FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA - UNIFOR

Wladimir Maia Furtado

LOCALIZAÇÃO DE DOCUMENTOS EM REDES P2P

Fortaleza

2006

FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA - UNIFOR

Wladimir Maia Furtado

LOCALIZAÇÃO DE DOCUMENTOS EM REDES P2P

Dissertação apresentada ao curso de Mestrado em Informática Aplicada da Universidade de Fortaleza como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática Aplicada.

Orientador: Prof. Dr. Pedro Porfírio Muniz Farias

Fortaleza

2006

FURTADO, Wladimir Maia

Localização de documentos em redes P2P. Fortaleza. Universidade de Fortaleza (UNIFOR), Dissertação de Mestrado, 2006.

125 p.: il. 210 x 297 mm (MIA/UNIFOR, M.Sc. Ciência da Computação)

1. redes peer-to-peer (P2P)
2. Recuperação da informação (IR)
3. Sistemas distribuídos

I. MIA/UNIFOR II. Título CDU:

Agradecimentos

À Patrícia, minha esposa e companheira, por tudo (e algo mais), que sempre, durante o dia, a noite e a madrugada, esteve ao meu lado dando o suporte necessário e me livrando de certas tarefas para que eu pudesse me dedicar à dissertação e à pesquisa. Valeu "baby"!

À minha mãe e irmãs, pelo incentivo e confiança. À vovozinha, que mesmo sem entender o objetivo do trabalho, sempre se mostrou interessada e me encorajou a avançar, mas sem descuidar da saúde e das outras dimensões da vida.

Ao meu professor e orientador, Dr. Pedro Porfírio Muniz Farias, pela tolerância, tranquilidade, argumentação, ponderação e orientação na condução da pesquisa. E também por compartilhar comigo o acesso à ACM *Digital Library*, uma importante fonte de referências e pesquisas.

Ao colega Mario Gibson Maia Pinto, acadêmico do curso de Informática da Unifor, pela importante contribuição na construção do protótipo, nos testes e nas dicas com o Eclipse.

Ao colega de turma, Fernando Siqueira, pela ajuda na obtenção de artigos do IEEE. Aos colegas de turma Arturo e Pablo, pelo incentivo mútuo durante as madrugadas. Aos demais colegas pelo incentivo mútuo para a conclusão do trabalho.

Ao MIA, pela oportunidade de tornar a meta de ser mestre possível. Ao conselho de professores do MIA, por nos conceder mais tempo para a conclusão das nossas dissertações. Ao corpo de apoio do MIA pela ajuda nas questões administrativas.

À Etice, empresa da qual sou funcionário, por ter me ajudado a custear o curso de mestrado.

Ao Ministério Público Estadual, nas pessoas dos Secretários Gerais (2004-2006), pela liberação dos dias de trabalho nas ocasiões em que precisei me ausentar para participar de conferências e encontros acadêmicos.

Resumo da Dissertação apresentada ao MIA/UNIFOR como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática Aplicada.

LOCALIZAÇÃO DE DOCUMENTOS EM REDES P2P

Wladimir Maia Furtado

Dezembro / 2006

Orientador: Prof. Dr. Pedro Porfírio Muniz Farias
Programa: Informática Aplicada

Resumo

Sistemas P2P populares que compartilham arquivos na Internet permitem pesquisas sobre o nome desses arquivos. Tal método tem alcançado grande utilização para arquivos multimídia, já para documentos, esse método pode deixar de recuperar arquivos desejados pelo usuário, especialmente quando o nome do arquivo não contém a *string* pesquisada. No entanto, para atender a consulta por conteúdo, vasculhar os documentos do *peer* em tempo de consulta, implica em alto esforço computacional e maior tempo para a obtenção da resposta. Nessa dissertação, propomos uma arquitetura para ajudar no problema da localização de documentos textuais, pelo seu conteúdo, em redes P2P. Tal arquitetura propõe a adição da capacidade de indexação do conteúdo dos documentos da coleção do *peer*, como uma forma de realizar consultas em tempos aceitáveis.

Palavras-chave: peer-to-peer. Consulta textual. Recuperação de informação.

Abstract of the Dissertation presented to MIA/UNIFOR as a partial fulfillment of the requirements for the degree of Master of Applied Informatics.

LOCALIZATION OF DOCUMENTS IN P2P NETWORKS

Wladimir Maia Furtado

December / 2006

Adviser: Prof. Dr. Pedro Porfírio Muniz Farias
Program: Applied Informatics

Abstract

Popular P2P systems that share files on the Internet allow queries using the file names as main criterion. This kind of search has been utilized for multimidia files, but for documents, this method can not retrieve some files the user may need, specially when the name of the file does not match the string used in the search. Otherwise, the examination of the content of the peers´ documents, at the search time, implies in a high computational effort and higher processing to obtain the final answer. In this word, we propose an architecture to help in the problem of localization of textual documents, by its content, on P2P networks. Such architecture, adds the capacity of indexing the contents of the shared documents to the peers, as a way to support searches that consider the documents´ contents, in a reansonable time.

Keywords: peer-to-peer. Textual search. Information retrieval.

Sumário

Lista de Figuras

Capítulo 1 – Introdução

Apresentaremos neste capítulo as motivações para a realização da pesquisa, os objetivos que pretendemos alcançar, as contribuições e a organização geral da dissertação.

1.1 Motivação

A Internet tem crescido de forma importante, interconectando países, organizações e pessoas, permitindo a realização de pesquisas, a prestação de serviços, a comunicação em tempo real, o comércio eletrônico etc. Tais serviços e aplicações são possíveis graças a uma combinação de infra-estrutura de rede, protocolos de comunicação e sistemas computacionais devidamente integrados.

A infra-estrutura de rede, composta por meios de transmissão e equipamentos, é a via para o transporte de dados entre computadores. Sobre esta via, estações de usuário e servidores de rede compõem o aparato de *hardware* necessário ao funcionamento dos serviços e aplicações.

Os protocolos de comunicação disciplinam a comunicação entre computadores da rede. O TCP – *Transmission Control Protocol* e o IP – *Internet Protocol* emprestam seus nomes para batizar a pilha de protocolos da Internet (Tanenbaum, 2003).

Os sistemas computacionais são diversos, mas devemos citar os distribuídos e os centralizados. No topo da pilha de comunicação está a camada de aplicação, que permite a construção de soluções e a oferta de serviços. Por exemplo, sites de busca como o Google prestam o serviço de pesquisa no imenso conjunto de páginas eletrônicas, instituições financeiras interagem com seus clientes efetuando transações e pagamentos, lojas virtuais vendem itens e realizam o comércio eletrônico etc.

Várias aplicações na Internet seguem o modelo cliente-servidor, por exemplo a navegação em hipertexto. O modelo cliente/servidor é caracterizado pela existência de duas

porções de software: um cliente e um servidor. O software cliente interage e solicita para uma outra porção de software, o servidor, o qual fornece os serviços solicitados. Em um ambiente cliente-servidor de duas camadas, o *software* cliente executa em estações convencionais e o *software* servidor executa em computadores mais robustos e dedicados, os servidores. Um exemplo seria o acesso feito por uma estação a um servidor solicitando uma página HTML a um servidor Apache[1.] (Client-server, 2006).

Apesar da maioria dos protocolos da camada de aplicação da arquitetura Internet TCP/IP ser baseada no modelo cliente/servidor, existem outras aplicações que usam a Internet como meio de transporte ou ambiente de execução. Um exemplo disto são os sistemas P2P - *peer-to-peer,* também conhecidos por sistemas ou redes "ponto-a-ponto"[2]. Tais sistemas permitem que os computadores interligados, ou simplesmente *peers,* possam compartilhar e consumir recursos de outros computadores participantes dessa rede.

Em redes P2P, os *peers* conectam-se a outros *peers* e formam um sistema sobre os protocolos da arquitetura Internet. Cada *peer* pode ser detentor de conteúdo na Internet. As redes P2P distribuem informação entre os membros ao invés de concentrá-la em um servidor (Parameswaram et al., 2001). O paradigma das redes P2P foi inicialmente debatido para os serviços de compartilhamento de arquivos como o Napster e o Gnutella (Hamra; Felber, 2005). Esse tipo de aplicação alcançou grande sucesso entre usuários, onde havia a troca de arquivos de música, dando origem ao jargão "Content at the edges" (Shirky, 2001). Cada usuário escolhe o que quer compartilhar, um ou vários arquivos, e o limite está nos recursos de armazenamento do *peer*.

O imenso tamanho atingido pela Internet se deve a um conjunto de variáveis que permitiram a criação de conteúdo abundante e diversificado, entre elas: o barateamento de dispositivos de armazenamento e dos recursos de rede, a "liberdade" que pessoas e organizações têm para publicar seus *sites*, a ausência de autoridade e de controle sobre a redundância de conteúdo (Chakrabarti 2, 2003).

Entretanto, com a Internet tão grande, um problema surgiu: como encontrar o que se quer? Tornou-se difícil encontrar o conteúdo desejado de forma rápida e garantida.

Na navegação *web,* os mecanismos de busca têm conquistado a atenção dos usuários

---

1 http://www.apache.org/

2 Os *peers,* no contexto da ciência da computação, representam um grupo de computadores interligados que possuem interesses comuns, onde não há hierarquia formal definida entre eles.

porque os auxiliam a encontrar as páginas desejadas, sem cobrar "nada" por isto. Os servidores *web* armazenam as páginas eletrônicas dos sites. Os mecanismos de busca indexam tais páginas e assim podem responder de forma mais rápida às consultas feitas sobre o conteúdo das páginas. Os mecanismos de busca permitem encontrar páginas que contenham certa palavra-chave ou frase (Chakrabarti 2, 2003).

Além do conteúdo "visível" da Internet, que pode ser recuperado por mecanismos de busca, também existe um conjunto de páginas, documentos, apresentações, bancos de dados etc. que estão "invisíveis" (Invisible web, 2006). Levantamentos recentes[3], sugerem que o tamanho da *web* esteja na ordem de 167 terabytes na chamada *"surface web"*. Já a *"hidden web"*, os números chegam a 91.850 terabytes (Invisible web, 2006).

Um importante conjunto de objetos não é recuperado por mecanismos de busca porque barreiras técnicas impedem o acesso aos documentos/objetos e pela política adotada nos mecanismos de busca para excluir certos documentos/objetos (Invisible web, 2006). Por exemplo; caso um ramo de um *web site* só esteja disponível para usuários cadastrados, o mecanismo de busca não conseguirá recuperar os arquivos contidos nele; ou ainda, documentos em formatos como PDF, PPT, ODT, Flash podem não ser analisados por causa da política do mecanismo de busca, que pode ou não vasculhar o conteúdo destes tipos de arquivos.

Nos parece razoável considerar que no conteúdo "invisível" da Internet possa haver documentos de interesse para comunidades de usuários, haja visto os tipos de documentos desconsiderados pelos mecanismos de busca e a existência de várias redes P2P.

Neste contexto, as redes *peer-to-peer* agrupam computadores com interesses comuns enquanto os mecanismos de busca na *web* não têm esta finalidade. Os mecanismos de busca atuam, preferencialmente, sobre servidores *web.* Por outro lado, vários computadores de usuários se conectam na Internet e estes não são consultados pelos mecanismos de busca. Assim, considerar que nos discos rígidos dos *peers* das redes P2P conectadas à Internet há conteúdo de interesse para comunidades de usuários também nos parece razoável. Por exemplo; um grupo de professores que ministrem uma certa disciplina em escolas ou universidades poderão ter um conjunto de documentos, apresentações etc. que interessem aos demais membros desta comunidade.

---

[3] http://www.sims.berkeley.edu:8000/research/projects/how-much-info-2003/execsum.htm#int , estimou o tamanho da *web* em 2003.

Em redes P2P, arquivos relacionados à música e software são facilmente identificados por seus títulos e podem ser compartilhados por muitos *peers*. Apesar da busca textual ser suficiente quando o nome do arquivo ou trechos de sua descrição são amplamente conhecidos e caracterizam bem o seu conteúdo, ela não é apropriada quando o nome do arquivo é, por exemplo, "aula1.txt", o qual não indica a natureza de seu conteúdo. Como localizar os documentos de um *peer,* independente do nome do arquivo? Como encontrar documentos pelo o seu conteúdo e não apenas pelo nome (ou parte dele)?

1.2 Objetivos

O presente trabalho trata da localização de documentos pelo conteúdo em redes P2P. Para isto, propomos uma arquitetura que tem por objetivo facilitar a localização de documentos textuais nos diversos *peers* da rede.

Os objetivos específicos que pretendemos alcançar com essa proposta são:

- Identificar os requisitos funcionais suficientes para a realização da localização de documentos em redes P2P. Os requisitos funcionais envolvem a preparação da rede P2P, incluindo os *peers,* para suportar consultas que se referem ao conteúdo dos documentos textuais compartilhados.
- Propor uma arquitetura que permita realizar a localização de documentos, com base no conteúdo, em redes P2P.
- Elaborar um protótipo que atenda aos requisitos funcionais de forma satisfatória.
- Realizar experimentos que fundamentem uma avaliação inicial da abordagem que escolhemos.

1.3 Contribuições

A localização de documentos pelo conteúdo não é realizada por programas P2P populares. Programas P2P como o Kazaa (Kazaa, 2006), Emule (Emule, 2006) e LimeWire (LimeWire, 2006a) permitem a realização de consultas sobre os arquivos compartilhados dos *peers.* Ter alguma forma de consulta pelo conteúdo de documentos facilitaria o trabalho dos

usuários de encontrar arquivos textuais armazenados em *peers* de uma rede P2P.

As contribuições do nosso trabalho iniciam com a criação de uma arquitetura que permita a realização de consultas pelo conteúdo de documentos em redes P2P, a construção de um protótipo para a prova de conceito da arquitetura proposta e a realização de experimentos iniciais em uma rede P2P com inundação.

Para tanto, realizamos uma investigação sobre as redes P2P envolvendo principalmente os protocolos de comunicação, a organização da rede, os programas P2P para troca de arquivos, as facilidades de busca *(search)* e o potencial para a distribuição de conteúdo. Além disto, realizamos estudos sobre a abordagem semântica na localização de documentos e sobre os conceitos e técnicas de recuperação da informação, tipos de índices e métodos de indexação, estudos e testes com a API Lucene[4] e o software LimeWire[5].

1.4 Estrutura da dissertação

A presente dissertação está estruturada em seis capítulos, incluindo o capítulo inicial. Os demais capítulos foram descritos como a seguir.

O capítulo 2 descreve outros detalhes sobre as redes P2P que serão úteis para a apresentação da nossa proposta. Apresentaremos algumas características dos sistemas P2P, uma classificação das arquiteturas P2P, o protocolo Gnutella em maiores detalhes, o compartilhamento de arquivos e outras aplicações das redes P2P.

No capítulo 3, traremos alguns conceitos do processo de recuperação da informação (*Information Retrieval – IR),* de indexação, de métodos de consultas textuais, de recuperação de informação distribuída e distribuição de conteúdo em redes P2P.

O capítulo 4 traz a nossa proposta em si. O capítulo inicia descrevendo os requisitos funcionais necessários para a solução do problema que pesquisamos, apresentaremos uma visão geral da nossa arquitetura e mostraremos como ela atende aos requisitos identificados. Comentaremos sobre o LimeWire e a API Lucene como partes integrantes da nossa proposta e os possíveis cenários de utilização da arquitetura.

No capítulo 5 apresentaremos maiores detalhes da implementação da arquitetura

[4] Http://lucene.apache.org
[5] Http://www.limewire.com

proposta, a qual descreve o protótipo construído para a prova de conceito da arquitetura, a realização dos experimentos premilinares, as APIs utilizadas e algumas questões quanto à integração.

O capítulo 6 encerra o trabalho trazendo as conclusões preliminares, os resultados iniciais com base na implementação e nos experimentos realizados. O capítulo traz ainda algumas sugestões de pesquisa futuras.

Capítulo 2 – Redes P2P

Neste capítulo, detalharemos alguns aspectos dos sistemas P2P que ajudam a introduzir a nossa proposta. Apresentaremos um histórico dos sistemas P2P, uma classificação das suas arquiteturas, o protocolo Gnutella, o compartilhamento de arquivos e outras aplicações dessas redes.

2.1 Histórico

A Internet concebida na década de 1960, então determinada Arpanet, era um sistema P2P (Minar; Hedlund, 2001). O objetivo da Arpanet era compartilhar recursos computacionais pelos Estados Unidos. Os primeiros *hosts* da Arpanet (UCLA, SRI, ECSB e Universidade de Utah) foram sites independentes e com igual *status*. A Arpanet os conectava como *peers* e não como sistemas *master/slave* ou cliente/servidor (Minar; Hedlund, 2001).

Dois sistemas da Internet são considerados precursores dos atuais sistemas P2P: Usenet e DNS (Minar; Hedlund, 2001). Fundamentalmente, Usenet é um sistema que, usando controle centralizado, copia arquivos entre computadores. Usenet foi baseado no UUCP – *Unix to Unix Copy*, mecanismo daquele sistema operacional que permite a cópia de arquivos entre computadores distintos. Atualmente, Usenet é implementado como o protocolo NNTP – *Network News Transport Protocol*[6].

O outro sistema, o DNS – *Domain Name System,* permite resolver nomes de *host* em endereços IP e vice-versa (DNS Reverso). Nos primórdios da Internet, havia apenas um arquivo *hosts.txt*[7] que era periodicamente copiado entre os servidores (Minar; Hedlund, 2001). Hoje em dia, o DNS continua em plena operação mas precisou de ajustes no seu projeto para acompanhar o crescimento de escala. O DNS se tornou hierárquico. Ele atua como servidor para *hosts* e como cliente de outros DNSs, quando necessário. O DNS ganhou utilização forte

---

[6] RFC 997.

[7] Arquivos *hosts* são usados no processo de resolução de nomes do DNS. Neles, há uma associação entre endereço IP e nome de *host*, ou de outros recursos, conforme seja o tipo (Resource Record – RR).

em intranets e se tornou o principal mecanismo de resolução de nomes dos atuais sistemas operacionais de rede, como o Windows 2003 Server, Netware Novell e Linux.

Em maio de 1999, o Napster[8], sistema P2P que permitia o compartilhamento de arquivos de música na Internet, surge e experimenta um crescimento de escala mundial, indo de encontro aos interesses da indústria musical. No início de 2000, o Gnutella (Gnutella, 2005) surge trazendo consigo a característica totalmente distribuída, ou seja, sem a participação de servidores/*clusters*, como no Napster.

Em março de 2001, por decisão judicial, o Napster interrompeu o serviço, vindo a ofertá-lo com a condição de pagar direitos autorais pelas músicas compartilhadas[9]. Isso motivou a implementação do protocolo Gnutella, que trazia consigo a característica de ser totalmente distribuído e o anonimato para os usuários da rede P2P. Surgiram depois, vários outros "compartilhadores de arquivo" como o Bearshare[10], LimeWire, Kazaa (Kazaa, 2006), Morpheus[11], Emule (Emule, 2006) etc.

Na seção seguinte, apresentaremos alguns conceitos e outras características dos sistemas P2P.

2.2 Conceitos

Os sistemas P2P não possuem uma conceituação definitiva, mas algumas definições que expressam suas características (Rocha, 2005). Para Shirky (Shirky, 2000), P2P é uma classe de aplicações que tira vantagem dos recursos – armazenamento, ciclos de processador, conteúdo, presença humana – disponíveis na borda da Internet.

Em (Stephanos; Diomidis, 2004), foi proposta uma definição que abrange duas importantes características dos sistemas P2P: o compartilhamento de recursos computacionais e a habilidade de tratar (e operar) na instabilidade e variabilidade de conectividade; "sistemas P2P são sistemas distribuídos consistindo de nós interconectados capazes de se auto-organizar em topologias de rede com o propósito de compartilhar recursos tais como conteúdo, ciclos de processador, armazenamento e largura de banda, capazes de se adaptar a falhas e acomodar

---

8 Http://www.napster.com
9 http://en.wikipedia.org/wiki/Napster
10 Http://www.bearshare.com
11 Http://morpheus.com

populações transientes de nós enquanto mantém conectividade e desempenho aceitáveis, sem requerer a intermediação ou o suporte de uma autoridade (ou servidor) centralizado".

O grupo de trabalho sobre redes P2P da Intel define os sistemas P2P como o compartilhamento de recursos computacionais e serviços pela troca direta entre os sistemas (Peer-to-peer working group, 2001).

O software P2P é chamado *servents*, um acrónimo para "*server*" e "*client*", ou seja, o software reune funções de servidor e de cliente. Dispondo graficamente como em (Ross; Rubenstein, 2005), temos:

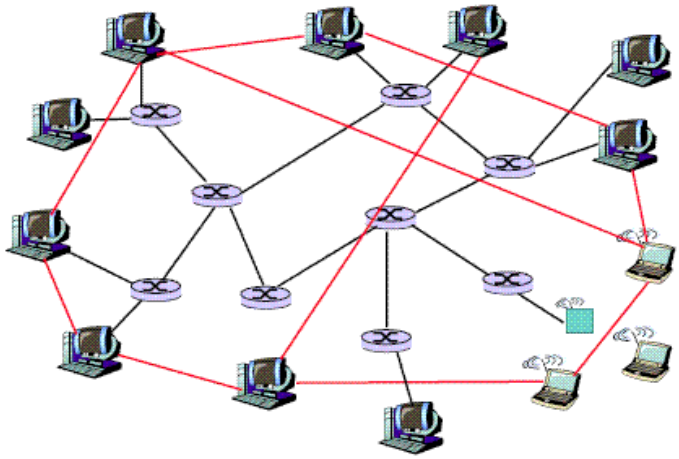

Figura 2.1 – Representação genérica de uma rede P2P (Ross; Rubenstein, 2005).

Observa-se pela figura 2.1 que existe um conjunto de enlaces que interliga os roteadores e computadores na Internet e que sobre eles os *peers* se organizam, formando uma topologia lógica de rede P2P diferente da topologia física. A figura 2.2 destaca a diferença de topologias lógica e física de uma rede P2P.

As arquiteturas de redes P2P geralmente especificam como se dá a organização dos *peers,* o anúncio e a localização de recursos e serviços, o roteamento das mensagens entre os *peers*, o transporte de mensagens etc., ou seja, o conjunto das funcionalidades que permitem o funcionamento de uma rede P2P.

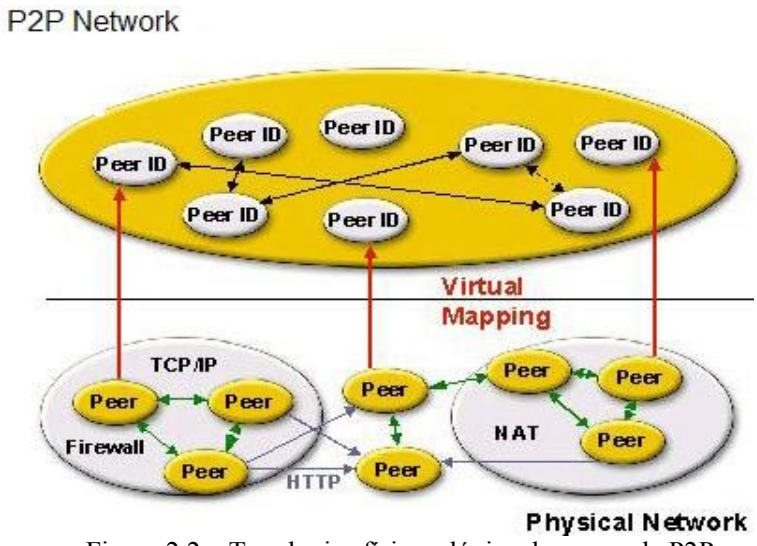

Figura 2.2 – Topologias física e lógica de uma rede P2P.
Fonte: http://www.sun.com/2002-0604/feature/diagram1.gif, com ajustes.

Como sugerido em (Vanwasi, 2001), para adquirir uma visão geral dos sistemas P2P, convém mencionar certos termos e conceitos básicos.

Rede P2P é a rede formada por nós-membros que encontram-se ativos (conectados). Em geral, tais nós-membros precisam estar conectados à Internet.

Nó-membro ou *peer* é o *host* ou o computador do usuário que pode atuar tanto como um servidor de recurso quanto cliente (consumidor) de recursos de outros nós-membros da rede P2P.

Identificador do *peer* é um número que identifica unicamente o *peer* dentro da rede P2P, independemente do endereçamento IP utilizado. A depender do protocolo P2P, esse identificador pode ter tamanhos diferentes. Por exemplo; no Gnutella, este identificador (GUID) possui 16 bytes.

Protocolo da rede P2P é o que permite o funcionamento da rede *peer-to-peer*, a troca de diálogos entre os nós-membros. O protocolo de rede P2P é responsável por incluir certo nó na rede, por rotear as mensagens entre os *peers* e por transportar os dados compartilhados. Uma outra característica é a capacidade de comunicação entre *peers,* mesmo que haja *firewall(s)* entre eles. A figura 2.3 (Ross; Rubenstein, 2005) detalha a sobrecarga de protocolos P2P sobre os protocolos da Internet.

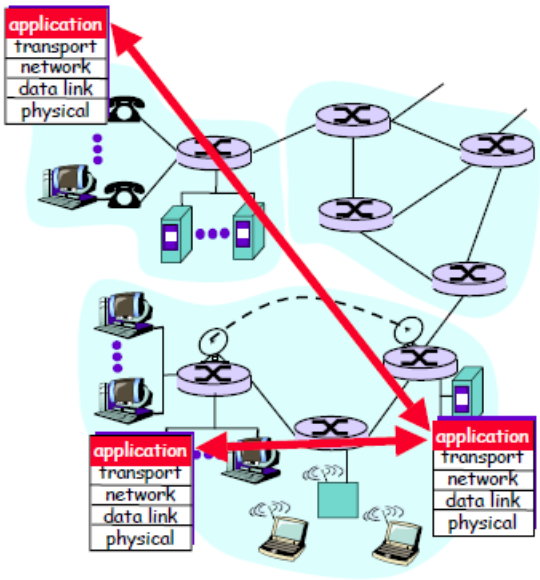

Figura 2.3 – Aplicação P2P sobre a Internet (Ross; Rubenstein, 2005).

Apesar de nos sistemas P2P não haver a necessidade de uma autoridade centralizada que controle a rede, os *peers* dependem entre si para que possam compartilhar e consumir os recursos computacionais e serviços. Na seção seguinte, comentaremos algumas características das redes P2P e questões relativas ao seu projeto.

2.3 Características dos sistemas P2P

Os sistemas P2P possuem características que às vezes os confundem com outros tipos de sistemas distribuídos. Nas subseções a seguir, comentaremos os aspectos relevantes dos sistemas P2P: descentralização, escalabilidade, anonimato, auto-organização, desempenho, conectividade *ad-hoc* e interoperabilidade.

2.3.1 Descentralização

O funcionamento de forma descentralizada é uma das principais características dos sistemas P2P. A total ausência de uma entidade centralizada dificulta o projeto dos sistemas P2P porque não existe um diretório central que informe quais os *peers* da rede e quais recursos compartilhados eles possuem (Milojicic et al., 2003).

Os *peers* são responsáveis pelo armazenamento e pela disponibilidade do conteúdo e dos serviços que eles possuem, diferentemente dos sistemas cliente/servidor tradicionais, onde os dados (ou o conteúdo) ficam armazenados em servidores e *clusters,* de forma centralizada.

Existem várias arquiteturas de redes P2P. As arquiteturas variam quanto ao nível de centralização (ou de descentralização). Maiores comentários sobre arquiteturas e sobre a forma de organização dos *peers* serão apresentados na seção seguinte.

2.3.2 Escalabilidade

O benefício imediato da descentralização é a melhoria da escalabilidade (Milojicic et al., 2003). A escalabilidade é limitada pelo poder de processamento/computação e pela capacidade de comunicação entre os nós.

Os sistemas P2P podem acomodar uma grande quantidade de *peers* e assim crescer em escala. Com mais fontes de dados ou conteúdo (*peers)*, os gargalos causados pela concorrência do acesso a um nó específico diminuem porque o acesso pode ser feito aos outros *peers*. Por outro lado, enlaces com pouca largura de banda criam gargalos e limitam o poder de crescimento de um sistema P2P. Outro fator que pode limitar o crescimento de uma rede P2P é a sua arquitetura.

2.3.3 Anonimato

O anonimato inerente dos sistemas P2P torna difícil o controle sobre aspectos legais e de direitos autorais dos arquivos compartilhados, especialmente nos arquivos multimídia. O anonimato, no contexto das redes P2P, possui vários aspectos, conforme (Dingledine et al., 2001): quem é o autor, quem publicou ou compartilhou, quem leu, quem realizou consulta sobre um certo conteúdo? São perguntas que podem ficar sem resposta em se tratando de redes P2P.

A identificação dos *peers* se torna difícil porque *hosts* podem receber endereços IP dinâmicos, reservados (frios)[12], ou ainda estarem "atrás" de um elemento que execute NAT – *Network Address Translation.* Além disso, com a troca de arquivos entre *peers* e a falta de

[12] Conforme a RFC 1918 – Endereçamento IP para Intranets, disponível em http://rfc.net/rfc1918.html .

registro das aplicações sobre a origem dos arquivos, a identificação de autores dos documentos também fica comprometida.

O anonimato também pode causar problema para os usuários das redes P2P. Uma vez que qualquer *peer* pode, inadvertidamente, publicar um arquivo com certo nome, atrativo para os demais usuários, mas que o seu conteúdo não seja o que o nome sugere; e assim outros *peers* ao solicitarem o *download*, podem receber o que não quiseram e além disto serem contaminados por vírus ou vermes digitais.

2.3.4 Auto-organização

A auto-organização, no contexto das redes P2P, diz respeito à capacidade que os *peers* têm de se adaptar às transformações sofridas pela rede, por exemplo, quando a rede recebe *peers* novos ou quando *peers* desconectam-se intermitentemente ou sem prévio aviso.

Os sistemas P2P podem escalar de forma imprevisível e isto aumenta a probabilidade de falhas que requeiram auto-manutenção e auto-reparo do sistema (Dingledine et al., 2001). Uma vez que não há, necessariamente, uma autoridade de controle nos sistemas P2P e nem pessoas para gerenciar a adaptação da rede, a capacidade de auto-organização de uma rede P2P precisa estar presente em cada *peer,* para assim garantir a continuidade da operação.

2.3.5 Desempenho

O desempenho é uma importante característica desejável nos sistemas P2P. Nestes sistemas, o desempenho pode ser melhorado através da adição de capacidade de armazenamento distribuído, de ciclos de processamento e de largura de banda (Dingledine et al., 2001).

Três abordagens podem otimizar o desempenho geral de uma rede P2P: replicação, *cache* e roteamento inteligente (Dingledine et al., 2001).

A replicação de objetos em outros *peers* da rede facilita a localização porque mais *peers* podem responder positivamente a uma consulta sobre um certo objeto. Além disso, a existência de réplicas de um objeto em mais *peers* permite que o objeto requisitado continue a

ser localizado na rede, mesmo quando alguns *peers* que o detêm saem da rede. Nas situações de compartilhamento de arquivos, quanto mais *peers* possuírem um certo objeto, maior será a chance de realização de *download* de várias fontes e assim diminuir o tempo de espera para o usuário.

O uso de *cache* reduz a quantidade de mensagens trocadas entre *peers* e objetiva minimizar a latência de acesso (Dingledine et al., 2001). A redução da quantidade de mensagens entre *peers* reduz eventuais gargalos de comunicação.

O roteamento "inteligente" deve levar em consideração as relações sociais dos *peers* e assim aproximar logicamente aqueles que possuem maior potencial de compartilhamento ou, efetivamente, apresentam maiores chances de interação. Dessa forma, o desempenho será melhorado porque o tempo de resposta será minimizado.

2.3.6 Conectividade *ad-hoc*

A natureza *ad-hoc* da conectividade tem forte efeito em todas as classes de sistemas P2P (Dingledine et al., 2001). Os sistemas P2P precisam conviver com o fato freqüente dos *peers* entrarem e saírem da rede, a qualquer momento e sem prévio aviso, diferentemente de outros sistemas em que isto acontece como uma exceção.

A rede P2P deve ter políticas que estimulem os *peers* a permanecerem conectados o maior tempo possível. O conteúdo ou os serviços dos *peers* podem deixar de existir, caso os seus provedores se ausentem da rede. Assim, a rede precisa estar preparada para tratar estes eventos e isto se reflete nos principais eventos e mensagens do seu protocolo.

2.3.7 Interoperabilidade

Apesar da existência de vários sistemas P2P, ainda não existe um padrão que permita a interoperabilidade entre *peers* de redes P2P diferentes. Há, por outro lado, software que permite a entrada em uma ou mais redes P2P, como é o caso do Emule e do Kazaa.

Em (Dingledine et al., 2001), são definidos alguns requisitos para a interoperabilidade entre sistemas P2P:

- Como os sistemas devem se comunicar, ou seja, qual protocolo deve ser usado: TCP ou UDP;
- Como os sistemas trocam requisições e respostas, como executam tarefas de alto nível como a troca de arquivos ou a busca por um objeto;
- Como os sistemas podem determinar a compatibilidade com outro sistema;
- Como um sistema efetua anúncios;
- Como um sistema mantém (ou garante) o mesmo nível de segurança de outro sistema.

Buscar a interoperabilidade entre redes P2P seria interessante pelo ponto de vista da população de *peers* que, a priori, poderia migrar de uma rede para outra ou ainda compartilhar recursos com *peers* de outra rede. Na ausência de um protocolo genérico para redes P2P, uma alternativa para a interoperabilidade seria o uso de *gateways* para outras redes P2P. Os *gateways*, neste contexto, seriam *peers* que traduziriam as mensagens de uma rede para outra e assim viabilizariam a interação entre *peers* de redes diferentes.

As características e questões relacionadas aos sistemas P2P estarão presentes, em maior ou menor proporção, nas arquiteturas de sistemas P2P comentadas na seção seguinte.

2.4 Arquiteturas P2P

Nesta seção comentaremos sobre a classificação das arquiteturas P2P quanto ao nível de descentralização e quanto ao nível de estruturação da rede. Estes dois parâmetros de classificação são úteis para analisar os diferentes aspectos dos sistemas.

A exemplo do que acontece com a conceituação das redes P2P, também não há consenso geral quanto à taxonomia das arquiteturas P2P. Foram propostas por (Figueiredo et al., 2003), (Schollmeier, 2002) e (Stephanos; Diomidis, 2004) taxonomias distintas para classificar as arquiteturas de sistemas P2P. Entretanto, tais classificações apresentam semelhanças.

A primeira nomenclatura, proposta por (Schollmeier, 2002), classifica as arquiteturas P2P quanto ao nível de descentralização como:

- Arquitetura P2P Pura: totalmente distribuída e não necessita de um elemento central para o seu funcionamento. O Gnutella (Gnutella, 2005) foi o primeiro grande exemplo de uma rede P2P pura.
- Arquitetura P2P Híbrida: não é inteiramente distribuída e existe a necessidade de um elemento central para garantir o seu funcionamento. O maior exemplo foi o Napster[13] inicial, onde há a presença do servidor (ou *cluster* de servidores*)*.

A taxonomia proposta por (Figueiredo et al., 2003), também quanto ao nível de descentralização, classifica em: CIA *(Centralized Indexing Architecture),* DIFA *(Distributed Indexing with Flooding Architecture)* e DIHA (*Distributed Indexing with Hashing Architecture),* comentadas a seguir.

- CIA: existe a presença de um servidor central (ou *cluster* de servidores*)* que tem o papel de responder as consultas feitas pelos *peers* e realizar tarefas de manutenção da rede, como, por exemplo, registrar um novo *peer*. Entretanto, as trocas de objetos entre *peers* são feitas sem a participação direta do servidor, mas apenas entre os *peers* envolvidos.
- DIFA: não existe a presença de um servidor central e a sua operação é totalmente distribuída. As tarefas de manutenção da rede e de consultas são realizadas pelos próprios *peers* através de inundação *(flooding).* Assim, cada *peer* possui uma lista e um índice dos objetos compartilhados.
- DIHA: também não existe a presença de um servidor central e a operação da rede também é totalmente distribuída. A diferença para a DIFA está na forma como são feitas as consultas. Em DIHA, cada *peer* possui um subconjunto dos índices de todos os índices dos demais *peers.* As consultas não são realizadas por inundação, mas com base no grupo de índices locais, sendo então direcionadas para o *peer* correto.

Já em (Stephanos; Diomidis, 2004), a classificação é: arquiteturas puramente descentralizadas, arquiteturas parcialmente centralizadas e arquiteturas descentralizadas híbridas, comentadas a seguir.

- Arquiteturas puramente descentralizadas: todos os nós da rede realizam as mesmas tarefas, agindo tanto como clientes e como servidores *(servents).* Não há uma

[13] Http://www.napster.com

coordenação central das atividades dos *peers.*

- Arquiteturas parcialmente centralizadas: semelhante à arquitetura puramente descentralizada, mas há alguns *peers* que desempenham um papel importante como provedores de índices dos arquivos compartilhados por *peers* próximos. Tais *peers* são chamados *supernodes* ou *ultrapeers* e são designados dinamicamente pela rede.
- Arquiteturas descentralizadas híbridas: há a presença de um servidor central que facilita a interação entre os *peers*, provendo o diretório de metadados e descrevendo os arquivos compartilhados por todos os *peers.* As trocas de objetos fim a fim são realizadas diretamente pelos *peers* envolvidos.

Observamos semelhanças nas classificações P2P pura, DIFA e DIHA; já a híbrida apresenta semelhanças com a CIA, por exemplo.

O outro aspecto relevante das arquiteturas P2P diz respeito ao nível de estruturação da rede P2P, ou seja, como é formada e mantida a rede, seguindo ou não regras específicas. Em (Stephanos; Diomidis, 2004), é sugerida a classificação das redes P2P quanto ao nível de estruturação como não-estruturadas e estruturadas, comentadas a seguir.

- Redes P2P não-estruturadas: neste tipo, a distribuição de arquivos e conteúdo é feita de forma não relacionada com a topologia, ou seja, um novo *peer* pode entrar na rede bastando para isso anunciar-se perante um outro *peer* ou servidor. Portanto, é fácil criar e manter a operação da rede. Por outro lado, os métodos de busca são custosos porque promovem inundação com a propagação das consultas e assim são mais apropriadas para comunidades transitórias de *peers.* São exemplos: Napster, Gnutella (Gnutella, 2005), Kazaa (Kazaa, 2006) e Edutella (Edutella, 2006).
- Redes P2P estruturadas: tentam resolver limitações na escalabilidade das redes P2P não-estruturadas. Nas redes P2P estruturadas há forte controle sobre a topologia e os objetos compartilhados são armazenados em locais específicos, através do mapeamento entre conteúdo e localização. Assim, as consultas podem ser rapidamente roteadas para um *peer* que detém certo conteúdo. Há portanto, grande potencial de crescimento em escala para consultas onde se conhece o valor de um identificador do objeto desejado. Por outro lado, uma forte desvantagens das redes P2P estruturadas é a complexidade da manutenção da rede, não sendo apropriada para altas taxas de *peers* transitórios. São exemplos: Chord (Chord, 2006) e Pastry

(Pastry, 2006).

Comentaremos na próxima seção algumas características do protocolo Gnutella a fim de ilustrar as arquiteturas distribuídas e introduzir o protocolo que utilizaremos na prova de conceito da nossa proposta.

2.5 Protocolo Gnutella

O Gnutella é um protocolo de rede P2P inspirado no Napster. O Gnutella é *ad-hoc,* aberto, totalmente distribuído e preserva o anonimato dos *peers*. Ele surgiu em meados de março de 2000 e foi criado por Justin Frankel e Tom Pepper (Kan et al, 2001).

Ele foi implementado por alguns programas de compartilhamento de arquivos na Internet, como o LimeWire (LimeWire, 2006a) e o Bearshare[14]. A versão atualmente estável é o Gnutella v0.4 (Gnutella, 2005), mas existem outras propostas de melhorias e extensões que deram origem à versão 0.6 (Klingberg; Manfredi, 2002) e ao Gnutella2[15].

O principal objetivo do Gnutella é compartilhar recursos dos *peers.* Os recursos podem ser arquivos de qualquer tipo. Cada integrante da rede, roda um programa compatível com o Gnutella que procura por outro *peer* para que possa se conectar à rede. Os *peers* da rede trocam diálogos que são, essencialmente, requisições e respostas sobre os objetos por eles compartilhados.

2.5.1 Definição do protocolo Gnutella

A definição do protocolo Gnutella se dá pelo conjunto de mensagens trocadas pelos *servents*, descritos em (Gnutella, 2005) e (Klingberg; Manfredi, 2002), comentadas a seguir.

- Ping: usada para descobrir ativamente outros *peers* na rede. É esperado que um *peer* que receba uma mensagem Ping responda ao remetente uma ou várias mensagens Pong.
- Pong: é a resposta a uma mensagem Ping. A mensagem Pong contém o endereço do *peer* (que respondeu), a porta definida para o recebimento das mensagens e demais

[14] Http://www.bearshare.com
[15] http://www.gnutella2.com

dados sobre o montante de recursos compartilhado por ele.

- Query: é a mensagem utilizada para lançar uma consulta na rede. É esperado que um *peer*, ao receber uma mensagem Query, realize a consulta interna e, caso haja algum objeto que atenda ao critério da consulta, envie uma resposta ao *peer* remetente.
- QueryHit: é a mensagem resposta a uma Query. QueryHit contém dados que atendem a consulta realizada.
- Push: é a mensagem que permite ao *peer* enviar arquivos para a rede.
- Bye: é uma mensagem opcional que informa ao *peer* remoto a saída (desconexão) da rede.

Na seção seguinte comentaremos o funcionamento geral do protocolo. Tal funcionamento é baseado nos tipos de mensagens apresentadas acima.

2.5.2 Funcionamento do protocolo Gnutella

Descreveremos o funcionamento do protocolo Gnutella[16] na entrada de um *peer* na rede, na realização de uma consulta, no recebimento de resposta a uma consulta, na troca de arquivos e na saída do *peer* da rede.

2.5.2.1 Entrada de um *peer* na rede Gnutella

Um novo *peer* se conecta na rede Gnutella através do estabelecimento de conexão com outro *peer* já conectado. O *peer* envia uma mensagem tipo Gnutella Ping para a rede. Um outro já conectado na rede, envia a resposta, o Gnutella Pong. Uma vez o endereço de outro *servent* seja obtido pelo novo *peer,* é estabelecida uma conexão TCP/IP com tal *servent* e o procedimento de *handshaking* acontece. Os passos gerais são:

1. O cliente estabelece conexão TCP com o *servent*;
2. O cliente envia “GNUTELLA CONNECT/0.6”;

[16] Uma vez que o protocolo Gnutella é aberto e recebe contribuição da comunidade, decidimos utilizar como referências as especificações da versão 0.4 e da versão 0.6 (*draft)*.

3. O *servent* responde com “GNUTELLA CONNECT/0.6 200”, que significa que a conexão ocorreu com sucesso;
4. O cliente envia a confirmação da conexão “GNUTELLA CONNECT/0.6 200”;

Com a realização desses passos, o novo *peer* entra na rede. O envio de mensagens do tipo Gnutella Ping permite que o novo *peer* encontre outros *peers.* A partir daí e enquanto o *peer* permanecer na rede poderá haver troca de mensagens do protocolo Gnutella.

Cada mensagem é precedida por um cabeçalho e um conjunto de *bytes* que representa o conteúdo da mensagem. O cabeçalho contém: a identificação da mensagem, o identificador do tipo de mensagem, o tempo de vida da mensagem (TTL - *Time To Live),* o número de vezes que a mensagem foi repassada entre os *peers* da rede (*hops)* e o tamanho do conteúdo da mensagem (*payload).*

As referências (Gnutella, 2005) e (Adar e Huberman, 2000) trazem mais detalhes sobre o procedimento de entrada de um novo *peer* na rede.

2.5.2.2 Realização de uma consulta na rede Gnutella

A realização de uma consulta na rede é feita pelo *peer* através do envio de uma mensagem Gnutella de tipo *Query* (*Payload Type* 0x80). A mensagem possui os seguintes campos: a velocidade de conexão mínima (em kb/s) que um *peer* deve ter para poder responder a consulta, critério de pesquisa e a extensão (opcional).

O critério de pesquisa é formado por caracteres ASCII que representam a *string* de palavras-chave. Os *peers* da rede devem responder apenas caso tenham arquivos que atendam as palavras-chave especificadas. Não há uma definição de como interpretar o critério de pesquisa na especificação do Gnutella, apenas recomendações, por exemplo: espaços são usados para separar palavras-chave, não é permitido o uso de expressões com caracteres coringas como o “*” ou “.”, caracteres maiúsculos e minúsculos são considerados os mesmos. Assim, cada software aderente ao Gnutella pode implementar a sua forma particular de pesquisa.

A consulta é propagada para outros *peers* vizinhos, exceto para o *peer* que propagou a consulta, que então verificam localmente se eles podem atender ao critério da pesquisa. Cada

peer que recebe a Gnutella Query decrementa o campo TTL (*Time to Live)* e incrementa o campo Hops. Enquanto o campo TTL da Query for maior que zero, a mensagem é propagada para os *peers* vizinhos. A figura 2.4 ilustra o processo.

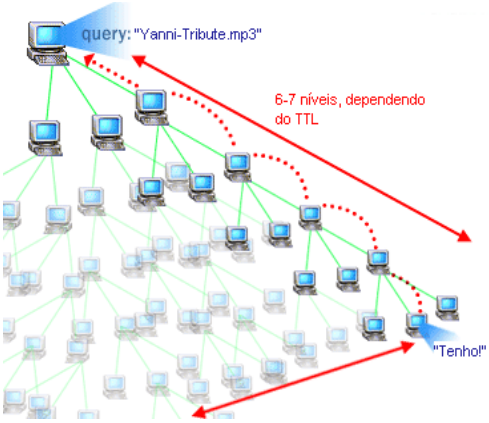

Figura 2.4 - Consulta na rede Gnutella.

Fonte: http://static.howstuffworks.com/gif/file-sharing1.gif, com ajustes.

As referências (Gnutella, 2005) e (Adar e Huberman, 2000) trazem mais detalhes sobre o envio de consulta na rede Gnutella.

2.5.2.3 Recebimento da resposta de uma consulta

Após um *peer* enviar uma consulta para a rede, outros *peers* poderão ter objetos que atendam ao critério desta consulta. Nesse momento, o *peer* detentor do objeto pode enviar uma mensagem de resposta Gnutella QueryHit para o *peer* solicitante.

A mensagem Gnutella QueryHit é formada pelos seguintes campos: número de *hits*, porta, endereço IP, velocidade de conexão, lista dos objetos que atendem ao critério de pesquisa. Tal lista contém ainda os seguintes campos: identificador do arquivo do *peer,* tamanho do arquivo (em *bytes),* nome do arquivo, o bloco de extensão e o bloco extra (opcional).

As mensagens QueryHit chegam de forma assíncrona ao *peer* solicitante que, de acordo com o programa que esteja utilizando, organiza a dados (*hits)* para que o usuário possa escolher qual objeto ele fará o *download.*

As referências (Gnutella, 2005) e (Adar e Huberman, 2000) trazem mais detalhes sobre o recebimento de QueryHits da rede Gnutella.

2.5.2.4 Troca de arquivos

A troca de arquivos entre *peers* acontece quando um *peer* recebe respostas a uma consulta realizada e o usuário comanda o *download* de um (ou mais) objeto(s) da lista *(hits).* O *download* acontece diretamente entre *servents,* ou seja, não há interferência de outro *peer* da rede Gnutella. O *peer* solicitante envia então uma mensagem Gnutella Push ao *peer* detentor do objeto desejado.

O protocolo utilizado para a transferência de arquivos é o HTTP, preferencialmente o HTTP 1.1. O *servent* inicia o *download* enviando ao servidor uma requisição contendo:

```
GET /get/<índice do arquivo>/<nome do arquivo> HTTP/1.1
User-agent: Gnutella
Host: <endereço IP>:<6346 ou outra porta>
Conection: Keep-alive
Range: bytes=0-
```

A resposta do servidor é:

```
HTTP/1.1 200 OK
Server: Gnutella
Content-type: application/binary
Content-lenght: <tamanho do objeto>
```

A exemplo do suporte do HTTP 1.1 e a depender do programa utilizado, um *peer* pode realizar o *download* por intervalo do arquivo de vários *peers*. Nesses casos, a requisição do *servent* deve ser:

```
GET /get/<índice do arquivo>/<nome do arquivo> HTTP/1.1
User-agent: Gnutella
Host: <endereço IP>:<6346 ou outra porta>
Conection: Keep-alive
Range: bytes=<intervalo inicial>-<intervalo final>
```

A troca de arquivos pode ficar comprometida se um dos *peers* estiver protegido por um *firewall* impedindo a conexão na porta do Gnutella. Nesse caso, o *peer* solicitante envia uma mensagem Push para o servidor descrevendo o arquivo desejado.

As referências (Gnutella, 2005) e (Adar e Huberman, 2000) trazem mais detalhes sobre transferência de arquivos entre *peers* da rede Gnutella.

2.6 Estrutura da rede Gnutella

Um problema importante no Gnutella versão 0.4 é o consumo de banda para o seu funcionamento, uma vez que os *peers* podiam aderir à rede de forma aleatória, bastando que algum *peer* próximo já estivesse conectado. Essa modalidade funciona bem para usuários com conexão de banda larga, mas não para usuários com acesso discado.

A fim de reduzir esse problema, a versão 0.6 do Gnutella categorizou os *peers* em peers-folha e *ultrapeers,* hierarquizando a rede Gnutella. A existência desses dois tipos de *peers* permitiu diminuir a quantidade de *peers*-folha da rede, tornando-a mais escalar e consumindo menor banda, uma vez que existe divisão de papéis entre *ultrapeers* e *peers*-folha (Klingberg; Manfredi, 2002). Nas subseções a seguir, comentaremos outros aspectos dos *peers* e dos *ultrapeers*.

2.6.1 Peer e ultrapeers

Nessa subseção, apresentaremos algumas características dos *peers e* dos *ultrapeers* na rede Gnutella versão 0.6.

Os *peers* mantém uma conexão aberta para um *ultrapeer,* que por sua vez pode possuir maior número de conexões, geralmente de 10 a 100, com outros nós-folha (Rohrs, 2002a) (Rohrs, 2002b). Os *ultrapeers* representam uma tentativa de dotar a rede Gnutella de maior escalabilidade e de economia de largura de banda, proporcionada pela redução da quantidade de mensagens Gnutella trocadas entre os *peers* (Rohrs, 2002b).

Os *ultrapeers* são *peers* especiais responsáveis por algumas operações da rede. Os *ultrapeers* se conectam entre eles e também aos *peers* considerados "normais", ou seja, *peers* puros de versões legadas do Gnutella. Os *ultrapeers* agem como um *proxy* da rede Gnutella, assim as consultas realizadas por um *peer* são encaminhadas aos seus *ultrapeers* conectados (Klingberg; Manfredi, 2002).

Na rede Gnutella não existe autoridade central, assim, cada nó decide se atuará como *peer*-folha ou como *ultrapeer,* preservando uma hierarquia distribuída na rede. A decisão do *peer* se tornar *ultrapeer* é tomada com base em alguns critérios:

- Não usar *firewall*. O *peer* deve poder receber várias conexões concorrentes.
- Sistema Operacional usado. Sistemas como Linux, Windows 2000/XP gerenciam melhor várias conexões (*sockets)* que outros, como Windows 9x.
- Largura de banda. Um *ultrapeer* deve possuir uma conexão razoável com a Internet para que possa desempenhar o seu papel e assim suportar várias conexões concorrentes. Para tanto, pelo menos 15Kb/s de *download* e 10Kb/s de *upload* são necessários.
- Conexão por longos períodos de tempo. *Ultrapeers* precisam estar conectados na rede Gnutella com uma freqüência e tempo de permanência superior a dos *peers*-folha.
- Memória RAM e velocidade de processamento. *Ultrapeers* precisam de memória disponível para armazenar as tabelas de roteamento e folga na utilização da CPU para tratar as consultas.

Com esses critérios atendidos, um *peer* pode vir a ser um *ultrapeer ("ultrapeer capable"),* cuja intenção de se tornar um *ultrapeer* precisa ser explícita, durante a sua entrada na rede. Assim, ele só se tornará um *ultrapeer* se houver necessidade de mais *ultrapeers* na rede. A figura 2.5 apresenta uma topologia com *ultrapeers.*

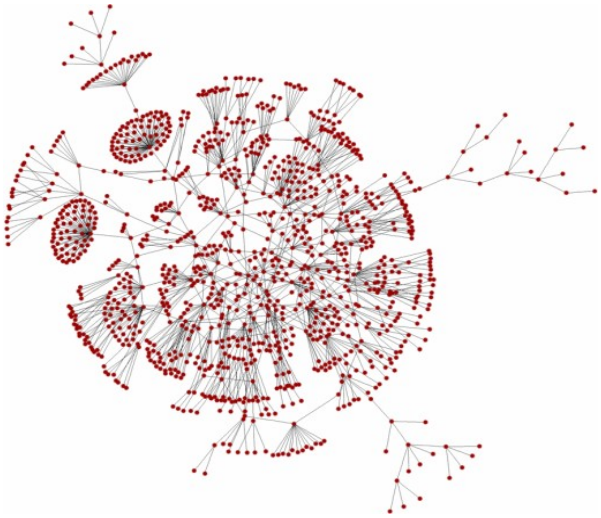

Figura 2.5 - Topologia Gnutella P2P versão 0.6 com *ultrapeers. Fonte:* *http://www.ee.ucl.ac.uk/netdist/gnutella.gif, com ajustes.*

Um *ultrapeer* decide se envia ou não uma consulta para um *peer*-folha. Ele só o faz se o *peer* tiver chance de responder a consulta, de acordo com o protocolo *Query Routing Protocol,* comentado a seguir.

2.6.2 Query Routing Protocol

O *Query Routing Protocol* – QRP é o protocolo de roteamento de consulta utilizado nos *peers* Gnutella versão 0.6. Essa foi uma importante evolução do protocolo Gnutella para se tornar mais escalar. O QRP orienta a operação dos *ultrapeers* quanto ao tratamento e encaminhando de consultas na rede Gnutella, impedindo que *peers* recebam consultas para as quais não poderão responder. Evitar esse tráfego representa redução da carga de processamento nos *peers* e economia no consumo de banda, com a redução das trocas de mensagens (Rohrs, 2002a).

O QRP desempenha operações diferentes quando em um *peer* e em um *ultrapeer.* O *peer* deverá: coletar todas as palavras que compõem o nomes dos recursos, aplicar uma função *hash* em tais nomes e o vetor resultante é enviado, em partes, para o *ultrapeer.* Para o *ultrapeer*, o trabalho é um pouco diferente. Cabe a ele: repassar as consultas ao *peer* enquanto esse não conclui o envio do vetor dos termos com *hash* que vão compor a *query routing table - QRT;* ao receber todo o vetor, o *ultrapeer* aplicará a mesma função *hash* para cada palavra e consultará a QRT, em busca de acertos (*hits)* e caso encontre, a consulta será repassada ao *peer* que detém tal entrada nessa tabela (Klingberg; Manfredi, 2002).

Uma vez que a coleção de arquivos de um *peer* poderá sofrer alteração (inserções e ou deleções), o vetor contendo as palavras dos nomes dos arquivos precisa ser atualizado, de tempos em tempos, no *ultrapeer.* A atualização é feita através do envio de mensagens Gnutella específicas (*Route_Table_Update)* que podem zerar (*Reset)* ou ajustar (*Patch)* a QRT no *ultrapeer*. Maiores detalhes sobre a função *hash* utilizada e o roteamento de consultas podem ser encontrados em (Rohrs, 2002a).

2.7 Compartilhamento de arquivos

O compartilhamento de arquivos é uma das principais funcionalidades de um software P2P. Apresentaremos algumas formas de compartilhamento de arquivos e, em seguida, compararemos com o compartilhamento realizado através de software P2P.

O compartilhamento de arquivos pode ser feito através de serviços de um sistema operacional de rede, por FTP, na web através de sites que cedem ou alugam espaço de armazenamento[17], por software de controle remoto ou por sistemas P2P.

O compartilhamento de arquivos não iniciou na Internet com os sistemas P2P. O FTP – *File Transfer Protocol* já permitia que usuários pudessem "baixar" arquivos de um servidor. Os usuários podem fazer acesso de forma anônima ou não. O tipo de acesso – leitura ou escrita sobre o servidor – é definido pelo administrador do serviço. A figura 2.6 mostra alguns arquivos compartilhados em um servidor FTP. Observa-se que o acesso aos arquivos compartilhados deve ser feito por um cliente-FTP.

Outra forma de compartilhamento de arquivos é a realizada em uma rede local. Tal compartilhamento difere do que é realizado através de sistemas P2P na Internet e do feito por FTP. Na rede local, o sistema operacional[18], que executa em estações, possui o serviço compartilhamento recursos na rede. O usuário pode eleger qual recurso deseja compartilhar. Os recursos podem ser: impressoras, diretórios (ou discos rígidos inteiros), unidades de CD etc.[19]

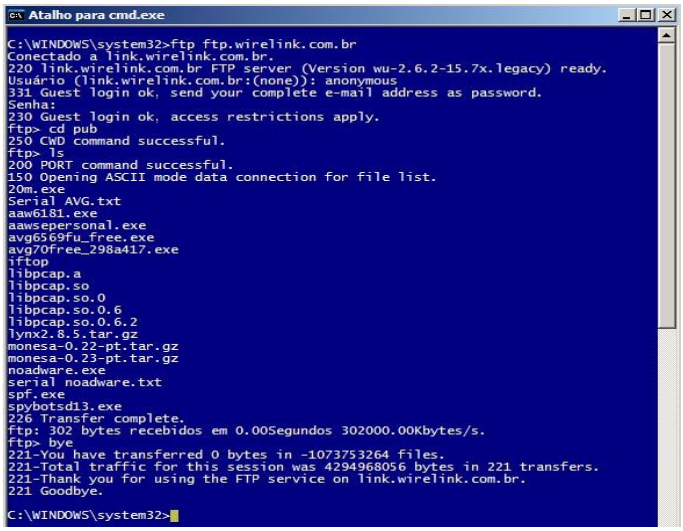

Figura 2.6 - Servidor FTP compartilhando arquivos.

[17] Www.xdrive.com e www.myspace.com são alguns exemplos.
[18] Realizamos os testes com o sistema operacional Microsoft Windows XP.
[19] No sistema operacional Windows XP, usado para os testes, não foi possível compartilhar recursos como processador e memória RAM com outros usuários da rede.

Em geral, esse compartilhamento pode ser feito associando-se uma senha para acesso ao recurso ou definindo quais usuários (ou grupo de usuários) da organização podem fazer acesso ao recurso, como também o tipo de acesso: leitura, gravação etc. A figura 2.7 ilustra este tipo de compartilhamento.

Uma vez compartilhada a pasta e definida a permissão de acesso, um outro usuários da rede que detenha as credenciais poderá fazer acesso ao recurso. O acesso é feito através dos redirecionadores, porção de software que executa em cada estação, que redireciona as requisições aos recursos compartilhados para os computadores detentores do recurso.

Na figura 2.8, mostramos a pasta que foi compartilhada e já disponível na rede. Observa-se que o acesso é feito por utilitário específico do sistema operacional, ou seja, seguindo um outro protocolo – SMB[20].

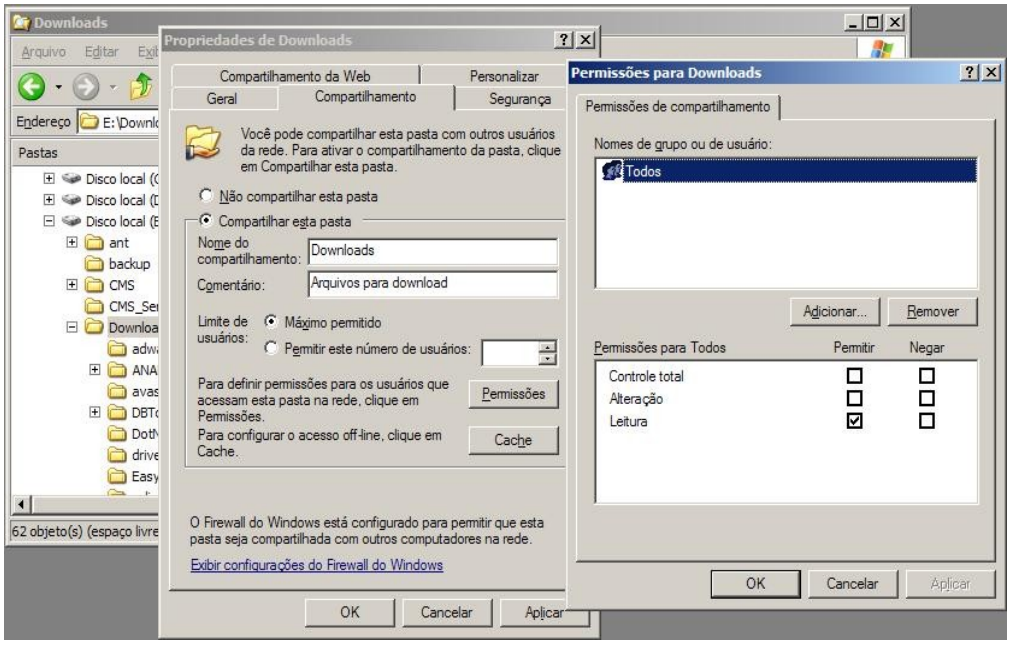

Figura 2.7- compartilhamento de arquivos na rede local com Windows XP.

Em sistemas P2P, o compartilhamento de arquivos é feito quando o usuário instala um

[20] SMB – *Server Message Block*, é um protocolo usado para o compartilhamento de arquivos, impressoras, portas de comunicação, usado em várias edições do Microsoft Windows e Linux.

software P2P e define qual pasta (ou pastas) compartilhar. O software P2P, ao executar, estabelece conexão com a rede P2P na Internet através de uma porta de comunicação específica. Por exemplo, no LimeWire, o usuário pode selecionar uma pasta que deseja compartilhar ou usar a pasta padrão (*"shared"*). A figura 2.9 apresenta a janela com os parâmetros de configuração do LimeWire, onde é possível definir quais pastas serão compartilhadas pelo *peer*.

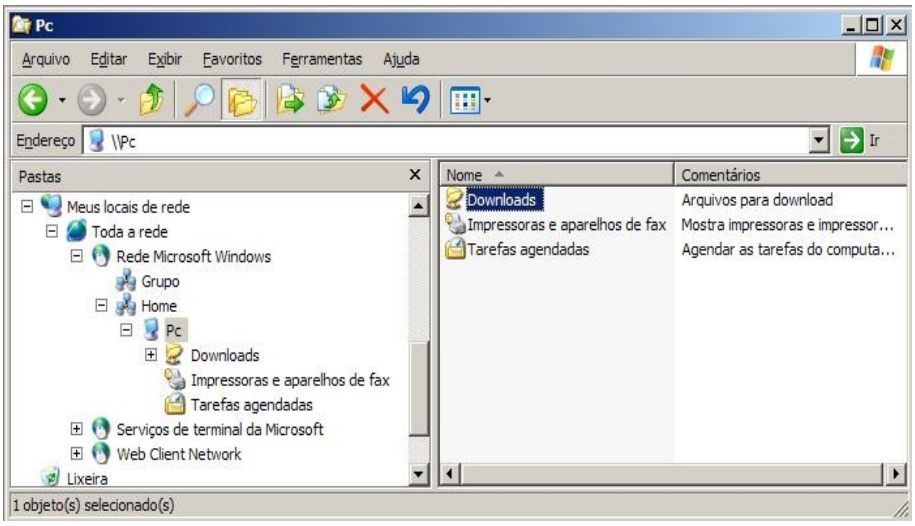

Figura 2.8 – Pasta Downloads compartilhada.

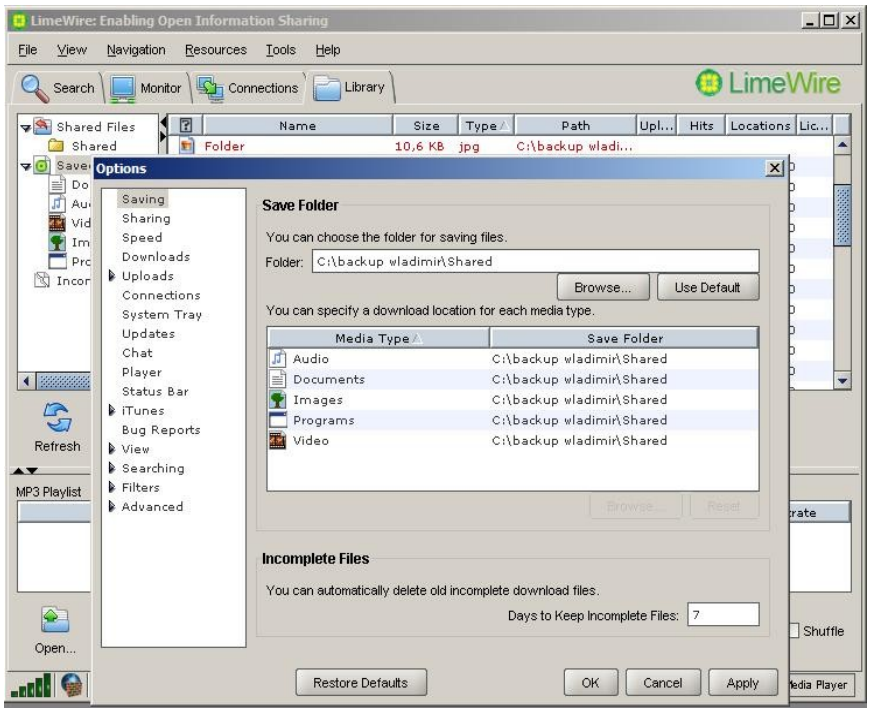

Figura 2.9 - Arquivos compartilhados na pasta selecionada pelo usuário.

O compartilhamento de arquivos em sistemas P2P difere do compartilhamento realizado pelas demais formas de compartilhamento. Destacamos algumas diferenças:

- O compartilhamento de arquivos é realizado pelo software P2P, enquanto ele está em execução na estação. Observamos que a pasta compartilhada no software P2P não permanece compartilhada quando ele para de executar, ou seja, não se trata de um compartilhamento do sistema operacional gerenciado pelo software mas um compartilhamento totalmente gerenciado pelo software P2P.
- Ainda não há padrão de interoperabilidade entre redes P2P. Cada rede tem a sua forma de incluir ou excluir um certo *peer,* qual porta utilizar e um conjunto de mensagens suportadas. No compartilhamento via FTP, a porta TCP utilizada é por padrão a 21, em qualquer utilitário. O Emule (Emule, 2006) utiliza, por padrão, a porta TCP 4662. Já o LimeWire escolhe uma porta aleatória e ainda permite que o usuário altere tal porta.
- O software P2P não trata de segurança, deixando para o usuário adotar as medidas cabíveis como o uso de anti-vírus, firewall pessoal e sensores de intrusão. A interação entre *peers* pode gerar uma série de vulnerabilidades, como a abertura de portas em protocolos de rede. Nas demais formas de compartilhamento de arquivos, via FTP e via rede, também há vulnerabilidades, como as senhas sem criptografia utilizadas no FTP. Em operação normal, um outro *peer* da rede poderá solicitar o *download* ou realizar consultas. Portanto, em operação normal, apenas o *peer* detentor dos arquivos poderá realizar operações de escrita na pasta apesar de não checar o que está escrevendo.
- Não há controle sobre a redundância e nem sobre versões de documentos. Pode-se encontrar arquivos duplicados em vários *peers.* Isso permite que os usuários da rede P2P tenham outras opções de *download*. Entretanto, não há controle sobre qual o momento que um *peer* deixa a rede. Em sistemas P2P, no FTP e nos compartilhamentos de rede local, só poderão existir arquivos com nomes iguais se esses estiverem em pastas diferentes.

Há serviços na Internet que permitem o armazenamento de arquivos de usuário para o *download* de outros usuários. São exemplos de tais serviços o Yahoo Porta Arquivos, o xDrive e o MySpace. O serviço é semelhante ao compartilhamento via FTP, onde os arquivos são colocados em um servidor e o acesso é feito através de um navegador (*browser*). O serviço pode ser pago ou não.

Diferentemente do simples compartilhamento de arquivos, a distribuição de conteúdo

requer mecanismos mais sofisticados para localização e recuperação de documentos segundo o seu conteúdo, segurança, publicação e indexação de documentos. Comentaremos mais sobre distribuição de conteúdo em redes P2P no próximo capítulo.

2.8 Outras aplicações em redes P2P

Os recursos compartilhados em redes P2P são em geral arquivos. Entretanto, existem outras aplicações, por exemplo:

- Computação em *grid* (Vanwasi, 2001): *peer-to-peer* e computação em *grid* são abordagens para a computação distribuída interessadas na organização do compartilhamento dos recursos nas comunidades (Vanwasi, 2001).
- Mensagens instantâneas: a troca de mensagens instantâneas em programas como o Microsoft Messenger[21], ICQ[22], Miranda[23] aproximam pessoas e organizações permitindo uma comunicação em tempo real e muitas vezes menos custosa, em termos financeiros, caso os *peers* estejam em regiões geográficas distantes. A comunicação com mensagens instantâneas ocorre por meio de *chats* individuais ou em grupo. Nesses programas também é possível a troca de arquivos entre *peers*[24].
- Voz sobre IP (VoIP): um exemplo de aplicação VoIP em redes P2P é o Skype[25]. Por meio desse software, um canal de comunicação por voz é estabelecido através da Internet permitindo que pessoas possam se comunicar com custo muito abaixo das tarifas de ligações interurbanas.
- Computação P2P: distribuição entre os *peers* de parte de um cálculo ou processamento que é feito quando o processador está ocioso. Um exemplo é a rede Seti @ Home[26], cujo programa Boinc, utiliza da ajuda do poder de processamento de vários computadores conectados na Internet que quando estão ociosos passam a analisar imagens captadas por telescópios afim de descobrir vida extra-terrestre inteligente.

---

21 www.microsoft.com/messenger.
22 www.icq.com
23 www.miranda-im.org
24 Apesar de haver servidor de mensagem instantâneas, ele atua prioritariamente como roteador das mensagens dos *peers* ao passo que a troca de arquivos ocorre diretamente entre os *peers*.
25 www.skipe.com
26 Search for Extraterrestrial Intelligence (SETI) , disponível em http://setiathome.berkeley.edu/

- Jogos: a arquitetura descentralizada dos sistemas P2P é particularmente interessante para jogos em rede. A ausência de uma autoridade centralizada, juntamente com a capacidade de comunicação entre os *peers* em rede, permite a criação de comunidades de jogadores ao redor do planeta e cria oportunidades de interação para os criadores de jogos. Além disso, os desenvolvedores podem ter foco no jogo em si e fazer uso dos *frameworks P2P* para o funcionamento da rede.

2.9 Sumário do capítulo

Nesse capítulo, abordamos o paradigma das redes P2P trazendo o seu histórico, alguns conceitos e características, as classificações das arquiteturas P2P, o protocolo Gnutella em maiores detalhes, as formas de compartilhamento de arquivo e aplicações das redes P2P.

No próximo capítulo, trataremos de recuperação da informação com maior ênfase na distribuição e recuperação de conteúdo em sistemas P2P.

Capítulo 3 – Recuperação de informação

Nesse capítulo abordaremos a recuperação de informação em coleções de documentos, com enfoque na distribuição de conteúdo em redes P2P.

3.1 Introdução

Após a apresentação dos aspectos dos sistemas P2P no capítulo anterior, faremos uma abordagem sobre a área de Recuperação de Informação para complementar a fundamentação teórica da nossa proposta. Faremos uso de técnicas e modelos de recuperação de informação para dotar a nossa arquitetura da capacidade de localizar documentos pelo seu conteúdo naquelas redes.

Recuperação de Informação (*Information Retrieval – IR)* é dedicada a pesquisa de tecnologias para manipulação e recuperação de informações em grandes coleções com diferentes formatos de apresentação. Nessa área são investigadas formas de representação, de armazenamento, de organização e de acesso a itens de informação de modo a permitir ao usuário fácil acesso à informação na qual ele está interessado (Callan et al., 1995).

Tradicionalmente, a informação toma forma de texto, o que sugere que a Recuperação de Informação atue majoritariamente sobre documentos, sejam eles informação em texto puro, administrativa, em diretórios, numérica ou bibliográfica (Ingwersen, 1992). Há porém outras formas e formatos em que a informação está armazenada, mas nos concentraremos em informação textual nesse trabalho.

No final da década de 60, surgiram os primeiros catálogos bibliográficos *on-line* que em alguns minutos permitiam a recuperação de informações armazenadas. Para manusear as informações, o usuário manipulava um ambiente de consulta utilizando um conjunto controlado de operações e linguagens pré-definidas. Naquela época os computadores tinham baixo poder de processamento sobre grandes quantidades de informação.

Entre os anos 1970 e 1980, o tamanho das coleções de informações armazenadas e

manipuladas pelos computadores cresceu de centenas para milhares de megabytes. Finalmente, na década de 90, grande quantidade de informações estava disponível devido à popularização da *web* (Guerreiro; Macedo, 2005).

Para acompanhar o crescimento do volume de informação disponível, indústrias trabalharam no desenvolvimento de máquinas com maior capacidade de armazenamento e processamento. Paralelamente, pesquisadores de diferentes áreas, inclusive de Recuperação de Informação *(Information Retrieval – IR)* e Hipermídia, concentraram esforços no desenvolvimento de mecanismos que promoviam a recuperação, a integração estruturada das informações armazenadas e a apresentação das mesmas (Guerreiro; Macedo, 2005).

No contexto da área de Recuperação de Informação, há distinção entre recuperação de dados e de recuperação de informação. Recuperação de dados consiste em determinar quais objetos de uma coleção contém certa palavra-chave ou valor, que em alguns casos, é suficiente para a realização de consultas diretas, mas não tanto para satisfazer a necessidade de informação do usuário. A recuperação de dados lida com uma linguagem que tem por objetivo recuperar todos os objetos que atendem as condições de consulta, claramente definidas, como em expressões regulares e expressões de álgebra relacional. Assim, um objeto ausente ou excedente em uma consulta de recuperação de dados, consiste em erro (Ribeiro-Neto; Baeza-Yates, 1999).

Na recuperação de informação, por outro lado, pequenos e eventuais erros são toleráveis. Essa diferença existe porque ela lida com textos em linguagem natural que podem ser semanticamente ambíguos, diferente das coleções estruturadas como as tabelas de um banco de dados (Ribeiro-Neto; Baeza-Yates, 1999). Assim, para que um sistema de Recuperação de Informação seja efetivo na satisfação da necessidade do usuário, é necessário que ele "interprete" os documentos da coleção e crie um *rank* de relevância dos documentos com a consulta do usuário. Essa "interpretação" dos documentos da coleção envolve a extração sintática e semântica de termos e palavras-chave. A dificuldade não está somente em como extrair informação, mas também em como usá-la para decidir a relevância do documento analisado (Ribeiro-Neto; Baeza-Yates, 1999).

As tarefas de representação da coleção de documentos e de consultas estão fortemente relacionadas. O tipo de modelo de recuperação de informação indica quais as formas de consultas possíveis sobre a coleção. Existem três modelos clássicos[27] de recuperação de

[27] Existem extensões a esses modelos, também disponíveis em (Guerreiro; Macedo, 2005) e (Ribeiro-Neto;

informação, comentados a seguir: o booleano, o vetorial e o probabilístico (Guerreiro; Macedo, 2005).

Modelo booleano. Representa documentos e consultas com base na teoria dos conjuntos e na álgebra booleana. Cada documento é representado por um conjunto de termos de índice aos quais podem ser atribuídos apenas pesos binários, onde: 1 indica se o termo de índice existe no documento e 0, caso não exista. As consultas são expressões booleanas especificadas por palavras-chave conectadas por operadores lógicos (AND, OR e NOT). Assim, a recuperação é baseada em decisão binária, não suportando portanto a similaridade parcial, ou seja, um documento é ou não é relevante. Para muitos usuários, a expressão de consulta é complexa (Guerreiro; Macedo, 2005).

Modelo vetorial. Documentos e consultas são representados como vetores multidimensionais. Cada documento é representado por um vetor numérico no qual cada elemento representa um único termo ou um termo de índice de um documento. A esses elementos são atribuídos pesos parciais. Os pesos indicam o grau de especificidade relativo de cada termo e são utilizados para calcular similaridade entre consultas e documentos, como também classificar os resultados (Guerreiro; Macedo, 2005).

Modelo probabilístico. O modelo utiliza um *framework* para representar o documento e a consulta baseado em probabilidades (Ribeiro-Neto; Baeza-Yates, 1999)(Guerreiro; Macedo, 2005). A idéia é classificar os documentos da coleção de acordo com a sua probabilidade de ser relevante à busca. O modelo assume que dada uma consulta, existe um conjunto-resposta ideal e o processo de busca está baseado na tentativa de encontrar as propriedades desse conjunto. Como as probabilidades não são conhecidas no início da consulta, o modelo sugere um processo de suposição, refinado na interação com o usuário, que deve informar quais documentos são pertinentes (Guerreiro; Macedo, 2005).

Na seção seguinte, comentaremos o processo de recuperação de informação para que possamos entender em qual o nível e como acontece a preparação e a "interpretação" dos documentos de uma coleção, como também apresentar técnicas que serão úteis para posicionar a nossa proposta.

Baeza-Yates, 1999).

3.2 Processo de recuperação de informação

O processo de recuperação de informação requer uma série de subprocessos e componentes de software para a sua realização. O objetivo é permitir que o sistema consiga responder às consultas formuladas por seus usuários. Uma vez que as arquiteturas de software dos sistemas de recuperação de informação podem ser diferentes, utilizaremos a arquitetura genérica de (Ribeiro-Neto; Baeza-Yates, 1999), presente na figura 3.1, para ilustrar esse processo.

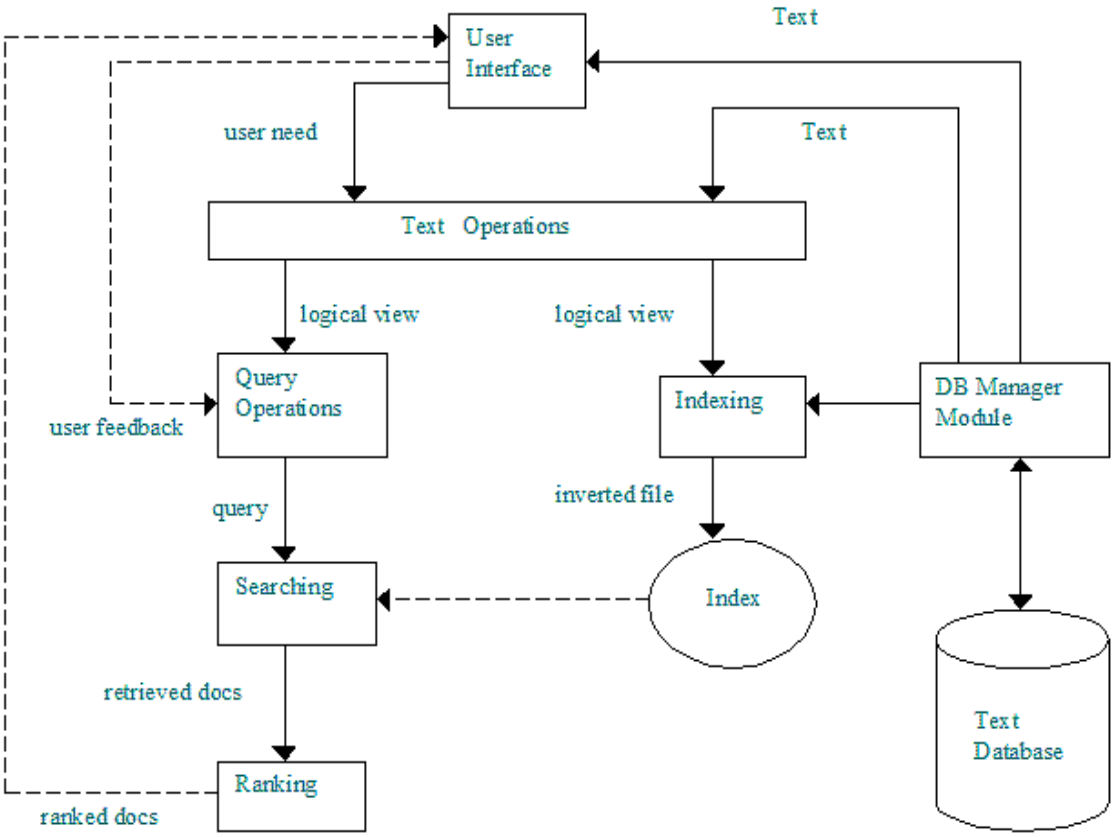

Figura 3.1 - arquitetura genérica de um software de recuperação de informação. *Fonte: (Ribeiro-Neto; Baeza-Yates, 1999), com ajustes.*

O processo inicia com a criação da base de dados (ou repositório) textual. Essa etapa envolve a definição: da coleção de documentos a serem usados, das operações a serem aplicadas sobre o texto, do modelo de estrutura e quais elementos podem ser recuperados. As operações sobre o texto criam a visão do lógica de cada documento

Com a visão lógica do documento definida, o índice é então construído. O índice é uma importante estrutura de dados na obtenção do resultado da consulta. É ele que vai permitir que as consultas sobre as coleções possam ser respondidas com menor tempo, apesar do esforço computacional (e tempo) empregado na sua construção (e atualização) e do espaço gasto para o seu armazenamento.

Uma vez que os documentos estão indexados, as consultas podem ser realizadas. O usuário entra com a consulta desejada através de uma interface. A consulta informada pelo usuário é verificada (*parsed)* e também passa pelas operações de texto para extrair os termos de consulta nos mesmos moldes dos termos previamente indexados, ou seja, é criada a visão lógica da consulta do usuário.

Sobre a visão lógica da consulta, operações de consulta são aplicadas gerando a *query,* que é a representação do sistema de recuperação de informação para a consulta do usuário. A *query* é processada (*Searching)* utilizando os índices. Os documentos que atendem ao critério são então relacionados, caso existam.

O resultado é classificado (*ranking)* de acordo com a probabilidade de relevância com a consulta do usuário. O usuário então examina o conjunto de documentos trazidos pelo sistema e pode dar início ao refinamento da sua consulta com sucessivas interações com o sistema (*feedback)*, através da reformulação da consulta.

Nas subseções a seguir, detalharemos as operações sobre textos, a indexação e as operações de consulta utilizadas no processo de recuperação de informação.

3.2.1 Operações sobre textos

Na linguagem escrita, algumas palavras trazem maior significado do que outras, geralmente os substantivos são mais representativos no conteúdo de um documento (Ribeiro-Neto; Baeza-Yates, 1999). Os sistemas de Recuperação de Informação realizam uma série de tratamentos e operações sobre o texto dos documentos da coleção com o objetivo de criar uma visão lógica dele e assim acelerar o acesso à informação nas consultas futuras como também reduzir o tamanho dos índices, já que nem todas as palavras do texto virarão termos do texto.

O pré-processamento sobre o conteúdo do documento é dividido nos seguintes passos (Ribeiro-Neto; Baeza-Yates, 1999), comentados adiante:

- Análise léxica: tratar dígitos, hífens, marcas de pontuação e maiúsculas/minúsculas;
- Eliminação de *stopwords*, comentada a seguir*;*
- Redução das palavras ao seu radical (*stemming),* comentada a seguir*;*
- Seleção de termos que comporão o índice;
- Construção de estruturas para categorizar os termos, tais como dicionário de sinônimos e conceitos.

A figura 3.2 demonstra os passos realizados no pré-processamento do texto.

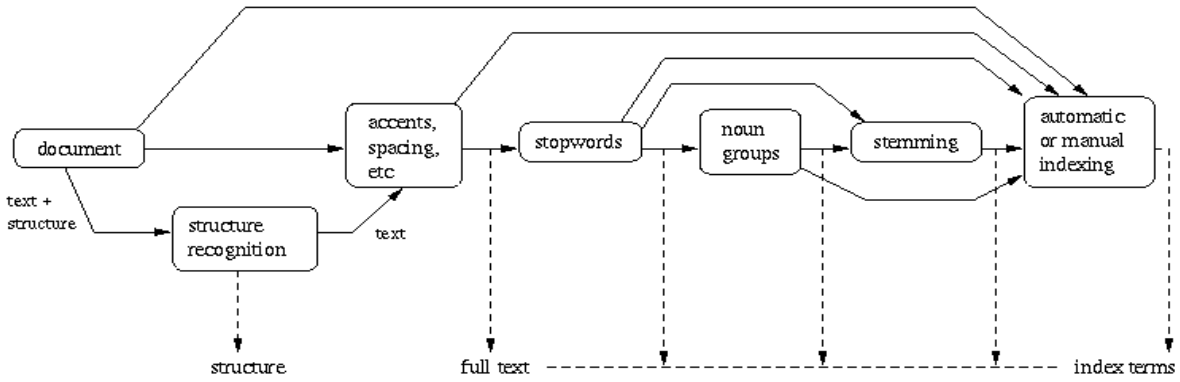

Figura 3.2 - Visão lógica de um documento, obtida após o pré-processamento do texto. *Fonte: (Ribeiro-Neto; Baeza-Yates, 1999).*

O primeiro passo no pré-processamento do texto é a análise léxica. Ela objetiva a identificação das palavras no fluxo de caracteres do texto (Ribeiro-Neto; Baeza-Yates, 1999). Essa análise também trata símbolos como os dígitos, hífens, marcas de pontuação e converte letras maiúsculas em minúsculas. O tratamento dado a esses símbolos varia com o sistema de recuperação de informação. Em geral, tais símbolos não são indexados.

O passo seguinte é a retirada de *stopwords*. ***Stopwords*** são palavras, geralmente artigos, preposições e conjunções presentes no texto que não devem ser indexadas porque são muito freqüentes e tem pouca representatividade na identificação e na semântica do documento. A eliminação das *stopwords* tem um benefício adicional: ela reduz o tamanho da estrutura de índice (Ribeiro-Neto; Baeza-Yates, 1999). As operações sobre o texto reduzem a complexidade da representação do documento e permitem a obtenção do conjunto de termos

desse texto (Ribeiro-Neto; Baeza-Yates, 1999).

A realização do *stemming* é o passo seguinte. ***Stemming*** é a técnica de redução de palavras a seus radicais (Guerreiro; Macedo, 2005). Ela é útil para melhorar a performance de sistemas de recuperação de informação porque há redução de variações de palavras para o seu radical (Ribeiro-Neto; Baeza-Yates, 1999). Freqüentemente, os usuários especificam uma certa palavra em uma consulta, mas apenas uma variação dessa palavra é encontrada em um documento relevante. Plurais, gerúndios, tempos verbais são exemplos de variações sintáticas que podem comprometer o resultado da consulta.

A seleção dos termos para compor o índice pode contemplar todas as palavras do texto ou apenas uma parte. Essa separação pode ser feita de duas formas: ou por um especialista (manualmente, no contexto da ciência bibliográfica) ou de forma automática, como proposto em (Callan et al., 1995). A estratégia utilizada é a eliminação sistemática dos verbos, adjetivos, advérbios, conectivos, artigos e pronomes (Ribeiro-Neto; Baeza-Yates, 1999); restando assim os substantivos que possuem maior semântica.

A construção de estruturas para categorizar os termos, tais como dicionários de sinônimos e conceitos, pode ser usada pelo sistema de recuperação de informação, em domínios específicos de conhecimento, com o objetivo de utilizar um vocabulário controlado para a indexação e para a consulta. Um vocabulário controlado tem a vantagem de permitir a normalização da indexação de conceitos, a identificação de termos de índice com semântica clara e assim efetuar a recuperação de informação baseada em conceitos e não apenas em palavras (Ribeiro-Neto; Baeza-Yates, 1999).

O último passo é a construção dos índices, que pode ser feita de forma manual ou automática. O uso de índices é a forma mais utilizada para acesso rápido dos sistemas de recuperação da informação (Chakrabarti, 2003). Entretanto, a depender do sistema de recuperação de informação e dos modelos utilizados por ele, nem todas as palavras do texto devem ser indexadas. Os índices permitem organizar o texto segundo um conjunto de critérios e, assim, acelerar a consulta sobre a coleção de documentos. Comentaremos mais sobre índices na seção seguinte.

3.2.2 Indexação

Uma opção de consulta sobre uma coleção de documentos seria realizá-la vasculhando diretamente cada documento, em tempo de consulta. Considerando que a coleção fosse composta por muito poucos documentos e de tamanho pequeno (alguns poucos *bytes*), talvez fosse eficiente. Entretanto, com o crescimento do tamanho das coleções tanto na quantidade quanto no tamanho dos documentos, vasculhar o conteúdo de cada texto, em tempo de consulta, implica em grande tempo de processamento para a construção do resultado para o usuário, como também esforço repetitivo quando várias consultas são feitas ao sistema.

Uma outra forma de consulta sobre tal coleção, seria construir uma estrutura de dados sobre os textos que permita reduzir o tempo e o esforço computacional do sistema de recuperação de informação possa apurar o resultado da consulta. Tal estrutura de dados é o índice.

O uso de índices é a forma mais utilizada para acesso rápido dos sistemas de recuperação da informação (Chakrabarti, 2003). Os índices permitem organizar o texto segundo um conjunto de critérios, indicando a seqüência dos objetos através de ponteiros, e assim aceleram a consulta sobre a coleção de documentos.

É justificável construir e manter índices quando a coleção de textos é grande e semi-estática. Tais coleções podem ter seus índices atualizados em intervalos de tempo na freqüência de uma vez por dia, por exemplo. Por outro lado, quando a coleção é pequena ou os documentos sofrem muitas alterações em intervalos pequenos de tempo, o esforço computacional para manter os índices atualizados pode ser superior ao benefício de utilizá-los, para um sistema de recuperação de informação.

Existem três principais técnicas de indexação: vetor de sufixos, arquivos de assinatura e arquivos invertidos (Ribeiro-Neto; Baeza-Yates, 1999). Na subseção seguinte, comentaremos mais sobre índices com arquivos invertidos por serem os mais utilizados nas aplicações de recuperação de informação e pela nossa proposta.

3.2.2.1 Índices com arquivos invertidos

Um índice com arquivo invertido (ou índice invertido) é um mecanismo orientado a palavras para indexação de uma coleção de textos (Ribeiro-Neto; Baeza-Yates, 1999). O nome índice invertido vem da estrutura desse índice; a partir de uma matriz documento-termo,

formada pelos documentos da coleção e seus respectivos termos, tal matriz é transformada por transposição em uma matriz termo-documento (Chakrabarti, 2003). Portanto, no lugar de responder quais os termos de um certo documento, que era favorecida pela antiga organização, a matriz fica organizada para melhor responder, dado um certo termo, quais os documentos em que ele está presente. A figura 3.3 ilustra o resultado.

docs	*t1*	*t2*	*t3*
D1	**1**	**0**	**1**
D2	**1**	**0**	**0**
D3	**0**	**1**	**1**
D4	**1**	**0**	**0**
D5	**1**	**1**	**1**
D6	**1**	**1**	**0**
D7	**0**	**1**	**0**
D8	**0**	**1**	**0**
D9	**0**	**0**	**1**
D10	**0**	**1**	**1**

Terms	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**...**
t1	1	1	0	1	1	1	0	
t2	0	0	1	0	1	1	1	
t3	1	0	1	0	1	0	0	

Figura 3.3 - Matriz termo-documento. *Fonte: http://www.sims.berkeley.edu:8000/courses/is202/f98/Lecture19/sld005.htm*

A estrutura de um índice invertido é composta pelo vocabulário e pelas ocorrências. O vocabulário é o conjunto de todas as palavras distintas do texto. Para cada palavra, uma lista de todas as posições onde ela aparece dentro do texto é armazenada. A lista dos locais onde a palavra aparece no texto é chamada de ocorrências (Ribeiro-Neto; Baeza-Yates, 1999).

Há índices invertidos que colocam o deslocamento ou a posição onde uma certa palavra aparece. Há outros porém que mencionam apenas o documento que certa palavra (ou termo) aparece. A figura 3.4 ilustra as duas formas.

As ocorrências demandam mais espaço do que os termos. Uma vez que cada termo só aparece uma vez no vocabulário, o mesmo não acontece com as ocorrências. É recomendado que o vocabulário e as ocorrências fiquem armazenados em arquivos diferentes. Essa divisão melhora o desempenho da técnica porque o vocabulário, muitas vezes, pode ficar em memória juntamente com o ponteiro para as suas ocorrências armazenadas no outro arquivo.

Na prática, as ocorrências consomem entre 30% e 40% do tamanho do texto (Ribeiro-Neto; Baeza-Yates, 1999). Assim, para reduzir o espaço ocupado pelas ocorrências, o índice invertido pode utilizar blocos de endereçamento.

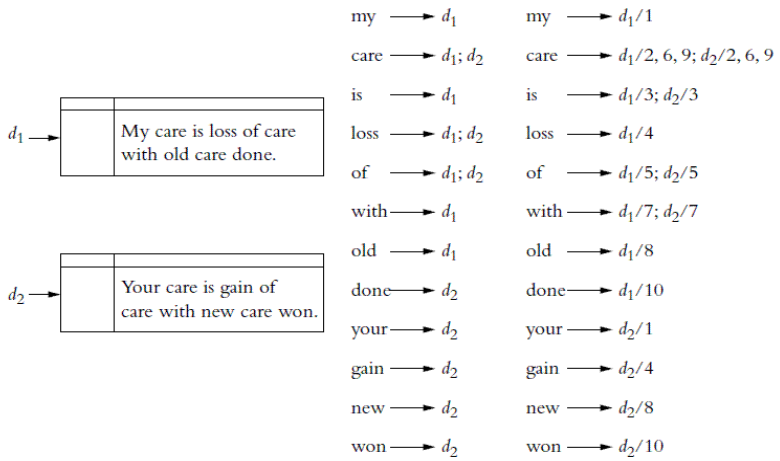

Figura 3.4 - Duas variantes do índice invertido, com e sem a posição do termo no documento. *Fonte: (Callan et al., 1995).*

Os blocos de endereçamento são criados a partir da divisão do texto. Cada parte do texto vai para um bloco específico. Isso permite economia de espaço para o endereçamento do bloco (ponteiros) já que existem menos ocorrências. Com essa técnica, 5% do tamanho do texto, em média, é gasto com o armazenamento das ocorrências (Ribeiro-Neto; Baeza-Yates, 1999). A desvantagem da técnica é que para encontrar a posição correta de certa palavra no texto, será necessária uma travessia dentro do bloco que a contém. Ainda em (Ribeiro-Neto; Baeza-Yates, 1999), há uma análise da complexidade computacional para a construção de índices invertidos em vários casos, com e sem o uso de blocos.

O algoritmo de consulta sobre índices invertidos tem três passos (Ribeiro-Neto; Baeza-Yates, 1999):

- Busca no vocabulário: as palavras são buscadas no vocabulário;
- Recuperação das ocorrências: a lista de todas as ocorrências das palavras é recuperada das ocorrências;
- Manipulação das ocorrências: as ocorrências são processadas para atender o tipo de consulta realizada.

A busca no vocabulário é feita de acordo com a estrutura de dados utilizada para armazenar o vocabulário. Podem ser utilizadas funções *hash, tries,* árvores-B ou ainda dispor os termos em ordem alfabética. Há diferença na complexidade computacional para as buscas

no vocabulário e no tipo de consulta suportado por cada tipo de estrutura de dados. Por exemplo, consultas por prefixos não são suportadas por funções *hash.*

A recuperação das ocorrências é o passo seguinte do algoritmo de consulta. Uma vez encontrado o termo dentro do vocabulário, haverá um ponteiro para a lista das ocorrências daquele termo nos documentos da coleção. Caso a consulta seja feita com uma frase, essa deve ser quebrada nas várias palavras que são então buscadas no vocabulário separadamente. O resultado será a união das ocorrências parciais. A operação de união das ocorrências é muito custosa para os índices invertidos (Ribeiro-Neto; Baeza-Yates, 1999).

O passo final do algoritmo de consulta é a manipulação das ocorrências, recuperadas no passo anterior, a fim de adapta-las para o tipo de consulta realizada, por exemplo: consultas feitas com frases, consultas por proximidade, ou ainda consultas feitas com operadores booleanos. Outro tratamento é necessário quando blocos forem utilizados. Nesse caso, será necessário percorrer o bloco recuperado para encontrar a posição exata da palavra pesquisada, se houver necessidade de exibição da posição da palavra.

Na seção seguinte, abordaremos outras questões acerca das consultas realizadas em sistemas de recuperação de informação, com maior enfoque nas consultas textuais, por essas terem maior proximidade com a nossa proposta.

3.2.3 Consultas textuais

Os sistemas de recuperação da informação são construídos objetivando a realização de consultas sobre as coleções. Para tanto, alguma estrutura de dados, geralmente os índices, é usada para organizar (e preparar) os itens da coleção para as futuras consultas que serão submetidas ao sistema.

O tipo de consulta que o usuário formulará ao sistema é dependente do modelo de recuperação de informação utilizado (Ribeiro-Neto; Baeza-Yates, 1999). Comentaremos a seguir três tipos de consultas textuais que se aproximam da nossa proposta: consulta por palavras, consulta por contexto e consultas booleanas. Em (Ribeiro-Neto; Baeza-Yates, 1999), são apresentadas outras linguagens de consulta.

Consulta por palavra. É a consulta mais elementar que pode ser formulada em um sistema de recuperação de informação. Ela é intuitiva e facilita a interação do usuário final

com o sistema. O resultado da consulta é a lista dos documentos da coleção que possuem tal palavra, contendo ou não a posição dentro do texto onde a palavra aparece.

A fim de criar um *rank* dos documentos recuperados, um sistema de recuperação de informação pode utilizar duas métricas: a freqüência do termo por documento ou a freqüência do termo na coleção. Para melhorar a precisão do resultado, um sistema pode também não realizar o *stemming* e a retirada das *stopwords,* indexando assim todas as palavras do texto.

Consulta por contexto. Complementando a consulta por palavras, alguns sistemas de recuperação de informação realizam consultas de palavras em um certo contexto, ou seja, palavras que aparecem próximas a outras. Existem dois tipos de consulta por contexto: por frase e por proximidade. Consulta por frase é uma seqüência de consultas por palavras onde o sistema adota o mesmo separador para o texto e para a consulta, retirando as *stopwords.* Já a consulta por proximidade é uma forma mais relaxada de consulta por frase onde uma lista de palavras e a distância máxima entre elas são passadas ao sistema (Ribeiro-Neto; Baeza-Yates, 1999). Com estes parâmetros o sistema compõe o resultado da consulta.

Consulta booleana. Ela combina os termos da consulta e operadores lógicos (AND, OR, NOT). Ela é bastante utilizada nos sistemas de recuperação de informação. Pode haver ainda composição de resultados, ou seja, operadores podem ser compostos com o resultado de outros operadores. Em geral, consulta booleanas não utilizam *rank,* uma vez que um documento satisfaz ou não a consulta (Ribeiro-Neto; Baeza-Yates, 1999).

Após apresentarmos as formas de consulta textual que mais se aproximam da nossa proposta, abordaremos em seguida a distribuição e a recuperação de conteúdo em redes P2P, como forma de aplicação das técnicas de recuperação de informação nessas redes.

3.3 Recuperação de Informação distribuída

Os sistemas de recuperação de informação distribuída - RID geralmente consistem de um conjunto de processos cliente, um conjunto de processos servidores, cada um executando em um nó de processamento separado, e um processo *broker* responsável por aceitar requisições de clientes, por distribuir a requisição do cliente entre os servidores, coletando o resultados intermediários dos servidores e consolidando em um resultado final para o cliente (Ribeiro-Neto; Baeza-Yates, 1999).

O particionamento da coleção de documentos no sistema RID influi na forma como as consultas são realizadas. As coleções podem ser distribuídas entre os servidores de forma aleatória ou por um particionamento semântico. Na distribuição aleatória, a consulta deve ser enviada a todos os servidores para que eles processem a consulta na sua coleção. No particionamento semântico, deve existir um algoritmo que faça uma consulta prévia para identificar qual coleção (ou coleções) deverão ser consultadas (Ribeiro-Neto; Baeza-Yates, 1999) em detrimento de outras.

O protocolo para transmitir as consultas e receber os resultados precisa definir a sintaxe e a semântica das mensagens trocadas entre o cliente e o servidor (Ribeiro-Neto; Baeza-Yates, 1999). A definição envolve a escolha de um ou mais protocolos de comunicação, como por exemplo o TCP/IP.

O protocolo deve (Ribeiro-Neto; Baeza-Yates, 1999):

- obter informações sobre os servidores, incluindo a lista das coleções disponíveis;
- submeter a consulta para uma ou mais coleções usando uma linguagem de consulta bem definida;
- receber os resultados da consulta formatados;
- recuperar os itens identificados no resultado da consulta.

Uma vez que os resultados sejam encontrados pelos servidores, eles precisam ser enviados ao cliente. O resultado final deve ser obtido com a junção dos resultados parciais. Uma das melhores formas é ordenar o resultado pela sua relevância em relação a consulta feita.

Sistemas RID têm semelhanças com sistemas P2P utilizados para distribuição de conteúdo. Ambos têm como finalidade a localização de informações (e documentos) em coleções de documentos que estão a disposição dos usuários. Do ponto de vista da arquitetura, também são semelhantes porque existe um cliente que faz a consulta, servidores que processam a consulta e encaminham o resultado de volta.

Os sistemas RID e P2P possuem diferenças; nos sistemas P2P, os *peers* operam como servidores e como clientes (*servents)* e podem entrar e sair da rede sem prévio aviso, tornando a tarefa de consulta mais complexa. Em sistemas P2P, não há a figura do *broker*, presente nos sistemas RID. Em redes P2P, o próprio *peer* se encarrega de receber e ordenar o resultado da

consulta, recebido de outros *peers* da rede.

Na seção seguinte, comentaremos sobre a distribuição de conteúdo e recuperação de informação em redes P2P.

3.4 Distribuição de conteúdo e Recuperação de informação em redes P2P

Apresentados os conceitos e técnicas de recuperação de informação que são úteis para a nossa proposta, comentaremos nessa seção alguns aspectos da distribuição de conteúdo, especialmente em redes P2P.

O paradigma das redes P2P foi inicialmente utilizado para serviços de compartilhamento de arquivo, como o Napster e o Gnutella. Entretanto, as redes P2P representam também uma alternativa importante para a distribuição de conteúdo (Hamra; Felber, 2005).

Na distribuição de conteúdo em redes P2P, a exemplo dos sistemas de recuperação de informação, o desafio é predizer quais *peers* detêm documentos que satisfaçam a consulta, de forma rápida e consumindo menos banda de rede possível. Assume-se que poderão existir várias coleções de documentos espalhadas pelos *peers*. Assim, para encontrar um certo documento, será necessário identificar o *peer* (ou *peers)* que detém tal documento na sua coleção local. Esse passo adicional é realizado por protocolos de redes P2P, e, de forma similar, por sistemas distribuídos de recuperação de informação.

Os *peers* precisam trocar mensagens entre si e consultar suas coleções para que o usuário solicitante possa receber a resposta à consulta realizada. Essa troca de mensagens pode ser intensa, como nos casos das redes P2P que geram inundação de tráfego. Isso ocorre por causa da forma como a rede P2P é organizada e porque os *peers* não sabem previamente onde estão os documentos que satisfazem a consulta.

Um sistema P2P adaptável pode suportar consultas ricas para um amplo conjunto de aplicações. Algumas aplicações para atingir o seu objetivo podem utilizar consultas simples, baseadas em palavras-chaves e palavras simples; já outros, com requisitos mais elaborados, requerem linguagens de consulta mais complexas (*SQL-like)* para encontrar documentos com múltiplas palavras-chave, ou para unir resultados oriundos de relações distribuídas (Risson;

Moors, 2004).

Conforme apresentado no capítulo anterior, o processo de recuperação de informação requer que, inicialmente, a coleção de documentos seja preparada para as consultas que serão feitas sobre ela. Os tipos de consultas são dependentes dos modelos de recuperação de informação utilizados. Um passo essencial na preparação da coleção é a construção dos índices para acelerar a recuperação do resultado das consultas. Uma vez terminada a construção dos índices, consultas podem ser feitas sobre a coleção.

Em redes P2P, quanto a localização dos índices, podem existir três tipos: locais, centralizados e distribuídos. Um *peer,* utilizando apenas índices locais, não recebe referências de índices de outros *peers.* Com índices centralizados, um servidor (ou *cluster)* mantém as referências para os documentos de vários *peers*. Já com índices distribuídos, ponteiros apontam para *peers* onde estarão os documentos (Risson; Moors, 2004).

As referências (Stephanos; Diomidis, 2004) e (Risson; Moors, 2004) apresentam várias comparações e classificações de trabalhos na área de distribuição de conteúdo em redes P2P, tratando da criação de índices livres ou não de semântica, e dos tipos de pesquisa: por faixa, com múltiplos atributos, por junção e por agregação (SUM, Average etc.). Esses recursos, apesar de não estarem implementados em programas P2P populares, aumentam a robustez funcional das consultas nessas. As melhorias propostas naqueles trabalhos buscam redes P2P mais escalares ou sugerem ajustes nas redes P2P existentes, através de um desenho melhor e que reflita em menor complexidade computacional para as operações básicas de uma rede P2P, para as operações de indexação e consultas. As melhorias na parte de consulta são em relação ao roteamento de mensagens e a redução do consumo de banda.

Entretanto, os pontos fracos também existem. Ainda não se consegue uma consulta rica em recursos e que seja de fácil utilização para os usuários; ainda não se consegue alcançar uma grande quantidade de *peers* sem comprometer o tempo de resposta de uma consulta; ainda não se consegue consultas abertas sem interferir na quantidade de resultados e sem consumir muita banda da rede.

Ao nosso ver, a maior preocupação com o desempenho parece ter deixado para um segundo plano as demais funcionalidades que envolvem as redes P2P e a recuperação de informação, por exemplo com as consultas textuais, que são tão utilizadas na *web.*

3.5 Outras soluções de localização de conteúdo

Nas nossas pesquisas encontramos referências de soluções que realizam distribuição de conteúdo em redes P2P. As subseções a seguir são dedicadas a elas. O primeiro grupo de soluções são os produtos tipo "*desktop search*". Uma outra solução, semelhante a nossa proposta tanto em termos de arquitetura quanto ao funcionamento é o PlanetP. Por fim, comentaremos também sobre o LionShare, software P2P utilizado para distribuição de conteúdo em ambiente acadêmico.

3.5.1 Desktop Search

No que tange a indexação e consulta de arquivos, soluções como o Google Desktop Search (GDS, 2006), Windows Desktop Search (WDS, 2006), Yahoo Desktop Search (YDS, 2006), Copernico Desktop Search (CDS, 2006) e Ask Desktop Search (ADS, 2006) têm finalidades comuns: indexar os arquivos do computador e assim realizar consultas sobre esses arquivos de forma mais rápida do que procurando diretamente no disco, no momento da consulta.

Nessas soluções, a indexação dos arquivos é feita em segundo plano, quando o computador fica ocioso (GDS, 2006). Uma vez criados os índices, as consultas sobre o conteúdo dos arquivos são então realizadas sobre os índices, permitindo assim uma redução importante no tempo total da consulta.

Outra característica dessas soluções é a capacidade de indexar arquivos de diversos tipos, como arquivos DOC, XLS, PPT, PDF, RTF, XML, TXT, e-mails etc. A indexação de arquivos que trazem no seu conteúdo informações de controle requer que o motor de indexação esteja preparado para separar *tags* e ou metadados do conteúdo. Assim, as soluções dispõem de *crawlers* para os vários tipos de arquivos. Os *crawlers* são rotinas que conhecem a estrutura interna do arquivo e conseguem separar o que é informação do usuário e o que é campo de controle (Hamra; Felber, 2005).

Uma vez que o objetivo das soluções de *desktop search* é a indexação dos arquivos locais e assim realizar consultas mais rápidas, não encontramos registro de algum sistema P2P

que faça uso de tais índices para acelerar suas buscas. Também não encontramos documentação sobre a estrutura de índices utilizadas por tais produtos. Se a estrutura dos índices fosse aberta ou o código da API de manipulação fosse livre e documentada, outras aplicações poderiam fazer uso dela. Isto permitiria que, por exemplo, a nossa arquitetura viesse a utilizar tal estrutura de índices.

Por outro lado, o Google Desktop Search, ilustrado na figura 3.5, criou[28] a possibilidade de uma consulta ser encaminhada a outro computador do mesmo usuário. Para que a consulta ocorra, o outro computador deve ter instalado e a opção "Pesquisar em outros computadores" precisa estar ligada.

Figura 3.5 - Google Desktop Search.

Existem algumas diferenças entre a nossa proposta e as soluções *Desktop Search*. Nossa proposta é essencialmente para utilização em rede P2P, enquanto as soluções *desktop search* foram projetadas para o funcionamento local (*stand-alone)*. Quanto a indexação, propomos indexar arquivos textuais da pasta compartilhada pelo usuário, ao passo que àquelas soluções podem indexar todo o conteúdo do(s) disco(s) rígido(s) local(ais) e com suporte a mais tipos de arquivos.

Na subseção seguinte, apresentaremos uma arquitetura semelhante a nossa proposta, o PlanetP.

[28] http://desktop.google.com/pt/BR/features.html#searchremote, acesso em abril de 2006.

3.5.2 PlanetP

O PlanetP (Cuenca-acuna et al., 2001) é uma infraestrutura P2P para o compartilhamento e localização de arquivos, pelo seu conteúdo. Ele foi projetado para suportar a inclusão e a remoção de nós na rede como também a existência de cópia de arquivos/documentos em mais de um nó.

A arquitetura do PlaneP é composta por: um diretório local, um *Brokerage Server* e *XML Store.* O diretório local é uma lista de todos os membros com os seus filtros sumarizados e algumas informações sobre o *peer.* O *Brokerage Server* implementa o serviço de corretagem e de distribuição de informação na rede. O *XML Store* mantém em memória uma tabela *hash* para os XML-*snippets* (Cuenca-acuna et al., 2001), que possuem metadados adicionais utilizados nas consultas. A figura 3.6 apresenta a arquitetura do PlanetP.

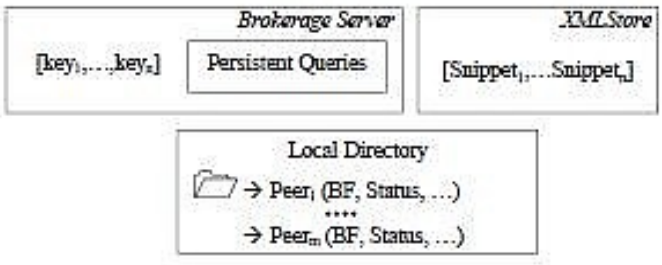

Figura 3.6 - Arquitetura PlanetP. *Fonte (Cuenca-acuna et al., 2001).*

Cada nó PlanetP cria e mantém um sumário do conteúdo indexado dos demais nós da rede (Cuenca-acuna et al., 2001). A distribuição do sumário de um nó para os demais não se dá por inundação, mas por uma espécie de "fofoca" na rede. O algoritmo funciona assim: um nó A escolhe aleatoriamente um outro nó B da rede para enviar seu sumário; quando o nó B recebe a mensagem de A, ele verifica se A "sabe" algo que ele não sabe; se sim, B atualiza o seu diretório local. Isso faz com que cada *peer* tenha como saber, em tempo de consulta local, qual *peer* detém a informação desejada.

As consultas suportadas pelo PlanetP são básicas: o usuário informa uma sequência de palavras-chave, separadas por espaços em branco. O nó consulta os filtros locais (bloom

filters[29]) para identificar quais *peers* são candidatos a possuir a informação desejada. Alguns *brokers* também podem ser contactados e retornam URLs que apontam para onde encontrar o dado procurado. Após a obtenção da lista dos *peers* candidatos, uma mensagem específica de consulta é encaminhada ao nó detentor, solicitando o arquivo.

As consultas são persistentes na rede PlanetP. Cada consulta feita fica armazenada. Cada novo filtro que chega ao nó é checado em relação as consultas armazenadas. Havendo batimento, uma mensagem é enviada ao nó (que realizou a consulta) avisando que um novo arquivo que satisfaz a uma certa consulta foi adicionado a rede (Cuenca-acuna et al., 2001).

Em um trabalho mais recente (Cuenca-Acuna; Nguyen, 2001), os pesquisadores utilizaram a infra-estrutura PlanetP com a técnica de espaço vetorial para prover uma localização de conteúdo mais precisa, juntamente com a regra para ranking TFxIDF.

A seguir, apresentaremos uma outra proposta para distribuição de conteúdo em rede P2P, o LionShare.

3.5.3 LionShare

O LionShare (LionShare, 2004) é um projeto da Universidade da Pensilvânia, nos Estados Unidos. O objetivo principal do LionShare é usar uma rede P2P para fomentar a interação entre alunos, pesquisadores e universidades para o compartilhamento de conhecimento.

A criação do protótipo inicial foi feita com base na versão 4.0 do LimeWire, mas com outros objetivos e funcionalidades que não são realizados pela rede Gnutella. As funcionalidades adicionais representam as principais características desse produto. Entre elas, podemos destacar (LionShare, 2004):

- Rede P2P privativa: as redes LionShare são confinadas em federações. Pode haver interação entre federações. Apesar de ser construído sobre a rede Gnutella, outros artifícios de segurança lógica foram incorporados ao produto, como por exemplo: mecanismos de autenticação, credenciais para os usuários e controle de acesso aos arquivos.

[29] Bloom Filters são arrays de bits, calculados por um mais funções *hash,* que indexam o conteúdo de um certo *peer.*

- PeerServer: *peer* especial para persistência de certos arquivos e administração centralizada de alguns aspectos da rede.
- Mecanismos e colaboração embutidos.
- Pesquisa em repositórios externos a rede LionShare.

A rede contempla dois tipos de peer: *peer* e *peerserver.* Arquitetura da rede LionShare pode ser considerada híbrida porque apresenta alguns aspectos de distribuição total e cliente-servidor. A estrutura da rede pode ser considerada distribuída no sentido em que cada *peer* pode fazer acesso a outro, sem a intervenção de uma entidade centralizada. Entretanto, com a presença dos *peerservers,* desempenhando algumas operações na rede, pode-se dizer existe uma interação cliente-servidor, entre eles e os *peers.*

O *peerserver* que permite a administração da rede presta serviços de autenticação e também armazena arquivos como uma biblioteca digital para os *peers*. Quando há necessidade de interação entre duas instituições (federações, nos termos do LionShare), o *peerserver* ativa o *trust fabric* e faz a ponte lógica entre as duas redes, deixando a comunicação transparente para os demais *peer* (LionShare, 2004). A figura 3.7 ilustra a estrutura dessa rede.

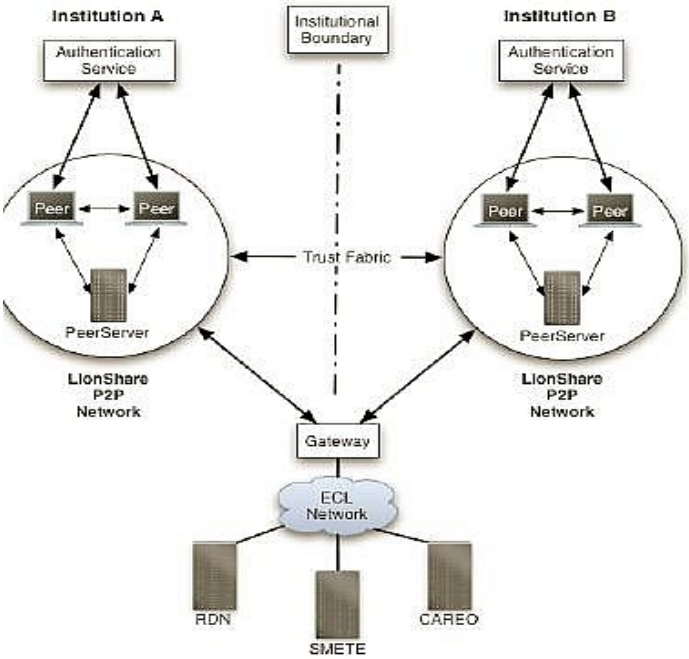

Figura 3.7 - rede LionShare. *Fonte (LionShare, 2004).*

Outra característica importante do LionShare é a sua capacidade de consultar outras fontes de recursos, como os vastos repositórios das instituições afiliadas ao projeto[30]. Nas consultas, o usuário pode escolher qual fonte ou repositórios utilizar (inclusive externos), o tipo de mídia e seus metadados associados (LionShare, 2004).

O modelo de segurança definido para o LionShare trata da autenticação, autorização e controle de acesso, e é baseado na reputação mútua entre os *peers* envolvidos no compartilhamento e no consumo de recursos (LionShare, 2004).

A autenticação de usuários é feita com a instituição a qual o usuário pertence[31], através de uma infra-estrutura de chaves públicas (PKI), onde cada *peer* deve possuir dois certificados que devem garantir a sua privacidade, são eles: certificados cliente e servidor. Ambos os certificados têm informações sobre a instituição de origem.

O certificado servidor é usado para o compartilhamento de arquivos. O certificado cliente permite que o *peer* solicite arquivos de outros *peers* (LionShare, 2004). A autorização de acesso aos recursos compartilhados depende da concordância do *peer* dententor do recurso. Desta forma, um *peer* de uma rede pode não ter acesso a um certo arquivo, caso o *peer* detentor assim deseje.

O controle de acesso, em complemento a autorização, pode ser feito por arquivo compartilhado e por atributos de acesso que o requisitante deve possuir. Por exemplo, certo arquivo poderia estar disponível para alguns usuários da federação nativa mas não para os usuários de outra federação. A lista dos atributos de acesso é chamada *Access Control List – ACLs* e fica armazenada no próprio *peer.* As ACLs nunca são enviadas pela rede (LionShare, 2004).

3.6 Sumário do capítulo

Nesse capítulo, comentamos as linhas gerais do processo de recuperação de informação e suas etapas, desde a identificação da coleção de documentos, sua preparação, indexação e os mecanismos de consulta. Detalhamos as principais etapas deste processo apresentando as

[30] Através do conector *Edusource Communications Layer – ECL,* uma vez que cada instituição pode ter seu próprio mecanismo de segurança. Uma vez autenticados, os usuários podem ter acesso aos repositórios Careo, RDN e Smete. Em (Gospodenetic; Hatcher, 2005), há uma seção explicando como acontece a integração do LionShare com o conector ECL.

[31] Uma vez que a rede LionShare permite que outras instituições participem da rede.

técnicas utilizadas que mais se aproximam da nossa proposta.

Em seguida, apresentamos os tipos de consultas textuais e alguns aspectos da distribuição de conteúdo em redes P2P. Finalizando o capítulo, investigamos algumas soluções de localização de conteúdo agrupadas em: localização de conteúdo local (*desktop search)* e em rede P2P, como as propostas PlanetP e LionShare.

No capítulo seguinte, apresentaremos a nossa proposta. Ela reuni aspectos de recuperação de informação e de redes P2P, com o objetivo de melhorar a localização documentos textuais pelo o seu conteúdo.

Capítulo 4 – Arquitetura proposta

Nesse capítulo apresentamos uma proposta de arquitetura para realizar a localização de documentos textuais pelo conteúdo em redes P2P.

4.1 Introdução

Neste capítulo apresentaremos a arquitetura que propomos para realizar a localização de documentos textuais em redes P2P, baseada no conteúdo dos mesmos (Farias e Furtado, 2005a) (Farias e Furtado, 2005b). Nas seções a seguir, contextualizaremos o problema, explicaremos os detalhes da arquitetura, o seu funcionamento e alguns cenários de utilização.

As redes P2P provêem um ambiente onde cada *peer* publica automaticamente quais arquivos deseja compartilhar, tão logo se conecte. Programas que implementam redes P2P como o Napster[32], LimeWire[33] e Emule[34], têm tido grande sucesso ao permitir a troca de arquivos na Internet. A principal motivação para o uso desses programas foi a troca de arquivos de música entre os usuários.

Arquivos de música e software são facilmente identificados por seus títulos e podem ser compartilhados por dezenas de *peers* que constituem a rede P2P. Os usuários são motivados a fazer o compartilhamento de arquivos, especialmente de música, porque: existe software gratuito de conversão para o formato MP3, o espaço de armazenamento está limitado a capacidade do *peer*, a oferta de acesso à Internet em banda larga tem crescido em nível mundial e há facilidade para os usuários utilizarem os programas P2P.

Problemas de direitos autorais surgiram com a criação e cópias não autorizadas de arquivos de música. A indústria fonográfica, por força judicial, conseguiu suspender o funcionamento do Napster, em meados de 2000, porque era possível identificar os arquivos compartilhados dos *peers*. Entretanto, outros tipos de arquivos poderão existir nos *peers*, arquivos que houvesse interesse no compartilhamento e que não tenham problemas com

[32] Http://www.napster.com
[33] http://www.limewire.org
[34] http://www.emule-project.net

direitos autorais com a reprodução não autorizada.

Para que um usuário possa "baixar" um arquivo de outro *peer,* é necessário que ele encontre o arquivo desejado na rede P2P. Nessas redes, os *peers* podem entrar e sair sem prévio aviso ou compromisso de permanência. Isso faz com que o resultado de uma consulta varie com o tempo, ou seja, se um usuário realizar uma consulta em um certo horário não há garantia de que o resultado de uma nova consulta, igual a primeira, seja o mesmo se esta for feita, minutos depois.

Na seção seguinte apresentaremos o problema da localização de arquivos em redes P2P que escolhemos pesquisar.

4.2 O problema

Como localizar os documentos de um *peer,* independente do nome do arquivo? Como encontrar documentos pelo o seu conteúdo e não apenas pelo nome (ou parte) nessas redes? O escopo do problema envolve a identificação do(s) peer*(s)* que detém documentos (textuais) que possam satisfazer o critério de pesquisa informado pelo usuário, através da interface do software P2P. Propomos a adoção do recurso de indexação dos documentos para que os *peers* possam responder a consultas feitas sobre o conteúdo de seus documentos compartilhados.

A busca textual tem se mostrado suficiente quando o nome do arquivo, ou trechos de sua descrição, são amplamente conhecidos e caracterizam bem o conteúdo. Por exemplo, consultando "yanni tribute" na rede Gnutella, através do LimeWire, recebemos a resposta de vários *peers* que detém arquivos cujos nomes satisfazem a tal critério de pesquisa. Entretanto, a consulta pelo nome do arquivo não é apropriada quando o critério é, por exemplo, "aula1.txt", o qual não indica a natureza do seu conteúdo.

Nos parece razoável considerar que no conteúdo "não-*web*" da Internet possa haver documentos de interesse para comunidades de usuários, haja visto os tipos de documentos desconsiderados pelos mecanismos de busca da *web* e a existência de várias redes P2P. Por exemplo, um grupo de professores que ministrem uma certa disciplina em escolas ou universidades poderão ter um conjunto de documentos, apresentações etc. que interessem aos demais membros dessa comunidade.

Não encontramos registro na documentação dos programas P2P[35] relativo a capacidade dos *peers* de realizarem consultas pelo conteúdo de arquivos textuais. Uma causa para a ausência dessa capacidade, na nossa visão, seria porque os *peers* preferencialmente detém arquivos tipo multimídia, que dificultam o mapeamento entre partes do seu conteúdo com parâmetros de consulta informados pelo usuário.

Além disso, consultas encaminhadas aos *peers* que considerem o conteúdo dos documentos compartilhados poderiam implicar em tempo de resposta elevado para a obtenção dos resultados, porque os arquivos dos *peers* teriam de ser vasculhados em tempo de consulta.

Uma alternativa para a obtenção do resultado da consulta através do conteúdo do arquivo seria a utilização de índices textuais e locais, previamente criados e periodicamente atualizados, pelo software P2P. Assim, não seria mais necessário percorrer o conteúdo dos arquivos do *peer* em tempo de consulta para encontrar certo texto em documentos textuais. Isso faria com que o tempo de consulta para este tipo de arquivo reduzisse em relação ao cenário que não utilize os índices.

Outro problema relacionado é fazer com que o resultado apurado pelo *peer* possa ser encaminhado para o *peer* solicitante, através da rede P2P. O tamanho dessa resposta poderia variar muito, principalmente se o usuário utilizasse um valor de pesquisa que implicasse em uma grande quantidade de arquivos, que tivesse tal palavra, dentro da coleção. Além disso, vários outros *peers* da rede, que também receberam a consulta, poderiam ter, em suas coleções locais, arquivos que satisfazem o critério da consulta, aumentando o tamanho da resposta para o *peer* solicitante.

A seguir, descreveremos os requisitos funcionais necessários aos *peers* para dotá-los da capacidade de realizar consultas nos arquivos compartilhados, baseadas no conteúdo.

4.3 Requisitos funcionais

A fim de dotar os *peers* da capacidade de localização de documentos textuais na rede P2P, será necessário estender as suas funcionalidades nativas. Os requisitos funcionais necessários para a nossa proposta envolvem:

a) Utilização (criação e atualização) de índices locais para que possam realizar

[35] LimeWire, Emule, Kazaa e Napster.

consultas rápidas ao conteúdo dos documentos textuais;

b) Realização de consultas textuais nos documentos compartilhados pelo *peer,* quando houver o recebimento de uma mensagem desse tipo;

c) Formatação do resultado da consulta local, contendo a lista dos documentos que satisfazem ao critério de pesquisa recebido*;*

d) Envio do resultado ao *peer* solicitante através da rede P2P;

e) Exibição do resultado da consulta para o usuário do *peer* solicitante;

f) Permitir que o *peer* realize o *download* do arquivo desejado a partir dos *peers* detentores.

Além desses, há um conjunto de requisitos funcionais e não-funcionais inerentes a maioria dos programas P2P já implementados, por exemplo:

- capacidade de se conectar na rede P2P compatível;
- realizar o compartilhamento dos arquivos e recursos com os demais *peers* da rede;
- realizar consultas e lançá-las na rede P2P;
- controlar o recebimento das respostas a essa consulta;
- responder as consultas realizadas por outros *peers;*
- executar arquivos de música;
- segurança lógica do *peer;*
- realizar o *download e upload* de arquivos.

Tais requisitos não se limitam aqui. A depender do software P2P, eles podem crescer bastante. Na seção seguinte, apresentaremos a arquitetura da nossa proposta para atender aos requisitos funcionais enumerados acima.

4.4 Arquitetura proposta

Essa seção apresenta a arquitetura que propomos como uma solução para o problema da localização de documentos textuais em redes P2P. Com base em uma arquitetura P2P genérica, idealizamos a integração com um motor de indexação e busca para dotar o *peer* da

capacidade de localizar documentos da sua coleção pelo conteúdo como também responder a consultas desse tipo, realizadas por outros *peers* da rede.

A arquitetura precisa atender aos requisitos funcionais listados na seção anterior. Utilizaremos a figura 4.1 para ilustrar a proposta.

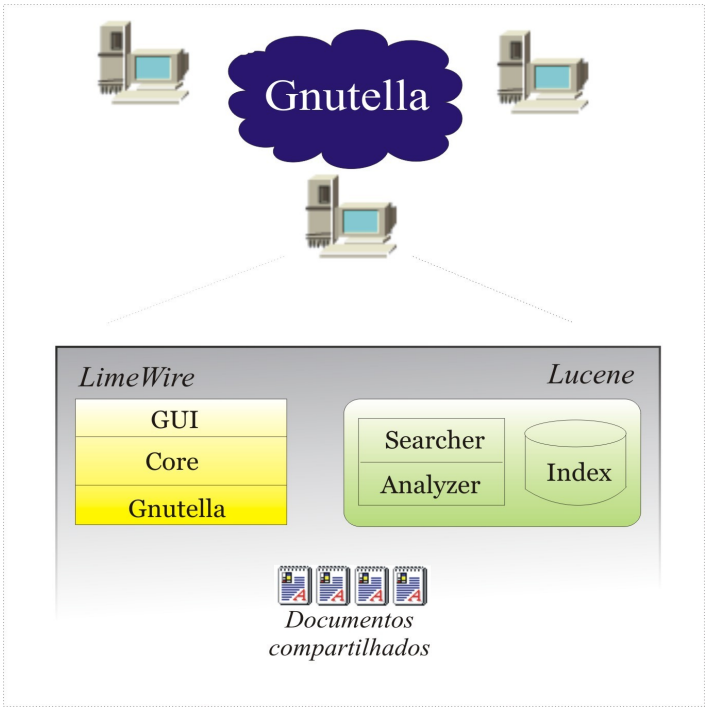

Figura 4.1 - Visão geral da arquitetura proposta.

A arquitetura possui duas partes principais: os componentes do software P2P e os componentes de indexação e consulta aos índices textuais. Essa é a principal característica da nossa proposta: vincular uma máquina de busca a um *servent* afim de dotá-lo da capacidade de localizar e recuperar conteúdo distribuído em uma rede P2P.

Decidimos utilizar o software LimeWire P2P e a API Lucene. As justificativas para a nossa escolha foram baseadas em três aspectos básicos:

a) Robustez funcional: o software P2P escolhido deveria possuir as funções básicas, como conexão e desconexão na rede, envio e recebimento de consultas, suportar *downloads e uploads concorrentes,* realizar consultas etc. Além disso, o *framework* de recuperação de informação deveria ser capaz de indexar arquivos, de tantos tipos o quanto possível, permitir a realização de consultas de vários tipos sobre a coleção

de documentos previamente indexada.

b) Código aberto: uma vez que um protótipo seria construído como prova de conceito da arquitetura proposta, nossa intenção seria aproveitar o máximo de código existente para reduzir o esforço e o tempo na desenvolvimento do protótipo. Assim, produtos de código aberto e bem estruturados facilitariam o trabalho de integração entre eles. Nesse aspecto, encontramos tanto no LimeWire quanto no Lucene, uma documentação e estruturação razoáveis dos seus principais componentes.

c) Linguagem de programação comum: um fator importante é que os produtos deveriam ter a linguagem de programação em comum. Isto iria facilitar a construção do protótipo. No nosso caso, ter os produtos escritos em Java seria um aspecto desejável pois havia interesse em aprofundar os nossos conhecimentos na linguagem.

Durante as pesquisas, encontramos referências para outros produtos e *frameworks* da área de indexação e pesquisa, como também da área de P2P. Alguns exemplos de produtos são: BearShare P2P, o *crawler* ht://Dig[36] e o mnoGoSearch[37].

Tais produtos possuem características importantes mas que contribuiriam menos com a nossa proposta. Por exemplo, o ht://Dig tem maior foco em páginas *web,* que atenderia parcialmente no cenário de troca de documentos em sistemas P2P*; o* mnoGoSearch requer o uso de um servidor de banco de dados padrão SQL para o armazenamento dos índices (Guerreiro; Macedo, 2005), que seria pouco apropriado para o cenário de sistemas P2P.

A opção pelo protocolo Gnutella veio por conseqüência, já que o LimeWire conecta-se nessa rede. Existem restrições técnicas quanto ao uso da rede Gnutella, principalmente porque ela não suporta alto nível de escalabilidade (Jordan, 2001), se considerarmos o tamanho da Internet. Apesar dessa limitação, existem muitos usuários[38] nessa rede (LimeWire, 2006a). Além disso, a simplicidade do protocolo Gnutella e a possibilidade de o utilizarmos com poucas alterações foram atrativos para a nossa proposta.

Desde o seu lançamento até hoje, o Gnutella passou por várias melhorias e talvez a maior delas tenha sido a incorporação dos *ultrapeers*. Tais *peers* promoveram melhoria

---

[36] Http://www.htdig.org

[37] Http://search.mnogo.ru

[38] A rede Gnutella tem tamanho variável, mas é frequente encontramos milhares de *hosts* conectados. Em http://www.limewire.com/english/content/netsize.shtml é possível consultar o tamanho desta rede, em termos de quantidade de computadores.

significativa no consumo de banda com a redução da inundação de tráfego. Em paralelo ao Gnutella, está o Gnutella 2, uma nova versão que pretende resolver alguns problemas do seu antecessor, mas ainda conta com poucos adeptos, em relação a versão anterior.

Por questões de escopo, priorizamos nesse trabalho os aspectos funcionais em detrimento dos não-funcionais, como o desempenho. Existem trabalhos que tratam especificamente da comparação de redes P2P para distribuição de conteúdo, sobre vários pontos relacionados ao desempenho (Hamra; Felber, 2005), (Stephanos; Diomidis, 2004) e (Theodore, 2001).

4.5 Funcionamento da arquitetura

Uma vez apresentada a arquitetura proposta, comentaremos o seu funcionamento ideal (*happy day)*. Consideraremos que dois *peers* estão executando o nosso protótipo e estão conectados à Internet e à rede Gnutella. A figura 4.2 auxilia no entendimento do funcionamento da arquitetura.

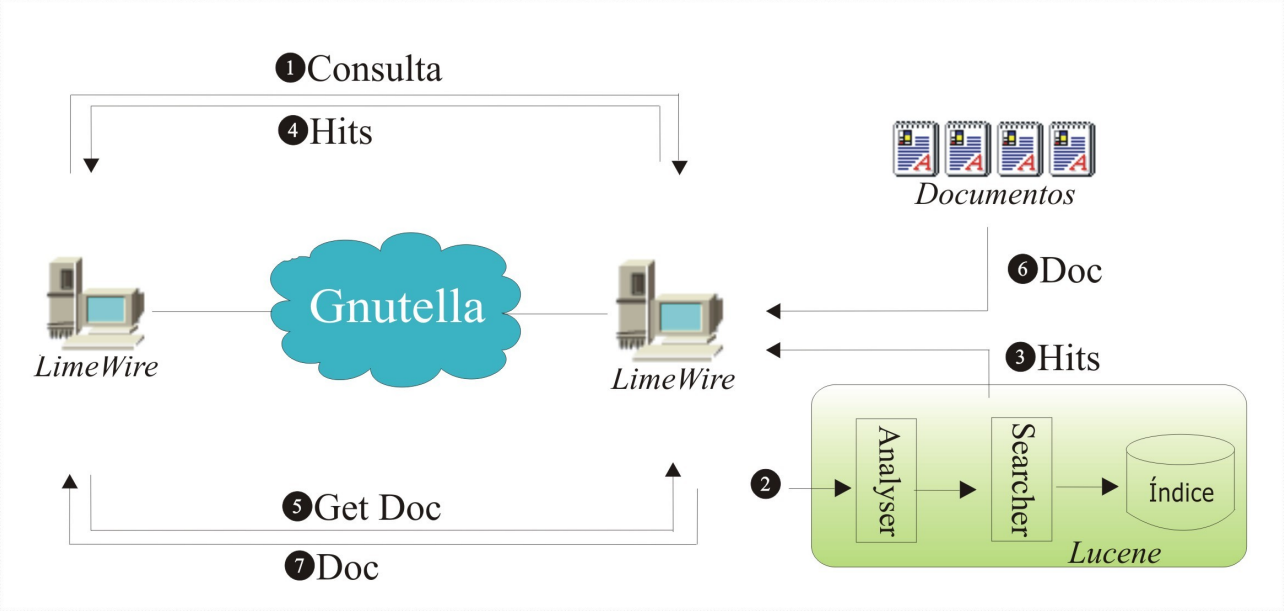

Figura 4.2 - Funcionamento da arquitetura.

Os números na figura 4.2 representam os passos do funcionamento da arquitetura.

1- O usuário do *peer* solicitante submete uma consulta passando o valor de pesquisa

informado pelo usuário, sinalizando que é uma consulta pelo conteúdo dos documentos compartilhados pelo *peer*. A rede Gnutella propaga a consulta entre os *peers*, segundo a especificação do protocolo;

2- A consulta chega a um outro *peer* que identifica (e suporta) o tipo de consulta especial, ou seja, que a consulta se refere ao conteúdo de documentos e não apenas ao nome dos arquivos;

3- O *peer* aciona a camada Lucene e o índice textual local, que já deve estar criado a esse momento, é consultado com o valor da consulta informado pelo usuário do *peer* remetente. A lista dos resultados obtidos na coleção local (*hits)* é montada;

4- O resultado (*hits*) é passado ao *peer* solicitante através de mensagem específica, encaminhando os dados básicos dos arquivos;

5- O usuário do *peer* solicitante, ao receber os *hits* dos *peers* envolvidos na consulta e que tenham arquivos que satisfaçam o critério, apresenta então a lista para o usuário que seleciona um arquivo, encaminhando o pedido do *download* para o *peer* detentor do arquivo;

6- O arquivo é lido da pasta compartilhada pelo *peer* detentor;

7- A transferência do arquivo inicia.

Comentaremos nas seções a seguir mais detalhes do LimeWire e do Lucene. Já os detalhes da implementação do protótipo estão no capítulo seguinte.

4.6 LimeWire

O LimeWire é um software P2P de código livre licenciado pela GNU - *General Public License* e escrito na linguagem de programação Java, podendo rodar em diversos sistemas operacionais. Está disponível gratuitamente para *download* e possui uma versão paga, o LimeWire PRO, que oferece algumas funcionalidades adicionais como *downloads* mais velozes e resultados de consulta mais completos. A última versão disponível do LimeWire é a 4.12.6.

A LimeWire LCC disponibiliza no site http://www.limewire.org o código do software para que possa ser modificado por qualquer programador. No site também estão disponíveis a documentação técnica do programa, fóruns, manuais de instalação do código e *wikis*.

O LimeWire (LimeWire, 2006a) tem despertado interesse de usuários de sistemas P2P nos *rankings* populares de *download*. Ele reuni várias funcionalidades úteis para a nossa proposta. A figura 4.3 ilustra a interface gráfica do LimeWire quando ele está em execução.

Comentaremos algumas características do produto para o melhor entendimento da nossa proposta.

Conexão em rede. O LimeWire usa o protocolo Gnutella versão 0.6. Há uma série de melhorias na versão 0.6 em relação a versão 0.4, principalmente em relação a performance e consumo de banda de rede, reduzindo o número de *peers* e tornando mais escalar, com a categorização de alguns *peers* em *ultrapeers.*

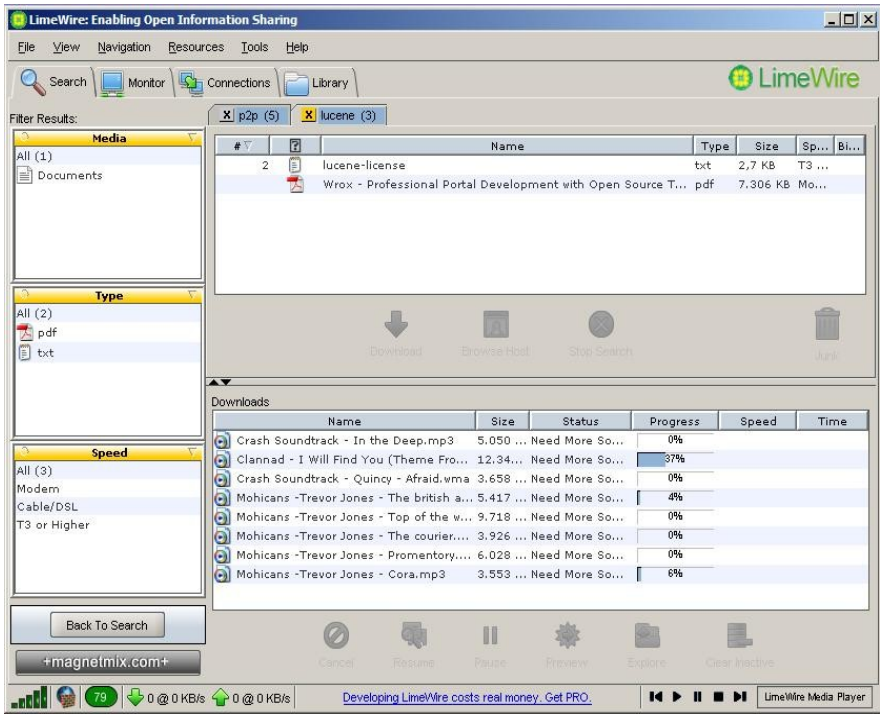

Figura 4.3 - LimeWire em execução.

Consulta sobre os arquivos. O LimeWire implementa o protocolo *Query Routing Protocol – QRP,* apresentado no capítulo 2. Esse protocolo tem o objetivo de apontar em qual *peer* (ou *peers)* está o arquivo que satisfaz o critério de busca. Para isso, o QRP utiliza-se da *Query Routing Table-QRT.* Essa tabela reuni as palavras dos nomes, submetidos a uma função *hash,* dos arquivos conectados aos *ultrapeers.* No peer, existe um outro algoritmo para fazer o casamento de *strings* e *substrings* dos nomes dos arquivos e do parâmetro de consulta,

rapidamente. O algoritmo utiliza uma árvore dos sufixos ordenados dos nomes dos arquivos e sobre ela há a comparação com o parâmetro de pesquisa (LimeWire, 2006b).

Agrupamento dos resultados de consultas. Outro algoritmo implementado pelo LimeWire é o de agrupamento do resultado da pesquisa. Para o LimeWire, arquivos são similares (e agrupáveis) se seus nomes não diferem em mais do que 5% (LimeWire, 2006b). Esse recurso é interessante porque permite ao usuário ver os arquivos mais populares, ou seja, presentes em vários *peers*. Assim, quando o usuário seleciona um arquivo presente em vários *peers,* o mesmo poderá ser baixado de mais fontes e em blocos, tornando o *download* mais rápido.

Transferência de arquivos. O LimeWire possui duas classes gerais para gerenciar as operações de *download e upload:* o *DownloadManager e o UploadManager.* Elas permitem realizar transferências em blocos do arquivo e de diferentes *peers*[39]*,* tornando-as mais rápidas. Todas as transferências são realizadas via HTTP, com ou sem a presença de *firewall* entre os nós. Em situações onde alguns nós estão atingiram o limite máximo de transferências concorrentes, o nó solicitante recebe uma mensagem HTTP 503 (ocupado) e persistindo a requisição, a mesma poderá ser colocada em fila (Klingberg; Manfredi, 2002).

Operação através de *firewalls*. Uma importante requisito para os atuais programas P2P de compartilhamento de arquivos é conseguir transferir arquivos através de um *firewall.* É comum a utilização de *firewall* local[40] nos computadores que têm acesso à Internet. Entretanto, eles podem negar certos pedidos de conexão, por porta ou por rede IP. Nesses casos, não será possível conectar com um *peer.* Assim, é comandado ao *peer* detentor do arquivo que o mesmo o envie para o *peer* solicitante, através da mensagem Gnutella Push (Klingberg; Manfredi, 2002). As transferências de arquivo são realizadas através do protocolo HTTP, geralmente liberado pelo *firewall* porque é o protocolo de navegação na *web.*

Reprodução de arquivo de música. O LimeWire permite a reprodução de arquivos de música no formato MP3. Isso permite que o usuário possa baixar um certo arquivo e durante o *download* já ouvir parte do arquivo e decidir se continuará ou não com a transferência. Os demais formatos de arquivos são encaminhados aos *players* nativos instalados no nó.

Nas subseções seguintes, apresentaremos mais detalhes sobre a arquitetura do LimeWire.

---

[39] A técnica é chamada *swarming.*

[40] Os sistemas MS Windows XP e Linux já incorporam *firewalls* locais para proteção do sistema.

4.6.1 Mensagens e conexões

O LimeWire possui um conjunto de classes que representam as mensagens do protocolo Gnutella. A classe abstrata *Message* representa uma mensagem Gnutella genérica enquanto as suas subclasses *QueryRequest, QueryReply, PingRequest, PingReply* e *PushRequest*, representam as mensagens Gnutella *Query*, *QueryHit*, *Ping*, *Pong* e *Push,* respectivamente. A figura 4.3 ilustra.

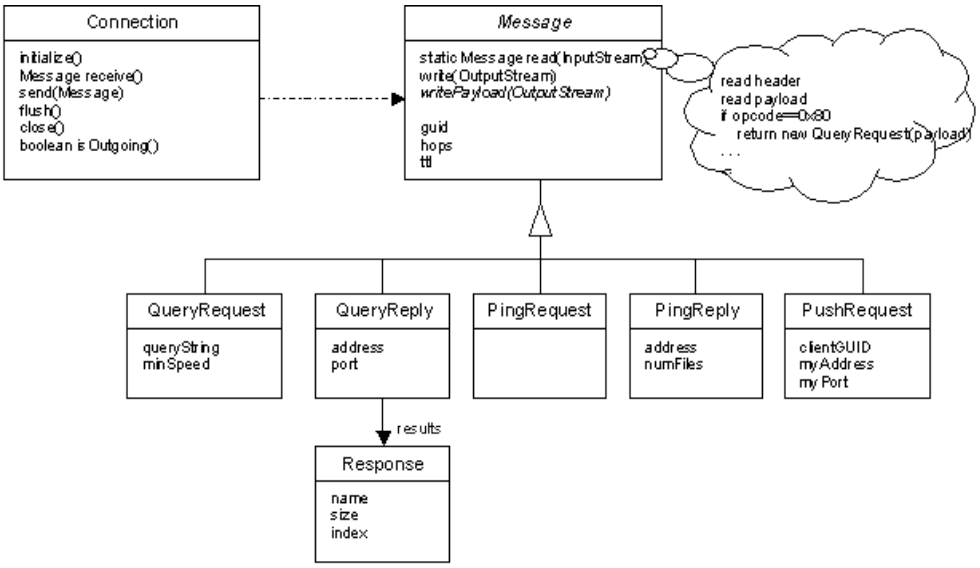

Figura 4.4 – Hierarquia de classes de mensagens e conexões da rede Gnutella. *Fonte:* *http://www.limewire.org/techdocs/design.html*

A classe *Connection* implementa uma conexão à rede Gnutella. Ela possui dois construtores, um para recebimento de mensagens, recebendo um endereço e porta; e outro para envio de mensagens recebendo um *Socket* criado por um *ServerSocket*. *Connection* também possui métodos de leitura e escrita de mensagens que são delegados para *Message.read* e *Message.write*, respectivamente. Na figura 4.4 mostramos a hierarquia de classes descritas. Maiores detalhes sobre tais classes podem ser obtidos em (LimeWire, 2006b).

4.6.2 Encaminhamento de mensagens

Nesta subseção descreveremos as classes que implementam o encaminhamento de mensagens. A classe abstrata *Connection,* comentada na subseção anterior, é implementada pela subclasse *ManagedConnection* que adiciona as seguintes funcionalidades:

- *Buffering* e *flushing* automático para mensagens de saída. As mensagens são encaminhadas na seguinte ordem: *Ping*, *Pong*, *Query*, *QueryHit* e *Push*;
- Método *loopForMessage* para a realização de leitura automática de mensagens, funciona através da *thread* da conexão;
- Filtro de spam implementado por subclasses de *SpamFilter* (como a *DuplicateFilter* e a *CompositeFilter*);
- Dados estatísticos como o número de *bytes* lidos.

A classe *ManagedConnection* é instanciada através de métodos de *factory* da classe *ConnectionManager* que mantém uma lista de instâncias desta classe. *ConnectionManager* é responsável pelo encaminhamento de mensagens de saída. Ela recupera endereços providos pela classe *HostCatcher* que dispara mensagens (através de *ConnectionManager*) para os servidores de *Pong* como o router.limewire.com. A classe *ConnectionWatchDog* busca conexões possivelmente inativas, matando as conexões que não respondem a um *Ping* com o TTL=1.

A classe *MessageRouter* é responsável pelo encaminhamento de mensagens. Cada mensagem lida pelo método *loopForMessage* da classe *ManagedConnection* é passada para uma instância de *MessageRouter* por um seus métodos *handle* (*handleQueryMessage(), handleQueryReply()*, etc). *MessageRouter* monitora *queries* e *pings* de todas as conexões realizadas por *ConnectionManager* e monta uma tabela de *reply* por meio de três instâncias da classe *RouteTable,* afim de que as repostas às *queries* sejam encaminhadas para a conexão correta.

Para responder as mensagens recebidas pelo *handle* (*queries* e *pings*) a classe *MessageRouter* possui dois métodos: *respondToQueryRequest()* e *respondToQueryReply()*. Esses métodos são implementados pela subclasse *StandardMessageRouter* que delega funções à classe *FileManager*.

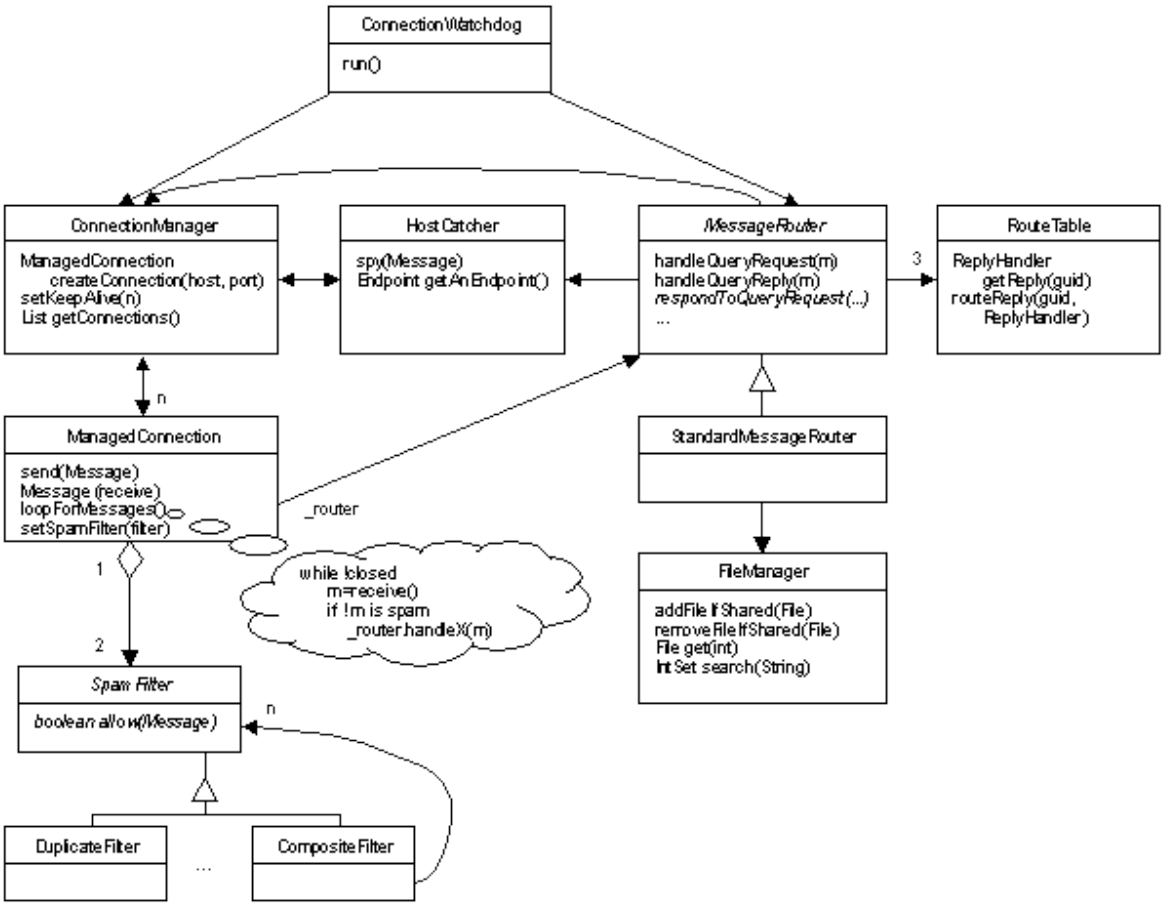

Figura 4.5 – Hierarquia de classes de encaminhamento de mensagens no LimeWire. *Fonte:* http://www.limewire.org/techdocs/design.html

A classe *FileManager* usa estruturas de dados para implementar *queries* eficientemente. Esta classe foi uma dos pontos-chaves na implementação do nosso protótipo. A figura 4.5 ilustra a hierarquia de classes desta subseção. Maiores detalhes sobre tais classes podem ser obtidos em (LimeWire, 2006b).

4.6.3 Uploads e Downloads

A classe *Acceptor* é responsável pela criação do *server socket* e monitoramento de conexões TCP. Caso o programa já possua uma quantidade suficiente de mensagens de conexão, a classe *ConnectionManager* instancia a classe *RejectConnection* ao invés da classe

ManagedConnection.

A *RejectConnection* estende a classe *Connection* mas instancia uma *thread* que espera por uma mensagem *Ping*, responde com mensagens *Pong,* vindas do *HostCatcher,* e realiza a desconexão. As instâncias de *RejectConnection* não são colocadas na lista de conexões da classe *ConnectionManager*.

A classe *DownloadManager* mantém uma lista dos *downloads* de arquivos. Cada *download* é implementado por uma instância única de *ManagedDownloader* que implementa a interface *Downloader*. *ManagedDownloader* provê a realização de *downloads* de uma lista de locais. *Downloads* realizados de um único local são feitos através de uma única instância da classe *HTTPDownloader*, caso contrário, *ManagedDownloader* fará referência a múltiplas instâncias de *HTTPDownloader*. *DownloadManager* usa a classe *FileManager* para detectar se o *download* de um determinado arquivo já foi realizado e para compartilhar o arquivo com outros usuários, após o *download*.

A classe *UploadManager* mantém uma lista de *uploads* de arquivos. Estes são implementados por uma instância única da classe *HTTPUploader* que implementa a interface *Uploader*. O estado do *upload* (ativo, pausado, cancelado, arquivo não encontrado, necessita de mais fontes) são representados pela classe *UploadState* que possui implementações para representar cada tipo de estado como *NormalUploadState* e *FileNotFoundUploadState*. Na figura 4.6 mostramos a hierarquia de classes descritas nesta seção. Maiores detalhes sobre tais classes podem ser obtidos em (LimeWire, 2006b).

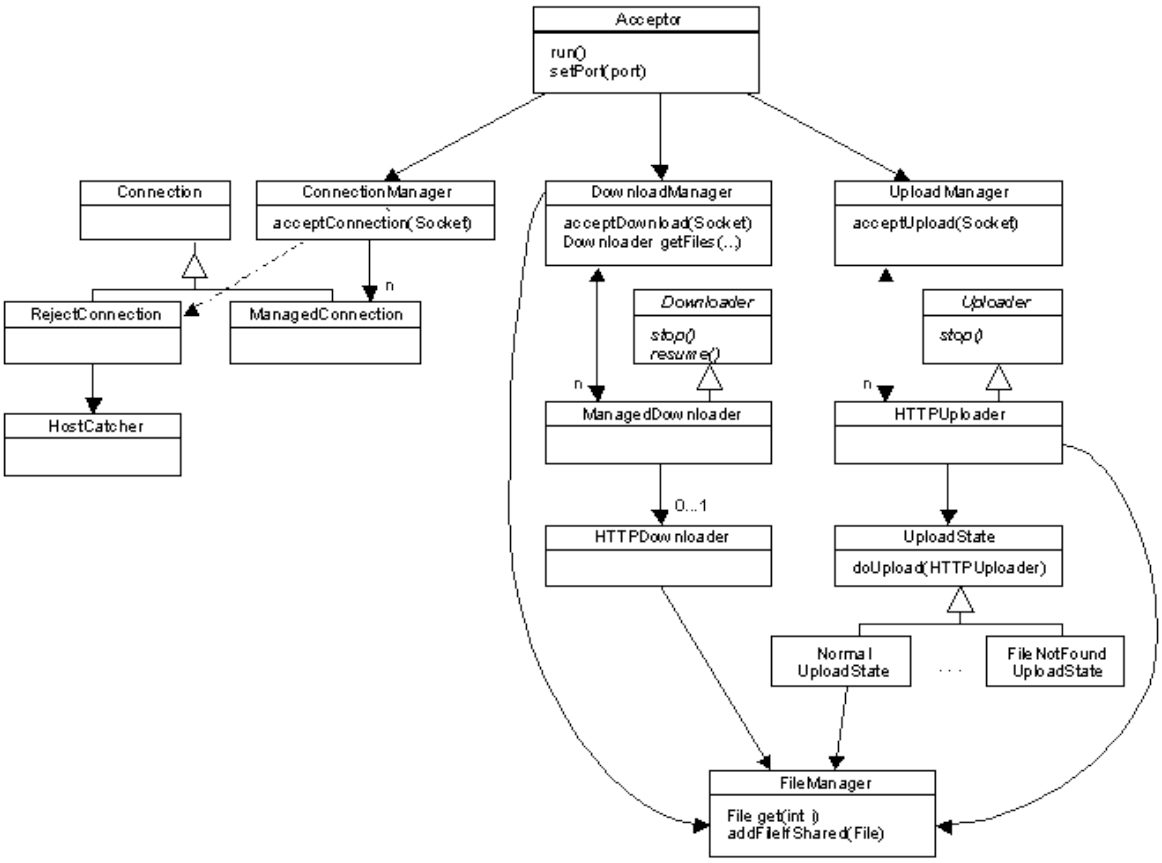

Figura 4.6 – Hierarquia de classes de controle de transferência de arquivos no LimeWire. *Fonte: http://www.limewire.org/techdocs/design.html*

4.6.4 GUI-Backend interface

As camadas *GUI* e *backend* do LimeWire são completamente desacopladas, possibilitando o desenvolvimento de outros tipos de interface e qualquer tipo de mudanças nas classes, sem afetar o restante da arquitetura do software.

A *GUI* dispara operações ao *backend* da aplicação através da instância da classe *RouterService*. Esta classe provê uma série de funcionalidades a *GUI*. Como por exemplo, conexões, desconexões, consultas e *downloads* de arquivos. O *backend* da aplicação realiza a comunicação com a *GUI* através da interface *ActivityCallback* usando o padrão de projeto *Observer*[41]. *ActivityCallback* é implementada pela classe *VisualConnectionCallback* que delega para os componentes *GUI* apropriados.

A classe *SettingManager* provê uma lista de propriedades de compartilhamento entre o

[41] http://c2.com/cgi/wiki?ObserverPattern

backend e a *GUI* como a lista de diretórios compartilhados pelo usuário. A hierarquia de classes apresentadas nesta subseção é ilustrada na figura 4.7. Maiores detalhes sobre tais classes podem ser obtidos em (LimeWire, 2006b).

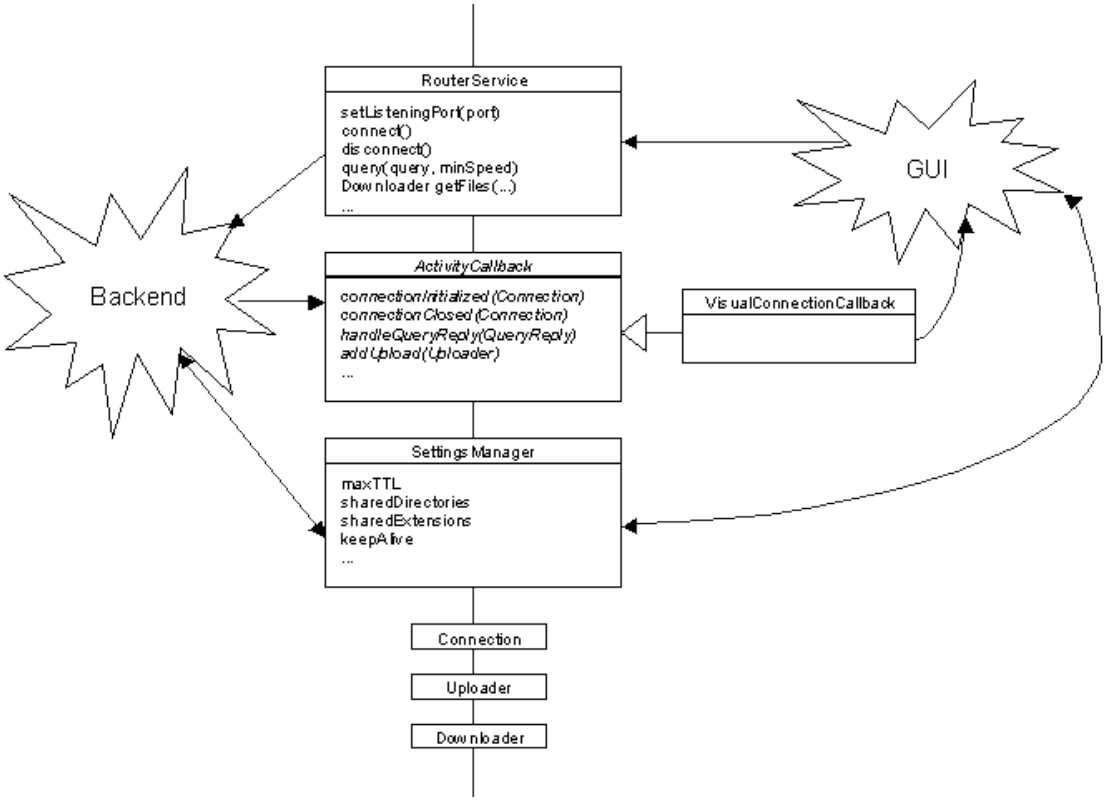

Figura 4.7 – Hierarquia de classes de interação entre as camadas de backend e GUI do LimeWire. *Fonte:* http://www.limewire.org/techdocs/design.html

4.6.5 Threads

De acordo com a documentação do LimeWire, várias *threads* são utilizadas no programa. A seguir listaremos algumas das *threads* relevantes:

- Para cada conexão aceita pela *thread Acceptor*, uma *thread Connection* é criada;
- Cada *ManagedConnection* possui duas *threads*: uma para a leitura de dados da conexão e outra para a fila de pacotes;
- *ConnectionManager* cria várias *threads* de *ConnectionFetcher* para encaminhamento de mensagens de destino;

- O *HostCatcher* possui *thread* chamada de *RouterConnectionThread* que estabelece conexões com os servidores de *pong* quando necessário.
- A *thread* de *ConnectionWatchDog* procura por conexões inativas e tenta substituir por uma conexão ativa quando possível;
- A transferência de arquivos é realizada por *threads* de *Downloader* e *Uploader*. Um arquivo é transferido por *thread*.
- A interface gráfica possui várias *threads* que provêem funcionalidades à GUI.

Maiores detalhes sobre tais *threads* podem ser obtidas em (LimeWire, 2006b).

Na seção seguinte, apresentaremos alguns detalhes da API Lucene que são úteis para a nossa proposta.

4.7 Lucene

O Lucene é parte do projeto Apache Jakarta[42]. Lucene é um motor de indexação e consulta de alta performance que fornece esses serviços à outras aplicações que venham a utilizá-lo (Gospodenetic; Hatcher, 2005).

A indexação é o processamento da fonte de informação (ou texto) na forma de referências cruzadas que permitam a realização de operações de consulta de forma mais rápida. A consulta sobre os índices é o processo de vasculhar o índice, segundo um critério de pesquisa, a fim de que se possa encontrar quais arquivos possuem tal palavra ou valor de pesquisa. Os processos de indexação e consulta são os alicerces de qualquer produto de recuperação de informação (Gospodenetic; Hatcher, 2005).

As métricas clássicas para a qualidade das consultas feitas em sistemas de recuperação de informação são a revocação (*recall) e* precisão (*precision).* O índice de revocação é definido como a fração de documentos relevantes recuperados pelo sistema de recuperação de informação. Esse índice indica o grau de exaustão de execução do sistema e o número de documentos recuperados que o usuário poderia querer (Hamra; Felber, 2005). O índice de precisão corresponde à fração de documentos recuperados que são relevantes. O índice de precisão mede a especialidade da busca e se todos os documentos recuperados são realmente

[42] http://jakarta.apache.org

relevantes (Hamra; Felber, 2005).

A figura 4.8 ilustra a forma geral de integração do Lucene com outras aplicações, através dos seus módulos de indexação e busca.

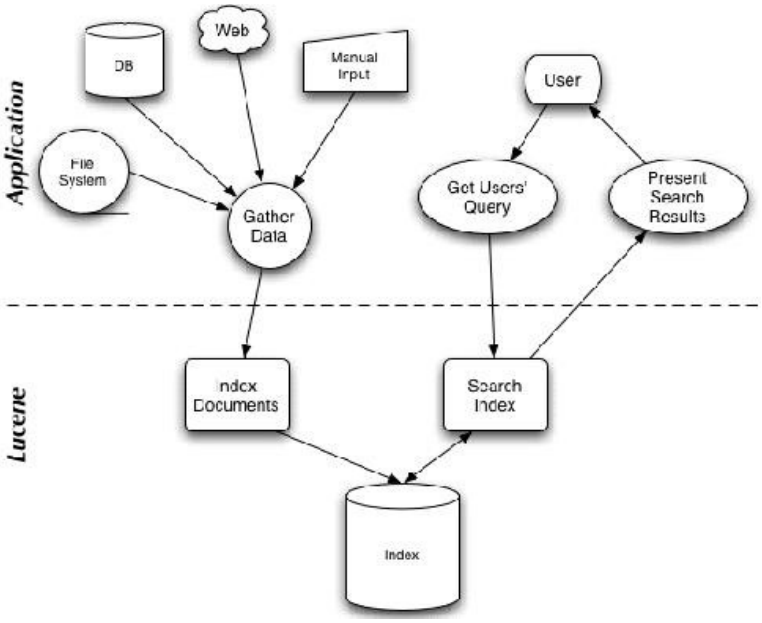

Figura 4.8 - Integração do Lucene com outras aplicações.
Fonte (Gospodenetic; Hatcher, 2005).

4.7.1 Indexação no Lucene

O processo de indexação no Lucene é feito em três etapas: conversão de dados em *strings*, análise e gravação do índice (Chakrabarti 2, 2003). Os documentos fonte são lidos e deles são extraídas *strings* e então convertidas em *tokens*. Os *tokens* são analisados e podem sofrer transformações. Por exemplo, retirando artigos, preposições, transformando caracteres especiais etc. Em seguida, os *tokens* são gravados no índice como termos. A figura 4.9 ilustra o processo de indexação do Lucene. Os módulos *parser, analysis* e *index* são comentados com maiores detalhes a seguir.

Parser. O Lucene pode indexar vários tipos de arquivos, por exemplo: documentos textuais, documentos do MS Word, arquivos XML, páginas HTML, documentos RTF, PDFs ou qualquer outro formato que possa ser retirada informação textual (Gospodenetic; Hatcher,

2005). Entretanto, para os documentos que não são em texto limpo, o Lucene requer *parsers* de terceiros que façam o trabalho de retirar os *tokens* dos arquivo. Os *parsers* conhecem a estrutura interna dos tipos de arquivos e assim conseguem separar dados de usuário de dados de controle e metadados. Uma vez separados, os *tokens* podem ser analisados e indexados.

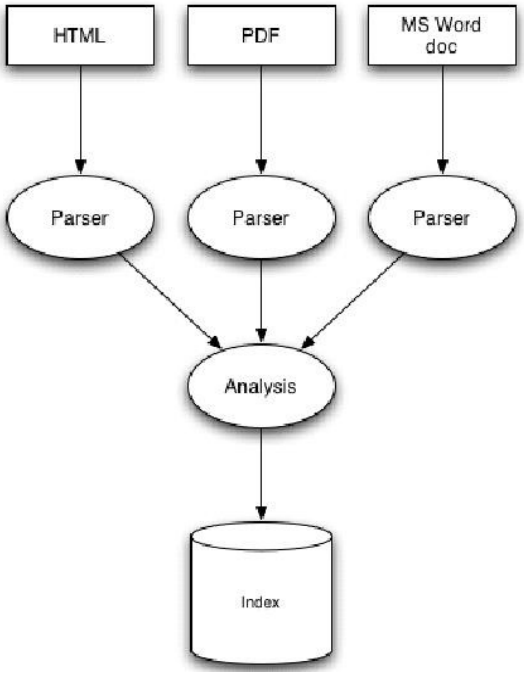

Figura 4.9 - Indexação Lucene. *Fonte (Gospodenetic; Hatcher, 2005).*

Análise. No Lucene, análise é o processo de converter texto em termos (Gospodenetic; Hatcher, 2005). Um termo é a forma mais básica de representação de uma unidade do índice. Os termos são usados para determinar quais documentos satisfazem uma consulta. O *Analyzer* encapsula o processo de análise. O *analyzer* cria *tokens* aplicando uma séria de operações sobre o texto, como: extração de palavras, descarte de pontuação, remoção de acentos, redução para letras minúsculas, retirada de palavras comuns, redução de palavras ao seu radical (*stemming)* (Gospodenetic; Hatcher, 2005). *Tokens,* combinados com os seus nome s de campo, são termos (Gospodenetic; Hatcher, 2005).

Índices. O Lucene suporta dois tipos de estrutura para os seus índices: multi-arquivo e composto. As duas estruturas utilizam os conceitos de segmentos, documentos, campos e termos. O tipo multi-arquivo, como o nome sugere, mantém vários arquivos abertos. Isso pode causar alguma limitação com algum sistema operacional e aplicação. Os índices são

armazenados, no sistema de arquivos, com segmentos. O número de arquivos de índices depende do número de campos a indexar. O tipo composto, por outro lado, utiliza menos arquivos abertos simultaneamente, mas tem menor desempenho (de 5-10%) que o multi-arquivo. O tipo composto é mais indicado para aplicações onde são necessário muitos índices e índices com um grande número de campos (Gospodenetic; Hatcher, 2005).

Outro tipo de arquivo de índice utilizado pelo Lucene é o índice invertido. Essa estrutura de índices permite saber rapidamente quais os arquivos possuem um determinado termo. Os índices suportam operações para a sua atualização (inclusão, alteração e exclusão), na medida que os documentos originais da coleção são atualizados. Maiores informações acerca da estrutura dos índices Lucene estão disponíveis em (Gospodenetic; Hatcher, 2005), (Gospodenetic; Hatcher, 2005) e sobre índices invertidos em (Chakrabarti 2, 2003).

Na figura 4.10 são apresentados quatro tipos de arquivos: .fnm, .tis, .frq e .prx. O .fnm contém todos os nomes de campos usados nos documentos. O .tis é o dicionário de termos em ordem alfabética, onde cada termo possui um ponteiro para a sua freqüência, armazenada em arquivos .frq. O .prx lista a posição de cada termo dentro do documento (Gospodenetic; Hatcher, 2005).

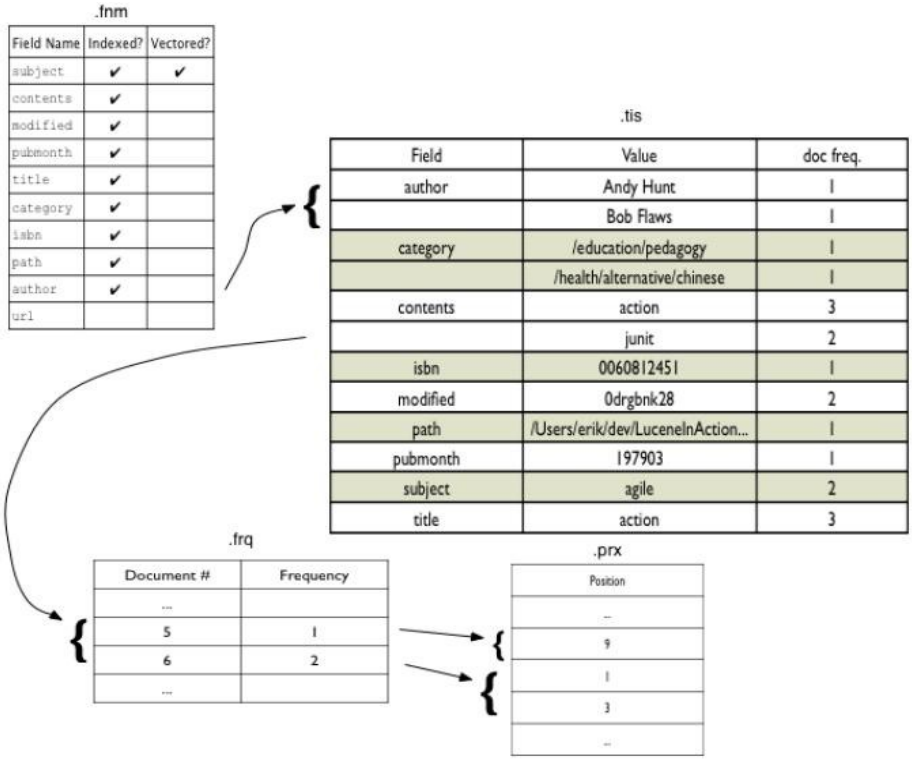

Figura 4.10 - Índices invertidos no Lucene. *Fonte (Gospodenetic; Hatcher, 2005).*

As classes principais envolvidas no processo de indexação são (Gospodenetic; Hatcher, 2005):

- *IndexWriter*: é a classe central para o processo de indexação. Ela permite a inclusão de entradas no índice.
- *Directory*: classe abstrata que representa a localização de um índice Lucene, que tanto pode ser feita em memória RAM *(RAMDirectory),* como em disco (*FSDirectory).*
- *Analyzer*: é outra classe abstrata Lucene que é responsável por retirar os *tokens* do texto e eliminar o restante. Se o conteúdo a indexar não é texto plano, é necessário primeiro transformá-lo, conforme a figura 4.9. As operações de retirada de *stopwords e stemming* do texto, descritas no capítulo 2, são realizadas.
- *Document*: representa uma coleção de campos. É a abstração do documento real ou dos metadados associados a esse documento.
- *Field*: um *Document* Lucene é composto de um ou vários *Fields.* Cada *Field* fica armazenado na classe *Field* e representa uma parte da informação que poderá ser usada no processo de recuperação de informação.

Os tipos possíveis de *Field* são: *Keyword, UnIndexed, UnStored* e *Text,* detalhados a seguir.

Keyword: Não é analisado, mas é indexado e armazenado no índice. Este tipo é ideal para campos que precisam ser preservados em sua totalidade como urls, caminhos de arquivo do sistema, datas, nomes de pessoas, número de telefones, entre outros.

UnIndexed: Não são analisados nem indexados, mas seus valores são armazenados no índice. Este tipo é ideal para campos que tem necessidade de serem mostrados nos resultados da busca como uma *url* ou uma *primary-key* de um banco de dados.

UnStored: São analisados e indexados, mas seus valores não são armazenados no índice. É ideal para a indexação de textos longos que não necessitam ser recuperados inteiramente como corpo de páginas web.

Text: É analisado e indexado. Os campos desse tipo são retornados nas buscas. Se o

dado for uma String ele é armazenado, mas se for um Reader ele não será.

A criação de um índice Lucene envolve alguns passos. O processo inicia quando instanciamos *IndexWriter* informando como parâmetros a localização (através da classe *Directory*, *File* ou informando o *path*) em que será o índice criado, o tipo de *Analyzer* que será utilizado e um parâmetro *booleano* (*true*, quando o índice é criado; *false*, quando são adicionados novos documentos a um índice existente).

Os *analyzers* são utilizados para conversão do arquivo no formato texto limpo. O documento (arquivo) a ser indexado é representado pela classe *Document*. As porções de texto de um documento são representadas através de campos (*Field*) estruturados através do par nome-valor. *Fields* são adicionados ao *Document* através do método *add* e possuem características que determinam, por exemplo, se o campo será ou não indexado. Finalmente, o índice é indexado através do método *addDocument* de *IndexAnalyzer* que recebe um *Document* como parâmetro.

4.7.2 Busca no Lucene

Uma vez criados os índices, pesquisas podem ser realizadas. A partir do critério de pesquisa, é feito um processo de transformação (*parser)* do parâmetro de consulta do usuário para uma instância de *Query.* O Lucene utiliza o objeto *Query* para realizar as consultas, com uma subclasse para cada tipo de consulta. O resultado é organizado no objeto *Hits*, que possui vários campos como o nome do arquivo, sua localização em disco etc. As principais classes Lucene envolvidas no processo de consulta são: *IndexSearcher, Term, Query, TermQuery e Hits,* comentadas a seguir. Maiores detalhes sobre estas classes podem ser obtidas em (Gospodenetic; Hatcher, 2005).

IndexSearcher. A classe *IndexSearcher* é o componente central para o processo de buscas no Lucene. Ela abre o índice para a leitura, sem modificá-lo e possui vários métodos para busca, muitos deles implementados a partir da classe abstrata *Searcher*. O método mais simples recebe como parâmetro um objeto *Query* e retorna o objeto *Hits* como resultado do processamento. Existem três construtores para a classe *IndexSearcher*, um recebendo um diretório (onde estão os índices) como parâmetro, outro passando como parâmetro um índice através da classe abstrata *IndexReader* que provê uma interface para acesso a um índice e por

último, um construtor recebendo um *path* em que o índice se encontra.

Term. A classe abstrata *Term* possui estrutura similar à classe *Field*. Contém um par nome-valor em que o nome identifica os campos *Field* que serão pesquisados e o valor corresponde a palavra-chave que será submetida para a busca. A Classe *Term* também é usada internamente durante o processo de indexação.

Query. A classe abstrata *Query* é recebida como parâmetro no método de *IndexSearch*. A implementação mais básica de *Query* é a *TermQuery*. Outros tipos de *queries* são:

- *BooleanQuery*: usada para combinar com outras *queries* e adicionar cláusulas lógicas do tipo *AND*, *OR* ou *NOT*.
- *PhraseQuery* e *PhrasePrefixQuery*: usam informações posicionais para buscar documentos que distam uma dada distância de outro.
- *RangeQuery*: busca termos dentro de uma faixa.
- *WildcardQuery*: usa os *wildcards* padrões * e ?
- *FuzzyQuery*: combina termos similares com termos específicos através do algoritmo Levenshtein Distance[43]. Isto permite a realização de consultas com termos e respostas com termos similares.

Hits. Os resultados de uma busca são acessados através da classe *Hits* que é obtida como retorno de um dos métodos *search* de *IndexSearcher*. Os resultados são organizados por relevância de acordo com um escore calculado a partir de uma fórmula matemática na classe abstrata *Similarity*. Para cada documento encontrado é atribuído um escore (variando de 0 a 1), quanto maior o escore, maior a possibilidade que o documento tem de atender à busca submetida pelo usuário.

Para que um documento seja resultado de uma busca ele deverá possuir um escore maior que zero. Outras fórmulas podem ser usadas para cálculo do escore disponíveis em subclasses de *Similarity*. Por padrão, o Lucene usa a implementação *DefaultSimilarity*.

O Lucene provê outras possibilidades de ordenação como: ordenação por *fields* (desde que os *fields* obedeçam regras de criação de *fields* ordenáveis), por múltiplos *fields*, por tipos de *fields* (*Integer*, *String*), por ordem inversa, entre outros.

---

[43] Http://www.merriampark.com/ld.htm

Cinco métodos são disponíveis nesta classe. O *lenght()* retorna o número de documentos retornados na pesquisa. Os métodos *doc(n), id(n)*, *score(n)* permitem o acesso aos documentos especificados. O Lucene não carrega todos os documentos de uma vez, apenas os 100 primeiros documentos são retornados e colocados em *cache* inicialmente. Esta aparente limitação, na prática, não causa problemas porque normalmente os documentos desejados estão entre os primeiros resultados (com maior escore). Por último, o método *iterator()* retorna um objeto *HitsIterator* para facilitar a navegação entre os *Hits*.

Uma outro tipo de busca suportada pelo Lucene é a busca vetorial. O espaço vetorial de termos é uma coleção de pares termos-freqüência (Gospodenetic e Hatcher-2, 2005). O espaço vetorial foi incorporado ao Lucene na versão 1.4 e esse é um conceito clássico na área de recuperação da informação, comentado no capítulo 2. Para que o Lucene opere com espaço vetorial é necessário ativar tal opção durante a indexação. Já para a consulta, será necessário utilizar um método *getTermFreqVector* (Gospodenetic; Hatcher, 2005).

A adição do recurso da indexação para o *peer* permite a realização de consultas sobre o conteúdo dos documentos compartilhados com tempo de resposta aceitável. Uma vez criados os índices, um *peer* poderia, por exemplo, solicitar a outro *peer* a lista (*hits)* dos arquivos que ele compartilha e que tenham "P2P" em seu conteúdo. Tal funcionalidade enriquece a forma de consulta e aprimora a troca de arquivos nessas redes.

Na seção seguinte, veremos como ocorre o funcionamento da arquitetura proposta em condições normais de utilização.

A seguir, apresentaremos alguns cenários para a utilização da nossa proposta.

4.8 Cenários de utilização

A arquitetura proposta requer uma rede P2P. Para efeito do protótipo e dos testes, optamos pela rede Gnutella, mas não precisamos ficar restritos a ela. Outras redes P2P poderiam servir para o funcionamento da arquitetura, com maiores ou menores ajustes no roteamento de mensagens, desde que se considere que os índices textuais da coleção estarão presentes no *peer.* Outras variações da arquitetura proposta podem ser consultadas no capítulo 6, como pesquisas futuras.

Os programas P2P populares e seus usuários efetuam o compartilhamento de qualquer

tipo de arquivo. Entretanto, a maioria dos arquivos desses *peers* são do tipo multimídia. Uma vez que o maior benefício da utilização da nossa proposta cai sobre os arquivos textuais, os cenários de utilização ideais são aqueles onde pode haver há troca intensa desses arquivos. Alguns exemplos seriam:

a) **No ambiente acadêmico:** pesquisadores, professores e alunos poderiam utilizar nossa proposta e assim encontrar arquivos textuais na rede pelo seu conteúdo. Assim, artigos, notas de aula, tutoriais, manuais (etc.) que pudessem interessar aos usuários dessa rede poderiam ser encontrados não apenas pelo o nome do arquivo.

b) **No ambiente corporativo:** colaboradores de uma organização, ou de um conjunto de organizações que atuem no mesmo setor de mercado, que utilizem arquivos semelhantes (ou modelos) para a realização das suas atividades, poderiam ter na rede P2P da companhia uma forma de trocar arquivos. Existem duas questões que precisamos comentar nesse cenário: quanto ao compartilhamento e quanto a localização dos arquivos. O compartilhamento de arquivos neste ambiente é comum e via de regra ocorre através de servidores de arquivos, que, a priori, armazenam arquivos de interesse das áreas organizacionais. Considerando um servidor de arquivos como um *peer* tornaria a consulta por conteúdo possível. Já quanto a localização de arquivos, observamos que existe certa latência pois mesmo estando em um computador (servidor), não há uma estratégia de indexação preparada para consultas textuais, solicitadas pelas demais estações da rede.

c) **Na Internet:** *peers* em diversas partes do planeta poderiam localizar arquivos textuais compartilhados por outros *peers,* pelo conteúdo. Nesse cenário, dada a possibilidade da quantidade de *peers* da rede ser grande, poderão surgir problemas com a carga de tráfego da rede P2P, em virtude da troca de mensagens e dos *hits*. Entretanto, o benefício dessa forma de consulta justificaria, na nossa visão, a sua utilização em larga escala pelo benefício advindo da consulta pelo conteúdo.

4.9 Sumário do capítulo

Nesse capítulo apresentamos uma proposta de arquitetura para realizar a localização de documentos pelo seu conteúdo em redes P2P.

Iniciamos com a contextualização do problema da localização de documentos em redes P2P ressaltando as principais dificuldades relacionadas ao objetivo. Definimos um conjunto de requisitos funcionais necessários aos nós da rede P2P para que esses possam ter a capacidade de enviar as consultas e realizar consultas textuais de forma mais rápida, como também enviar respostas (*hits)* de volta aos *peers* solicitantes.

Detalhamos alguns aspectos do LimeWire e do Lucene, software P2P e API de indexação e consulta, respectivamente, utilizados como componentes da nossa proposta, ressaltando a oportunidade e a necessidade de integração entre os dois.

Em seguida, apresentamos o funcionamento da arquitetura considerando as condições normais ("*happy day"*) e por fim, elencamos alguns cenários de utilização como exemplos da aplicação da arquitetura.

No capítulo seguinte, explicaremos alguns detalhes e desafios envolvidos na elaboração do protótipo e a avaliação inicial da arquitetura.

Capítulo 5 – Implementação

Nesse capítulo apresentaremos os detalhes de implementação do protótipo da nossa arquitetura para realizar a localização de documentos textuais, pelo conteúdo, em redes P2P.

A fim de ilustrar o funcionamento da arquitetura proposta, apresentaremos os detalhes da implementação do nosso protótipo. A integração dos dois produtos escolhidos como alicerces, LimeWire e Lucene, dá aos *peers* a capacidade de localizar e recuperar documentos textuais em uma rede P2P pelo conteúdo e não apenas pelo nome do arquivo ou o seu tipo.

Para o desenvolvimento do nosso protótipo foi utilizada a linguagem de programação Java[44], desenvolvida e disponibilizada gratuitamente pela Sun MicroSystems, em sua versão Standard Edition 5.0 (pacote J2SE Development Kit 5.0 Update 8) juntamente com sua documentação. Para o desenvolvimento nesta linguagem, utilizamos a IDE Eclipse[45] em sua versão 3.2.1.

Utilizamos o código-fonte do *software* de compartilhamento de arquivos LimeWire[46] em sua versão 4.12 (download em agosto de 2006). Ele foi totalmente desenvolvido em Java e seu código é disponibilizado gratuitamente pela empresa desenvolvedora do programa, a LimeWire LCC. Fizemos um intenso uso de sua documentação técnica, tutoriais e de fóruns disponíveis pelo *site* oferecido aos desenvolvedores[47]. Para a compilação do código-fonte foi necessária a utilização da ferramenta Apache ANT[48]. Também utilizamos a API de Recuperação de Informação Lucene[49], mantida projeto Apache, em sua versão 2.0.0 e sua documentação.

No capítulo anterior, apresentamos a lista de requisitos necessários ao protótipo para que ele pudesse alcançar o objetivo desejado. São eles:

---

44 http://java.sun.com
45 http://eclipse.org
46 http://www.limewire.com
47 http://www.limewire.org
48 http://ant.apache.org/
49 http://lucene.apache.org

a) Criação e atualização de índices locais (aos *peers)* para que possam realizar consultas rápidas sobre o conteúdo dos documentos textuais compartilhados;

b) Realização de consultas textuais nos documentos compartilhados pelo *peer,* quando houver o recebimento de uma mensagem desse tipo;

c) Formatação do resultado da consulta local, contendo a lista dos documentos que satisfazem ao critério de pesquisa recebido*;*

d) Envio do resultado ao *peer* solicitante através da rede P2P;

e) Exibição do resultado da consulta para o usuário do *peer* solicitante;

f) Permitir que o *peer* realize o *download* do arquivo desejado a partir do(s) *peer(s)* detentor(es) do documento solicitado.

Tendo em vista a diferença de papéis praticados por nós-folha e por nós-*ultrapeers* na rede Gnutella*,* como também o funcionamento do protocolo de roteamento de consultas (*Query Routing Protocol - QRP*)*,* será necessário que os dois tipos de *peer* realizem as operações de indexação e consulta aos índices Lucene locais, para que o protótipo possa funcionar corretamente.

Com efeito, no capítulo 2, foi dito que os *ultrapeers* representam um nível acima dos *peers* na hierarquia da rede Gnutella e que a presença deles ajuda a conter a inundação de tráfego, ocasionada pela propagação indiscriminada das consultas; foi dito também, no mesmo capítulo, que cada *peer* pode vir a se tornar um *ultrapeer (capable),* caso a rede necessite e o usuário assim deseje[50]. Optamos por "forçar" que o protótipo realize consultas aos índices Lucene, mesmo que o computador do usuário tenha status de *peer.* Isto permitiu celeridade nos testes do protótipo já que não foi necessário aguardar a rede precisar de um *ultrapeer*, mas deve ser uma condição de testes em laboratório e não uma definição para a arquitetura que propomos.

Na seção seguinte, detalharemos a parte de codificação da implementação apresentando as classes criadas para integrar o Lucene ao LimeWire.

---

[50] A anuência do usuário para tornar o seu *peer* um *ultrapeer capable* é feita através configuração das opções do LimeWire.

5.1 Implementação do protótipo: integração entre LimeWire e Lucene

Nesta seção apresentaremos a implementação do nosso protótipo baseado na arquitetura proposta. Apresentaremos as classes envolvidas na integração entre o LimeWire e o Lucene, como também as classes que sofreram modificações afim de implementar as funcionalidades necessárias.

Para a implementação da criação de índices e realização de busca sobre os arquivos, criamos três classes: *IndexFiles*, *SearchFiles* e *FileDocument*. Estas classes foram reunidas no pacote *br.com.unifor.informatica.lucene*, conforme ilustrado na figura 5.1, localizadas no diretório *core* do projeto LimeWire.

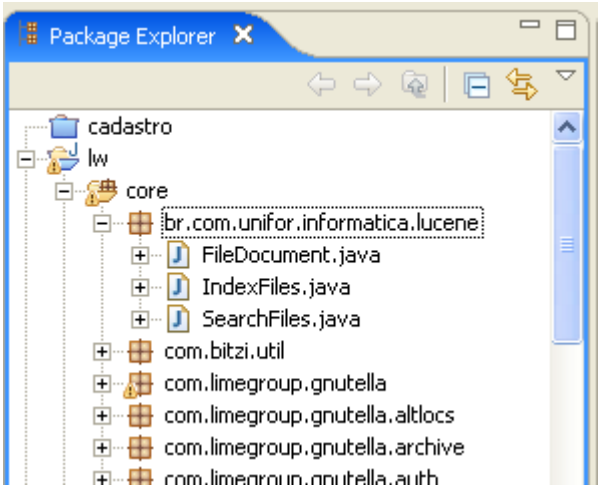

Figura 5.1 – Pacote br.com.unifor.informatica.lucene no Eclipse.

A classe *IndexFiles* é responsável pela criação de índices sobre os arquivos. A busca sobre esses índices é realizada pela classe *SearchFiles*. A classe *FileDocument* é usada para montar o objeto *Document* usado para representar o arquivo. Nas subseções a seguir, apresentaremos os trechos de código das classes.

5.1.1. Classe IndexFiles

A seguir mostramos o código da classe *IndexFiles* responsável pela criação e atualização dos índices Lucene dos documentos da coleção do *peer*. Esta classe possui o método *indexar* que recebe como parâmetros os atributos nomeIndice e listaDiretorios indicando o nome do índice a ser criado e a lista de diretórios de arquivos compartilhados pelo LimeWire.

Em nossa implementação, o índice é criado em um diretório dentro do diretório *bin* da instalação do LimeWire. A cada nova inicialização do programa cria um novo índice, sobrepondo o índice criado anteriormente. A listagem do código-fonte da classe encontra-se no anexo A.

No processo de indexação os documentos são representados pela classe *FileDocument.* O código-fonte da classe encontra-se no anexo A. Nos documentos (*documents)* adicionados ao índice são incluídos os seguintes campos (*field*s): *path* que indica o caminho de armazenamento de arquivo no sistema de arquivos local; *modifield* que indica a data de modificação do arquivo; e *contents* que indica o conteúdo do arquivo indexado.

5.1.2. Classe SearchFiles

A classe *SearchFiles* é responsável pela busca no índice Lucene. As classes básicas do Lucene usadas para este fim foram descritas no capítulo 4. A classe *SearchFiles* possui o método *procurar* que recebe como parâmetro o nome do índice e o critério de consulta a ser pesquisado. Efetuada a consulta, o método percorre os resultados (*hits)* encontrados e devolve uma lista contendo o *path* dos arquivos. O código-fonte da classe encontra-se no anexo A.

A fim de facilitar o entendimento dos detalhes da implementação da integração entre os produtos LimeWire e Lucene, adotamos uma trilha de alterações baseada no funcionamento desejado da arquitetura, conforme os passos: entrada do *peer* na rede Gnutella, indexação dos documentos da coleção do *peer* e a consulta por conteúdo sobre a coleção dos *peers*.

5.2 Entrada na rede Gnutella

Uma vez iniciado, o LimeWire tenta estabelecer conexão com a rede Gnutella através da troca de mensagens Gnutella Ping e Pong. O processo de conexão na rede Gnutella não

será afetado pela implementação do protótipo. Manteremos a especificação original do protocolo utilizando o serviço *bootstrapping*[51] para localizar *peers* na rede e em seguida o *handshaking*[52] para efetivamente incluir o novo *peer* na rede.

Uma versão mais elegante do protótipo poderia, no futuro, concomitantemente ao processo de conexão com a rede Gnutella, indagar ao usuário quanto a criação ou atualização dos índices Lucene locais; ou ainda, realizá-la em segundo plano sem indagar ao usuário, mediante a checagem da necessidade de indexação de alguns documentos da coleção. A seguir, comentaremos a indexação dos documentos da coleção.

5.3 Indexação dos documentos textuais compartilhados

A criação dos índices dos arquivos textuais da pasta compartilhada do *peer* é um passo essencial para que o *peer* tenha a capacidade de realizar consultas sobre o conteúdo desses documentos de forma rápida.

A criação e atualização dos índices serão realizadas pelo Lucene. A depender do tamanho da coleção, o processo de criação e atualização dos índices pode necessitar de maior tempo de processamento. As principais classes do Lucene envolvidas na criação de índices são: *IndexWriter, Directory, Analyzer, Document* e *Field.*

A estrutura do índice reflete os tipos de consultas suportadas por ele. No Lucene, o desenvolvedor tem a disposição várias opções de índices e tipos de consultas, por exemplo: índices para texto, índices para campos em formato de data, índices para números etc. Para efeito do protótipo, utilizaremos a forma de indexação padrão do Lucene, ou seja, índices textuais de arquivos texto (.txt) das pastas compartilhadas. Tal tipo de índice permite consultas baseadas em palavras-chave ou texto limpo.

O Lucene também permite a indexação de arquivos de outros formatos. Há extensões para o Lucene poder indexar documentos do MS-Word, arquivos PDF, RTF, XML etc. Isto é feito através de *parsers* específicos. Maiores informações sobre indexação de arquivos de outros formatos podem ser obtidas em (Gospodenetic; Hatcher, 2005).

Os índices também precisam ser atualizados sempre que a coleção muda. Desta forma,

---

[51] http://rfc-gnutella.sourceforge.net/developer/testing/bootstrapping.html
[52] http://rfc-gnutella.sourceforge.net/developer/testing/handshake.html

as consultas poderão recuperar o resultado (*hits)* correto. A manutenção dos índices precisa acontecer quando o conteúdo dos arquivos for alterado, quando algum arquivo for adicionado ou excluído da coleção.

A fim de facilitar a integração com o código-fonte do LimeWire, optamos por encapsular as várias classes do Lucene em classes mais genéricas. Uma dessas classes é *IndexFiles.java*, que é responsável por acionar o processo de indexação. O seu código-fonte está disponível no anexo A.

No LimeWire, optamos por inserir a chamada ao método *IndexFiles.indexar* na classe *Initializer.java*, que é responsável por instanciar as principais classes do LimeWire, no momento em que ele é iniciado. Tais classes são: *ResourceManager, GUIMediator, SettingsManager, FileManager* e *RouterService.* A figura 5.2 traz um trecho de *Initializer.java* onde houve a nossa intervenção.

```
//Realiza a indexação da(s) pasta(s) compartilhada(s)
System.out.println("Indexando arquivos das pastas compartilhadas...");
File diretorios[] =
SharingSettings.DIRECTORIES_TO_SHARE.getValueAsArray();
IndexFiles.indexar(SharingSettings.NOME_INDICE,diretorios);
//
```

Figura 5.2 - Trecho de Initializer.java onde é acionada a indexação do Lucene.

O LimeWire permite que o usuário defina uma ou mais pastas compartilhadas no computador do usuário. Dessa forma, o método *IndexFiles.Indexar* utilizará as pastas compartilhadas na indexação.

Ao final desses passos, os arquivos textuais compartilhados do *peer* estarão indexados e prontos para receber consultas sobre o seu conteúdo. Na seção seguinte, mostraremos os ajustes para o processo de consulta.

5.4 Consulta por conteúdo no LimeWire

Uma vez criados os índices, o *peer* já poderá realizar consultas pelo conteúdo dos arquivos textuais, localizados na(s) pasta(s) compartilhada(s).

O processo de consulta pode ser dividido nas seguintes etapas, comentadas em

subseções específicas: o usuário informa o parâmetro de pesquisa, o *peer* envia uma mensagem para rede solicitando arquivos que contenham o parâmetro de pesquisa, um outro *peer* (ou mais) recebe a mensagem e realiza a pesquisa pelo conteúdo na sua coleção, envia de volta ao *peer* solicitante o resultado da pesquisa local, o *peer* solicitante recebe o resultado consulta.

5.4.1. Recebimento do valor a pesquisar

O processo de consulta inicia propriamente quando o usuário deseja submeter uma consulta por uma palavra ou grupo de palavras. Na interface gráfica do LimeWire, um painel contém campos para vários tipos de pesquisa, conforme a figura 5.3.

Para efeito do protótipo, optamos por realizar as consultas pelo conteúdo coletando o parâmetro de pesquisa do campo *Topic* do LimeWire. Uma versão melhorada do protótipo poderia ter um campo específico (*text-box)* para receber o parâmetro de pesquisa por conteúdo e um *check-box,* desativado inicialmente, cujo título poderia ser "Consulta pelo conteúdo", onde o usuário indicaria explicitamente se deseja ou não fazer a consulta pelo conteúdo dos documentos dos *peers*.

Figura 5.3 - Opções de consulta do LimeWire para documentos.

Optamos por não incluir esta facilidade no protótipo porque necessitaríamos modificar os *peers* da rede para que pudéssemos testá-la. Basta que nosso *peer* seja modificado para que

ele possa incluir resultados da pesquisa baseada no conteúdo dos documentos da coleção como também, submeter consultas aos demais *peers*, de acordo com o protocolo original. Desta forma, não seria necessário alterar todos os *peers* para a nossa versão e então proceder os experimentos. Isto propiciou a realização dos testes diretamente na rede Gnutella e não em um ambiente simulado. Por outro lado, se a interface gráfica nativa do LimeWire fosse modificada, poderia acarretar em algum tipo de modificação nos demais *peers*.

A inclusão do *check-box* nas opções de consulta de documentos do LimeWire seria uma extensão conveniente porque o usuário indicaria explicitamente que deseja realizar consultas por conteúdo dos arquivos compartilhados dos *peers,* já que esse tipo de consulta tende a envolver maiores recursos computacionais dos *peers* e recuperar um conjunto resposta maior, ocupando também maior largura de banda de rede.

De posse do parâmetro de consulta informado pelo usuário, o protótipo enviará para a rede Gnutella uma mensagem *GnutellaQuery*, que é descrita na subseção seguinte.

5.4.2.Envio da mensagem de consulta por conteúdo

De posse do parâmetro de pesquisa, digitado pelo usuário, o protótipo enviará uma consulta para a rede. O protocolo Gnutella não possui uma mensagem específica para o tipo de consulta pelo conteúdo dos arquivos. Os tipos de mensagens suportados na versão 0.6 são (Gnutella, 2005)(Klingberg; Manfredi, 2002):

Mensagem	***Descrição***
0x00 = Ping	Usada para ativamente descobrir outros *peers* na rede. É esperado que um *peer* que receba uma mensagem Ping, responda ao remetente uma ou várias mensagens Pong.
0x01 = Pong	é a resposta à uma mensagem Ping. A mensagem Pong contém o endereço do *peers*, a porta definida para o recebimento das mensagens e demais dados sobre o montante compartilhado por esse *peer.*
0x02 = Bye	é uma mensagem opcional que informa ao *peer* remoto a saída (desconexão) da rede.
0x40 = Push	é a mensagem que permite ao *peer* enviar arquivos para a rede.
0x80 = Query	é a mensagem utilizada para lançar uma consulta na rede. É esperado

Mensagem	***Descrição***
	que um *peer*, ao receber um Query, realize a consulta local e caso haja algum objeto que atenda ao critério da consulta, seja enviada uma resposta ao remetente.
0x81 = Query Hit	é mensagem resposta a uma Query. QueryHit contém os dados dos arquivos que atendem ao critério da consulta realizada.

Na especificação do Gnutella v0.6 (Klingberg; Manfredi, 2002), identificamos a possibilidade de estendermos o protocolo para suportar o envio de mensagens de busca por conteúdo para outros *peers*. Desta forma, incluiríamos um tipo de mensagem que sinalize aos outros *peers* uma consulta ao conteúdo dos documentos compartilhados do *peer*. Poderíamos adotar o valor 0x70 = QueryByContent para identificar o novo tipo de mensagem. A mensagens poderia ter o seguinte cabeçalho:

Bytes	***Descrição***
0-15	Identificação da mensagem
16	Ttipo da mensagem (*payload type)*
17	TTL
18	Quantidade de saltos (*hops)*
19-22	Comprimento do *payload*

O campo tipo da mensagem *(payload type)* seria alterado para o valor 0x70 *(QueryByContent),* que informaria aos *peers* que receberão a mensagem, que essa se trata de uma consulta pelo conteúdo dos documentos compartilhados e não apenas uma consulta pelo nome do arquivo (*GnutellaQuery*). Em seguida, precisaríamos definir o corpo da mensagem 0x70. Nesse caso, poderíamos aproveitar a estrutura da mensagem 0x80 (*GnutellaQuery*) porque ela possui o formato que atende ao nosso objetivo. Tal mensagem seria assim definida (Klingberg; Manfredi, 2002):

Bytes	***Descrição***
0-1	Velocidade mínima do *servent* para poder responder a consulta
2-	Critério de pesquisa
REST	Opcional.

O primeiro campo, Velocidade Mínima do *servent*, indica a velocidade mínima de conexão que o *peer* deve ter. Este atributo é usado nos filtros de consulta. Na interface gráfica do LimeWire, o usuário pode configurar a sua velocidade de conexão com a Internet e a velocidade mínima dos *peers* que responderão às suas consultas devem ter. Isso permite que o usuário possa receber respostas de *peers* com conexões lentas, atrasando a troca de arquivos. O segundo campo, Critério de pesquisa, é onde colocaríamos o parâmetro de pesquisa informado pelo usuário para a consulta pelo conteúdo dos arquivos. Mais informações sobre a estrutura das mensagens Gnutella versão 0.6 podem ser encontradas em (Klingberg; Manfredi, 2002).

O LimeWire possui uma hierarquia de mensagens, descrita na figura 5.4. Estenderíamos esse conjunto de mensagens para suportar o envio das mensagens de consulta por conteúdo. Assim, a partir da classe *QueryRequest,* definiríamos a classe *QueryByContentRequest* que enviaria a mensagem 0x70 para rede Gnutella. Uma vez implentada a criação da mensagem *QueryByContent,* o seu funcionamento estaria condicionado a capacidade de processamento dessa mensagem por parte de outros *peers* da rede.

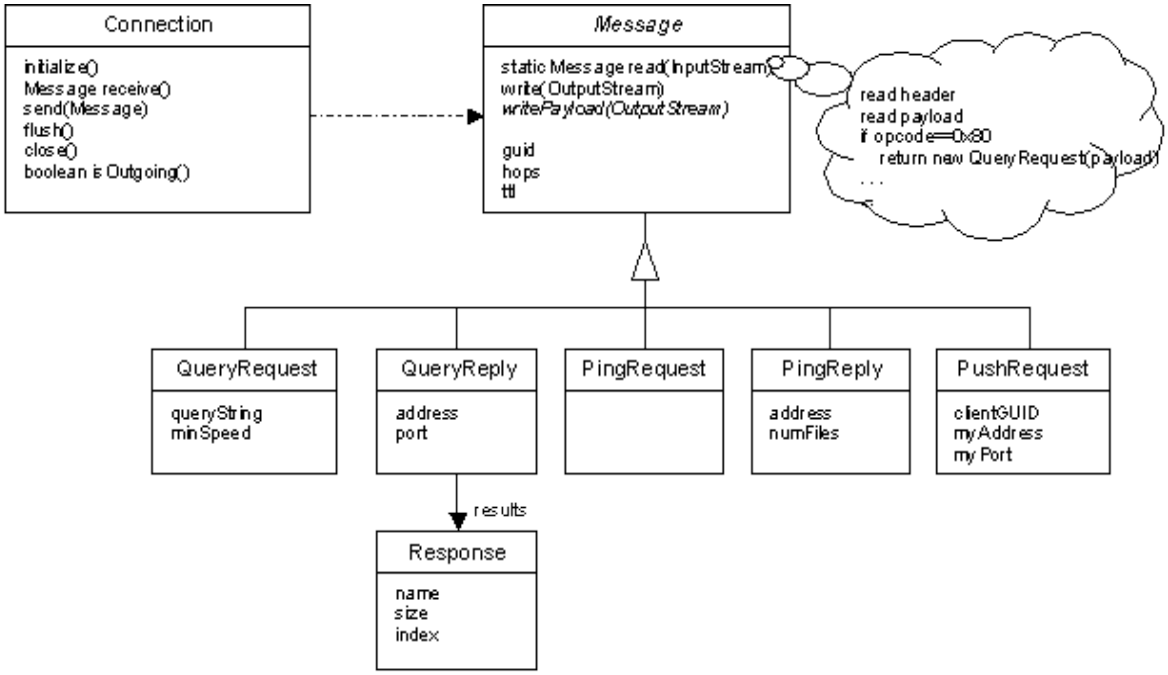

Figura 5.4 - Hierarquia de mensagens do LimeWire. *Fonte (Rohrs, 2000).*

Para facilitar a implementação do protótipo e poder utilizá-lo diretamente na rede Gnutella interagindo com os demais *peers* (tradicionais), optamos por aproveitar a mensagem original *GnutellaQuery* para transportar o pedido de consulta por conteúdo, de forma semelhante ao transporte da consulta pelo nome do arquivo. Portanto, o nosso protótipo não faz distinção entre uma mensagem de consulta tradicional e uma de consulta por conteúdo. O resultado esperado será um conjunto de arquivos compartilhados, cujos nomes contém o parâmetro de consulta e, cujos conteúdos contém tal parâmetro.

A classe do LimeWire responsável pelo envio de mensagens para a rede Gnutella é a *RouterService.java.* Tal classe não precisou ser alterada com a decisão de usar a mensagem *GnutellaQuery* para o transporte da consulta por conteúdo.

Considerando a propagação de mensagens pela rede Gnutella, apenas um grupo de peers receberá a consulta. Tal passo será comentado na subseção seguinte.

5.4.3. Recebimento da mensagem de consulta

Um *peer* conectado na rede Gnutella pode receber uma mensagem de consulta a qualquer instante. Como apresentado na subseção anterior, as mensagens de consulta Gnutella tem a identificação 0x80 (*Query).*

A classe do LimeWire responsável pelo recebimento de mensagens da rede Gnutella é a *RouterService.java.* Tal classe não precisou ser alterada, com a decisão de usarmos a mensagem *GnutellaQuery* para o transporte da consulta por conteúdo.

Entretanto, precisamos estender o LimeWire para ele poder acionar a camada Lucene para efetuar uma consulta pelo conteúdo dos documentos textuais da coleção. Isto será apresentado na subseção seguinte.

5.4.4. Busca local e apuração do resultado

Após o tratamento da mensagem Query, o valor do campo Critério de Pesquisa é usado como parâmetro de pesquisa na coleção de documentos textuais compartilhados. O *peer* deverá proceder a consulta em três partes: a) consulta do LimeWire, b) consulta do Lucene, c)

integração dos resultados das consultas do LimeWire e do Lucene.

O LimeWire utiliza um algoritmo de casamento de *strings*[53] para escolher os arquivos que atendem ao critério da consulta. O *peer* fará em seguida a consulta ao conteúdo dos documentos textuais compartilhados. Para realizar consultas textuais, o *peer* aciona as classes do Lucene, para ler os índices criados previamente.

As buscas no LimeWire, originalmente, são processadas pela classe *FileManager,* do pacote *com.limegroup.gnutella,* que as realizam através do método *Search,* comparando a *query* recebida com o nome dos arquivos compartilhados.

Para cada documento compartilhado é atribuído um identificador inteiro. Ao realizar a busca, o *FileManager* armazena os indicadores dos arquivos que satisfazem a consulta usando um objeto chamado *IntSet*. Este objeto traz uma coleção de inteiros e tem como uma de suas características o armazenamento de intervalos de resultados. Por exemplo, um resultado que retornasse os arquivos 5, 6, 7 e 8 são armazenados em uma posição do *Intset* como o intervalo [5-8].

A consulta no Lucene começa a partir deste ponto. Fizemos a extensão do código do LimeWire para que a busca contemplasse o conteúdo dos documentos através do uso do índice Lucene. Neste ponto, fizemos uma inserção no código do método *search* para que incluísse na sua lista de resultados o processamento da busca realizada pelo Lucene. Acrescentamos ao método *search* da classe *FileManager* uma chamada ao método *searchLucene,* também inserido nesta classe. O código-fonte de *SearchLucene* encontra-se no anexo A.

A terceira etapa do processo de consulta é a integração dos resultados das consultas do LimeWire e do Lucene. De posse da lista de arquivos, chamamos o método *getFileDescForFile* que recebe um *path* como parâmetro e retorna um objeto *FileDesc* que é usado para a representação de um arquivo no LimeWire. Finalmente, para cada *FileDesc* utilizamos o método *getIndex* para retornar o identificador do documento compartilhado. Esses identificadores são armazenados em uma coleção de objetos do tipo *IntSet*, que é retornado como resultado de *searchLucene*.

As duas listas de resultados, a lista proveniente da busca por *string* e da busca realizada pelo método *searchLucene* são concatenadas em uma só lista através dos método *addAll* da

[53] http://www.limewire.org/techdocs/KeywordMatching.htm

classe *IntSet*, sendo desprezados os resultados repetidos.

No protótipo, tanto o resultado da busca nativa do LimeWire quanto o resultado apurado na consulta do Lucene são enviados de volta para o *peer* solicitante. Isso será apresentado na seção seguinte.

5.4.5. Envio da resposta

O resultado da consulta realizada nos índices Lucene do *peer* contém a lista dos documentos que satisfazem ao critério de pesquisa recebido. Esse resultado é manipulado através da classe Lucene *Hits*. O resultado é ordenado, por padrão, pelo escore do Lucene.

A classe do LimeWire que gerencia os arquivos compartilhados do usuário é a *FileManager.java.* Esta classe possui diversos métodos agrupados em *Search, Utility, Accessors*, e *Loading*. A classe sofreu alterações para poder atender ao objetivo do protótipo.

Internamente, tanto o Lucene quanto o LimeWire, atribuem identificações *(IDs)* para os arquivos da coleção, no caso do Lucene (Gospodenetic; Hatcher, 2005), e para os arquivos compartilhados, no caso do LimeWire (LimeWire, 2006b). Os identificadores não são padronizados ou mesmo compartilhados por eles. Assim, um arquivo da pasta compartilhada terá dois identificadores, um para o Lucene e outro para o LimeWire.

Desta forma, uma vez identificado um arquivo no *Hits* Lucene será possível encontrar o seu respectivo identificador *(Id)* no LimeWire. Para isto, modificamos *FileManager.java* incluindo o método *SearchLucene,* que consulta os índices Lucene, e para cada arquivo encontrado o mesmo é passado para *getFileDescForFile* de *FileManager.java* para poder pegar o seu identificador no LimeWire. O método *Search* de *FileManager.java* e *SearchLucene* foram listados no anexo A.

O identificador do arquivo é suficiente para o *peer* localizar todas os dados do arquivo. O identificador de um arquivo indexa uma matriz que indica outras características do arquivo, como por exemplo o seu tamanho, tipo e seus metadados. Tais características são exibidas ao usuário na interface do LimeWire, dentro do painel de resultado da consulta.

Nativamente, o LimeWire utiliza duas classes para o envio da resposta: *QueryReply e Response.* Essas classes mapeiam partes das mensagem Gnutella *QueryHits.* No conteúdo da

mensagem, estarão os identificadores dos arquivos que satisfazem o critério de pesquisa do usuário, recuperados pelo Lucene, em sua classe *Hits,* além dos resultados encontrados pelo LimeWire.

A mensagem Gnutella *QueryHit* (0x81) possui a seguinte estrutura (Klingberg; Manfredi, 2002):

Bytes	***Descrição***
0	Número de hits encontrados na coleção e presentes na resposta;
1-2	Porta. O número da porta que o *peer* que enviou a resposta utiliza para receber requisições HTTP, para a transferência de arquivos;
3-6	Endereço IP do *peer* que enviou a resposta;
7-10	A velocidade de conexão do *peer* que enviou a resposta (em kb/s);
11-	Lista do resultado da respectiva consulta.

Esse resultado contém tantas ocorrências quanto as presentes nos *Hits* Lucene, além das ocorrências encontradas pelo LimeWire. Cada ocorrência possui a seguinte estrutura:

Bytes	***Descrição***
0-3	Identificador do arquivo para o LimeWire;
4-7	Tamanho do arquivo;
8-	Nome do arquivo;
X	Blocos de extensão que podem ser do tipo HUGE[54], GGPE[55] e metadados em texto plano.

O conjunto de identificadores (*IntSet*) representa os arquivos do *peer* que atendem ao critério de pesquisa. O conjunto é passado para a classe *RouterService.java* que cuida do envio da mensagem para a rede Gnutella. Tal classe não precisou de alterações para se adequar ao protótipo. Na seção seguinte, veremos como a resposta é recebida pelo *peer* solicitante.

[54] *Hash URN Gnutella Extension – HUGE*
[55] *Gnutella Generic Extension Protocol - GGEP*

5.4.6. Recebimento da resposta

A exibição do resultado da consulta para o usuário do *peer* solicitante se dá quando o *peer* solicitante recebe a mensagem Gnutella *QueryHit* da rede. Tal mensagem conterá a lista dos arquivos do *peer* (remetente da resposta) que atendem ao critério da pesquisa informado pelo usuário.

Uma vez que o *peer* solicitante poderá receber respostas de vários outros *peers,* o LimeWire deverá consolidar as respostas parciais na interface gráfica, montando a resposta total. A consolidação dos resultados parciais dos *peers* é feito através do algoritmo de agrupamento, descrito em (Rohrs, 2001).

A lista dos resultados é apresentada na interface do LimeWire para o usuário decidir se solicitará ou não o *download* de algum arquivo. Após o recebimento da resposta, o usuário poderá realizar o *download* do arquivo desejado a partir do(s) *peer(s)* detentor(es). Para isso, o LimeWire possui a classe *FileManager* que é responsável por todo o gerenciamento dos arquivos compartilhados do *peer.*

As classes *DownloadManager* e *UploadManager* gerenciam todas as operações e eventos envolvidos no *download* e no *upload* de arquivos, respectivamente. Nessa parte, não precisaremos intervir, uma vez que o LimeWire já realiza *downloads e uploads* de arquivos, inclusive de forma concorrente. Maiores detalhes sobre *downloads e uploads* no LimeWire poderão ser encontrados em (Christopher, 2002).

Na seção seguinte, apresentaremos o cenário e os testes realizados com o protótipo.

5.5 Testes preliminares

O cenário dos testes realizados com o protótipo foi composto de dois computadores, um notebook e um PC Desktop, interligados em rede local e com o acesso à Internet. Nos dois computadores rodavam o nosso protótipo.

No na pasta compartilhada do notebook foram criados alguns arquivos texto (.txt) com os nomes “Texto1.txt”, “Texto2.txt”, “Texto3.txt” e “Texto4.txt”. O arquivo “Texto1.txt” é composto por um conjunto de palavras aleatórias, entre elas a palavra “semantic”. Tal palavra

estava presente apenas neste arquivo. Nenhum dos arquivos texto da pasta compartilhada do notebook continha a palavra “texto”. A figura 5.5 mostra o conteúdo dos arquivos no notebook.

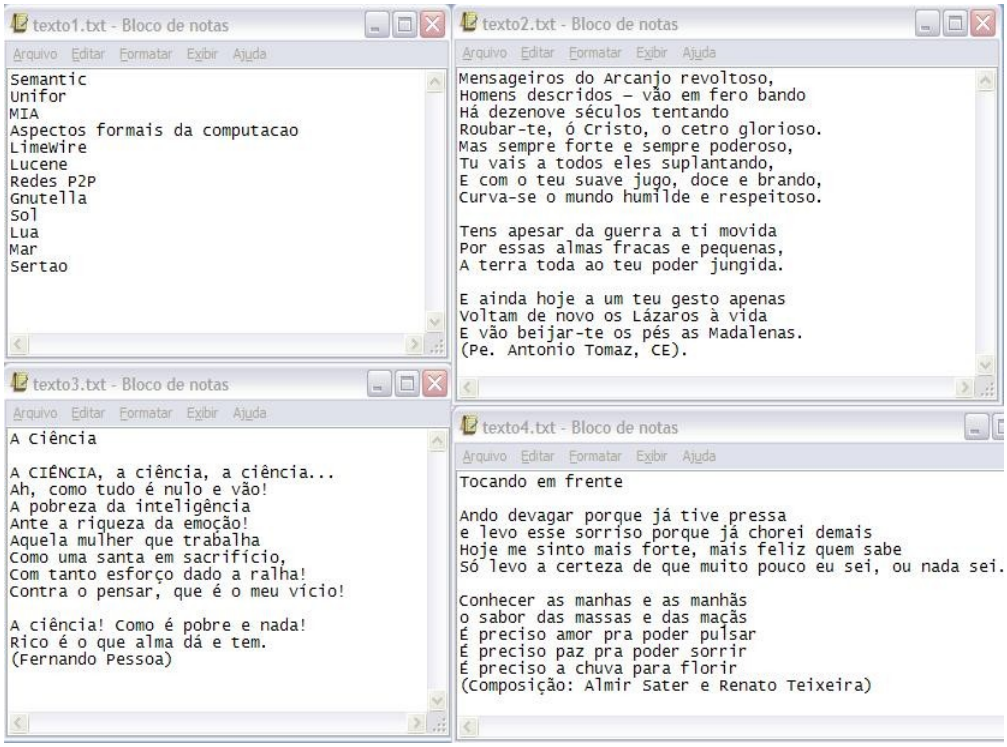

Figura 5.5 – Conteúdo dos arquivos compartilhados por notebook.

No na pasta compartilhada do PC Desktop foram criados dois arquivos: “Documento1.txt” e “Documento2.txt”. O arquivo “Documento1.txt” é composto por um conjunto de palavras aleatórias, entre elas a palavra “semantic”. Tal palavra estava presente apenas neste arquivo. Nenhum dos arquivos texto da pasta compartilhada do PC Desktop continha a palavra “texto”. A figura 5.6 mostra o conteúdo dos arquivos do PC.

Figura 5.6 – Conteúdo dos arquivos compartilhados por PC Desktop.

O protótipo do Desktop foi iniciado e após a sua conexão à rede Gnutella iniciamos o protótipo rodando no notebook. Com ambos computadores conectados à rede Gnutella, foram elaborados três experimentos iniciais descritos a seguir.

Teste 1. Procedemos uma consulta através da interface gráfica do Desktop. O parâmetro de consulta escolhido foi a palavra “texto”. Através do monitor de consultas do LimeWire, pudemos constatar o recebimento do parâmetro de consulta por parte do notebook. A consulta foi respondida pelo notebook informando os quatro arquivos “Texto1.txt”, “Texto2.txt”, “Texto3.txt” e “Texto4.txt”, aparecendo como resposta na interface do LimeWire do Desktop, indicando “Ethernet” como o tipo de conexão de notebook. A figura 5.7 ilustra o resultado.

O resultado apresentado no primeiro teste demonstrou que o protótipo respondeu corretamente, segundo a operação tradicional do LimeWire, ou seja, realizando consultas pelo nome (ou parte) dos arquivos compartilhados. De fato, conforme foi implementado, o protótipo também realizou a consulta de “texto” nos índices Lucene, mas não havia nenhum arquivo com contivesse o parâmetro.

Teste 2. Foi submetida uma consulta com a palavra “semantic”, a partir do Desktop. A consulta foi recebida pelo notebook, conforme mostra o seu monitor de consultas recebidas. O notebook respondeu e no Desktop pudemos observar o arquivo “Texto1.txt” como resultado da consulta, indicando “Ethernet” como o tipo de conexão do notebook. A figura 5.8 ilustra o resultado.

O resultado apresentado mais uma vez está correto. Desta vez, nenhum arquivo da pasta compartilhada do notebook possuía nome contendo “semantic”, mas o arquivo “Texto1.txt”

continha a palavra “semantic” no seu conteúdo.

Teste 3. Foi submetida uma consulta a partir do PC Desktop, tendo como critério de pesquisa “almir sater”. A consulta foi recebida pelo notebook, conforme demonstrou o seu monitor de consultas. O PC Desktop recebeu a resposta mostrando o arquivo “Texto4.txt” como resultado, indicando “Ethernet” como tipo da conexão de notebook. A figura 5.9 ilustra o resultado.

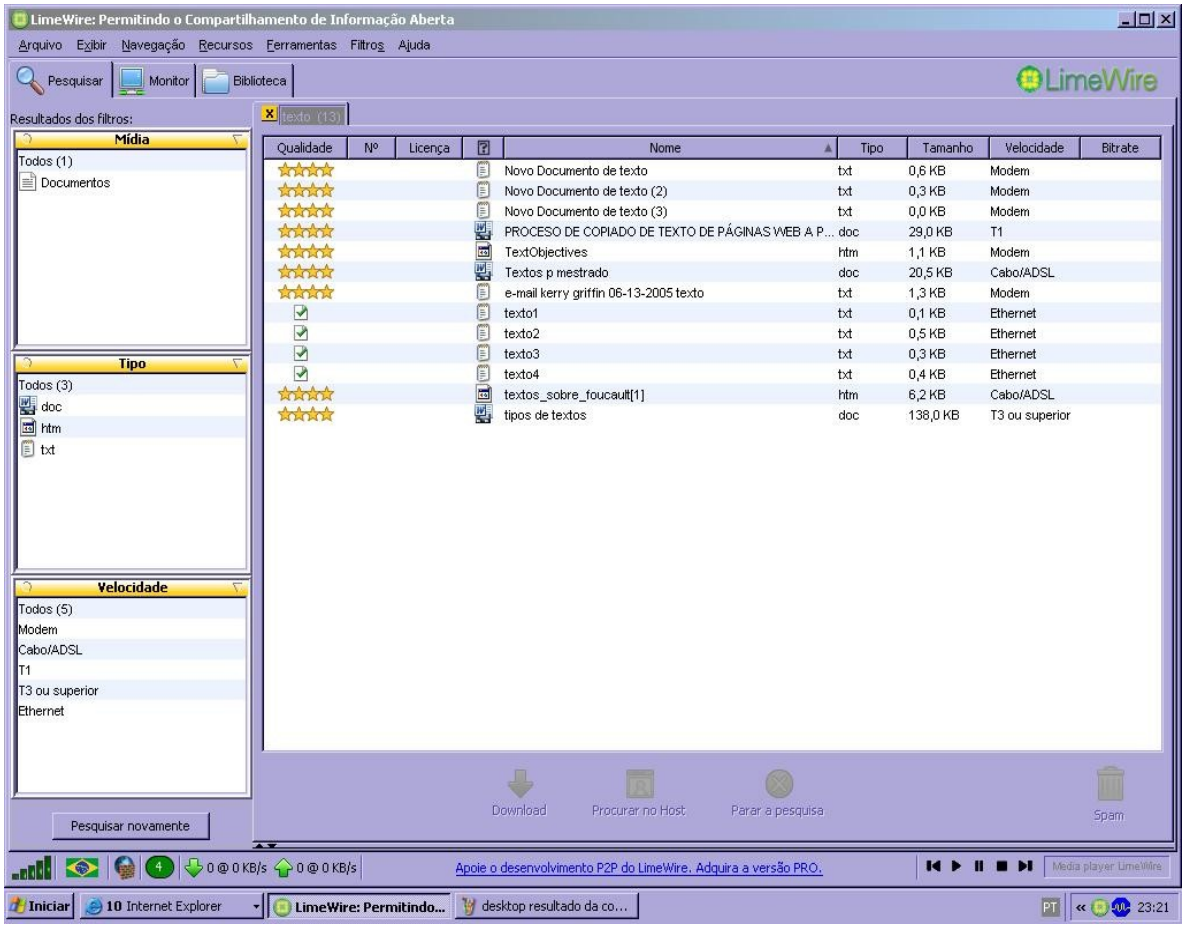

Figura 5.7 – Em PC Desktop, resultado da consulta por “texto”.

O resultado apresentado mais uma vez está correto. Nenhum arquivo da pasta compartilhada do notebook possui nome que tivesse “almir sater” como parte, mas o arquivo “Texto4.txt” possui tal *string* no seu conteúdo.

Em seguida, foi realizada uma outra consulta, utilizando o mesmo critério (“almir sater”), mas, desta vez, foi solicitado ao protótipo apenas arquivos de música (*audio files*). A

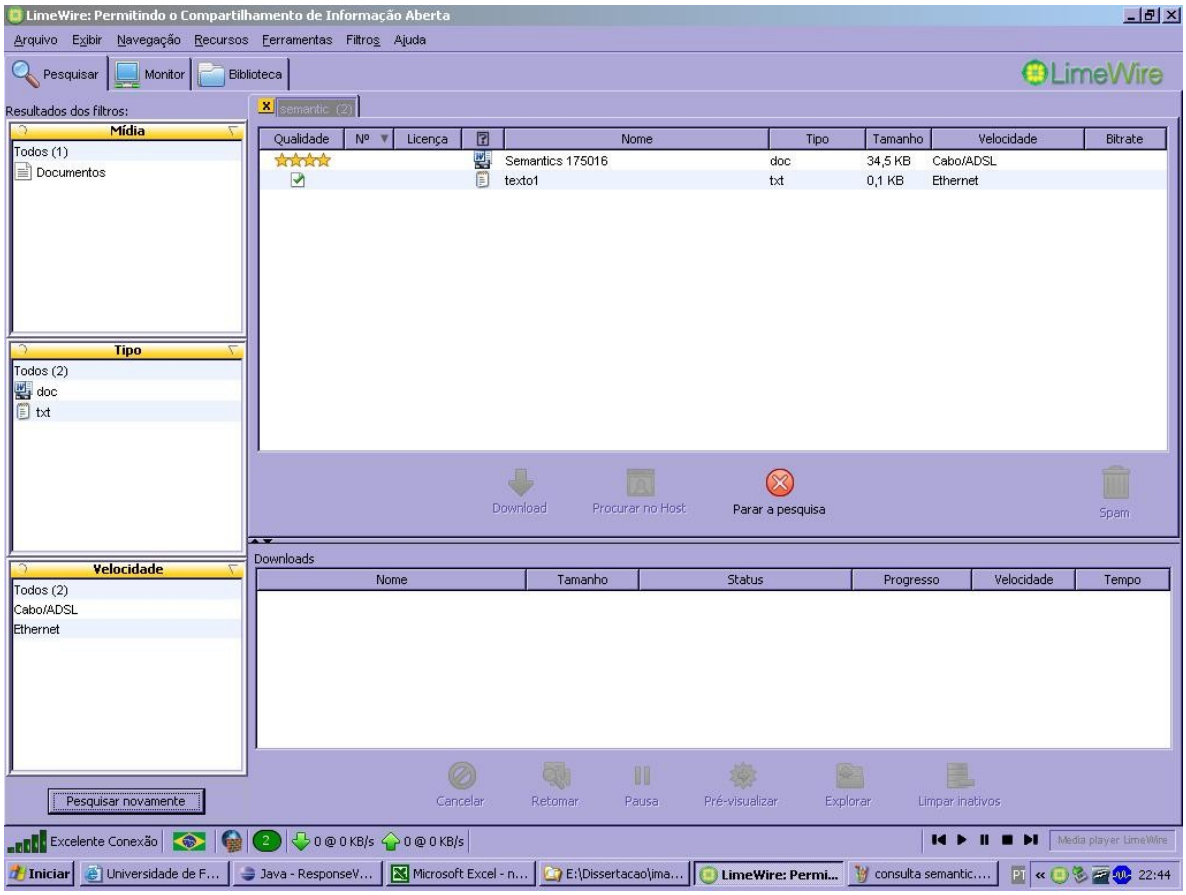

figura 5.10 ilustra o resultado.

Figura 5.8 – Em PC Desktop, resultado da consulta por “semantic”.

O resultado mais uma vez está correto. Nenhum arquivo textual da pasta compartilhada do notebook foi apresentado como resposta no Desktop, uma vez que foi solicitado apenas arquivos de música.

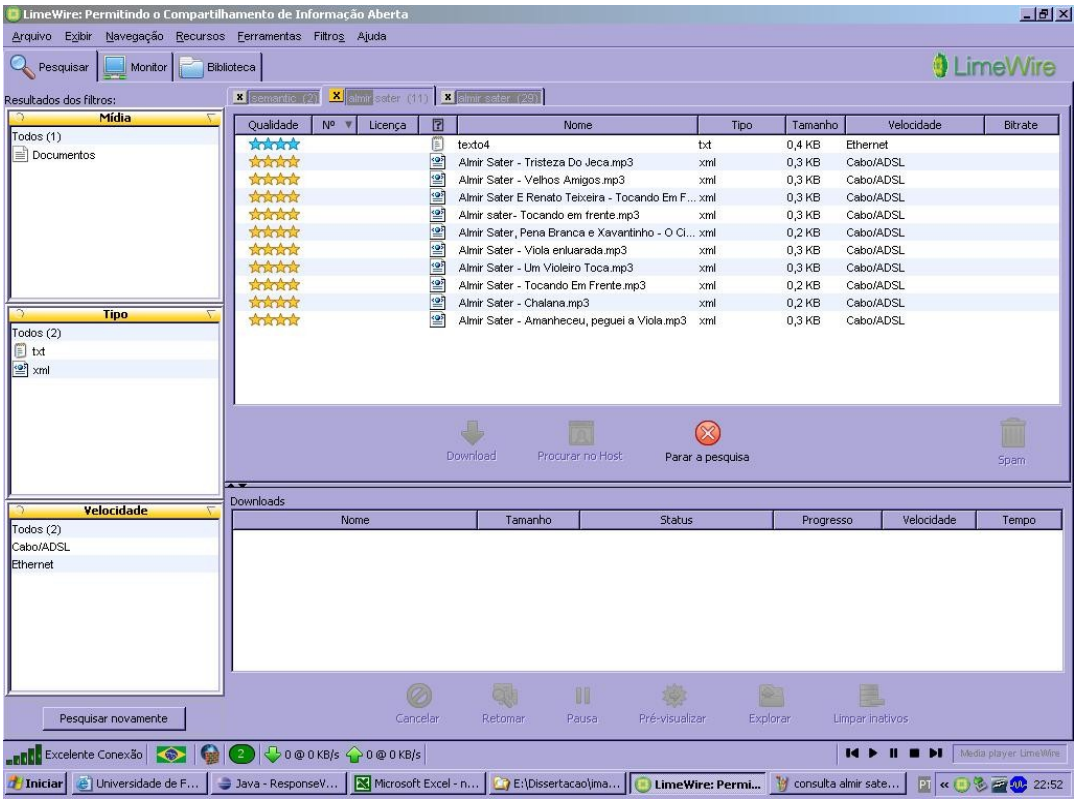

Figura 5.9 – Em PC Desktop, resultado da consulta por “almir sater”.

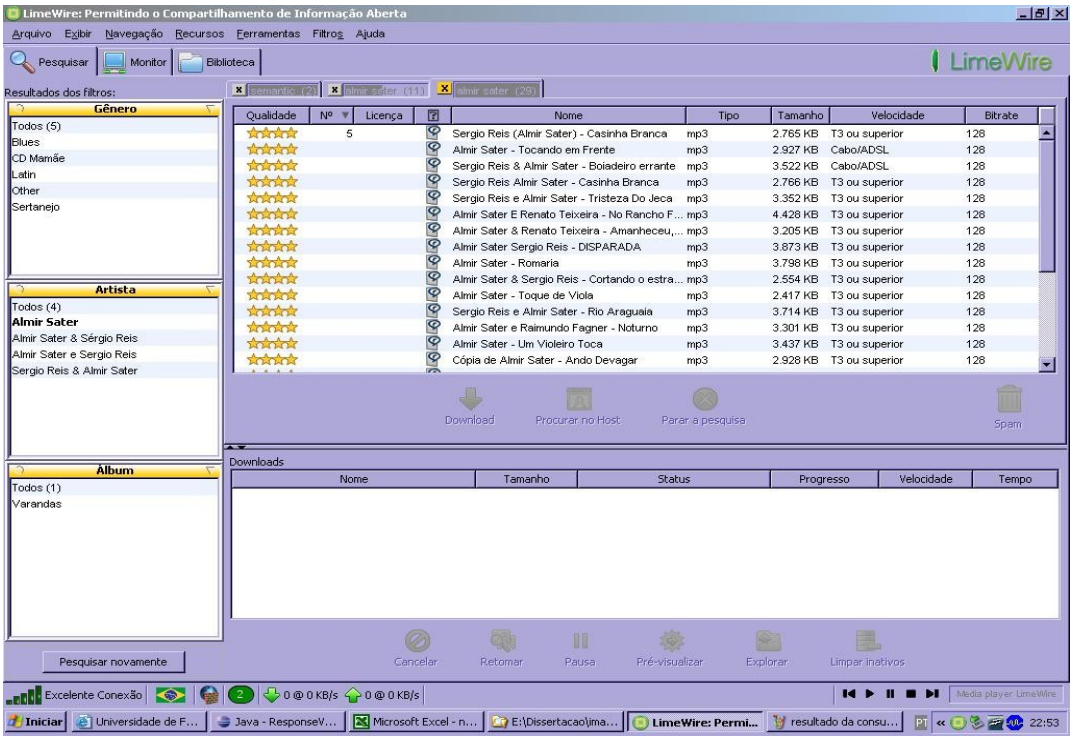

Figura 5.10 – Em PC Desktop, resultado da consulta por “almir sater” (*audio files*).

Observamos que as respostas de notebook e de PC Desktop sempre chegavam primeiro do que as respostas de outros peers da rede. Atribuímos isto a maior velocidade de conexão entre o notebook e o Desktop, porque estavam em rede local, enquanto os demais peers estavam dependentes das conexões com a Internet.

5.6 Sumário do capítulo

Neste capítulo, descrevemos uma série detalhes da implementação do protótipo como prova de conceito da arquitetura que propomos para o problema da localização e recuperação de documentos pelo o seu conteúdo em redes P2P.

Apresentamos as alterações e extensões necessárias à integração dos produtos LimeWire e Lucene considerando o funcionamento desejado da arquitetura. Assim, comentamos como se dá a entrada do *peer* na rede Gnutella, a indexação dos documentos textuais da pasta compartilhada do *peer,* o processo de consulta.

Dividimos o processo de consulta para facilitar o entendimento da nossa proposta. Assim, o processo possui as seguintes partes: recebimento do valor de pesquisa, preparação e envio para a rede de mensagem de consulta por conteúdo e o seu recebimento por um outro *peer*, busca local com a apuração do resultado, formatação da resposta com o resultado parcial (do *peer)* como mensagem de volta ao solicitante, e o recebimento desta e a apresentação ao usuário.

Realizamos um pequeno conjunto de testes entre dois computadores (um notebook e um PC Desktop), conectados na rede Gnutella, para apurar o correto funcionamento do protótipo criado, como prova de conceito de arquitetura proposta.

No capítulo seguinte, apresentaremos algumas conclusões iniciais, alguns resultados preliminares e algumas sugestões de pesquisas futuras.

Capítulo 6 – Conclusão

Este capítulo apresenta a conclusão do trabalho, destacando suas contribuições, principais resultados obtidos e sugestões para pesquisas futuras.

As redes P2P representam um paradigma na computação distribuída, especialmente quando comparadas ao modelo cliente-servidor clássico. As redes P2P ganharam muitos adeptos, em escala planetária, desde o seu surgimento com o Napster. Muitas pesquisas surgiram transformando uma rede de compartilhamento de arquivos de música para uma plataforma para diversas aplicações.

A utilização de redes P2P para distribuição e localização de conteúdo fez surgir sistemas distribuídos de recuperação de informação. A exemplo da *web*, nesses sistemas, a quantidade e o tamanho das coleções, sejam centralizadas ou totalmente distribuídas, pode variar e impor desafios nos processos de distribuição e recuperação de informação.

Nosso principal objetivo com a pesquisa nessa área foi propor uma solução para o problema de localização de documentos textuais pelo seu conteúdo em redes P2P. Essa lacuna foi observada nos programas de compartilhamento de arquivos populares e vem sendo alvo de pesquisas que unem técnicas de recuperação de informação sobre infra-estrutura de redes P2P. Alguns exemplos são as referências (Risson; Moors, 2004), (Cuenca-Acuna; Nguyen, 2001), (Qin et al, 2002), (Ratnasamy et al, 2001), (Lu; Callan, 2003).

A fim de definir o escopo do problema que pesquisamos, identificamos um conjunto de requisitos funcionais básicos para que um *peer* tivesse a capacidade de localizar e recuperar arquivos textuais através de consultas pelo o seu conteúdo.

Desenhamos uma arquitetura para atender aos requisitos identificados e realizamos a construção de um protótipo, como prova de conceito da arquitetura proposta. O protótipo foi resultado da integração entre o LimeWire e a API Lucene. Com o protótipo, realizamos experimentos iniciais para verificar o seu funcionamento.

6.1 Contribuições e resultados preliminares

As contribuições do nosso trabalho foram a criação de uma arquitetura que permitisse a realização de consultas pelo conteúdo de documentos compartilhados em redes P2P, a construção de um protótipo para a prova de conceito do funcionamento da arquitetura proposta e a realização de experimentos iniciais.

Os resultados experimentais preliminares, obtidos utilizando os produtos e o protótipo inicial na forma de prova de conceito, demonstraram ser possível localizar e recuperar documentos textuais pelo o seu conteúdo em redes P2P.

Portanto, podemos concluir que o uso da indexação textual em redes P2P possibilita uma forma de recuperação de documentos mais eficiente porque permite que arquivos espalhados entre os diversos *peers* da rede, que não eram retornados às consultas tradicionais, possam compor o resultado.

Por outro lado, é esperado que o processo de localização adotado na nossa proposta provoque um aumento no consumo de banda de rede, provocado pelo tamanho das respostas em consultas pelo conteúdo dos documentos sobre uma rede que propaga mensagens por difusão e pela forma como o protótipo foi implementado. Além disso, requer uma pequena alteração nos papéis dos nós-folha e dos nós-*ultrapeers.* Da forma como implementamos o protótipo, as respostas são obtidas pela composição dos resultados da busca tradicional (pelo nome dos arquivos) e pelo conteúdo dos documentos. A nova modalidade de consulta impõe uma mudança no papel dos *peers* do Gnutella.

A possibilidade de inundação da rede P2P advém da forma como as consultas são lançadas na rede e do recebimento de respostas que poderão conter vários *hits* de vários *peers.* O protocolo de roteamento de consultas utilizado no Gnutella v0.6 restringe a propagação de consultas aos *ultrapeers,* que por sua vez só encaminham uma consulta aos *peers* conectados a ele quando há chance desses *peers* responderem positivamente à consulta.

A motivação por trás da existência dos *ultrapeers* é reduzir o consumo de banda e assim permitir que rede possa escalar melhor em nível mundial. Por outro lado, os *peers* têm forte potencial para armazenar documentos que podem ser de interesse para outros usuários da rede P2P, mas as formas de consulta disponíveis não permitem a localização de arquivos textuais pelo seu conteúdo. Por isso, dentre as alternativas que identificamos, optamos por estender o

papel dos *peers e ultrapeers* para lidarem com consultas baseadas no conteúdo.

As outras alternativas que identificamos foram:

a) Permitir que os *peers* encaminhassem consultas diretamente aos outros *peers*: essa alternativa teria grande viabilidade se estivéssemos utilizando a versão 0.4 do Gnutella. Nessa versão, não há a presença dos *ultrapeers* e as consultas são realizadas diretamente entre *peers,* provocando inundação na rede, o que vai de encontro a visão evolutiva do Gnutella que busca melhor escalabilidade e desempenho. Além disso, o LimeWire, software escolhido como base para a criação do protótipo, utiliza a versão 0.6 do Gnutella. Outra opção seria adotar outro software P2P que fosse aderente a versão 0.4. Não encontramos software que fosse robusto, de código aberto e capaz de substituir o LimeWire. Entretanto, essa opção nos remeteria ao caminho inverso dos esforços dos desenvolvedores e pesquisadores envolvidos com o Gnutella.

b) Fazer com que o *peer* encaminhasse ao *ultrapeer* o índice textual Lucene dos documentos compartilhados: os *peers* encaminham o índice compactado dos nomes dos arquivos por ele compartilhados para o *ultrapeer* que ele se conecta. Isso faz parte do funcionamento do *Query Routing Protocol – QRP*. Desta forma, se alterássemos o funcionamento do *peer* para ele poder enviar os índices textuais, produzidos pelo Lucene, para o *ultrapeer,* incorreríamos nos seguintes problemas: forte possibilidade de maior utilização de rede por causa do envio e atualização dos índices textuais, que tendem a ser maiores que os índices dos nomes dos arquivos e; forte chance dos índices se tornarem obsoletos, já que alguns *ultrapeers* poderiam passar a ser *peers* e vice-versa.

Para amenizar esse eventual impacto no consumo da rede, poderemos adotar algumas medidas no sentido de limitar o tamanho das respostas, comentados na seção seguinte.

Tivemos várias dificuldades durante a elaboração do trabalho, entre elas podemos citar:

- A documentação da arquitetura do LimeWire é resumida, dando apenas uma visão geral do funcionamento do sistema. Encontramos vários pontos da documentação que estavam desatualizados;
- Algumas instruções contidas nos tutoriais de instalação do código-fonte não correspondiam à realidade, como por exemplo, a versão da linguagem Java que deveria ser usada para que o programa fosse compilado;

- Em razão de filtros de *spam* utilizados pelo LimeWire, em alguns momentos as consultas pelo conteúdo não eram apresentadas pelo *peer solicitante*, embora fosse processado no *peer solicitado*. Após a identificação da origem do problema, desligamos os filtros de *spam* utilizados para que os resultados fossem exibidos com sucesso;
- O código-fonte do LimeWire é extenso. O código-fonte, depois de baixado, descompactado e configurado no Eclipse, possui aproximadamente 110Mb. Localizar as classes e os pontos de alteração exigiu uma varredura em boa parte do seu código consumindo várias horas.

Apesar do protótipo construído ter atendido aos requisitos iniciais da arquitetura, há limitações na atual *release,* causadas por decisões de implementação e por limitação de tempo, que relacionamos a seguir:

- Não há possibilidade de permitir ao usuário a ativação/desativação do recurso de indexação de arquivos;
- O processo de indexação ocorre toda vez que o protótipo é inicializado, mesmo se a coleção de documentos não tiver sido alterada;
- Não há possibilidade de permitir ao usuário a escolha de usar ou não a procura pelo conteúdo do documento;
- Todas as consultas estão sendo feitas com utilização da busca pelo índice Lucene, mesmo que o tipo de arquivo solicitado seja, por exemplo, arquivo de música;
- O índice dos arquivos compartilhados somente é criado ou atualizados no momento da inicialização do programa. Assim, qualquer alteração na coleção durante a execução do programa não serão cobertas pelo índice, fazendo com que as informações contidas nos arquivos possam ser desatualizadas e portanto retornar resultados de buscas errados;
- O processo de indexação pode consumir muito tempo, dependendo do número de arquivos a ser indexado e do tamanho dos mesmos.

6.2 Pesquisas futuras

Vislumbramos várias oportunidades de melhorias e extensões do trabalho reportado nesta dissertação. Dentre elas, destacamos:

- Ajustes na interface gráfica do LimeWire para o usuário comandar, explicitamente, a consulta pelo conteúdo de documentos, através de um *check box* no seu painel de consulta. Dessa forma, nem todas as consultas realizadas seriam por conteúdo, permitindo assim a utilização nativa do Gnutella versão 0.6 e a economia de recursos computacionais e de rede.
- Imposição de limites para o tamanho do resultado (*hits*) por *peer*: os *peers* teriam um limite de *hits* por consulta, por exemplo até 10 (dez) ou 20 (vinte) ocorrências na lista resultado. Os arquivos seriam relacionados descrescentemente pelo *score* Lucene. Assim, os arquivos com maior relevância com o termo da consulta seria apresentados primeiro. A rede seria poupada do transporte de listas resultado maiores mas que representariam menor (ou nenhuma) relevância com o consulta.
- Redução do *Time To Live – TTL* das mensagens Gnutella de consultas por conteúdo: poderia haver redução do valor do campo TTL nas mensagens de pedido de consulta pelo conteúdo dos arquivos da coleção, mensagem Gnutella tipo 0x70 - *QueryByContent*, proposta por nós. As mensagens com TTL menor ficam menos tempo na rede P2P e também são propagadas em menor extensão.
- Dar continuidade à implementação enriquecendo o protótipo com uma melhor nível de integração entre os componentes-chave da arquitetura: o software LimeWire P2P e a API Lucene. Algumas melhorias poderiam ser: realizar a indexação e a atualização dos índices acionada diretamente pela classe *FileManager* do LimeWire, gerenciadora dos arquivos compartilhados, para quando o arquivo mude de conteúdo, ou seja excluído ou ainda quando um novo arquivo textual surja na pasta; ampliar os tipos de arquivos textuais indexados (PDFs, DOC, XML, HTML, RTF, PPT etc.); estender os metadados utilizados pelo LimeWire para incorporar alguns campos de indexação utilizados no Lucene.
- Realizar novos experimentos para investigar sistematicamente as vantagens e limitações da adoção da arquitetura em cenários específicos, como em redes empresariais, ambientes acadêmicos e na própria Internet.
- Avaliar a arquitetura quanto aos parâmetros *precision e recall* apurando números

quanto a qualidade dos resultados das consultas.

- Estender o esquema de consulta para suportar linguagens de consultas semanticamente mais ricas.
- Evoluir a arquitetura proposta para utilizar índices semânticos com a técnica LSI, o que poderia permitir a realização de consultas com resultados por contexto. Outra abordagem para a melhoria semântica seria a adoção de ontologias para representar conceitos e assim mapear contextos de forma mais natural.
- Evoluir a arquitetura para operar em redes P2P estruturadas, como Chord (Chord, 2006) e CAN, para dotá-las da capacidade de consultar documentos pelo conteúdo, ou com maior requinte semântico. Isso permitiria o uso mais eficiente do que em redes P2P baseadas em inundação.
- Organização dos *peers* segundo o seu interesse de consulta ou do seu conteúdo. Isso permitiria aproximar logicamente os *peers* que detém e compartilham documentos de um mesmo domínio de conhecimento. Esta proximidade diminuiria o tempo de resposta das consultas e tenderia a melhorar a qualidade das respostas porque apenas um conjunto de *peers,* logicamente próximos e que detém documentos de interesse ao *peer* solicitante.

Referências bibliográficas e bibliografia

(Adar e Huberman, 2000) ADAR, Eytan; HUBERMAN, Bernado A. Free Riding on Gnutella. First Monday, volume 5, number 10, October 2000. Disponível em: <http://firstmonday.org/issues/issue5_10/adar/index.html>. Acesso em: janeiro/2006.

(ADS, 2006) Ask Desktop Search, 2006. Disponível em: <http://ask.com/?tool=des>. Acesso em: abril/2006.

(Cachelogic, 2006) CACHELOGIC. Understanding peer-to-peer: history and background. Disponível em: http://www.cachelogic.com/p2p/p2phistory.php . Acesso em: fevereiro/2006.

(Callan et al., 1995) J. P. Callan, W. B. Croft, and J. Broglio. TREC and TIPSTER experiments with INQUERY. Information Processing & Management, 31(3):327--343, May/June 1995. <http://citeseer.ist.psu.edu/callan94trec.html>. Acesso em: janeiro/2006.

(CDS, 2006) Copernico Desktop Search, 2006. Disponível em: <http://copernic.com>. Acesso em: abril/2006.

(Chakrabarti, 2003) CHAKRABARTI, S. Mining the web – Discovering knowledge from hypertext data, 2003. Capítulos 1 e 3. p. 1-13, 45-76. Indian Institute of Technology, Bombay.

(Chord, 2006) Projeto CHORD: Site oficial. Disponível em <http://pdos.csail.mit.edu/chord/> . Acesso em: dezembro/2006.

(Christopher, 2002) Christopher, R. LimeWire Download Code, novembro 2002. Disponível em: <http://www.limeodelwire.org/techdocs/downloads.htm>. Acesso em: abril/2006.

(Client-server, 2006) Modelo cliente-servidor, Wikipedia. Disponível em: <http://en.wikipedia.org/wiki/Client-server>. Acesso em: janeiro/ 2006.

(Cuenca-acuna et al., 2001) Cuenca-acuna F. M., Perry C., Martin R.P., Nguyen T. D. PlanetP: infrastructure support to P2P information sharing, 2001. Techcnical Report DCS-TR-465. Novembro 2001. Disponível em: <http://www.panic-lab.rutgers.edu/Research/planetp/Publications/DCS-TR-465-PlanetP/>. Acesso em: abril/2006.

(Cuenca-Acuna; Nguyen, 2001) Cuenca-Acuna, F.; Nguyen, T. Text-based content search and retrieval in ad hoc P2P communities, 2001. Disponível em: <http://www.panic-lab.rutgers.edu/Research/planetp/Publications/content_search_on_

P2P_02(FINAL)/paper.html>. Acesso em: abril/2006.

(Dingledine et al., 2001) Dingledine, R.; Freedman, M.; Rubin, A.; Oram, A. Chapter 12 - Free Haven, 2001. In:Peer-to-Peer: Harnessing the Power of Disruptive Technologies. p. 159-187.

(EDonkey, 2006) Edonkey: site oficial, 2006. Disponível em: <www.edonkey2000.com> . Acesso em: janeiro/2006.

(Edutella, 2006) Projeto Edutella: Site oficial. Disponível em http://sourceforge.net/projects/edutella/. Acesso em: dezembro/2006.

(Emule, 2006) Emule: site oficial, 2006. Disponível em <www.emule-project.net>. Acesso em: janeiro/2006.

(Farias e Furtado, 2004) FARIAS, Pedro Porfírio Muniz; FURTADO, Wladimir Maia. CABO-P2P Compartilhamento de arquivos baseado em ontologias sobre infra-estrutura P2P. In: WebMedia 2004, 2004, Ribeirão Preto. WebMedia - PhD & MSc Research Reports, 2004.

(Farias e Furtado, 2005a) FARIAS, Pedro Porfírio Muniz; FURTADO, Wladimir Maia. Indexação Textual em redes P2P. In: Webmedia 2005, 2005, Poços de Caldas. Anais do IX Simpósio Brasileiro de Sistemas Multimia e Web. Poços de Caldas : Sociedade Brasileira de Computação, 2005. p. 201-203.

(Farias e Furtado, 2005b) FARIAS, Pedro Porfírio Muniz; FURTADO, Wladimir Maia. Cabo P2P - Compartilhamento de Arquivos baseados em Ontologias sobre redes P2p. In: SBRC - 23 Simpósio Brasileiro de Redes de Computadores - I Workshop de Peer-to-Peer, 2005, Fortaleza. SBRC - 23 Simpósio Brasileiro de Redes de Computadores - I Workshop de Peer-to-Peer. Fortaleza : Sociedade Brasileira de Computação, 2005. p. 109-118.

(Figueiredo et al., 2003) Figueiredo, Z. Ge, D. R.; Jaiswal S.; Kurose, J.; Towsley, D. Modeling peer-peer file sharing systems, 2003. In : Proceedings of Infocomm 2003.

(Foster; Iamnitchi, 2003) Foster, Yan; Iamnitchi, Adriana. On death, taxes, and the convergence of peer-to-peer and grid computing, 2003. Proceedings of the 2nd International Workshop on Peer-to-peer Systems (IPTPS´2003). Berkeley, CA-US.

(GDS, 2006) Google Desktop Search, 2006. Disponível em: <http://desktop.google.com>. Acesso em: agosto/2005.

(Gnutella, 2005) Gnutella v0.4 Protocol. Disponível em: <http://rfc-gnutella.sourceforge.net/developer/stable/index.html>. Acesso em: dezembro/2005.

(Gospodenetic; Hatcher, 2005) Gospodenetic, O.; Hatcher, E. Lucene in Action: a guide to the java search engine, 2005. Manning Publications. ISBN: 1-932394-28-1. 421p.

(Guerreiro; Macedo, Guerreiro, Camacho; Macedo, J.A. Capítulo 6 – Tecnologias de

2005) Recuperação de Informação na Web. In: WebMedia 2005 - Web e Multimídia: desafios e soluções. Dezembro 2005. PUC Minas – Campus Poços de Caldas, MG-Brasil. p. 167-196.

(Hamra; Felber, 2005) Hamra, A. A.; Felber, Pascal A. Design choices for content distribuition in P2P networks. ACM Sigcomm Computer Communication Review, Volume 35, number 5. October 2005.

(Ingwersen, 1992) Ingwersen, P. Information Retrieval Interaction. London: Taylor Graham, 1992. X, 246p.

(Invisible web, 2006) Invisible web: what it is, why it exist, how to find it, and its inherent ambiguity. In: Finding information on the web: a tutorial. UC Berkeley – Teaching Library Internet Workshop, 2006. Disponível em: <http://www.lib.berkeley.edu/TeachingLib/Guides/Internet/InvisibleWeb.html>. Acesso em: janeiro/2006.

(Jordan, 2001) Jordan, R. Why Gnutella can´t scale - No, really. 2001. Disponível em: <http://www.darkridge.com/~jpr5/doc/gnutella.html>. Acesso em: fevereiro/2006.

(Kan et al, 2001) Kan, G.; Gnutella; GoneSilent.com. Chapter 8 – Gnutella, In: Peer-to-Peer Harnessing the Power of Disruptive Technologies, 2001. p. 94-122.

(Kazaa, 2006) Kazaa: site oficial, 2006. Disponível em <www.kazaa.com>. Acesso em: janeiro/2006.

(Klingberg; Manfredi, 2002) Klingberg,T.; Manfredi, R. Gnutella v0.6 Protocol, 2002. Disponível em: <http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html>. Acesso em: dezembro/2005.

(Lee, 2003) Lee, Jintae. An end-user perspective on file-sharing systems. Communications of the ACM, volume 46, number 2 (2003), pp 49-53.

(LimeWire, 2006a) LimeWire: site oficial, 2006. Disponível em: <www.limewire.com>. Acesso em: janeiro/2006.

(LimeWire, 2006b) LimeWire: Technical Documents, 2006. Disponível em: <http://www.limewire.org/techdocs.shtml>. Acesso em: agosto/2005.

(LionShare, 2004) LionShare Whitepaper, 2004. Disponível em: <http://lionshare.its.psu.edu/main/info/docspresentation/LionShareWP.pdf>. Acesso em: abril/2006.

(Lu; Callan, 2003) Lu, Jie; Callan, Jamie. Content-based retrieval in hybrid peer-to-peer networks. CIKM´03, November 3-8, 2003, New Orleans, Louisiana-US. ACM 1-58113-723-0/03/0011.

(Milojicic et al., 2003) Milojicic, Dejan S.; Kalogeraki,Vana; Lukose, Rajan; Nagaraja, Kiran; Pruyne, Jim; Richard, Bruno; Rollins, Sami; Xu, Zhichen. Peer-to-Peer computing. HP Laboratories Palo Alto. Julho 2003. Disponível em: <http://web.cs.wpi.edu/~goos/Teach/cs4513-

d05/papers/p2p-tutorial.pdf>. Acesso em: janeiro/2006.

(Minar; Hedlund, 2001) Minar, Nelson; Hedlund, Marc. Chapter 1 – A network of peers Peer-to-Peer. In: Harnessing the power of disrruptive technologies. O´Reilly. March/2001.p. 3-20. Disponível em: <http://www.oreilly.com/catalog/peertopeer/chapter/ch01.html>. Acesso em: fevereiro/2006.

(Muher, 2006?) Muher, Paul. File sharing. Disponível em: <http://www.practicallynetworked.com/howto/fileshare/fileshare_intro.htm>. Acesso em: janeiro/2006.

(Murphy et al., 2002) Murphy, Declan; Kelly, Jarlath; Curley, Keith Vickery; O´Keeffe, Jonh Dan. P2P Security, 2002/03. Disponível em: <http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/p10.html>. Acesso em: janeiro/2006.

(Oppenheirmer, 1999) Oppenheirmer, Priscilla. Capítulo 4 – Caracterização do tréfego da rede. In: Projeto de redes top-down. p. 83-87. Editora Campus, 1999.

(Orfali et al., 1999) Orfali, Robert; Harkey, Dan; Edwards, Jeri. Client/Server Survival guide, 1999. Third edition. Wiley Publications. 800p.

(Parameswaram et al., 2001) Parameswaram, Manoj; Susarla, Anjana; Whinston, Andrew B. P2P networking: an information-sharing alternative. IEEE Computer Practices, julho 2001, p. 31-38.

(Pastry, 2006) Projeto Pastry: Site oficial. Disponível em <http://research.microsoft.com/~antr/Pastry/>. Acesso em: dezembro/2006.

(Peer-to-peer working group, 2001) Peer-to-peer working group, 2001. Disponível em: <http://www.p2pwg.org>. Acesso em: fevereiro/2006.

(Qin et al, 2002) Qin Lv; Pei Cao; Edith Cohen; Kai Li; Scott Shenker. Search and Replication in Unstructured Peer-to-Peer networks, 2002. ACM 1-58113-483-5/02/0006

(Ratnasamy et al, 2001) Ratnasamy, Sylvia; Francis, Paul; Handley, Mark; Karp, Richard; Shenker, Scott. A scalable content-addresseble network. Sigcomm ´01, August 27-31, 2001, San Diego, California-US. ACM 1-58113-411-8/01/0008.

(Ribeiro-Neto; Baeza-Yates, 1999) Ribeiro-Neto, B.; Baeza-Yates, R. Modern Information Retrieval. 1999. ACM Press Books. ISBN 0-201-39829-x. 513p.

(Risson; Moors, 2004) Risson, John; Moors, Tim. A survey of research towards robust peer-to-peer networks: search methods. Technical Report UNSW-EE-P2P-1-1, University of New South Wales. September 2004.

(Rocha, 2005) Rocha, Rafael R. Redes peer-to-peer para compartilhamento de arquivos na Internet. Disponível em: http://www.gta.ufrj.br/~rafael/tutp2p.pdf . Acesso em: dezembro/2005.

(Rohrs, 2000) Rohrs, Christopher. LimeWire Design: Keyword Matching.

Dezembro/2000. Disponível em: <http://www.limewire.org/techdocs/KeywordMatching.htm>. Acesso em: janeiro/2006.

(Rohrs, 2001) Rohrs, Christopher. LimeWire Design: Search Result Grouping. Disponível em: <http://www.limewire.org/techdocs/result_grouping.htm>. Acesso em: abril/2006.

(Rohrs, 2002a) Rohrs, Christopher. Query routing for Gnutella network, maio/2002. Disponível em: <http://www.limewire.com/developers/query_routing/keyword%20routing.htm>. Acesso em: abril/2006.

(Rohrs, 2002b) Rohrs, Christopher, 2002, Ultrapeers: Another step towards Gnutella scalability. Disponível em <http://www.limewire.com/developer/Ultrapeers.html>. Acesso em abril/2006.

(Ross; Rubenstein, 2005) Ross, Keith W.; Rubenstein, Dan.Tutorial P2P. Disponível em: <http://cis.poly.edu/~ross/tutorials/P2PtutorialInfocom.pdf>. Acesso em: dezembro/2005.

(Schollmeier, 2002) Schollmeier, Rudiger. "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications". In: IEEE Internet Computing, 2002.

(Shirky et al., 2001) Shirky, Clay; Truelove, Kelly; Domfest, Rael; Gonze, Lucas. Chapter 1 – All the pieces of pie. In: Peer-to-peer networking overview: the emergent P2P platform of presence, identity, and edge resources. October 2001. 289p. Disponível em: <http://www.oreilly.com/catalog/p2presearch/chapter/ch01.html>. Acesso em: fevereiro/2006.

(Shirky, 2000) Shirky, Clay. What is p2p... and what isn't. Disponível em: <http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html>. O'Reilly Open P2P.com, 2000. Acesso: em janeiro/2006.

(Shirky, 2001) Shirky, Clay; The Accelerator Group. Chapter 2 - Listening to Napster. In: Peer-to-Peer : Harnessing the power of disrruptive technologies, pp. 21-37. O´Reilly. March/2001.

(Stephanos; Diomidis, 2004) Stephanos Androutsellis-Theotokis,;Diomidis S. A survey of peer-to-peer content distribution technologies. ACM Computing Surveys, Vol. 36, No. 4, Dezembro/2004. p. 335-371.

(Tanenbaum, 2003) Tanenbaum, Andrew S. Capítulo 1 – Introdução. In: Redes de Computadores, 4a Edição, 2003. p. 1-85.

(Theodore, 2001) Theodore, H., 2001, Chapter 14 – Performance. In: Peer-to-Peer: Harnessing the power of disrruptive technologies. O´Reilly, March/2001. pp. 203-241.

(Vanwasi, 2001) Vanwasi, A.K. Unleashing the power of P2P, 2001. Network

Magazine. Disponível em: <http://www.networkmagazineindia.com/200204/200204focus3.shtml>. Acesso em: janeiro/2006.

(WDS, 2006) Windows Desktop Search, 2006. Disponível em: <http://addins.msn.com/devguide.aspx>. Acesso em: agosto/2005.

(YDS, 2006) Yahoo Desktop Search, 2006. Disponível em: <http://desktop.yahoo.com>, Acesso em: abril/2006.

Anexo A – Códigos-fonte

Classe IndexFiles

```
public class IndexFiles {

  private IndexFiles(){};


  public static void indexar(String nomeIndice, File[] listaDiretorios) {

      File dirIndice = new File("c://lw//bin//"+nomeIndice);

    if (dirIndice.exists()) {
      dirIndice.delete();
    }

    for(int i=0; i < listaDiretorios.length; i++){
        File dirArquivos = listaDiretorios[i];
        if (!dirArquivos.exists() || !dirArquivos.canRead()) {
                                System.out.println("Document   directory   '"
+dirArquivos.getAbsolutePath()+  "'  does  not  exist  or  is  not  readable,
please check the path");
          System.exit(1);
        }
    }

    Date start = new Date();
    try {
                IndexWriter   writer   =   new   IndexWriter(dirIndice,   new
StandardAnalyzer(), true);
      System.out.println("Indexing to directory '" +dirIndice+ "'...");
      for(int i=0; i < listaDiretorios.length; i++){
        File diretorio = listaDiretorios[i];
        indexDocs(writer, diretorio);
      }
      System.out.println("Optimizing...");
      writer.optimize();
      writer.close();

      Date end = new Date();
           System.out.println(end.getTime()  -  start.getTime()  +  "  total
milliseconds");

    } catch (IOException e) {
      System.out.println(" caught a " + e.getClass() +
       "\n with message: " + e.getMessage());
    }
  }

  static void indexDocs(IndexWriter writer, File file)
    throws IOException {

    if (file.canRead()) {
      if (file.isDirectory()) {
        String[] files = file.list();
        if (files != null) {
          for (int i = 0; i < files.length; i++) {
            indexDocs(writer, new File(file, files[i]));
          }
        }
```

```
      } else {
        System.out.println("adding " + file);
        try {
          writer.addDocument(FileDocument.Document(file));
        }
        catch (FileNotFoundException fnfe) {
            ;
        }
      }
    }
  }
}
```

Classe FileDocument

```
public class FileDocument {


  private FileDocument() {}

  public static Document Document(File f)
       throws java.io.FileNotFoundException {

    Document doc = new Document();
          doc.add(new   Field("path",   f.getPath(),   Field.Store.YES,
Field.Index.UN_TOKENIZED));
    doc.add(new Field("modified",
                                DateTools.timeToString(f.lastModified(),
DateTools.Resolution.MINUTE),
        Field.Store.YES, Field.Index.UN_TOKENIZED));
    doc.add(new Field("contents", new FileReader(f)));
    return doc;
  }

}
```

Classe SearchFiles

```
public class SearchFiles {


      private SearchFiles() {}


      public static List procurar(String nomeIndice, String consulta)
throws Exception {


            List <String>lista = new ArrayList<String>();
            String field = "contents";
            IndexReader reader = IndexReader.open(nomeIndice);
            Searcher searcher = new IndexSearcher(reader);
            Analyzer analyzer = new StandardAnalyzer();
                  QueryParser parser = new QueryParser(field,
analyzer);
                  Query query = parser.parse(consulta);
                  Hits hits = searcher.search(query);
```

```
                        for(int i=0; i < hits.length(); i++){
                        Document doc = hits.doc(i);
                        String path = doc.get("path");
                        lista.add(path);
                        }
                        return lista;
            }
           }
     }
```

Classe SearchLucene

```
        protected IntSet SearchLucene(String query) {

            IntSet ret = new IntSet();
            List result = null;

            try{
                  result =
SearchFiles.procurar(SharingSettings.NOME_INDICE, query);
            }catch(Exception e){
                  e.printStackTrace();
            }

            if (result != null){

                  for (int i = 0; i < result.size(); i++) {
                        try {
                              String path = (String)result.get(i);
                              if (path != null) {
                                    FileDesc fd = getFileDescForFile(new
File(path));

                                    System.out.println("id: " +
fd.getIndex());

                                    ret.add(fd.getIndex());
                              } else {
                                    System.out.println("Não há path para
este documento");
                              }

                        } catch (Exception e) {
                              e.printStackTrace();
                        }
                  }
            }
            return ret;
      }
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas