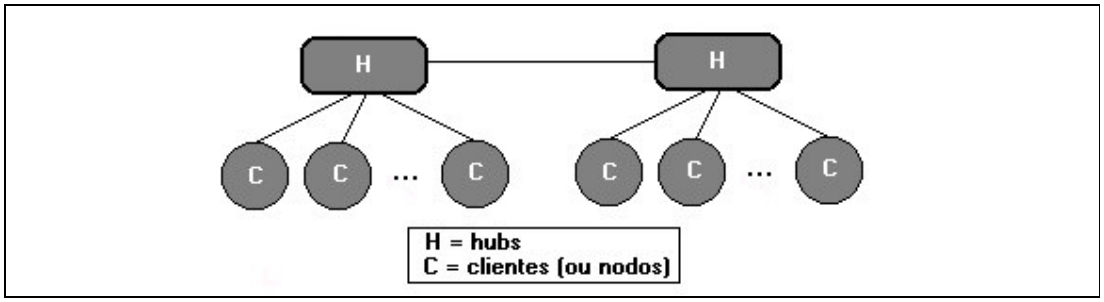

FIGURA 13 – Topologia de uma rede DirectConnect

A **comunicação cliente-servidor** se dá através de uma conexão TCP/IP na qual o cliente, para fazer parte da rede, deve se conectar a algum *hub* existente. O comportamento padrão do *hub* é aguardar pela conexão do cliente no porto quatrocentos e onze, não havendo, entretanto, empecilhos para a utilização de um outro porto qualquer. Após a conexão do cliente, o protocolo define um procedimento de autenticação. Inicialmente, o cliente propaga uma mensagem contendo uma seqüência de bytes aleatória. O *hub*, então, aplica uma rotina aritmética padrão sobre esta mensagem, retornando o resultado para o cliente. Caso o conteúdo retornado esteja correto, o cliente identifica o *hub* como válido. Então, este mesmo processo se repete, mas com os papéis invertidos, com o objetivo de permitir ao *hub* a identificação da validade do cliente. Caso este processo também tenha sucesso, o *hub* envia seu identificador para o cliente, usualmente seguido de uma mensagem de boas-vindas. Em seguida, ocorre um processo de autenticação do usuário presente no cliente através do tradicional mecanismo de *login* e *senha*. Sendo positiva a identificação, o *hub* retorna a versão de seu *software* e recebe do cliente um descritivo sobre suas características (velocidade de conexão, bytes compartilhados, endereço de correio eletrônico do usuário e outros). Esta mensagem é propagada a todos os clientes conectados.

O protocolo define algumas mensagens que podem ser utilizadas pelo cliente para a consulta dos nodos atualmente conectados. Também existe o suporte para envio de mensagens instantâneas entre os clientes ativos. Em termos de busca por conteúdo, o *DirectConnect* define duas classes: busca de conteúdo apenas nos clientes conectados no *hub* local; busca de conteúdo nos *hubs* vizinhos. De qualquer modo, o cliente sempre envia a busca para o *hub*. Entretanto, são definidos dois modos de recepção de resultados: ativo; passivo.

No **modo ativo**, o cliente aguarda pelas respostas diretamente dos clientes que possuem o conteúdo pesquisado em um porto especificado no comando de consulta. Este é o método tradicional. Já no **modo passivo**, os clientes encaminham as respostas para o *hub*, que as propaga ao cliente solicitante. Este modo é útil quando o cliente encontra-se atrás de um *firewall*. Sendo assim, são identificadas quatro variantes de consultas, conforme FIG. 14: busca local ativa; busca local passiva; busca *inter-hubs* ativa; busca *inter-hubs* passiva.

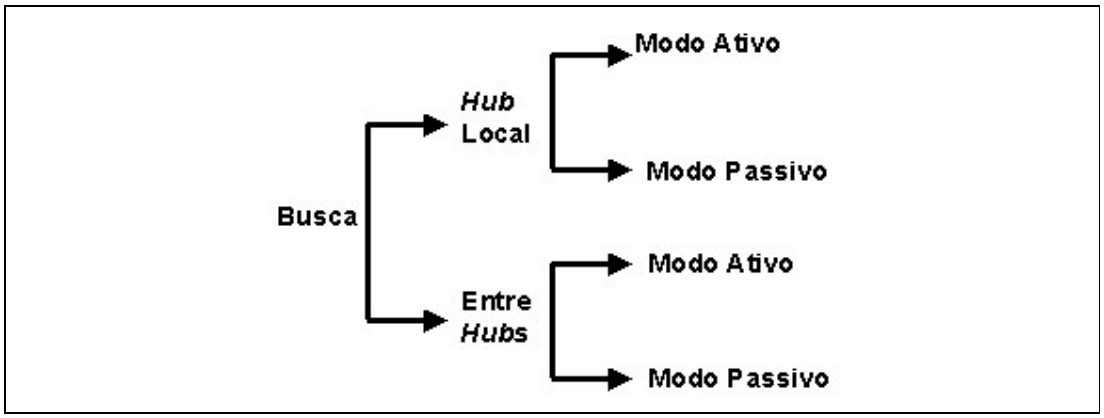

FIGURA 14 – Tipos de Buscas do Protocolo DirectConnect

A **comunicação cliente-cliente** ocorre através de uma conexão TCP/IP na qual dois clientes se conectam para a realização de transferências de conteúdo. Para que esta

conexão ocorra, uma das partes deve solicitá-la ao servidor. Existem duas mensagens distintas de conexão. Uma utilizada para conexão com clientes conectados ao mesmo *hub* e outra utilizada para conexão com clientes registrados em *hubs* vizinhos. Após a conexão, ocorre um processo de autenticação similar ao caso cliente-servidor. O protocolo restringe a comunicação cliente-cliente à listagem de arquivos compartilhados e transferência de conteúdo de arquivos. O protocolo também define um mecanismo de desempate para o caso dos dois clientes solicitarem operações de transferências simultaneamente.

A **comunicação servidor-servidor** é realizada através de pacotes UDP (*User Datagram Protocol*), com cada *hub* interrogando seus vizinhos periodicamente (*heartbeat*). Além destas mensagens de *ping*, os *hubs* também trocam mensagens de solicitação de conexão entre nodos conectados em servidores distintos, e, também, buscas entre *hubs*.

5.2.1.1 Plugin DirectConnect

Para a construção do *plugin* responsável pelo gerenciamento da rede *DirectConnect* deve-se escolher uma entidade para atuação como um agente do sistema de replicação. A organização das redes *DirectConnect* apresenta duas entidades bem definidas e com papéis distintos: *hubs*; clientes. ***Hubs*** são entidades concentradoras de clientes e, por isso, centrais à rede. Desta forma, possuem uma visão mais ampla de sua estrutura. Entretanto, *hubs* não servem arquivos ou respondem a consultas. Clientes são os responsáveis por servir conteúdo e responder a consultas. Por outro lado, possuem uma visão limitada da estrutura da rede. Para a implementação do sistema de replicação necessita-se escolher uma entidade bem posicionada na rede, de forma que esta possa

interceptar as requisições. Entretanto, o sistema de replicação também deve estar apto a servir conteúdo. Sendo assim, o *plugin* deve ser uma entidade mista, possuindo um lado *hub*, mas também, um lado cliente. Inicialmente, o agente do sistema de replicação se comporta como um *hub* permitindo que clientes se conectem a ele e realizem suas operações. Mas, em momentos específicos, esta entidade executa atividades de clientes, respondendo a consultas, buscando e servindo conteúdo. A FIG. 15 ilustra a arquitetura do *plugin*.

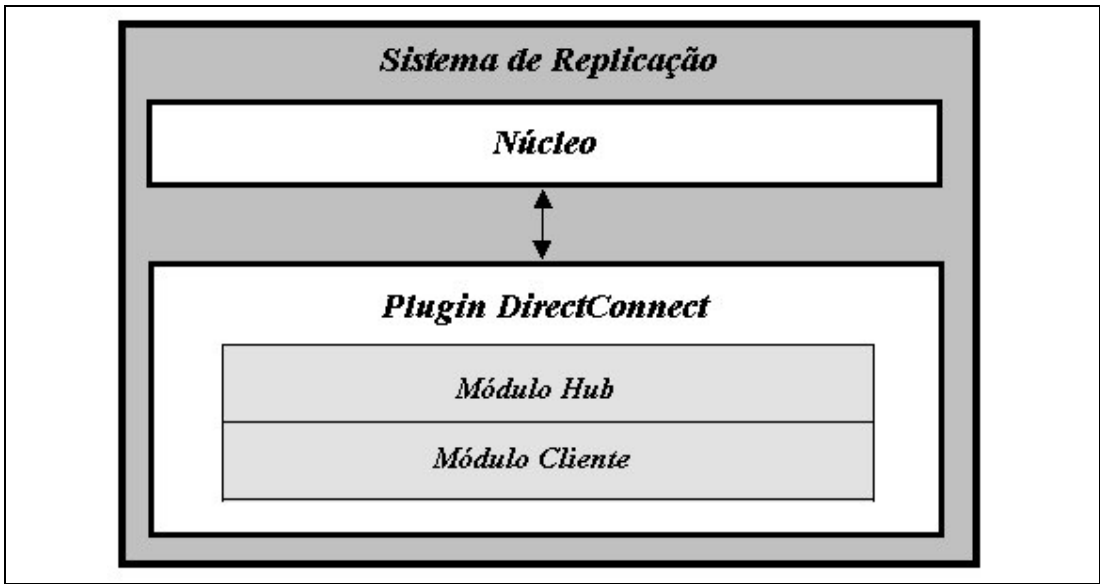

FIGURA 15 – Plugin DirectConnect

Para implementar a funcionalidade de sistema de replicação conjugada com a funcionalidade de *hub* no protocolo *DirectConnect*, alterações foram necessárias no tratamento de algumas mensagens enviadas pelos clientes.

Mensagem *Hello* é enviada pelo *hub* para todos os clientes conectados após a conexão de um novo cliente. Nela são descritos alguns dados do novo nodo como apelido, endereço de correio eletrônico, quantidade de bytes e lista de arquivos compartilhados. Nas fases iniciais deste trabalho considerou-se a possibilidade de não

repassar tal tipo de informação aos clientes, mantendo-os sem o conhecimento da presença de outros nodos nesta rede. Entretanto, desta forma a rede poderia se tornar pouco interessante aos seus usuários. Para esses, a ausência de clientes conectados representa ausência de conteúdo compartilhado. Além do mais, o *DirectConnect* não é um protocolo restrito apenas à troca de arquivos, contemplando, também, o serviço de mensagens instantâneas. Sendo assim, na implementação realizada neste trabalho, o *hub* apresenta aos clientes conectados a identificação do nodo entrante, bem como sua lista de arquivos compartilhados.

Mensagem *ConnectToMe* é utilizada quando um cliente (A) deseja se conectar a outro (B) para a realização de uma transferência de arquivos. Neste caso, o *hub* dispara uma linha de execução que irá simular o cliente solicitado (B). Sempre que o cliente solicitante (A) enviar alguma mensagem, este cliente virtual (V) a interceptará e tentará respondê-la (por exemplo, consulta ao tamanho de um determinado arquivo ou solicitação de obtenção de arquivo). Caso este cliente virtual (V) não tenha os dados necessários para a resposta (ou seja, arquivo não encontrado no sistema de replicação), este poderá se conectar no nodo originalmente solicitado (B) e repassar a consulta. Caso o cliente solicitante (A) requisite a obtenção de um arquivo, o cliente virtual (V) o solicitará ao agente do sistema de replicação, que ficará encarregado de buscar este arquivo na rede.

Mensagem *MultiConnectToMe* possui tratamento similar ao da mensagem *ConnectToMe*, mas destinada a um cenário *inter-hubs*.

Mensagem *RevConnectToMe* é enviada por um cliente (A) que deseja solicitar que um outro cliente (B) lhe envie um pedido de conexão. Quando o lado *hub* do *plugin* recebe esta mensagem, este instancia um cliente virtual (V) que enviará a solicitação de

conexão. Após o estabelecimento da conexão, este cliente virtual (V) responderá às solicitações enviadas pelo cliente solicitante (A). Caso seja necessário, o cliente virtual (V) poderá se conectar ao nodo originalmente solicitado (B), de forma a responder às solicitações do cliente (A) originador da comunicação. No caso do cliente (A) originador solicitar uma transferência de arquivos, o cliente virtual (V) repassará a requisição ao sistema de replicação que ficará encarregado de buscar o arquivo na rede.

Mensagem *Search* é enviada para o *hub* de um cliente que deseja realizar uma busca por conteúdo na rede. O *hub* repassa esta mensagem para seus clientes. Dependendo dos parâmetros enviados, esta mensagem pode assumir o caráter ativo ou passivo. Na busca ativa, o cliente especifica um endereço de retorno para envio das respostas através de datagramas (UDP). Na busca passiva, as respostas são encaminhadas, através do *hub*, pela conexão já existente. De qualquer forma, o sistema de replicação, ao receber tal mensagem, verifica em seu repositório por resultados para a consulta. Caso sejam encontrados, estes são retornados de forma apropriada para o cliente solicitante. No entanto, caso não existam respostas replicadas, uma busca deve ser propagada na rede. Sendo assim, o *plugin* instancia um cliente virtual e o associa a um determinado endereço. Este cliente virtual fica responsável por receber as respostas. Uma mensagem de busca ativa é então propagada do agente do sistema de replicação *hub* para todos os clientes da rede, contendo como endereço de retorno o endereço previamente associado ao cliente virtual. Este cliente virtual recebe as respostas, realiza a replicação no agente do sistema de replicação e as encaminha para o cliente solicitante.

Mensagem *MultiSearch* possui tratamento similar à mensagem *Search*, mas é destinada a um cenário *inter-hubs*. Outras mensagens presentes no protocolo e não

mencionadas devem ser consideradas como inalteradas, ou, com alterações mínimas irrelevantes para a discussão aqui apresentada.

5.2.2 Estudo de Caso *eDonkey*

O protocolo *eDonkey*[9] é considerado um dos primeiros mecanismos especializados na transferência de recursos muito grandes. Sua estratégia é o particionamento destes arquivos em diversos agrupamentos de dados (*chunks*) que podem ser transferidos de forma independente e simultânea, acelerando o processo de obtenção do arquivo e eliminando a dependência de um único nodo específico, consagrando a eficiência do processo de *swarmed retrieval* na distribuição de conteúdo par-a-par [41]. Com cada agrupamento de dados tratado de forma independente pela rede, é possível que nodos que ainda não receberam todo o arquivo comecem a distribuir agrupamentos já armazenados, resultando no aumento da dinamicidade do processo de transferência, na medida em que um nodo não precisa mais aguardar por todo o conteúdo antes de iniciar sua distribuição. Após a recepção de todos os agrupamentos de dados, estes são finalizados em um único arquivo, idêntico ao original, o que pode ser checado através de seu valor *hash*. A rede *eDonkey* apresenta uma arquitetura par-a-par híbrida com sua rede composta das entidades cliente[10] e servidor. O **cliente *eDonkey*** permite o compartilhamento e obtenção de arquivos. O **servidor *eDonkey*** opera na forma de um

---

[9] *Observe que alguns detalhes do protocolo utilizado pelo* eDonkey *não foram encontrados na literatura, por se tratar de um protocolo fechado e proprietário.*

[10] *Na descrição do protocolo eDonkey são utilizados os termos 'nodo' e 'cliente' com o mesmo significado.*

indexador de localizações de arquivos e endereços distribuídos de outros servidores. A FIG. 16 apresenta a visão de uma rede *eDonkey* típica:

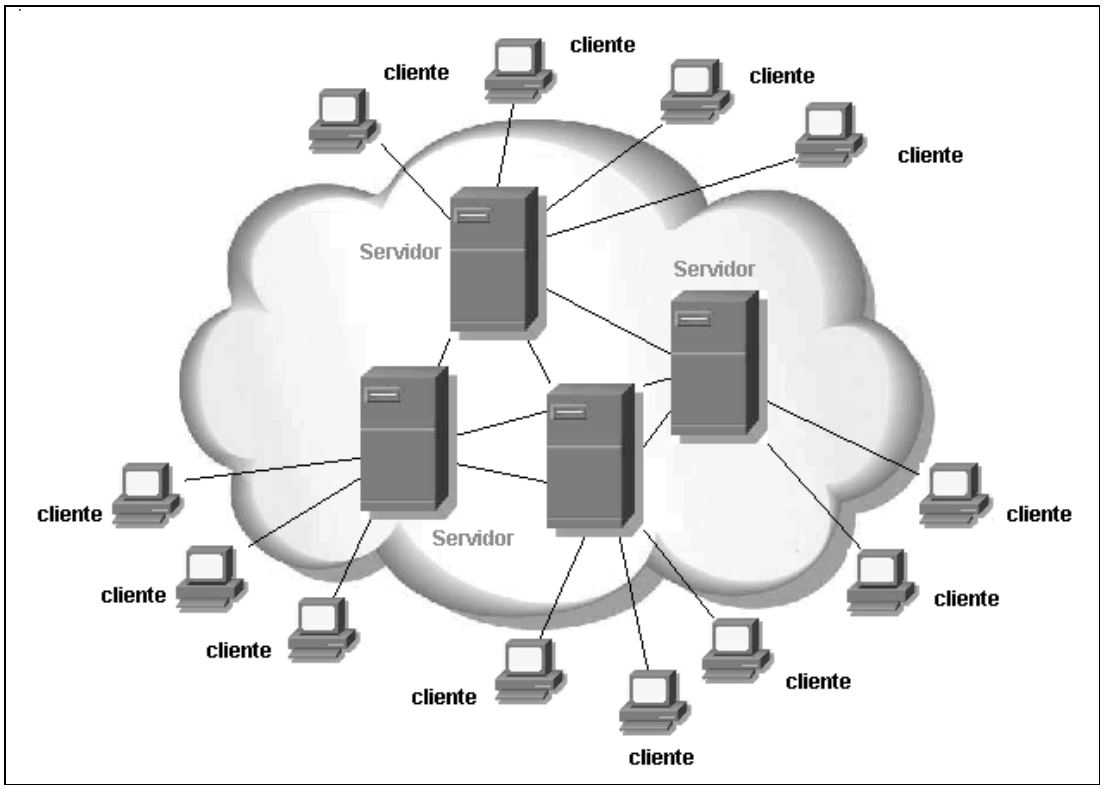

FIGURA 16 – Rede eDonkey

Embora o protocolo *eDonkey* não seja oficialmente documentado, *Lohoff* [17] apresenta seus detalhes obtidos através de engenharia reversa. Neste protocolo são contempladas comunicações servidor-servidor, servidor-cliente e cliente-cliente [6], onde, periodicamente, servidores e clientes trocam requisições de *ping* de forma a identificar a existência da outra parte. Para ingressar na rede, um cliente deve se conectar a algum servidor. Para cada ação, o protocolo exige a definição de canais específicos para o envio e recebimento de mensagens. Por convenção, servidores se comunicam através do porto TCP 4661 e porto UDP 4665, enquanto os clientes utilizam os portos TCP 4662 e UDP 4666. Neste protocolo, um nodo se anuncia ao servidor

através da abertura de uma conexão e envio de uma mensagem de *início de sessão*. Então, o servidor abre uma conexão em retorno ao cliente e verifica se este encontra-se apto a enviar e receber arquivos. Logo após, a conexão de retorno é fechada com o servidor, que realiza o cadastro do novo nodo e retorna (pelo canal originalmente aberto pelo nodo) uma mensagem de *lista de arquivos conhecidos*. De forma similar, o cliente informa ao servidor de sua *lista de arquivos compartilhados*. Após a conexão, o nodo passa a ser identificado apenas pelo seu endereço IP e o nome do usuário conectado. A partir deste ponto, para obter um arquivo o nodo deve localizar trechos deste presentes em outros nodos através do nome do arquivo, tamanho e chave identificadora (*hash MD4* do conteúdo do arquivo registrado no servidor).

Para localizar arquivos, o cliente deve submeter uma requisição de busca para o servidor que retorna uma lista de nodos que potencialmente podem serví-lo. Caso esta consulta não retorne resultados suficientes, o cliente pode se conectar em outro servidor e submeter novamente a consulta.

Para obter um arquivo, o cliente deve transmitir uma mensagem especificando o identificador do arquivo solicitado. O servidor responde com uma mensagem indicando quais clientes possuem o arquivo solicitado. O cliente realiza uma conexão TCP/IP com os nodos retornados e verifica quais trechos cada nodo possui. Posteriormente, o cliente contata um nodo de interesse solicitando um canal de comunicação (*slot*) para a obtenção do trecho do arquivo. Ao receber esta solicitação, o cliente remoto realiza seu enfileiramento em uma lista de atendimento regida por um complexo mecanismo de pontuação (*score*) que decide qual é a próxima requisição a ser servida [18]. Assim que o nodo remoto decide por servir a requisição, uma confirmação será enviada ao nodo solicitante. Após esta confirmação, o cliente solicitante já encontra-se pronto para

receber o trecho solicitado. Novos trechos podem ser solicitados, desde que o nodo solicitante envie uma mensagem de *requisição de próximo trecho*. Para finalizar o *download*, o nodo solicitante deve enviar uma mensagem de *fim de transferência* e fechar a conexão com o nodo remoto.

Cada arquivo é armazenado nos clientes em agrupamentos de dados denominados *chunks* compostos por 9.500 KBytes, com exceção do *chunk* final, que pode conter um valor inferior. Quando um *chunk* é compartilhado, sua transmissão se dá através de blocos de 180 KBytes, conforme a FIG. 17.

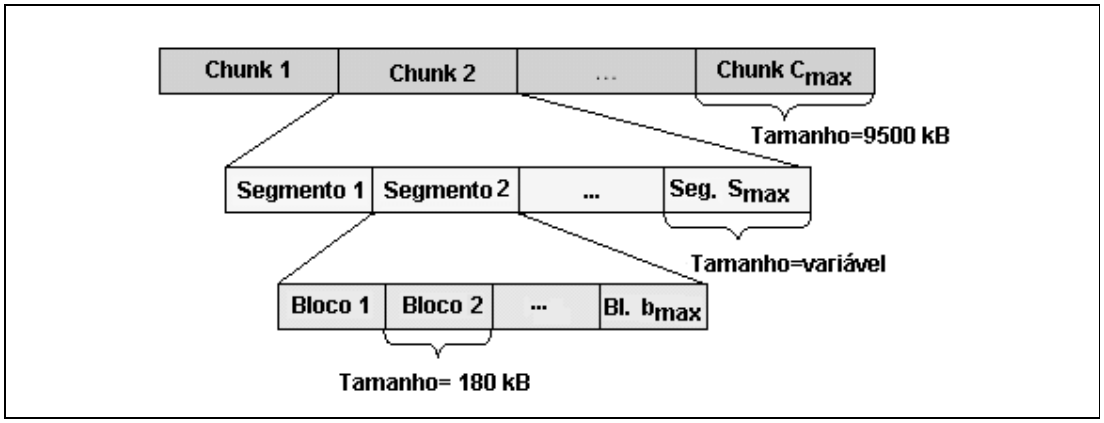

FIGURA 17 – Armazenamento de arquivos no eDonkey

De forma a preservar a corretude dos dados transferidos, *eDonkey* apresenta mecanismos internos de detecção e correção de erros. Em sua versão inicial, estes mecanismos operam na unidade de *chunks*. Com a evolução da rede, novos mecanismos de detecção e correção de erros foram incorporados para tratamento de unidades de dados menores, denominadas de segmentos, onde o tamanho de cada segmento depende do mecanismo de correção empregado na transmissão [11].

5.2.2.1 Plugin eDonkey

O projeto do *plugin eDonkey* envolve três módulos: módulo de tratamento de consultas; módulo de obtenção de arquivos da rede; módulo de envio de arquivos para a rede. A FIG. 18 apresenta esta organização:

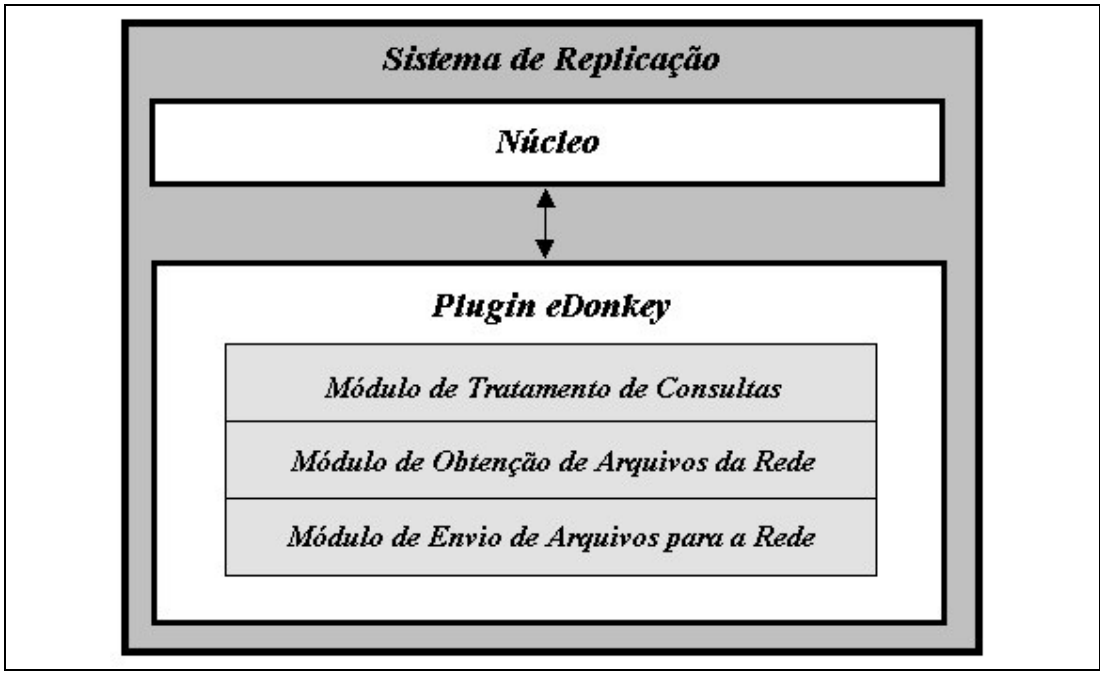

FIGURA 18 – Plugin eDonkey

O **módulo de tratamento de consultas** atua de forma similar a um servidor *eDonkey*, tendo como objetivos receber requisições de consultas e enviar respostas para estas requisições. Como um servidor *eDonkey*, este módulo deve manter internamente uma tabela de índices de arquivos por cliente, bem como a lista dos clientes conectados. O protocolo *eDonkey* especifica diversas variantes para requisições de consultas, entretanto, como simplificação, a implementação deste protótipo trata apenas um tipo de consulta baseada no nome do arquivo (trafegada através de pacotes UDP). Ao receber

uma consulta, o *plugin* realiza seu repasse ao núcleo do sistema de replicação. Caso o núcleo tenha um resultado para esta consulta, ele é propagado para o *plugin* que realiza seu repasse ao nodo solicitante, associando como detentor do arquivo um endereço correspondente ao módulo de envio de arquivos[11]. No caso do núcleo do sistema de replicação não possuir resultados para a consulta, uma solicitação de busca é propagada do núcleo para os *plugins*. Ao receber esta solicitação, o *plugin eDonkey* realiza uma pesquisa na sua tabela de índice de arquivos, retornando o resultado para o núcleo. O núcleo realiza o armazenamento da resposta da consulta no repositório de replicação e a retorna para o *plugin eDonkey*. O *plugin eDonkey*, por sua vez, propaga a resposta ao nodo solicitante, atribuindo o nodo especial como detentor do arquivo.

O **módulo de obtenção de arquivos da rede** opera como um cliente *eDonkey*, tendo a tarefa de realizar conexões com nodos e solicitar a transferência de arquivos. Este módulo é invocado sempre que o *plugin eDonkey* recebe uma solicitação do núcleo do sistema de replicação para obter um arquivo na rede. O processo de obtenção de arquivos é custoso devido às peculiaridades do *eDonkey* associadas à possibilidade do arquivo estar distribuído entre diversos nodos. Após a obtenção do arquivo, este é transferido para o núcleo que realiza sua replicação.

O **módulo de envio de arquivos para a rede** também opera como um cliente *eDonkey* na função de entregar conteúdo para nodos solicitantes. Sempre que resultados de uma busca são enviados para a rede, o *plugin eDonkey* associa o módulo de envio de arquivos para a rede como detentor do arquivo. Este módulo atua como um nodo típico permitindo que clientes *eDonkey* se conectem a ele e recebam o arquivo solicitado.

---

[11] *Módulo de envio de arquivos - responsável por realizar a entrega efetiva do conteúdo solicitado.*

5.3 Resultados

Nesta seção são apresentados os experimentos realizados com o sistema de replicação desenvolvido. Para estes experimentos foram desenvolvidos clientes geradores de carga, cuja tarefa consiste em obter a seqüência de arquivos que devem ser solicitados ao sistema de replicação (através do *log* cedido por um provedor) e realizar sua solicitação. Foi desenvolvido um cliente para cada *plugin* (*DirectConnect* e *eDonkey*), conforme demonstra a FIG. 19.

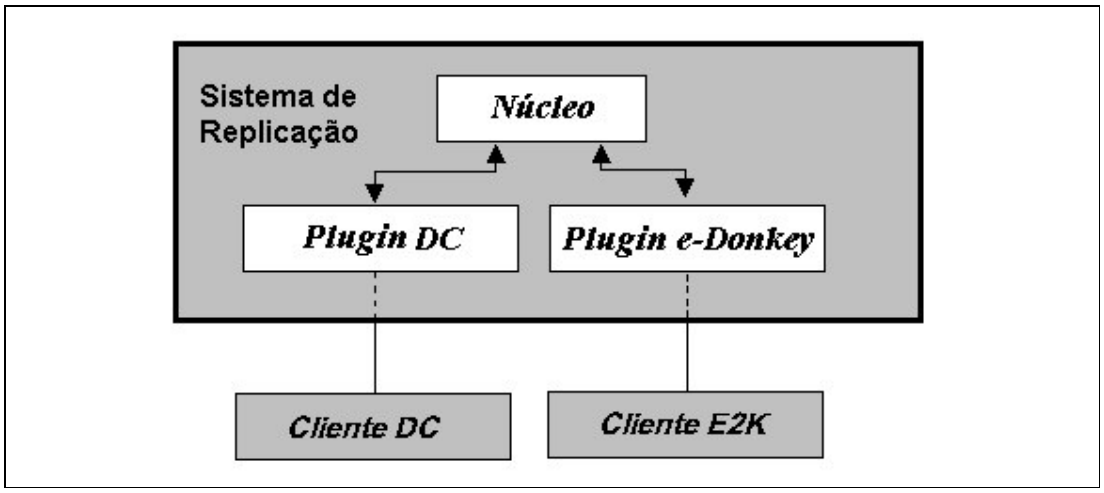

FIGURA 19 – Arquitetura para os experimentos

O arquivo de *log* é composto por registros separados pelo caractere de quebra de linha. Cada registro contém o nome do arquivo a ser solicitado, tamanho do arquivo e intervalo em segundos em relação à requisição imediatamente anterior, sendo estes campos separados pelo caractere de tabulação. O arquivo é composto de 2.665 requisições com um total de 543 arquivos distintos referenciados. A soma do tamanho de todos os arquivos distintos presentes na amostra eqüivale a 167 GBytes, enquanto

que a soma de todos os bytes solicitados chega a 1.277 GBytes. A TAB. 3 sumariza as características da amostra de dados utilizada.

TABELA 3 – Caracterização da amostra

Atributo	**Valor**
Número de requisições	2.665
Número de arquivos distintos presentes na amostra	543
Quantidade de bytes solicitados	1.277 GBytes
Tamanho de todos os arquivos	167 GBytes

O GRAF. 4 apresenta a curva de popularidade dos arquivos presentes na amostra.

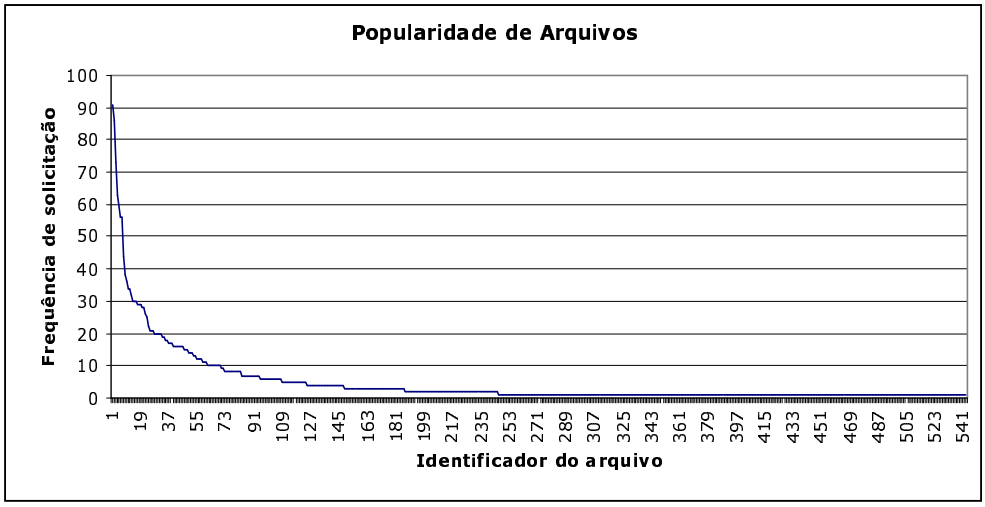

GRÁFICO 4 – Freqüência de solicitação de arquivos da amostra

O arquivo mais popular foi solicitado por 91 vezes, sendo que os 40 arquivos mais populares (7,3% de todos os arquivos) são responsáveis por 50% das requisições.

No entanto, a cauda da curva não pode ser desprezada, já que os outros 504 arquivos (92,7% dos arquivos) também são responsáveis por 50% das requisições, indicando que a popularidade segue uma distribuição severa, com alguns itens muito procurados e uma cauda extensa e importante. O GRAF. 5 apresenta a distribuição dos tamanhos de arquivos presentes na amostra.

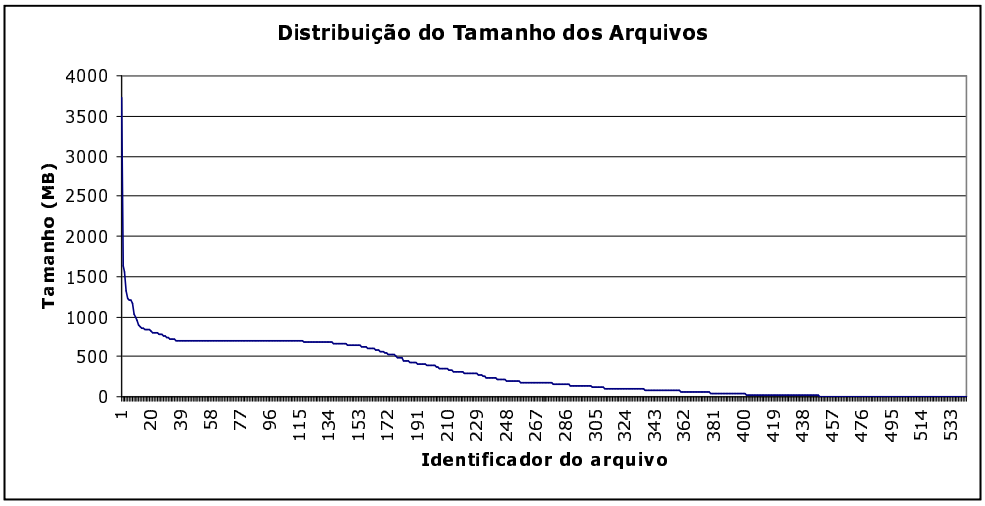

GRÁFICO 5 – Distribuição do tamanho dos arquivos da amostra

Pelo GRAF. 5 percebe-se que existe grande variação do tamanho dos arquivos, com o menor possuindo 283 bytes (provavelmente um arquivo de índice no formato texto) e o maior 3,7 GBytes, conforme a TAB. 4 demonstra:

TABELA 4 – Tamanho dos arquivos da amostra

Atributo	**Valor**
Tamanho médio	315 Mbytes
Tamanho mínimo	283 bytes
Tamanho máximo	3,7 GBytes
Desvio padrão	347,0 Mbytes

De forma a avaliar os ganhos obtidos com a utilização do sistema de replicação, são realizados experimentos avaliando a *taxa de acertos de arquivos* e a *taxa de acertos de* bytes. Estes experimentos variam o espaço de armazenamento, disponibilizado ao sistema de replicação, com limiar inferior de 5 GBytes e superior de 167 GBytes. O limiar inferior foi escolhido de forma a comportar o maior arquivo presente na rede (3,7 GBytes), enquanto o superior foi definido em função do tamanho total de todos os arquivos solicitados (167 GBytes). Utilizou-se como unidade de incremento o valor de 5 GBytes, pelo fato desta unidade oferecer precisão suficiente para análise dos dados.

A taxa de acertos é uma métrica tradicional para avaliação de desempenho de sistemas de replicação. Embora a técnica de replicação já seja conhecida e utilizada em diversos ambientes (Web, sistemas operacionais, bancos de dados e outros), os efeitos de sua utilização no ambiente par-a-par ainda não são bem conhecidos. O ponto de partida deste experimento é o cálculo da máxima taxa de acertos possível para a amostra de dados utilizada. São 2.665 requisições para 543 arquivos distintos. Como o sistema

de replicação não apresenta nenhum arquivo armazenado em seu estado inicial, a maior taxa que pode ser atingida é 79,6%.

O GRAF. 6 demonstra a curva de variação da taxa de acertos, conforme o tamanho designado para a área de armazenamento do sistema de replicação. O maior valor de tamanho do repositório de replicação (eixo x) corresponde a 167 GBytes, oferecendo resposta a 2.122 requisições, que é o maior valor de requisições que podem ser atendidas.

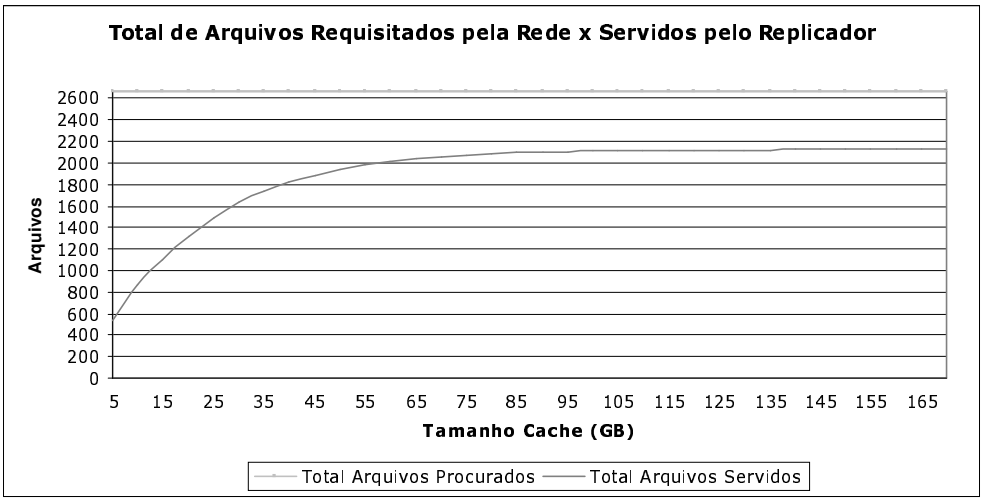

GRÁFICO 6 – Arquivos requisitados x arquivos servidos pelo replicador

A curva representativa da taxa de acertos cresce rapidamente, convergindo para o valor máximo possível. Com a utilização de um repositório de dados de tamanho referente a 50% do total de arquivos distintos[12] manipulados pela rede, obtém-se uma economia do canal de comunicação de 79%. Os percentuais de tamanho do repositório de replicação apresentados nos gráficos e tabelas são calculados com base neste segundo valor, que representa apenas 13% de todos os bytes requisitados.

Como em uma rede de computadores trafegam bytes ao invés de arquivos, considera-se interessante apresentar a taxa de acertos de bytes que pode ser obtida através do sistema de replicação. O GRAF. 7 apresenta este mesmo experimento sob a perspectiva de bytes economizados.

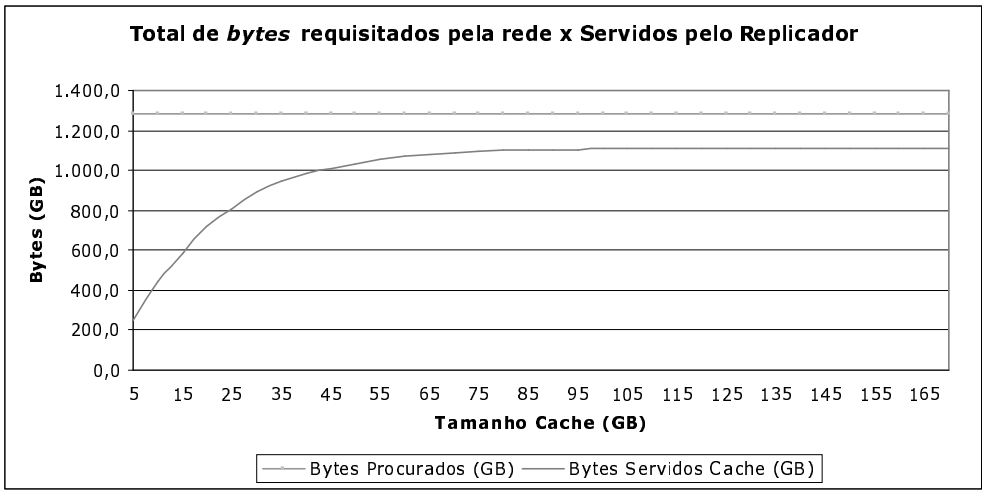

GRÁFICO 7 – Bytes requisitados x bytes servidos pelo replicador

Um arquivo é apenas uma abstração realizada sobre um conjunto de bytes organizados de alguma forma. Como arquivos não possuem o mesmo tamanho, embora a taxa de acertos de arquivos seja um indicador de economia de tráfego, esta não deve ser considerada como grandeza definitiva para a análise dos ganhos de banda de rede. Neste caso, para 1.277 GBytes de dados solicitados, 1.111 GBytes são atendidos através do sistema de replicação. Neste cenário, é possível obter uma economia de 87% na transferência de bytes. Em valores absolutos, isto significa que empregando-se um

---

[12] *É oportuno ressaltar a diferença conceitual que existe entre a* soma do tamanho dos arquivos presentes nas requisições *(1.277* GBytes*) da* soma de todos os arquivos **distintos** presentes nas requisições *(165* GBytes*).*

sistema de replicação com 80 GBytes é possível obter uma redução de 1.111 GBytes no tráfego de rede.

5.4 Sumário

Neste trabalho são avaliados os ganhos de otimização de transferências de conteúdo par-a-par que podem ser obtidos através do posicionamento do sistema de replicação na fronteira da rede corporativa com a Internet. Nesta seção foi apresentado o sistema computacional desenvolvido para atuar como replicador de conteúdo par-a-par, bem como os experimentos realizados para comprovação de sua eficácia na redução da utilização de banda de rede. De forma geral, o sistema de replicação deve ser percebido pela rede como um nodo típico, destacando-se apenas por sua capacidade de oferecer conteúdo e alta disponibilidade, tendo como características a operação com redes par-a-par distintas, suporte à requisição de intervalos e facilidade e praticidade para inclusão do suporte para novas redes.

Sugere-se a modelagem do sistema de replicação em duas camadas: a primeira camada trata da implementação de um nodo genérico de uma rede par-a-par. Já a segunda camada trata da interoperação do sistema com as diversas redes par-a-par alvo da replicação. Esta modelagem é interessante por ter o efeito prático de criar um sistema par-a-par genérico de compartilhamento de arquivos.

Para a elaboração dos estudos de casos e obtenção de resultados experimentais, optou-se pela implementação dos *plugins* referentes às redes *DirectConnect* e *eDonkey*. Neste estudo realizado com dados reais, com a disponibilização de um repositório de replicação correspondente a 50% do tamanho de todos os arquivos distintos, conseguiu-

se uma taxa de 87% de economia de bytes, um bom resultado quando levado em conta os custos de armazenamento em comparação com os custos de uma ligação com a rede mundial de computadores concluindo-se que a utilização de um sistema de replicação em um provedor de acesso à Internet ou qualquer outra organização que permita a existência de tráfego par-a-par em sua rede pode oferecer relevantes ganhos de economia de banda de rede.

CAPÍTULO 6

CONCLUSÃO

A principal contribuição da tecnologia par-a-par no campo da ciência da computação é a mudança de paradigma que ela trouxe em relação aos modelos existentes de processamento distribuído e compartilhamento de recursos. Pode-se afirmar, também, que o grande catalisador desta mudança foi a popularização da Internet. Os principais conceitos e tecnologias que hoje constituem as redes par-a-par são antigos, mas só puderam ser colocados em prática, de forma efetiva, com o surgimento de uma rede de dados aberta e de baixo custo.

Com a utilização maciça desta tecnologia, surge a necessidade de mecanismos que possibilitem ganhos de desempenho, escalabilidade e economia de banda de rede. Com esta motivação, foi desenvolvido este trabalho que avaliou o potencial de um

sistema de replicação para o ambiente par-a-par. Através de diversos experimentos foi possível demonstrar sua viabilidade.

Considera-se como principal contribuição deste trabalho a demonstração da viabilidade prática e os ganhos potenciais de uma arquitetura de replicação para o ambiente par-a-par, juntamente com o desenvolvimento de uma metodologia de projeto de sistemas de replicação para este ambiente.

6.1 Trabalhos Futuros

Sugere-se como trabalhos futuros a implantação desta solução em um provedor de acesso à Internet, como forma de avaliar a robustez e os ganhos de desempenho em um ambiente de operação real. A busca por políticas mais adequadas para a reposição de páginas do sistema de replicação também deve ser avaliada. Atualmente, o núcleo utiliza a política LRU. Outra sugestão é verificar o comportamento de um sistema de replicação par-a-par que consiga servir trechos de arquivos não armazenados completamente (*swarmed retrieval*). Por fim, considera-se interessante verificar, também, novas políticas de reposição de páginas para este cenário.

REFERÊNCIAS

[1] APACHE SOFTWARE FOUNDATION.
Disponível em: <http://www.apache.org>
Acesso em: 22 fev 2002.

[2] AURORA DOCUMENTATION.
Disponível em: <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/FreeNet/aurora/>
Acesso em: 05 fev 2002.

[3] BARISH, OBRACZKA. A Survey of World Wide Web Caching. Information Science Institute, Marina Del Rey, USA, 2000

[4] BUSINESSES EMBRACE INSTANT MESSAGING.
Disponível em: <http://enterprise.cnet.com/enterprise/0-9534-7-4403317.html>.
Acesso em: 03 fev. 2002.

[5] COHEN, LI, SHENKER. Search and replication in unstructured peer-to-peer networks. In Proceedings of the 16th annual ACM International Conference on Supercomputing, New York, NY, 2002

[6] EDONKEY DOCUMENTATION.
Disponivel em: <http://www.edonkey2000.com/documentation/overview.html>.
Acesso em: 05 fev 2002.

[7] EMULE PROJECT TEAM WEB SITE.
Disponível em: <http://www.emule-project.net/>
Acesso em: 22 jun. 2003

[8] FARSITE. Disponível em: <http://research.microsoft.com/sn/Farsite>.
Acesso em: 01 mar. 2003

[9] FREENET. Disponível em: <http://FreeNet.sourceforge.net>.
Acesso em: 15 abr. 2003.

[10] Gnutella Web Site. Disponível em: <http://www.gnutella.com>.
Acesso em: 03 abr. 2003.

[11] HOßFELD, LEIBNITZ, PRIES, TUTSCHKU, TRAN-GIA AND PAWLIKOWSKI. Information Diffusion in eDonkey Filesharing Networks, Report No. 341, 2004. Disponível em: <http://www3.informatik.uni-wuerzburg.de/TR/tr341.pdf>. Acesso em: 05 abr. 2003.

[12] HOßFELD, TUTSCHKU AND ANDERSEN. Mapping of File-Sharing onto Mobile Environments: Feasibility and Performance of eDonkey with GPRS, White Paper. Disponível em: <http:// www3.informatik.uni-wuerzburg.de/TR/tr338.pdf>. Acesso em: 05 nov. 2003.

[13] IYER, ROWSTRON AND DRUSCHE. Squirrel: A decentralized Peer-to-peer Web cache. Proceedings of the 21st Symposium on Principles of Distributed Computing (PODC), Monterey, CA, July 2002.

[14] TUTSCHKU, A measurement-based traffic profile of the eDonkey filesharing service. 5th Passive and Active Measurement Workshop (PAM2004), Antibes Juan-les-Pins, France, Apr. 2004.

[15] KAZAA WEB SITE. Disponível em: <http://www.kazaa.com>. Acesso em: 02 mai. 2003.

[16] KRISHNAMURTHY, ZHANG. P4P: Proxies for P2P systems. Unpublished white paper, 2001.

[17] LOHOFF, Lowlevel documentation of the eDonkey protocol, 2004. Disponível em: <http://silicon-verl.de/home/ o/software/donkey/>. Acesso em: 01 fev. 2005.

[18] MLDONKEY WEB SITE. Disponível em: <http://mldonkey.org/>. Acesso em: 01 fev. 2005.

[19] MORPHEUS WEB SITE. Disponível em: <http://www.morpheus-os.com/>. Acesso em: 02 jun. 2003.

[20] NAPSTER WEB SITE Disponível em: <http://www.napster.com>. Acesso em: 02 jun. 2003.

[21] P2P WORKING GROUP. Disponível em: <http://www.p2pwg.org, 2002>. Acesso em: 03 mar. 2003.

[22] PEER-TO-PEER: AN OVERVIEW, Computing Department, Lancaster University, UK, Unpublished white paper, 2001. Disponível em: <http://www.atc.gr/p2p_architect>. Acesso em: 25 mar. 2003.

[23] PUBLIC KEY CRYPTOGRAPHY FOR THE FINANCIAL SERVICES INDUSTRY: THE SECURE HASH ALGORITHM (SHA-1). American National Standards Institute

[24] ROWSTRON, DRUSCHEL. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms, 2001.

[25] SALTZER, REED AND CLARK. The end-to-end argument in Systems Design. Disponível em: <http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf>. Acesso em: 01 jul. 2003.

[26] SETI@HOME. Disponível em: <setiathome.ssl.berkeley.edu>. Acesso em: 01 abr. 2003.

[27] SRIPANIDKULCHAI. The popularity of Gnutella queries and its implications on scalability. Unpublished white paper, 2001. Disponível em: <http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html>. Acesso em: 15 fev. 2003.

[28] *STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In* Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, páginas 149–160. ACM Press.*.

[29] TRADITIONAL AND NON-TRADITIONAL APPLICATIONS; PEER-TO-PEER NETWORKS, 2001. Disponível em: <http://www.ida.liu.se/~TDTS43/tdts43-10-peer-to-peer.pdf>. Acesso em: 03 jan. 2003.

[30] WALKERDINE, MELVILLE, SOMMERVILLE. Dependability properties of P2P architectures. Computing Department, Lancaster University, UK, 2003. Disponível em: <http://www.atc.gr/p2p_architect>. Acesso em: 01 jun. 2003.

[31] WANG. A Survey of Web Caching Schemes for the Internet. Department of Computer Science, Cornell University, NY, 2001

[32] WHATISP2P... AND WHAT ISN'T. O'Reilly Network, 2003. Disponível em: <http://www.openp2p.com/lpt/a//p2p/2000/11/24/shirky1-whatisp2p.html>. Acesso em: 19 jun. 2003.

[33] ZIPF, POWER-LAWS, AND PARETO – A RANKING TUTORIAL. Xerox Palo Alto Research Center, 2003. Disponível em: <http://ginger.hpl.hp.com/shl/papers/ranking/ranking.html>. Acesso em: 30 mar. 2004.

[34] ANDROUTSELLIS-THEOTOKIS, S. AND SPINELLIS, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*.

[35] BALAKRISHNAN, H., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND *STOICA, I. (2003). Looking up data in P2P systems.* Communications of the ACM.

[36] BARISH, G. AND OBRACZKA, K. (2000). World wide web caching: Trends and techniques. In *IEEE Communications Magazine - Internet Technology Series*.

[37] *BENEVENUTO, F., JÚNIOR, J. I., AND ALMEIDA, J. (2005). Avaliação de mecanismos avançados de recuperação de conteúdo em sistemas P2P. In* Anais do 23º Simpósio Brasileiro de Redes de Computadores, SBRC2005.

[38] *CHANKHUNTHOD, A., DANZIG, P. B., NEERDAELS, C., SCHWARTZ, M. F., AND WORRELL, K. J. (1996). A hierarchical internet object cache. In* USENIX Annual Technical Conference*, páginas 153–164.*

[39] *CHAWATHE, Y., RATNASAMY, S., BRESLAU, L., LANHAM, N., AND SHENKER, S. (2003). Making gnutella-like P2P systems scalable. In* SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, páginas 407–418, New York, NY, USA. ACM Press.*

[40] CHU, J., LABONTE, K., AND LEVINE, B. N. (2002). Availability and locality measurements of peer-to-peer file systems.

[41] *TUTSCHKU, K. (2004). A measurement-based traffic profile of the edonkey filesharing service. In* 5th Passive and Active MeasurementWorkshop (PAM2004)*, volume 3015 of* Lecture Notes in Computer Science*, Antibes Juan-les-Pins, France. Springer.*

[42] *CORMEN, T. H., RIVEST, R. L., AND LEISERSON, C. E. (1989).* Introduction to Algorithms. *McGraw-Hill, Inc., New York, NY, USA.*

[43] *CÁCERES, R., DOUGLIS, F., FELDMANN, A., GLASS, G., AND RABINOVICH, M. (1998). Web proxy caching: the devil is in the details.* SIGMETRICS Performance Evaluation Review*, 26(3):11–15.*

[44] *GUMMADI, K. P., DUNN, R. J., SAROIU, S., GRIBBLE, S. D., LEVY, H. M., AND ZAHORJAN, J. (2003). Measurement, modeling, and analysis of a peer-to-peer filesharing workload. In* Proceedings of the nineteenth ACM symposium on Operating systems principles*, páginas 314–329. ACM Press.*

[45] *TANENBAUM, A. S. AND STEEN, M. V. (2001).* Distributed Systems: Principles and Paradigms. *Prentice Hall PTR, Upper Saddle River, NJ, USA.*

[46] *SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. (2002). A measurement study of peer-to-peer file sharing systems. In* Proceedings of Multimedia Computing and Networking 2002 (MMCN ’02)*, San Jose, CA, USA.*

[47] KARAGIANNIS, T., FALOUTSO, M., BROIDO, A., BROWNLEE, N., AND CLAFFY, K. C. (2004A). Is P2P dying or just hiding. In *Globecom 2004*.

[48] KLEMM, A., LINDEMANN, C., VERNON, M. K., AND WALDHORST, O. P. (2004). Characterizing the query behavior in peer-to-peer file sharing systems. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, páginas 55–67. ACM Press.

[49] *LEIBOWITZ, N., BERGMAN, A., BEN-SHAUL, R., AND SHAVIT, A. (2002). Are file swapping networks cacheable? Characterizing P2P traffic. In* 7th International Workshop onWeb Content Caching and Distribution.

[50] *RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SCHENKER, S. (2001). A scalable content-addressable network. In* Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, páginas 161–172. ACM Press.*

[51] LIANG, J., KUMAR, R., XI, Y., AND ROSS, K. W. (2005). Pollution in P2P file sharing systems. In *Proceedings of IEEE Infocom 2005*.

[52] LOO, B. T., HUEBSCH, R., STOICA, I., AND HELLERSTEIN, J. M. (2004). The case for a hybrid P2P search infrastructure. In *Third International Workshop Peer-to-Peer Systems Workshop (IPTPS’04)*, volume 3279 of *Lecture Notes in Computer Science*, páginas 141–150. Springer.