PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

HEVERSON BORBA RIBEIRO

WEB2PEER: UMA INFRA-ESTRUTURA BASEADA EM REDES PEER-TO-PEER PARA PUBLICAÇÃO DE PÁGINAS NA INTERNET

CURITIBA
2007

HEVERSON BORBA RIBEIRO

WEB2PEER: UMA INFRA-ESTRUTURA BASEADA EM REDES PEER-TO-PEER PARA PUBLICAÇÃO DE PÁGINAS NA INTERNET

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito para obtenção do título de Mestre em Informática.

Área de Concentração: *Ciência da Computação*

Orientador: Prof. Dr. Lau Cheuk Lung

CURITIBA

2007

RIBEIRO, Heverson Borba

Web2Peer: Uma Infra-Estrutura Baseada em Redes Peer-to-Peer para publicação de Páginas na Internet. Curitiba, Fevereiro de 2007.

Dissertação de Mestrado – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática.

1. Redes *Peer-to-Peer* 2. Protocolos JXTA 3. Publicação de Páginas na Internet. 4. Disponibilidade na Internet. 5. Sistemas Distribuídos.
I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática Aplicada II-t

Dedico este trabalho a Deus e à minha família.

Agradecimentos

Primeiramente a *Deus* por tudo que Ele me concede, sem Ele nada seria possível.

À *minha família*, pois são as pessoas mais importantes deste mundo. Com eles compartilhei este momento abdicação, dedicação, persistência e finalmente colheita dos frutos.

Aos *verdadeiros amigos*, em especial ao colega de jornada Neander Brisola que vivenciou esta mesma experiência.

Aos *professores* que, direta ou indiretamente, contribuíram com o progresso deste trabalho. Em especial ao meu orientador Professor Doutor Lau Cheuk Lung, pela disposição em orientar, questionar, corrigir, incentivar e confiar, em todos os momentos, com o intuito de direcionar este trabalho ao sucesso. Ao Professor Doutor Altar Olívo Santin, por suas sugestões e questionamentos enfáticos que contribuíram diretamente com o resultado obtido. Ao Professor Doutor Carlos Maziero, pela motivação a pesquisa científica.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

HTTP	*Hypertext Transfer Protocol*
URL	*Uniform Resource Locator*
URI	*Uniform Resource Identifier*
JXTA	*Acrônimo para a Palavra Justapor, no inglês Juxtapose.*
DHT	*Distributed Hash Table*
IP	*Internet Protocol*
API	*Application Programming Interface*
GUI	*Graphical User Interface*
XML	*Extensible Markup Language*
TCP	*Transmission Control Protocol*

Resumo

Este trabalho apresenta uma infra-estrutura descentralizada que possibilita a publicação de páginas na internet através das redes *peer-to-peer*. Diferentemente da forma utilizada para a publicação de páginas na internet atual, a abordagem proposta independe de servidores centralizados ou endereços HTTP. Através de um *Web Browser*, o qual foi chamado *Web2Peer*, é fornecido um conjunto de funcionalidades para publicação, localização e replicação de páginas na Internet, aumentando a disponibilidade dos conteúdos publicados. A utilização desta infra-estrutura permite que qualquer pessoa conectada à Internet, mesmo através de uma conexão doméstica, possa publicar qualquer página *Web* através do seu computador pessoal, sem nenhum custo adicional.

Palavras-Chave: Peer-to-Peer, JXTA, DHT, Internet, Publicação.

Abstract

This work presents a decentralized infrastructure which makes possible to publish *Web* pages on Internet through peer-to-peer networks. Different from the way used for publishing *Web* pages on the current internet, the proposed approach does not depend on neither centralized *Web* Servers nor HTTP addresses, for instance. By using a *Web* Browser called *Web2Peer* it is provided a set of functions for publishing, locating, and replicating *Web* pages on Internet while the availability of the published *Web* pages is increased. By using this infrastructure anyone who has a computer connected to the internet, even through a domestic internet connection, can to publish *Web* pages through their own machines without any additional cost.

Keywords: Peer-to-Peer, JXTA, DHT, Internet, Publising.

Capítulo 1

Introdução

1.1 Motivação

A internet tornou-se, na última década, a principal referência para busca e publicação de informações no mundo. Com sua abrangência global, a rede tornou-se o lugar para todas as respostas, onde qualquer pessoa pode encontrar qualquer informação a qualquer instante.

Apesar das grandes mudanças ocorridas em relação às tecnologias associadas ao acesso à internet, como por exemplo, a evolução dos computadores pessoais e o aumento significativo no número de conexões domésticas de alta velocidade [2], algumas características básicas do funcionamento da WWW (*World Wide Web*) permanecem inalteradas.

A WWW mantém a sua estrutura centralizada, baseada no modelo cliente/servidor, onde um usuário quando precisa acessar um determinado conteúdo deve acessar um servidor *Web* específico que provê este conteúdo, através de *URLs* [3] e *URIs* [4].

Quando um usuário ou organização deseja criar um sítio *Web* (ou *Web site*) para publicar seus conteúdos (ex. páginas *Web*) na internet deve, necessariamente, recorrer a um provedor de serviços *Web*, seja pago ou gratuito, ou recorrer às entidades responsáveis [5, 6] para adquirir o seu próprio endereço IP e nome de domínio, a fim de criar o seu próprio provedor para abrigar o *Web Site*.

É importante observar que um provedor de conteúdo *Web* gratuito não fornece nenhuma possibilidade de administração do ambiente, somente acesso e publicação do conteúdo do usuário. Isto é, o usuário faz a carga (*upload*) de suas páginas *Web* nesse provedor e os aspectos de configuração do sistema são limitados ou inexistentes.

Este cenário apresenta dois fatores que devem considerados: o custo e a administração da infra-estrutura provedora das páginas *Web*.

Decidindo-se pela contratação de serviços de terceiros (um provedor) para publicação de páginas, acarretará ao usuário a dependência na administração dos conteúdos. Optando-se por servidores gratuitos, este usuário terá que conviver com a limitação dos recursos disponíveis. Por último, caso opte pela utilização de um provedor próprio, terá como conseqüência o custo com a contratação de infra-estruturas especificas como *link* com o provedor de rede de dados, manutenção do endereço *IP* e compra de nomes para o seu próprio domínio, bem como a dependência das entidades reguladoras que fornecem todos estes requisitos necessários.

Independente da opção escolhida é importante observar que através do modelo *Web* convencional, no qual a internet está fundamentada, não é possível garantir que os conteúdos estejam disponíveis sempre que necessário. Estruturas centralizadas criam pontos específicos de falha que põe sistemas computacionais em risco de indisponibilidade.

A estrutura centralizada da internet, baseada no modelo cliente/servidor [7], faz com que o provedor do conteúdo (o servidor *Web*) seja um ponto único passível de falhas, que pode tornar o conteúdo indisponível para acesso em qualquer momento. Em alguns tipos de aplicações a ocorrência de faltas pode ser aceitável, porém em outras, principalmente em aplicações críticas, isso não é admitido.

Algumas entidades e empresas tentam contornar este problema utilizando mais de um servidor para prover suas páginas *Web* na Internet ao mesmo tempo. Ou ainda, deixam servidores em estado de alerta (em *standby*) aguardando que um possível problema ocorra, caso isto aconteça o servidor em alerta deve assumir as funções do servidor principal no menor tempo possível [8]. O que também não garante que na fração de segundo em que uma página é requisitada o servidor reserva já esteja pronto para responder pelo principal.

Portanto, o modelo *Web* atual é caracterizado pela sua **susceptividade à indisponibilidade dos conteúdos**, devido a sua estrutura centralizada, e também pela sua **dependência de entidades reguladoras e os custos por elas impostos**.

A arquitetura de redes *peer-to-peer*, alvo de pesquisas intensas nos últimos anos, cria uma camada de rede sobre as topologias convencionais, permitindo que uma nova rede seja criada sobre as demais, eliminando assim os limites físicos da rede subjacente existente [9]. Nas redes *peer-to-peer* todos os nós são consumidores e fornecedores de recursos, com isso, a disponibilidade destes na rede é multiplicada. Isto é obtido através das técnicas de replicação de conteúdo passiva ou ativa

que ocorre entre os nós presentes na rede. Quando um nó consumidor acessa um conteúdo de um nó provedor, torna-se automaticamente um novo provedor deste conteúdo, criando uma cópia do mesmo. O método de publicação de páginas na internet pode fazer uso deste benefício para aprimorar o modelo *Web* atual.

1.2 Objetivos

As redes *peer-to-peer* foram propostas inicialmente para compartilhamento de recursos multimídia (arquivos de música e vídeo) e mais adiante para outras aplicações distribuídas. Neste trabalho propomos uma infra-estrutura distribuída para publicação, localização e replicação de páginas *Web* na internet, baseada em redes *peer-to-peer*. A solução proposta elimina os dois problemas citados anteriormente referentes à dependência de entidades reguladoras, além de apresentar uma solução, baseada em replicação pelos usuários, para aumentar a disponibilidade dos conteúdos fornecidos, sem a necessidade de criação de infra-estruturas redundantes no lado dos servidores.

O presente trabalho oferece uma nova abordagem para acesso e localização de conteúdos *Web*, mudando a forma com a qual publicamos esses conteúdos na internet e tirando proveito de tecnologias já existentes, porém não utilizadas conjuntamente para tal objetivo. Desta forma, a solução proposta permite que qualquer pessoa conectada à internet, mesmo através de uma conexão ADSL doméstica, possa publicar e replicar páginas *Web* a partir da sua própria máquina, sem a necessidade de recorrer a provedores ou infra-estruturas mais sofisticadas para disponibilizar o conteúdo que desejar na rede.

Para atingir tais objetivos foi criado um *Web browser* chamado de *Web2Peer*, que para o usuário final pode ser visto como um navegador *Web* (*browser*) comum, para acessar a internet através de uma rede *peer-to-peer*. Porém, agregado a este navegador há uma infra-estrutura definida para oferecer todas as funcionalidades necessárias, citadas anteriormente, sem que haja necessidade de configurações complexas ou impactos maiores na sua utilização. Restando ao usuário apenas usufruir dos benefícios da infra-estrutura *Web2Peer*.

Além disso, a infra-estrutura proposta oferece um mecanismo de busca de páginas *Web* através de palavras-chave baseado em tabelas *hash* distribuídas (DHT [10]), o qual atua como um servidor de buscas, similar ao *Google* [11]. Porém, neste caso, as informações ficam distribuídas em vários nós, permitindo ao usuário localizar conteúdos (páginas *Web*) por assuntos específicos e escolher de qual *peer* provedor deseja baixar a página *Web* procurada.

A forma com a qual a infra-estrutura foi definida, utilizando uma rede *overlay* [12], permite também que mecanismos de segurança, que podem dificultar a publicação de páginas por usuários isolados em redes privadas, não sejam obstáculos presentes no modelo proposto. Mesmo protegido por Firewall e NAT, um usuário do *Web2Peer* também pode publicar as páginas que desejar.

Além das novas funções, o *Web2Peer* manteve todas as funcionalidades já presentes, permitindo também acessar servidores *Web* convencionais sem nenhuma necessidade de configuração específica.

1.3 Organização

Este trabalho está dividido nos seguintes capítulos:

Capítulo 2: Redes *Peer-to-Peer*

Como base conceitual do trabalho proposto, este capítulo aborda os conceitos fundamentais e a principais características das redes *peer-to-peer*, do conjunto de protocolos *JXTA* e das tabelas *hash* distribuídas, bem como alguns exemplos já bem difundidos dessas tecnologias.

Capítulo 3: Trabalhos relacionados

Apresenta trabalhos existentes que motivaram o desenvolvimento da infra-estrutura proposta.

Capítulo 4: *Web2Peer*: Uma Nova Proposta para Publicação de Páginas na Internet

Descreve detalhadamente a infra-estrutura e suas principais funções.

Capítulo 5: Implementação e Avaliação dos Resultados

Mostra detalhes da implementação e algumas avaliações dos resultados obtidos em relação às tecnologias utilizadas atualmente.

Capítulo 6: Conclusão e Trabalhos Futuros:

Conclui a dissertação apontando os benefícios atingidos e ressaltando alguns trabalhos futuros relacionados ao projeto que podem melhorar ainda mais a solução proposta.

Capítulo 2

Redes *Peer-to-Peer*

2.1 Introdução

Arquiteturas de computação distribuídas chamadas *peer-to-peer* são projetadas para o compartilhamento de recursos por troca direta, sem precisar de intermediação ou suporte de uma autoridade central. As redes *peer-to-peer* são orientadas a conteúdo e são caracterizadas pela escalabilidade e adaptação às variações do número de nós presentes. Como características das arquiteturas *peer-to-peer* podem ser citadas:

- Resistência ao controle centralizado.
- Motivação à distribuição das tarefas.
- Favorecimento ao compartilhamento de recursos.
- Adaptação à variação do número de nós presentes.

A motivação para a utilização da tecnologia *peer-to-peer* vem da sua capacidade de trabalhar de forma escalável e auto-organizada na presença de um número variável de nós, num ambiente onde frequentemente ocorrem falhas computacionais, sem a necessidade de um servidor central. Por este motivo são inúmeras as aplicações para as redes *peer-to-peer*. Esta tecnologia permite, por exemplo, que computadores pessoais trabalhem, de maneira coordenada, como um meio de armazenamento distribuído, fornecendo uma capacidade total de armazenamento superior à capacidade individual de cada nó presente na rede[13].

Neste capítulo são apresentadas algumas definições de redes *peer-to*-peer encontradas na literatura atual, bem como algumas formas de classificar as redes, mostrando suas principais

características. Também é apresentado neste capítulo o conjunto de protocolos JXTA, que tem como objetivo maior a padronização da forma de desenvolver aplicações e serviços para redes *peer-to-peer*.

Por fim, o capítulo apresenta alguns conceitos de tabela *hash* distribuída, que também é utilizada na infra-estrutura sugerida. Ao longo deste estudo serão utilizados os termos nó e *peer* de forma similar, pois todos estão relacionados à mesma entidade de uma rede *peer-to-peer*.

2.2 Definições das Redes Peer-to-Peer

A literatura apresenta uma série de definições de redes *peer-to-peer*[9, 13-15]. Por exemplo:

"**O *peer-to-peer* puro é um sistema totalmente distribuído onde os nós são completamente iguais em relação às tarefas que executam**"[9].

Porém, esta definição falha quando aplicada aos sistemas que utilizam os modelos baseados em supernós. Supernós são nós especiais que executam, além das tarefas comuns aos outros nós, tarefas diferenciadas. Um exemplo de um sistema que utiliza supernós é o *E-mule*. O *E-mule* é um sistema amplamente difundido para compartilhamento de arquivos que utiliza alguns nós especiais para a localização dos conteúdos desejados. Outra definição mais ampla cita:

"***Peer-to-peer* é uma classe de aplicações que tiram proveito de recursos como armazenamento, ciclos de CPU e conteúdo disponíveis por toda a internet**"[9].

Porém, esta definição exclui os sistemas que dependem de servidores centralizados para operar, como, por exemplo, o *Napster*. No *Napster*, toda a indexação dos conteúdos presentes na rede fica armazenada num nó central. A interação direta entre os nós ocorre após a consulta a este nó central.

Por isso não seria exagero afirmar que há uma falta de concordância na questão conceitual do que é ou não um sistema *peer-to-peer*. Isto é explicado pelo fato de que os sistemas *peer-to-peer* atuais surgiram a partir de diferentes implementações que visavam resolver problemas distintos e não a partir de modelos pré-definidos e estudados para uma solução unificada. As experiências

práticas surgiram inicialmente, só depois os estudiosos passaram a se dedicar a analisar tais tecnologias.

Os sistemas são chamados de *peer-to-peer* não por causa da sua arquitetura, ou operação interna, e sim com base em como eles são percebidos externamente, se fornecem ou não a impressão de interação direta entre computadores.

Há duas características que tentam sintetizar as principais funcionalidades de um sistema *peer-to-peer*[9, 13]:

- Compartilhamento de recursos por interação direta, sem a necessidade de intermediação de um servidor centralizado. Há a possibilidade de utilização de servidores centrais para tarefas específicas, mas nunca para tarefas consideradas fundamentais, como, por exemplo, a manutenção do índice global para busca de nós.

- Habilidade em trabalhar com instabilidade e variação da conectividade, adaptação em casos de falhas de conexões de redes de computadores, bem como as variações no número de nós presentes. Esta característica de tolerância a faltas sugere que o sistema seja adaptativo às mudanças que possam ocorrer no sistema.

Com base nessas características é proposta a seguinte definição:

"Sistemas *peer-to-peer* são sistemas distribuídos compostos de nós interconectados, que estejam aptos a se auto-organizar em topologias de rede, com o propósito de compartilhar recursos, como conteúdo, ciclos de CPU, armazenamento e largura de banda, com a capacidade de adaptação a faltas e acomodação a um número variável de nós. Ao mesmo tempo, que mantém a conectividade e a performance em níveis aceitáveis, sem a necessidade de suporte ou intermediação de um servidor global centralizado"[9].

Esta definição atinge vários níveis de centralização, saindo dos sistemas chamados puros, passando pelos sistemas parcialmente centralizados e chegando aos sistemas totalmente descentralizados.

2.3 Classificações das Redes *Peer-to-Peer*

O funcionamento de um sistema *peer-to-peer* depende de uma rede de nós computacionais e das conexões entre eles. Esta rede é criada sobre a camada de transporte, gerando uma topologia virtual, chamada de rede *overlay*, sobre a topologia utilizada. A topologia, a estrutura, os mecanismos de localização e roteamento de mensagens e conteúdo, são essenciais para o funcionamento do sistema. As redes *peer-to-peer* são classificadas com relação a sua estrutura e a sua topologia virtual, conforme descrito a seguir[9, 13]:

2.3.1 Centralização da Rede *Peer-to-Peer* (Topologia *Overlay*)

Embora na sua forma mais pura, ou conceitual, uma rede *peer-to-peer* seja totalmente descentralizada, na prática vários graus de centralização são encontrados. Este parâmetro classifica as redes *peer-to-peer* de acordo com o grau de centralização das redes, definindo assim as topologias virtuais das redes *overlay*. Há três categorias definidas:

Arquitetura Puramente Descentralizada: Não há coordenação central das atividades, todos os nós executam exatamente as mesmas tarefas e agem tanto com clientes como servidores. Surgindo assim o conceito de *servents*, onde os nós executam tanto as tarefas de servidores (*servers*) quanto as tarefas de clientes (*clients*) (*servents* = *servers* + *clients*). A figura 2.1 mostra um exemplo de rede Puramente descentralizada, onde um nó (*servent*) faz uma consulta na rede por um determinado recurso enviando a mensagem para todos os nós da rede (passos 1) e o nó que possui o recurso procurado responde a solicitação (passo 2).

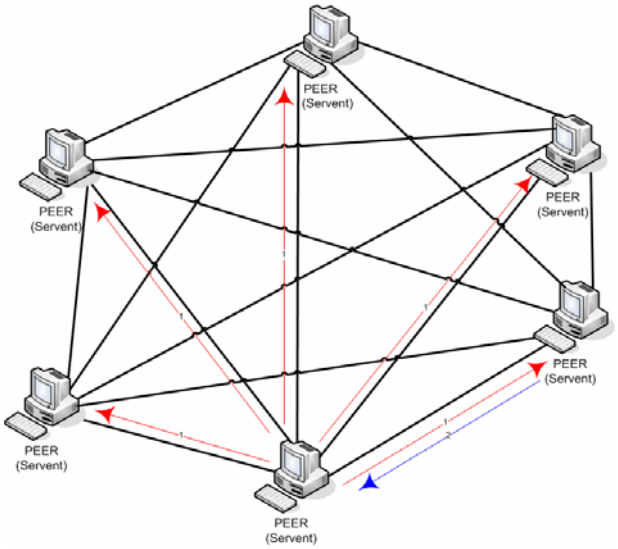


Figura 2.1: Arquitetura *Peer-to-Peer* Puramente Descentralizada

Arquitetura Parcialmente Centralizada: As características são similares as da abordagem anterior, porém alguns nós recebem um papel diferenciado, sendo chamados de supernós. A forma com a qual o supernó é eleito pode criar um ponto único de falha na rede. Isso pode ser evitado alocando o supernó dinamicamente, pois caso este falhe, um novo nó é alocado para substituí-lo.

A figura 2.2 mostra uma rede parcialmente centralizada onde um nó consulta um supernó, questionando-o sobre a localização de determinado conteúdo e este supernó responde ao nó requisitante qual é o nó detentor de tal conteúdo (passos 1 e 2). Em seguida, o nó requisitante contata diretamente o detentor do conteúdo desejado (passos 3 e 4), sem precisar questionar todos os nós da rede, somente o supernó.

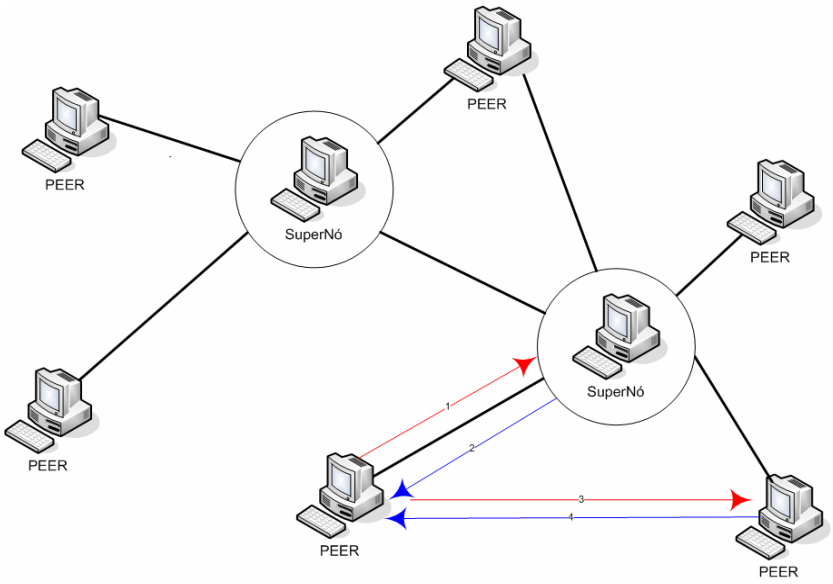


Figura 2.2: Arquitetura *Peer-to-Peer* Parcialmente Centralizada

Arquitetura Descentralizada Híbrida: Neste caso há um servidor central que busca para facilitar a interação entre os nós. Este servidor possui diretórios de dados que descrevem os arquivos compartilhados e armazenados nos nós. A troca de arquivos propriamente dita ocorrerá entre os nós, mas sempre com a interação do servidor central.

Uma desvantagem desta abordagem é que o servidor central constitui um ponto único de falha, que limita a escalabilidade do sistema e torna o sistema vulnerável aos ataques maliciosos. A figura 2.3 mostra o nó central da arquitetura híbrida como um ponto único de falhas, mas a troca de arquivos sendo realizada pelos nós envolvidos.

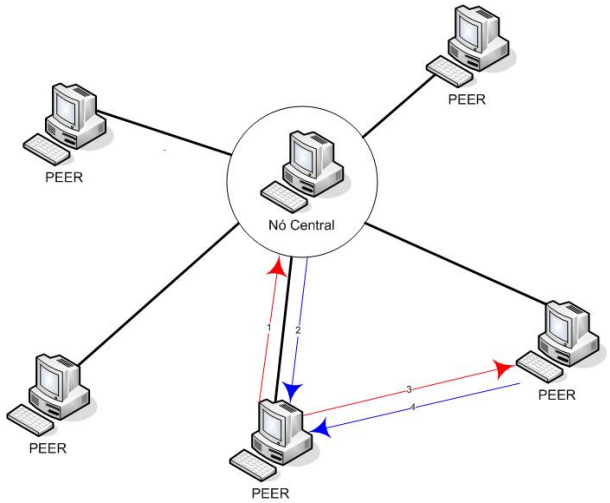


Figura 2.3: Arquitetura Descentralizada Híbrida

2.3.2 Estrutura da Rede

Por estrutura entendemos a forma com a qual a rede é criada, se determinística ou não. Este determinismo está relacionado à adição dos nós e recursos na rede. Se a adição destes obedece a regras específicas, a rede é considerada determinística, portanto estruturada. Caso não obedeça a nenhuma regra, é chamada de rede não-estruturada. Portanto as redes *peer-to-peer* são classificadas da seguinte maneira, de acordo com a sua estrutura:

Não-Estruturada: Neste caso, a localização dos recursos (arquivos, por exemplo) é totalmente desvinculada da topologia da rede *overlay*. O conteúdo precisa ser procurado por toda a rede, para isso podem ser utilizados mecanismos de busca que enchem a rede de mensagens, esta técnica é conhecida como *flooding.*

Estruturada: A topologia da rede *overlay* é rigorosamente controlada e os arquivos e ponteiros para eles são colocados em locais específicos. Estes sistemas fornecem um mapeamento entre conteúdo e localização. Dessa forma, a consulta é feita diretamente ao nó que contém o conteúdo desejado. São sistemas mais complexos, portanto a dificuldade de manutenção das estruturas necessárias se acentua, principalmente com relação à entrada e saída constante de nós do sistema, mas em ambientes em que a precisão é um fator determinante torna-se viável. Tabelas *hash* distribuídas são exemplos destas redes.

2.4 Replicação em Redes *Peer-to-Peer*

A replicação dos recursos providos por um nó na rede *peer-to-peer* pode ocorrer de três maneiras:

2.4.1 Replicação Passiva

A replicação passiva, a forma mais simples de replicação, segue o conceito tradicional das redes *peer-to-peer*¸ onde o recurso é replicado pela rede conforme é acessado por nós remotos. Por exemplo, um *peer* A procura na rede *peer-to-peer* por um recurso disponibilizado por um *peer* B, ao acessar este recurso diretamente no *peer* B, cria uma cópia local deste recurso, tornando se uma nova fonte deste recurso na rede.

2.4.2 Replicação Baseada em Cache

Na replicação baseada em cache, o recurso também é replicado quando um nó qualquer acessa um nó remoto, porém neste caso a cópia não é criada apenas no nó que acessou o recurso, mas também, em todos os nós que fizeram parte da localização do recurso na rede. Na prática, o recurso oferecido por um nó é copiado em todos os nós do caminho utilizado pela mensagem de localização deste recurso. Por exemplo, se para achar um recurso no *peer* F, uma mensagem M parte de um *peer* A e caminha pelos *peers* intermediários B, C, D e E, são criadas cópias do recurso em todos esses *peers* do caminho (B, C, D, E) e mais uma cópia no *peer* requisitor (A).

2.4.3 Replicação Ativa

Na replicação ativa, diferentemente das anteriores, não há necessidade de acesso ao recurso para que este seja replicado. A partir de políticas próprias, pré-determinadas, o nó detentor do recurso escolhe alguns nós na rede, para os quais enviará cópias do recurso disponibilizado, aumentando de forma ativa a disponibilidades destes recursos. A escolha destes nós segue regras próprias, por exemplo, o nó pode escolher os seus vizinhos para enviar as cópias dos recursos ou ainda, procurar por nós que possuem determinadas características, como largura de banda, recursos computacionais entre outros, para criar as cópias.

2.5 Redes *Peer-to-Peer* Difundidas

2.5.1 *Napster*

O *Napster* foi criado originalmente como um serviço de compartilhamento de arquivos de músicas. Foi o pioneiro entre as arquiteturas *peer-to-peer* atuais e ficou mundialmente conhecido por ter sido acusado de violação de direitos autorais[16].

O *Napster* possui uma arquitetura centralizada, onde o nó conecta-se a um computador central que possui um índice completo de todos os arquivos disponíveis em todos os nós presentes na rede. Apesar disso, os arquivos não ficam armazenados neste servidor central e sim nos nós participantes[17].

Quando o nó entra na rede *peer-to-peer* ele publica todos os arquivos que deseja compartilhar neste servidor central. Quando outro nó procura por determinados arquivos o servidor central informa ao nó requisitante qual é o nó da rede que possui o arquivo desejado (passos 1 e 2 da figura 2.4). Em seguida o nó vai até o nó informado e solicita que este lhe envie o arquivo desejado (passos 3 e 4).

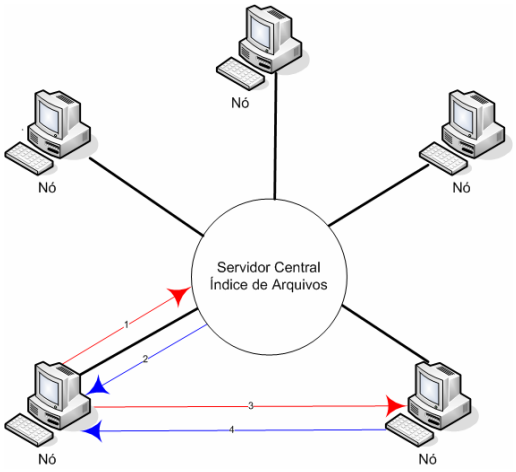


Figura 2.4: Busca de Arquivos no *Napster*

Conforme as características conceituais vistas anteriormente, podemos classificar o *Napster* como: uma rede *overlay* não-estruturada com topologia descentralizada hibrida.

A grande desvantagem deste modelo usado pelo *Napster* é que se o servidor central de índices por algum motivo deixar de funcionar, toda a rede também ficará inoperante.

2.5.2 *Gnutella*

Seguindo o caminho do *Napster*, a rede *Gnutella*[18] foi criada para o compartilhamento de arquivos de música na internet, apresentando-se, porém, como uma arquitetura puramente descentralizada, onde todos os nós executam as mesmas tarefas (servents) e as buscas não são realizadas em um nó centralizado e sim em toda a rede.

Quando um nó deseja entrar na rede *Gnutella* inicialmente pesquisa, através de um domínio publico na internet (*www.gnutellahosts.com)*, quais são os nós usados para conectar-se à rede. A partir disso o nó envia uma mensagem a um destes nós informando que deseja conectar-se na rede. Sendo a conexão estabelecida o nó já pode pesquisar e fornecer arquivos para a rede.

Quando o nó deseja pesquisar algum arquivo na rede ele consulta os seus vizinhos, caso estes não possuam o arquivo solicitado reenviam a solicitação aos seus vizinhos, desta forma a propagação da solicitação ocorre por toda a rede até que o arquivo seja localizado, caso exista. Este tipo de consulta inunda a rede com requisições (*flooding*) e não pode oferecer respostas rápidas. A figura 2.5 mostra como uma consulta é realizada na rede.

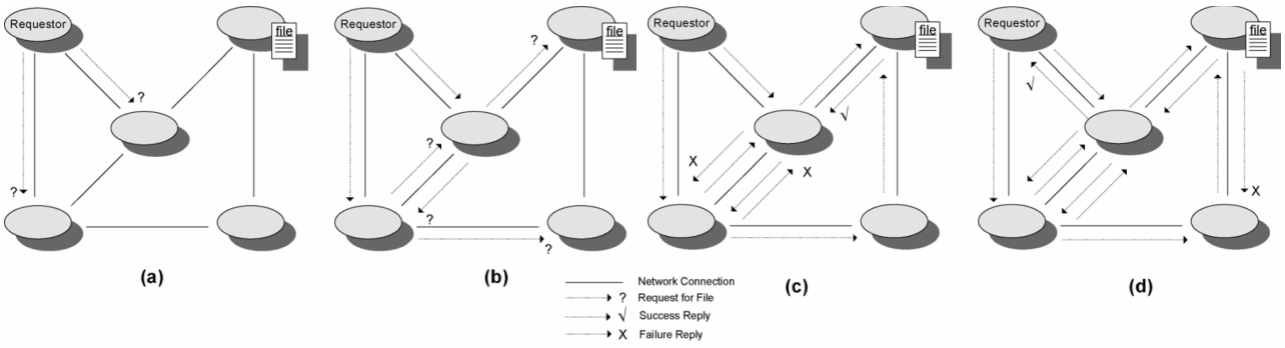


Figura 2.5: Consulta de Arquivos na Rede *Peer-to-Peer* [9]

O *Gnutella*[19] faz uso de quatro mensagens específicas para a comunicação entre os servents:

- ***Ping*:** Mensagem utilizada pelo nó para anunciar a sua entrada na rede.
- ***Pong*:** Resposta a uma mensagem *Ping*.
- ***Query*:** Uma requisição de busca de arquivos na rede. Esta mensagem contém o nome do arquivo procurado.
- ***Query Hits*:** Resposta à mensagem *Query*. Esta mensagem contém o IP e a porta do nó que possui o arquivo procurado.

2.5.3 *FreeNet*

O *Freenet*[20] destacou-se inicialmente por propor-se a impossibilitar o rastreamento da origem de um arquivo publicado na rede, bem como quem acessa este arquivo. Com isso ficou ressaltada a característica do anonimato. O *Freenet* propõe-se a criar uma internet anônima e sem censuras. Quando um nó disponibiliza um arquivo na rede, este arquivo, diferentemente das implementações até então citadas, fica distribuído em vários nós da rede.

O processo de busca de um arquivo na rede *Freenet*[13] é similar a abordagens vistas anteriormente, pois o nó, a partir de uma chave qualquer, pesquisa localmente se esta não está armazenada em seu repositório local. Caso não esteja, o nó determina um valor para o HTL (*hope-to-live)*, que é um valor que determinará por quantos nós a mensagem poderá passar antes de ser ignorada. Após determinar o HTL e não localizar a chave localmente, o nó procura em sua tabela de rotas pelos nós mais próximos, baseado em funções de distância conforme será discutido posteriormente na seção 2.6 Tabelas *Hash* Distribuídas, e envia a requisição do arquivo referente a tal chave para os nós mais próximos, e assim consecutivamente até localizar o arquivo procurado. O nó que possui tal arquivo responde ao nó requisitante, fazendo o caminho inverso pelo qual foi localizado. Após receber o arquivo armazena-o localmente em seu repositório e cria um ponteiro para este arquivo na sua tabela de roteamento. Durante o envio do arquivo, cópias são criadas em todos os nós envolvidos na transmissão do arquivo.

Um aspecto interessante no *Freenet* é a criação das chaves que identificam os arquivos na rede. Estas chaves são representadas por 160 *bits* e obtidas a partir da aplicação de uma função *hash* SHA-1 sobre o arquivo. Existem três diferentes formas de criar as chaves no *Freenet*: *Keyword-Signed Key* (KSK), *Signed-Subspace Key* (SSK) *e Content Hash Key* (CHK). As figuras 2.6, 2.7 e 2.8 descrevem como é feita a criação das chaves em cada uma das formas citadas[20].

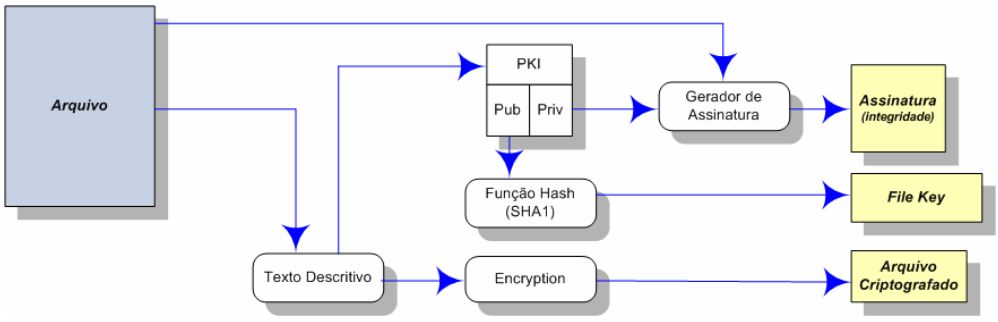


Figura 2.6: Criação da Chave usando KSK

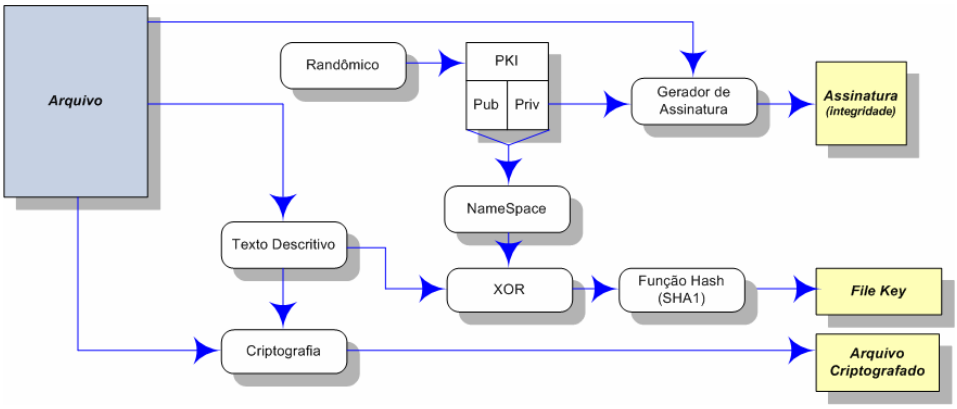


Figura 2.7: Criação da Chave usando SSK

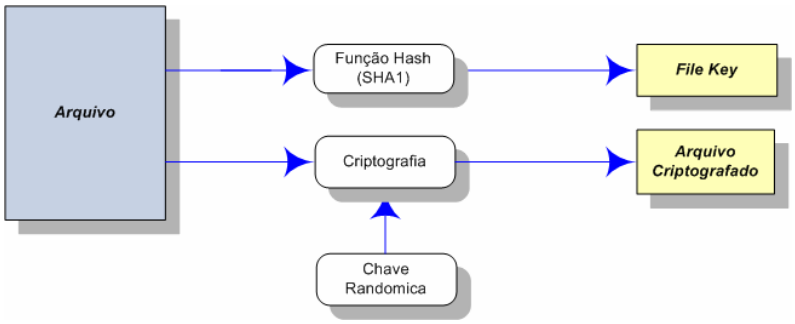


Figura 2.8: Criação da Chave usando CHK

Com suas características focadas ao anonimato o *Freenet* ganhou relevância entre as aplicações *peer-to-peer* e diversas aplicações passaram a utilizar a rede como mecanismo de armazenamento distribuído.

2.6 O Protocolo JXTA

Quando alguns protocolos e aplicações *peer-to-peer* começaram a surgir eles não interagiam entre si. O conjunto de protocolos *JXTA* foi concebido para padronizar o desenvolvimento de redes *peer-to-peer* e permitir a interoperabilidade entre essas aplicações[21]. O projeto JXTA foi concebido inicialmente pela *Sun Microsystems* em 2001 e desenvolvido com a participação de um grupo de instituições acadêmicas. O nome JXTA é derivado da palavra *juxtapose*, e foi escolhido com o intuito de mostrar que as soluções *peer-to-peer* poderiam conviver lado a lado com a arquitetura cliente/servidor, sem a intenção de substituí-la[22].

Os protocolos JXTA estabelecem uma rede *overlay* sobre a internet, permitindo que os nós interajam diretamente e se auto-organizem independentemente da conectividade ou topologia das

suas redes de origem. O *JXTA* foi especificado para ser independente da linguagem de programação, plataforma operacional, dos serviços e também dos protocolos de rede. O *JXTA* define os requisitos mínimos para que os nós formem a rede *peer-to-peer*, com isso possibilita que os programadores determinem qual a topologia da rede é mais adequada às suas aplicações[23]. A figura 2.9 mostra uma rede virtual JXTA sobre duas outras redes criando uma nova topologia virtual.

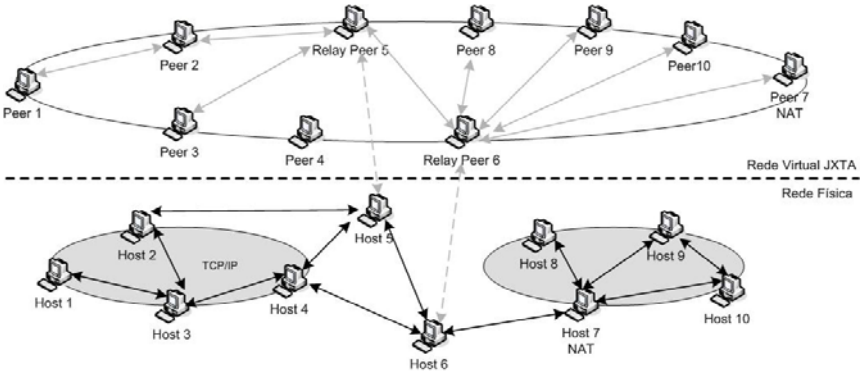


Figura 2.9: Rede Overlay JXTA [24]

O grupo de protocolos JXTA permite que os nós troquem mensagens entre si, independentemente da sua localização na rede. Um dos objetivos da arquitetura é esconder toda a complexidade da topologia de rede física, para que as mensagens sejam roteadas transparentemente através de *firewalls* e *NATs* e outras redes não-*IP* até alcançarem o seu destino[22].

O *JXTA* organiza os seus nós em *peergroups*, que tem como função isolar as aplicações em domínios específicos da rede *peer-to-peer*. Existem dois grupos pré-definidos na rede JXTA, o *World-PeerGroup* e o *Net-PeerGroup.* Todos os nós presentes na rede JXTA fazem parte obrigatoriamente do primeiro.

Ao criar um grupo o programador pode inseri-lo em um grupo específico e nomeá-lo de acordo com a sua própria vontade, para isolar a sua aplicação, porém os dois nomes pré-definidos, citados anteriormente, não podem ser utilizados. Caso nenhum outro grupo seja criado pelo usuário o próprio JXTA adiciona o nó ao segundo grupo pré-determinado, o Net-PeerGroup, este grupo é um subconjunto, já criado, do grupo World-PeerGroup.

O JXTA possui também *peers* especiais chamados de *rendezvous* e *relay-peers* que são supernós com funções diferenciadas.

Um *peer rendezvous*[25] é usado para executar propagação de mensagens em grupo, dentro da rede *peer-to-peer*. Isto possibilita a redução da quantidade de mensagens na rede, uma vez que

as mensagens podem ser enviadas a um grupo especifico de nós (redução do *overhead*). *Rendezvous* peers são utilizados, por exemplo, em mensagens de busca de recursos na rede *peer-to-peer*.

Um *relay peer*[25], anteriormente chamado de *router peer*, mantém informações sobre as rotas para outros *peers* e roteia mensagens para os *peers*. Por exemplo, quando um *peer* deseja enviar mensagens para outro, primeiramente procura no seu repositório (*cache*) local para verificar se possui alguma rota para o *peer* desejado. Se não possui a rota desejada ele envia uma consulta de rota para um *relay peer*, para descobrir a rota para o *peer* de destino. Os *relay peers* também são responsáveis por encaminhar mensagens em nome de *peers* que não podem encaminhá-las diretamente. Por exemplo, *peers* que estão em redes isoladas através de NAT e *Firewalls*.

A figura 2.10 mostra como um *relay peer* atua para enviar mensagens em nome de outros *peers*. Nesta figura o *peer 2* envia uma mensagem para o *relay peer* enviar ao *peer 1* em seu nome (passo 1). Periodicamente, o *peer 1* conecta-se ao seu *relay peer* para verificar se existem mensagens para ele (passo 2). O *relay peer* usa as conexões periódicas do *peer 1* para encaminhar as mensagens para dentro da rede privada e entrega a mensagem ao *peer* de destino, *peer 1*.

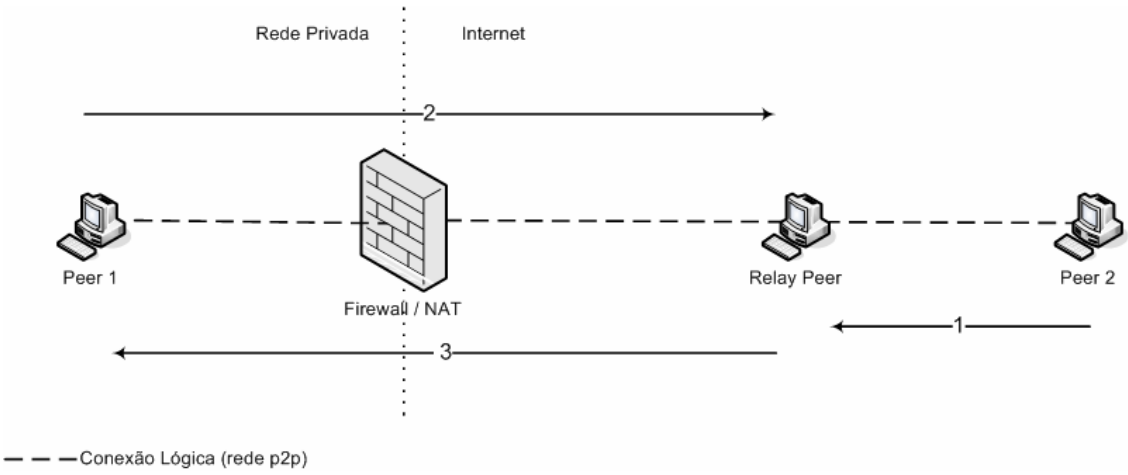


Figura 2.10: Função dos *Relay Peers*[25]

Outro elemento importante do JXTA é o e*ndpoint. Endpoints* são os pontos extremos em uma comunicação entre nós da rede JXTA. Qualquer dado sendo transmitido terá como origem e o destino um *endpoint*. O *endpoint* corresponde à interface de rede utilizada para enviar e receber dados[21].

As mensagens são transmitidas na rede através de *pipes*, que são canais virtuais de comunicação usados para enviar e receber mensagens entre serviços e aplicações. Os *pipes* são

chamados de canais de entrada e saída de acordo com a tarefa que estão executando no momento, se enviando (*output pipe*) ou recebendo mensagens (*input pipe*)[21].

É importante salientar que os *pipes* não são responsáveis por enviar os dados entre os nós, isto é apenas uma abstração usada para representar que dois *endpoints* estão conectados um ao outro, e estes sim, possuem os mecanismos necessários para a transmissão dos dados (mensagens) na rede.

O JXTA possui um conjunto com seis protocolos distintos, conforme mostrado a seguir. A figura 2.11 mostra como os protocolos interagem entre si[22]:

- ***Peer EndPoint Routing Protocol*:** Fornece um conjunto de mensagens usadas para possibilitar o roteamento de mensagens de um nó de origem até um nó de destino.
- ***Rendezvous Protocol*:** Permite a propagação de mensagens em grupo.
- ***Peer Resolver Protocol*:** Permite que os nós enviem e processem mensagens genéricas.
- ***Peer Discovery Protocol*:** Possibilita que os nós publiquem e descubram recursos na rede.
- ***Peer Information Protocol*:** Permite ao nó obter informações sobre outros nós da rede.
- ***Pipe Binding Protocol*:** Fornece um mecanismo para ligar um canal de comunicação virtual (*pipe*) a uma interface de rede (*endpoint*).

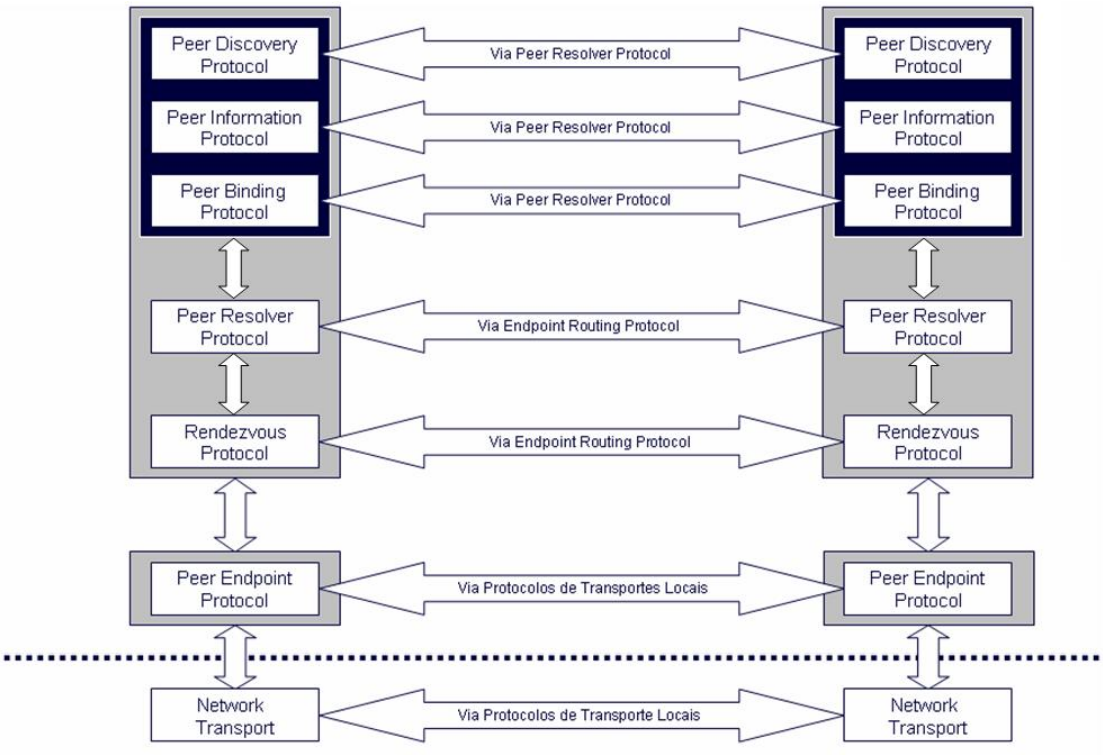


Figura 2.11: Os Protocolos JXTA[25]

Todos estes elementos apresentados são representados por suas respectivas publicações na rede JXTA. As publicações (*advertisements)* são representações estruturadas através de documentos XML que descrevem cada uma das entidades, seja ela *peer*, *pipe*, *endpoint*, *peergroup*, etc. As mensagens enviadas na rede JXTA também são criadas a partir de documentos XML, o que fornece a interoperabilidade dos protocolos.

2.7 Tabelas Hash Distribuídas (DHT)

2.7.1 Conceitos

Tabelas *hash* distribuídas[10], também conhecidas como DHT, são tabelas que fornecem o mapeamento de chaves em valores como as tabelas *hash* tradicionais, com o diferencial de particionar a tabela entre os diversos nós participantes da sua rede. Como uma tabela *hash* tradicional, as chaves da tabela são obtidas a partir da aplicação de uma função *hash* no dado que se deseja armazenar. Criando assim o par <*chave*, *valor*> que é armazenado na tabela.

Cada nó é responsável por fornecer o mapeamento para um grupo de chaves e estas chaves são distribuídas na DHT de acordo com o identificador dos nós. O identificador único (ID) do nó é obtido através da mesma função *hash* aplicada para criar as chaves da DHT. Além das informações da tabela de pares <*chave*, *valor*>, os nós possuem também uma tabela de roteamento com informações de outros nós, que é utilizada para a localização dos nós conhecidos, também chamados de nós adjacentes, ou vizinhos[26].

Veja o exemplo da figura 2.12, onde é mostrada uma tabela *hash* que está particionada sobre os nós presentes na DHT.

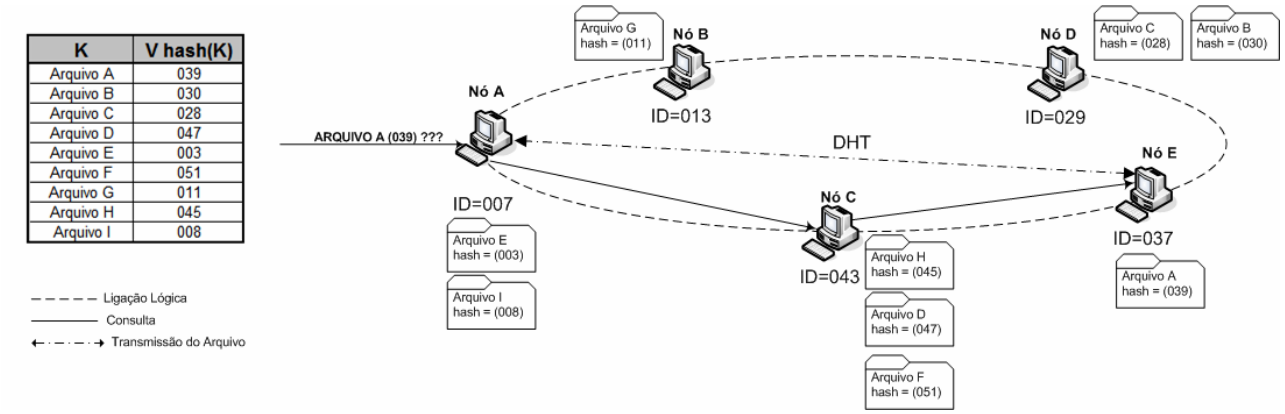


Figura 2.12: Visão Geral de uma Tabela *hash* Distribuída

No exemplo, a tabela é composta de vários arquivos nomeados de A até I, que são os valores (k) que devem ser armazenados na DHT. Sobre estes arquivos é aplicada uma função *hash* que determinará o valor da chave de cada arquivo que também será armazenada na tabela, criando assim o par (*chave*, *valor*). Para armazenar o par na tabela, a chave do par é usada para identificar o nó de destino deste par. O par é armazenado no nó que possui a menor distância entre a chave e o ID do nó. No exemplo, podemos constatar que o *Arquivo C* possui a *chave 028* e por isso foi armazenado no *Nó D* que possui *ID=029*. Desta forma todos os outros arquivos são distribuídos pela DHT, particionando o espaço de chaves.

As buscas na DHT seguem o mesmo raciocínio, a busca é realizada consultando-se os nós conhecidos e procurando por aqueles nós que estejam mais próximo do nó desejado. A partir disso, vai se aproximando cada vez mais do nó de destino, até atingi-lo. Recuperando assim o valor referente à chave pesquisada na tabela *hash* distribuída.

No exemplo da figura 2.12 podemos verificar uma busca ocorrendo quando o *Nó A* procura por uma *chave 039* referente ao *Arquivo A*. A busca inicia consultando o nó adjacente mais próximo da chave desejada, neste caso o *Nó C* (*ID=043*). Em seguida o *Nó C* consulta o seu vizinho mais próximo da chave desejada, neste caso o *Nó E* (*ID=0037*). Sendo o *Nó E* responsável pelo armazenamento do par desejado, ele envia o arquivo desejado ao nó requisitante (*Nó A*) e encerrando a busca.

Para evitar problemas com entradas e saídas constantes de participantes da DHT, o conjunto de pares também é replicado nos nós adjacentes ao *peer*, pois estes nós adjacentes serão responsáveis pelo armazenamento dos pares em caso de saída do nó em questão.

Ao criar algoritmos para tabelas *hash* distribuídas alguns fatores devem ser sempre lembrados, tais como:

Balanceamento de Carga: tanto as chaves quanto os nós são geralmente identificados por um número único de *n* bits (ID). Ao distribuir as chaves pelos nós presentes na rede, os IDs destas chaves devem ser armazenados em nós que possuam identificadores (ID) próximos aos seus.

Encaminhamento de Mensagens ao Nó Adequado: ao receber uma consulta por uma determinada chave, o nó precisa enviar esta mensagem ao nó mais próximo possível do nó responsável por armazenar o conjunto de chaves ao qual a chave procurada pertence.

Construção Dinâmica de Tabelas de Roteamento: para que as mensagens possam ser transmitidas e os nós e os conteúdos sejam localizados na rede, os nós precisam de um mecanismo que lhes permitam conhecer outros nós presentes. Esta informação é fornecida através de tabelas de

roteamento armazenadas em cada um dos nós e que precisam ser alteradas constantemente para refletir o estado dos nós que, assincronamente e concorrentemente, entram e saem da rede.

Definição da Função Distância: trata da definição de qual métrica será aplicada para determinar o quão distante uma chave está de um nó, para que seja armazenada no nó responsável pelo grupo de chaves adequado. Isso é o que criará o particionamento da tabela *hash* distribuída. Cada implementação define a sua própria Função Distância. Algumas implementações utilizam a diferença numérica como função distância, outras alguns bits pré-definidos do ID, outras aplicam uma função XOR entre os IDs, e assim por diante. Cabe ao projetista da DHT definir a metodologia que seguirá.

2.7.2 Implementações Difundidas

As primeiras abordagens de DHT foram apresentadas por volta de 2001 por *Chord[27]*, *CAN*[28], *Pastry[29]* e *Tapestry[30]*. Desde então, inúmeras abordagens e diferentes implementações, ou variações das iniciais, vem sendo apresentadas.

a) Chord: Desenvolvido pelo Instituto de Tecnologia de Massachusetts (MIT) é um sistema escalável, simétrico e totalmente distribuído, desenvolvido com o intuito atingir os seguintes objetivos:

- **Escalabilidade:** o sistema deve funcionar eficientemente mesmo quando a rede crescer.
- **Disponibilidade:** o sistema deve funcionar mesmo quando a rede dividir-se ou alguns nós falharem.
- **Balanceamento de carga:** os pares (*chave*, *valor*) são distribuídos igualmente pelo sistema.
- **Dinamicidade:** o sistema pode suportar mudanças rápidas na topologia da rede.
- **Localização de Acordo com a Proximidade:** buscas devem retornar os dados mais próximos do nó solicitante, em vez de percorrer grandes distâncias na rede.

Topologicamente o Chord é um anel virtual e o ID do nó e da chave do par é representado por um número *n* de *m* bits. No Chord, cada nó armazena somente um subconjunto das chaves, como mostrado no item 2.5.1 *Conceitos*.

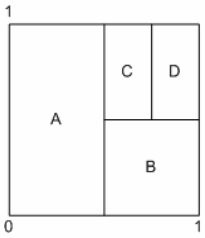
Para localizar uma chave cada nó precisa conhecer somente o seu nó sucessor no anel. Assim o nó solicitante pesquisa o seu sucessor para saber se ele é responsável pela chave que está procurando. Se não possui, a mensagem é propagada para o próximo sucessor, até encontrar o nó

que possui a chave procurada. Quando a chave for encontrada, o nó devolve a informação diretamente ao nó solicitante[27].

Para que não ocorra uma possível situação onde todos os nós precisem ser consultados para localizar uma informação, os nós armazenam também algumas informações dos outros nós, reduzindo assim o caminho da pesquisa. Estas informações são armazenadas em uma tabela chamada tabela *finger* presentes em todos os nós.

b) CAN – Content Addressable Network: é uma rede onde os nós estão organizados em um sistema de coordenadas cartesianas de *n* dimensões. O espaço é dinamicamente particionado entre os nós presentes na rede e estas partições do espaço são subdivididas ou reagrupadas conforme os usuários entram ou saem da rede[28].

Veja a figura 2.13 que mostra uma tabela com quatro nós (A, B, C e D) num plano cartesiano CAN.



	X	Y
A	(0, 0.5)	(0,1)
B	(0.5, 1)	(0, 0.5)
C	(0.5, 0.75)	(0.5, 1)
D	(0.75, 1)	(0.5, 1)

Figura 2.13: Plano Cartesiano CAN

Para que um novo nó entre na rede, primeiramente deve conectar-se a um nó que faça parte da rede, a partir disso escolhe de forma aleatória um ponto qualquer do espaço cartesiano do sistema.

O novo nó envia, através do nó no qual se conectou inicialmente, uma requisição ao nó responsável pela partição do espaço que contém o ponto selecionado anteriormente.

Ao receber esta requisição, o nó responsável por este espaço quebra a sua partição em duas partes e designa uma delas ao nó entrante, informando a ele as chaves pelas quais será responsável. Veja o exemplo apresentado na figura 2.14.

Caso o nó saia da rede, ele deve enviar aos seus vizinhos a parte da tabela distribuída que está sob a sua responsabilidade. Este nó assumirá a responsabilidade por esta tabela unindo-a com a sua tabela original, ficando responsável pela partição do nó que saiu.

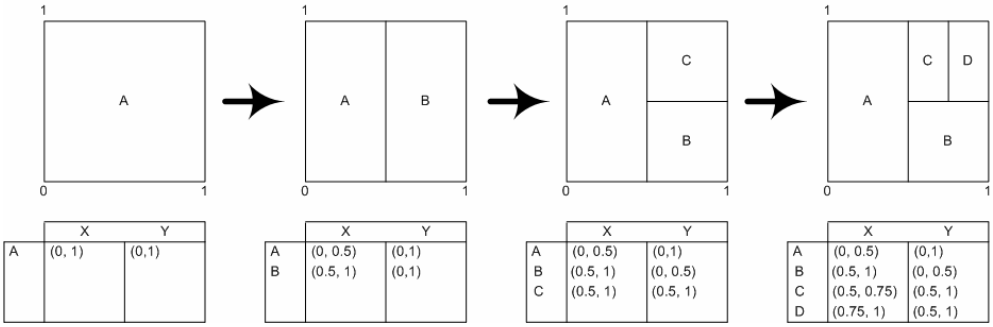


Figura 2.14: Inserção de nós na rede CAN

Caso haja algum problema e o nó saia da rede sem antes comunicar a sua saída, os nós vizinhos detectarão a ausência, por meio de mensagens de verificação trocadas entre eles. Caso isso seja detectado, um dos vizinhos assume o controle da partição pertencente ao nó que saiu. O CAN possui um protocolo interno com cinco mensagens, conforme mostrado na tabela 2.1.

Tabela 2.1: Mensagens do Protocolo da rede CAN[9]

Ping	Utilizada para verificar o status de um nó da rede.
Pong	Resposta enviada a um nó quando recebida uma mensagem de *ping*. Esta mensagem possui também informações sobre o nó que está respondendo ao *ping*.
Query	Mensagem de busca enviada à rede em busca de um conteúdo de uma chave específica
QueryHit	Resposta ao *Query*, esta mensagem contém também todas as informações necessárias para o envio das informações desejadas.
Push	Mensagem utilizada por nós que estão localizadas sob a proteção de *Firewalls* para enviar informações a DHT.

c) Pastry: Descrita como uma rede completamente descentralizada, escalável e auto-organizável, o *Pastry* é mais uma das primeiras e principais, em termos conceituais, tabelas *hash* distribuídas surgidas por volta de 2001.

Cada nó do *Pastry* tem um identificador único (ID) chamado de *nodeId*. Esses identificadores, bem como as chaves dos pares (*chave, valor)*, são representados por números de 128 bits na base 2, gerados randomicamente quando o nó entra na rede a partir do endereço IP ou da chave pública do nó, e dispostos num espaço circular. Como as demais implementações de DHT, os nós numa rede *Pastry* armazenam informações referentes às chaves distribuidamente nos nós, procurando colocar o par (*chave, valor*) no nó com ID mais próximo ao valor da chave[29].

Para armazenar e localizar estes pares o *Pastry* utiliza uma tabela de roteamento com ponteiros para os demais nós da rede. Esta tabela armazena um conjunto de nós vizinhos e nós chamados folhas. Os nós vizinhos são os nós mais próximos do nó em questão e são utilizados para controlar a entrada e saída dos nós na rede, já os nós folhas são usados com a intenção de roteamento de mensagens.

d) OpenDHT: É um projeto desenvolvido para criar uma DHT de acesso público em forma de serviço[31]. Esta rede possui uma interface simples, com funções básicas de inserção, recuperação e remoção de pares (*chave, valor*) e que, ao mesmo tempo, dispõe de todos os recursos oferecidos pelas tabelas *hash* distribuídas aqui apresentadas[32].

Sendo uma DHT de uso compartilhado por várias aplicações, buscou-se facilitar ao máximo a sua utilização, trazendo à tona um novo conceito chamado de *DHT-Gateway*. Este *gateway* possibilita que computadores que não fazem parte direta da DHT possam usar os mesmos recursos dos nós presentes na rede. O *gateway* age em nome do nó externo, executando as operações desejadas sem que haja necessidade de autenticação para tal. A rede pode ser acessada utilizando tanto Sun RPC[33] sobre TCP ou XML RPC[34] sobre HTTP. A figura 2.15 apresenta uma visão da função dos *DHT-Gateways.*

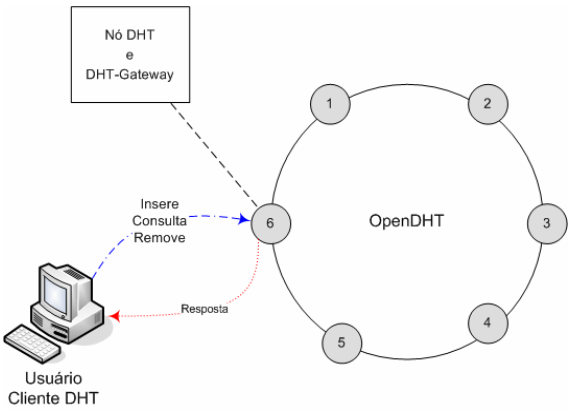


Figura 2.15: Acessando a OpenDHT

A OpenDHT utiliza uma implementação chamada Bamboo, que é uma implementação dos protocolos da tabela *hash* distribuída *Pastry*. Estes nós da OpenDHT estão alocados no PlanetLab[35] que é um conjunto de laboratórios de instituições científicas, geograficamente distribuídos pelo mundo, e que trabalham de forma colaborativa. Atualmente, cerca de 734 nós espalhados em 357 diferentes locais fazem parte da DHT, conforme mostrado na figura 2.16.

Figura 2.16: Distribuição Geográfica dos Nós da OpenDHT[35]

Para inserir um par na OpenDHT o cliente criado deve enviar as seguintes informações à rede:

- ***application***: *String* que define a operação a ser executada (*put*).
- ***key***: *Array* de *bytes*, com o máximo de 20 *bytes*, que informa a chave a ser armazenada.
- ***value***: *Array* de *bytes*, com o máximo de 1024 *bytes*, informa o valor a ser armazenado.
- ***ttl_sec***: Inteiro de 4 *bytes*, com o máximo de 604,800 segundos, que determina o tempo que a informação ficará armazenada na DHT.
- ***secret_hash***: *String* opcional gerada a partir de uma função *hash* SHA-1, usada para remover valores da DHT.

Como retorno desta requisição há três possíveis valores inteiros: 0 sucesso; 1 capacidade esgotada; 2 erro. *Para* buscar informações na OpenDHT o cliente deve enviar as seguintes informações:

- ***application***: *String* que define a operação a ser executada (*get*).
- ***key***: *Array* de *bytes*, com o máximo de 20 *bytes*, que informa a chave a ser pesquisada.
- ***maxvals***: Inteiro de 4 *bytes*, com o máximo de (2^{31} - 1), que especifica quantos valores devem retornar numa única pesquisa.
- ***placemark***: *Array* de *bytes*, com o máximo de 100 *bytes*, usado como um mecanismo de iteração para consultas com múltiplas respostas.

  Como resposta a essa requisição é retornado um *array* de *arrays* de *bytes* com os valores referentes à chave pesquisada.

Por apresentar simplicidade no desenvolvimento dos clientes, por conseqüência facilidade no acesso dos pares (*chave, valor*) na DHT, flexibilidade na escolha de chaves, ter controle, mesmo que simples, de autenticação para remoção dos valores e também por ser uma rede de acesso público constantemente testada, a OpenDHT foi a tabela *hash* distribuída utilizada pelo *Web2Peer*, como veremos nos capítulos posteriores.

2.8 Conclusão do Capítulo

A tecnologia *peer-to*-peer, apesar de ter seu primeiro registro histórico oficial datado de 1979 com o projeto Usenet, que permitia que computadores ligassem uns para os outros para trocar arquivos, tornou-se alvo das pesquisas de forma mais enfática nos últimos anos e continua em processo de melhoramento.

Este capítulo abordou os aspectos conceituais fundamentais referentes às tecnologias *peer-to-peer* utilizadas nesta pesquisa, envolvendo redes *peer-to-peer*, o conjunto de protocolos *JXTA* e as tabelas *hash* distribuídas.

Capítulo 3

Trabalhos Relacionados

3.1 Introdução

Embora alguns trabalhos já tenham tentado algumas soluções que, por um ou outro aspecto, são similares a infra-estrutura proposta, nenhum deles apresentou o conjunto de funcionalidades aqui apresentadas. Alguns desses trabalhos relacionados serão apresentados nesse capítulo.

3.2 O Projeto *FreeWeb*

Em 2001, o projeto *FreeWeb*[36] apresentou-se como uma alternativa para a distribuição de páginas *Web*, sem que houvesse forma de identificar o publicador das páginas, promovendo o direito de anonimato do usuário. Em termos práticos, o *FreeWeb* resume-se a uma interface gráfica sem muitos recursos que fornece acesso a rede *Freenet*[20], esta sim fornecendo as principais características e funcionalidades.

O *Freenet*, como citado anteriormente, é uma rede *peer-to-peer* que tem como característica principal o compartilhamento de espaço em disco para armazenamento distribuído de forma que os usuários não precisem ser identificados. Cada nó da rede *Freenet* possui uma tabela de armazenamento local, na qual são guardadas as chaves e os arquivos fornecidos pelo nó, e uma tabela de roteamento dinâmico, que possui endereços IP dos outros nós e as chaves que representam os arquivos armazenados por eles.

O fato de localização basear-se no endereço IP dos nós faz com que esta abordagem seja dependente da arquitetura de rede utilizada, neste caso, dependente do protocolo TPC/IP. A infra-estrutura apresentada neste trabalho, baseada no conjunto de protocolos JXTA, não é dependente de

tecnologias específicas e plataformas operacionais por basear-se num conjunto de protocolos independente de plataforma.

Para entrar na rede *Freenet* basta conhecer um dos nós presentes e então este nó terá uma rota para o nó que acaba de entrar na rede. Em seu funcionamento básico, o *Freenet* faz com que requisições sejam passadas nó a nó através de caminhos, chamados correntes, até atingir o nó desejado. Desta mesma forma as páginas são localizadas no *Freeweb*, onde ao procurar por determinado arquivo a requisição deve passar por toda a rede até atingir o nó fornecedor. A figura 3.1 mostra como um arquivo é localizado na rede *Freenet*.

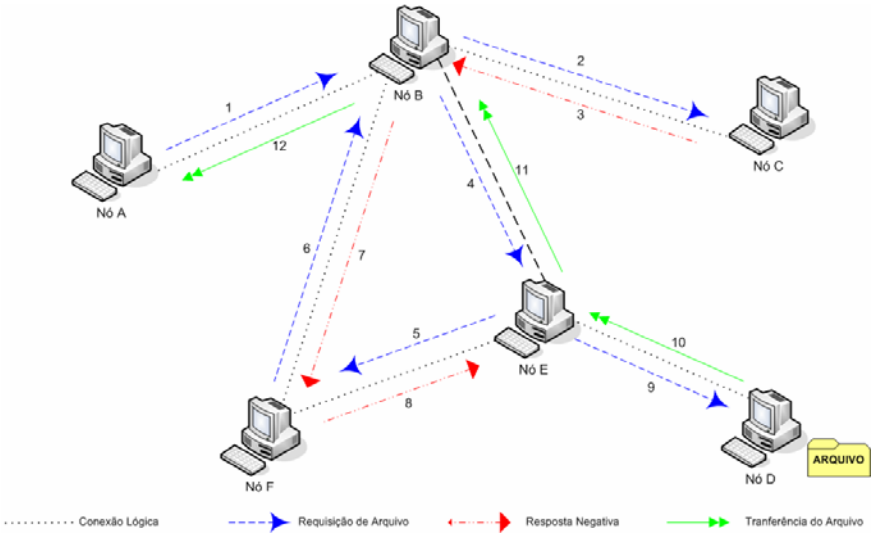


Figura 3.1: Busca de arquivos no *FreeWeb(Freenet)*

É importante observar que ao trazer o arquivo pelo caminho percorrido para localizá-lo os nós criam cópias deste para satisfazer pesquisas futuras. Este mecanismo de busca é bastante lento se comparado às buscas baseadas em DHT oferecidas pelo *Web2Peer*.

O *FreeWeb* oferece uma interface gráfica que interage com a rede *Freenet* limitando os arquivos publicados a arquivos HTML. Para visualizar estes arquivos, o usuário deve utilizar seu navegador tradicional, abrindo a página localmente após fazer o *download* da mesma.

O *website* do projeto *FreeWeb*[36] informa que apenas uma versão do projeto foi lançada e seu desenvolvimento ocorreu entre março e novembro de 2001 e desde então nenhum outro melhoramento foi aplicado. Recentemente uma mensagem foi publicada no *website* do projeto informando que ele estava encerrado e nenhuma melhoria seria implementada.

3.3 O *Web Browser* Compartilhado baseado no JXTA

Em 2003, outro projeto criou um navegador sobre uma rede *peer-to-peer*, utilizando a tecnologia JXTA. Porém, este *Web Browser* possui um objetivo específico de servir como navegador colaborativo para a execução de atividades em grupo para pesquisa e acesso à internet. Baseado em redes *peer-to-peer*, este *browser* permite que um grupo de usuários compartilhe as páginas acessadas de forma síncrona. Onde um documento visualizado por um determinado usuário é compartilhado com todos os demais[37].

Além de permitir que as operações realizadas por um dos usuários sobre este documento, de forma independente, sejam transmitidas aos demais fazendo com que todos visualizem as ações. Por isso o *browser* foi chamado por seus criadores de um sistema colaborativo de compartilhamento de tela e eventos.

O navegador foi implementado utilizando a linguagem Java e C/C++ e um pequeno protocolo próprio de mensagens, que são trocadas entre os nós, para que haja a sincronização das tarefas. Estas mensagens são transmitidas através dos recursos providos pelo conjunto de protocolos JXTA. Sobre a rede JXTA os clientes comunicam-se trocando informações sobre os seus estados. A figura 3.2 mostra os principais módulos do *browser* compartilhado. As mensagens são enviadas com *pipes* chamados de *propagate pipes*[22], que transmitem uma mesma mensagem para um grupo de outros nós ao mesmo tempo. (*broadcast* no *peergroup*)

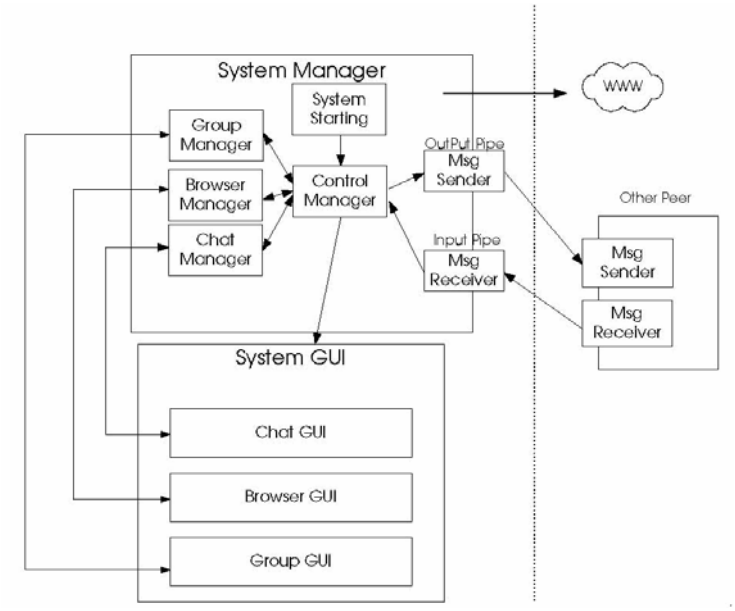


Figura 3.2: *Web Browser* Colaborativo em Blocos[37]

Os principais módulos do *browser* e suas respectivas funções são[37]:

- ***System Starting***: Responsável pela configuração de rede, segurança e conexão JXTA.
- ***Group Manager***: Responsável pela criação de grupos de trabalho através das funções *search(), creation(), join()* and *leave()*. Estes grupos servem para isolar grupos específicos, caso contrário o *browser* poderia ficar compartilhado com toda a internet.
- ***Chat Manager***: Mecanismo auxiliar para a comunicação através de mensagens entre os usuários do sistema.
- ***Browser Manager***: Trata das funções básicas do *browser*, como tratamento de *links* e eventos do *mouse*.
- ***Control Manager***: Controla os gerenciadores acima citados e também a interface gráfica e os canais de comunicação.
- ***System GUI***: São as três interfaces existentes: *Chat GUI*, *Browser GUI* e *Group GUI*.
- ***MsgSender* e *MsgReceiver***: São os canais de comunicação do *browser* que utilizam os *pipes* do JXTA.

Para a comunicação entre os nós o *browser* compartilhado utiliza um conjunto de sete mensagens pré-definidas, que são mostradas na tabela 3.1.

Tabela 3.1: Mensagens do Web Browser Colaborativo[37]

Mensagem	Nome do Elemento	Conteúdo do Elemento
0	SenderName	Logout Message
1	SenderMessage	Chat Message
2	WebURL	Internet URL
3	ScrollHValue	Horizontal Scroll Bar Position
4	ScrollValue	Vertical Scroll Bar Position
5	X,Y	Mouse Position
6	SenderName	Calling Parent Mode
7	WinHeight, WinWidth	Window Size

Esta abordagem não fornece nenhuma possibilidade de publicação de conteúdo por parte dos nós. Todas as páginas acessadas estão em servidores centralizados na internet. Desta forma, limita o uso deste navegador a um serviço colaborativo sem funções de publicação.

3.4 Coral: Uma Rede de Distribuição de Conteúdos

Com o objetivo de aumentar a disponibilidade dos conteúdos na internet, a solução apresentada pelo projeto Coral[38], visa aumentar a quantidade de cópias dos conteúdos para que os usuários não se deparem com problemas, nos servidores centralizados, que os impeçam de acessar os conteúdos que desejam.

Apesar de possuir um objetivo similar ao projeto aqui apresentado, o Coral sugere uma solução baseada em uma rede *peer-to-peer* que é executada apenas no lado dos servidores, na arquitetura cliente/servidor atualmente utilizada para publicar e localizar conteúdos na internet.

A solução apresentada pelo projeto Coral é cria uma camada de rede entre o cliente e o servidor, e esta camada construída sobre uma rede *peer-to-peer* replica os conteúdos providos pelos servidores, além de procurar estabelecer um balanceamento de carga, direcionando as requisições dos clientes para réplicas fisicamente mais próximas.

A solução é estruturada sobre três componentes principais, são eles:

- ***Coral Network***: gerenciador de todos os elementos da camada de rede entre os clientes e servidores de conteúdos tradicionais.
- ***Coral DNS Servers***: baseado no nome da URL solicitada procura um HTTP *Proxy* mais próximo do cliente.
- ***Coral HTTP Proxy***: armazenam cópias dos conteúdos disponibilizados pelos servidores tradicionais, associados à rede.

A rede Coral gerencia a interação entre *Coral DNS Ser*vers e *Coral HTTP Proxys* criando uma rede *overlay* entre os servidores de conteúdo e os clientes (*web browsers* tradicionais), conforme mostrado na figura 3.3.

Além dos servidores de DNS da rede Coral a estrutura depende ainda de servidores de DNS tradicionais, para que as requisições cheguem aos DNS da rede Coral. Para que isso ocorra, um requisito da rede Coral é pós-fixar à URL, de um determinado endereço, a seguinte expressão: ***.nyud.net:8080***. Com isso o endereço é encaminhado para os servidores de DNS da própria rede Coral, que faz o cliente chegar ao conteúdo desejado.

A figura 3.3 mostra o funcionamento da rede Coral. Neste exemplo temos um cliente representado por um *Web Browser* Tradicional, que deseja acessar uma página provida por um *Web Server* Tradicional através da rede Coral.

Para que esse acesso ocorra, é necessário que o cliente acrescente a expressão padrão, citada anteriormente, *".nyud.net:8080" a* URL desejada. Com isso, o DNS Tradicional encaminha a requisição a um servidor, na árvore ***.net***,, que faz parte dos servidores DNS Coral (passos 1 e 2 da figura 3.3). Este servidor DNS Coral é responsável por verificar qual o HTTP *Proxy* está mais próximo do cliente que solicitou a página, através de consultas à rede Coral (passos 3 e 4 da figura 3.3).

Após descobrir qual é o HTTP *Proxy* Coral mais próximo do cliente que requisitou a página (passo 5 e 6 da figura 3.3), o servidor DNS Coral informa ao cliente requisitante, através do DNS tradicional, qual é o HTTP *Proxy* Coral mais próximo do conteúdo e do próprio cliente (passo 7 e 8 da figura 3.3).

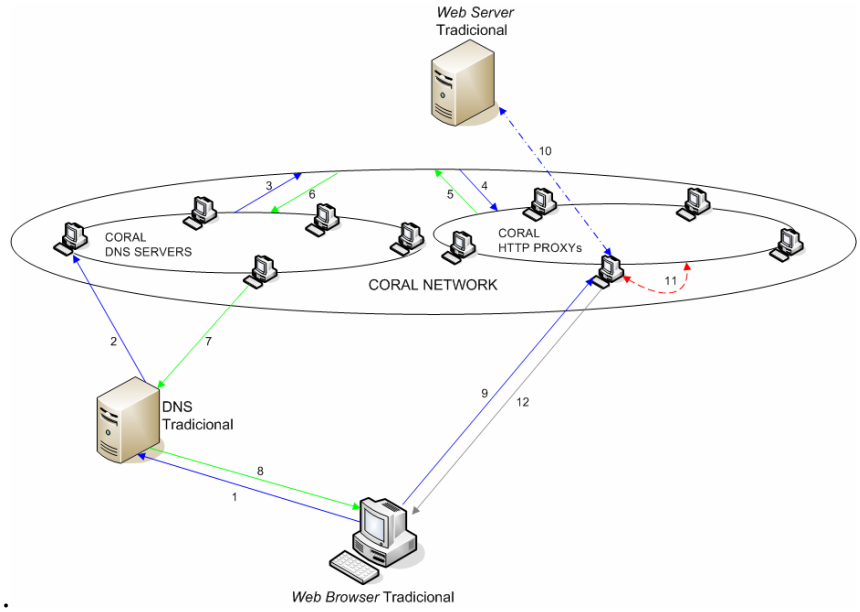


Figura 3.3: Rede de Distribuição de Conteúdo Coral[38]

Ao receber a informação sobre o *proxy* do seu DNS Tradicional (passo 8 da figura 3.3) o cliente solicita ao HTTP *Proxy* Coral o envio da página desejada (passo 9 da figura 3.3). Nesse momento há três possíveis caminhos, um deles é tomado sem que o cliente tome conhecimento.

Caso o HTTP *Proxy* Coral consultado possua uma cópia da página desejada ele simplesmente retorna essa página para o cliente (passo 12 da figura 3.3).

Caso o HTTP *Proxy* Coral não possua tal página, ele consulta a rede de proxys para saber quem é o proxy responsável pela página. Ao identificar este responsável, o HTTP *Proxy* Coral copia a página do responsável, faz uma cópia para si, e envia a página ao cliente. Na próxima requisição

por esta página o *proxy* terá a página e responderá mais agilmente à consulta (passo 11 e 12 da figura 3.3).

Caso nenhum dos *proxys* possua a página solicitada, o HTTP *Proxy* Coral consulta o servidor *Web* Tradicional, fornecedor primário da página, para obter uma cópia da mesma. Após Armazenar esta página localmente, envia a página solicitada ao cliente (passo 10 e 12 da figura 3.3).

Como dito anteriormente, essa é uma abordagem que procura aumentar a disponibilidade dos conteúdos, alterando algumas características funcionais do lado dos servidores. Neste caso uma camada entre os servidores e os clientes é acrescentada, e as URL tradicionais devem ser modificadas para que essa camada seja acessada, adicionando novos participantes nas operações atuais, onerando o tempo de consulta por páginas.

3.5 Outras Aplicações

Existem muitas aplicações difundidas entre as redes *peer-to-peer*, muitas delas clientes para acesso às redes e outras reimplementações das redes e concepções iniciais. Como exemplos de algumas dessas implementações poderíamos citar[15]:

- ***BitTorrent***: Aplicação de compartilhamento de arquivos integrada ao *Web browser*. É caracterizada por distribuir um simples arquivo em muitos *peers*.
- ***eDonkey***: Uma das primeiras aplicações *peer-to-peer* disseminadas na internet.
- ***Kademlia***: Rede *peer-to-peer* para compartilhamento de arquivos baseada em tabelas hash distribuídas.
- ***E-mule***: Surgiu inicialmente como um cliente para redes *eDonkey* criando algumas extensões ao protocolo, mais tarde aderiu também a rede *Kademlia*.
- ***FastTrack***: Rede *Peer-to-Peer* commercial implementada para diferentes aplicações, tais como o *Kazaa*. Por se tratar de uma tecnologia proprietária, vários detalhes da sua concepção são desconhecidos.
- ***Overnet***: Uma implementação da rede *Kademlia.*
- ***PeerCast*:** Uma aplicação baseada no *Gnutella* para propagação de conteúdo multimídia.

3.6 Conclusão do Capítulo

Neste capítulo foram apresentados alguns projetos existentes com focos parcialmente similares à infra-estrutura que aqui propomos. O *FreeWeb* propõe-se a oferecer uma publicação de

páginas mais simplificada, porém com recursos extremamente fracos. O *browser* compartilhado associando a tecnologia JXTA a um navegador para trabalhos colaborativos. O Coral oferecendo aumento na disponibilidade de conteúdos através de uma rede *peer-to-peer* localizada entre o servidor e o cliente.

Apesar disso, nenhum deles apresenta benefícios mais amplos, como os oferecidos pelo projeto *Web2Peer*, que oferece um ambiente simples para acesso e publicação das páginas na internet sem que haja a dependência de nenhum outro mecanismo externo. A própria infra-estrutura tem todos os mecanismos necessários para que o usuário comece a prover e acessar páginas assim que inicia o sistema. Ao mesmo tempo, estes trabalhos também serviram direta ou indiretamente como motivação para a criação da solução aqui descrita.

Capítulo 4

Web2Peer: Uma Proposta para Publicação de Páginas na Internet

4.1 Introdução

A infra-estrutura aqui apresentada utiliza as vantagens das redes *peer-to-peer* para aumentar a disponibilidade e reduzir a dependência de entidades reguladoras atualmente presentes na publicação de páginas na internet. Para atingir tal objetivo, foi criado um *Web browser*¸ chamado *Web2Peer*, que utiliza o conjunto de protocolos JXTA, para criar uma rede virtual para a distribuição das páginas e uma tabela *hash* distribuída (DHT), como serviço de localização de páginas nesta rede.

Este capítulo apresenta uma visão geral do modelo proposto, apresentando e detalhando o funcionamento dos seus principais componentes.

4.2 A Infra-estrutura *Web2Peer*

Todos os usuários desta rede são nós que podem publicar páginas ao mesmo tempo que acessam outras páginas. Cada nó tem internamente um *Web Server*, que é responsável por fornecer as páginas, e um *HTTP Client*, que acessa o conteúdo fornecido pelo *Web Server* através da rede *peer-to-peer*. Tudo isso implementado dentro de um *Web Browser* tradicional, mantendo as características convencionais de navegação na internet e acrescentando novas funcionalidades para a rede *peer-to-peer*. Sendo assim, conforme os nós vão acessando os conteúdos das páginas automaticamente, réplicas destas páginas vão sendo criadas na rede, tornando o modelo

independente de servidores centralizados, pois haverá várias fontes para um mesmo conteúdo. E tudo isso sendo feito a partir de um navegador, todos podem utilizar esta infra-estrutura sem se preocupar com detalhes de configuração.

A infra-estrutura é apresentada na figura 4.1, e em linhas gerais os seus principais componentes são:

- ***Web Browser***: interface do usuário com o sistema.
- **Editor HTML**: mecanismo para criação de páginas;
- ***Publisher***: mecanismo que publica as páginas na rede *peer-to-peer*.;
- **DHT**: tabela *hash* distribuída que indexa as páginas através dos seus conteúdo e localização na rede. É através da DHT que a busca das páginas é realizada.
- ***HTTP Client***: módulo utilizado para transferência das páginas do nó provedor ao nó requisitante.
- ***Web Server***: módulo provedor das páginas na rede *peer-to-peer*.
- **JXTA API**: módulo que fornece todas as funcionalidades da rede JXTA.

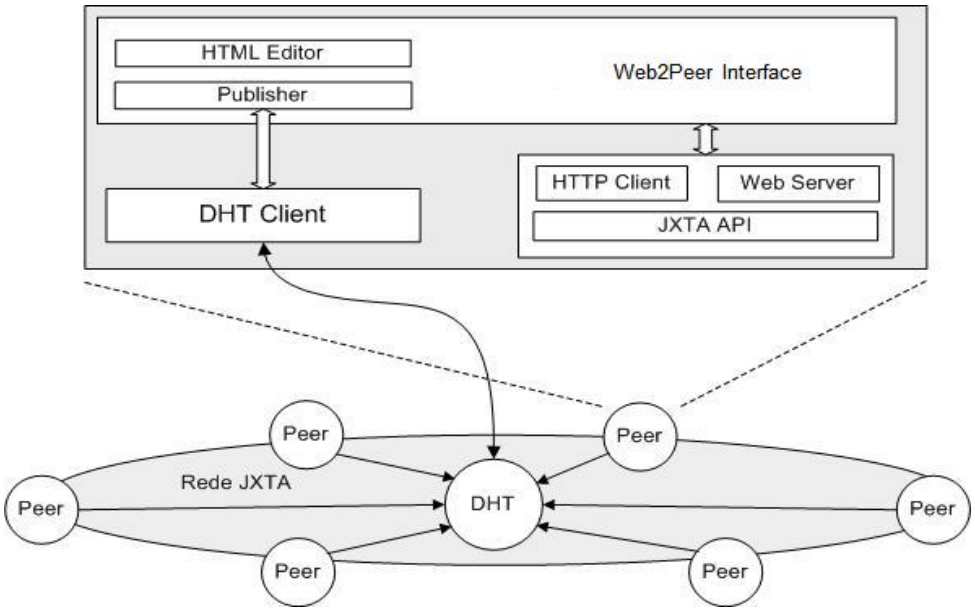


Figura 4.1: Arquitetura em Camadas do *Web2Peer*

Na figura 4.1 vemos a rede *peer-to*-peer com um elemento central, a DHT. A rede *JXTA* possui por si só um protocolo para busca de recursos, o *PDP* (*peer discovery protocol*), mas esta busca é realizada por inundação da rede (*flooding)*, o que prejudica muito o tempo de resposta. Por isso, como dito anteriormente, optou-se pela utilização de um serviço de indexação de conteúdos

provido por uma DHT para realizar buscas mais precisas das páginas, em comparação à forma tradicional de busca do JXTA. A indexação cria um relacionamento entre a localização do conteúdo na rede JXTA (um endereço P2P) e uma ou mais palavras-chave na DHT referentes à página publicada. Na seção *Publicação de Conteúdos*, deste capítulo, será abordado este assunto com mais detalhes.

O *"Publisher"* é um dos principais mecanismos do *Web2Peer*, pois é o responsável pela publicação direta das páginas na rede, tornando-as disponíveis para os demais *nós.*

Esta arquitetura proposta, do ponto de vista conceitual, pode ser classificada tanto como uma arquitetura parcialmente centralizada como descentralizada. Por ser baseada nos protocolos JXTA e com isso possuir alguns nós com características especiais, a rede *peer-to-peer* se caracteriza como uma arquitetura parcialmente centralizada. Mas ao agregar à infra-estrutura um serviço de indexação central caracteriza-se também como uma arquitetura descentralizada híbrida. Portando, esta infra-estrutura usa duas classificações conceituais ao mesmo tempo, podendo ser chamada de arquitetura **parcialmente centralizada e híbrida** [9].

Aqui fica clara uma diferença entre o modelo *Web* convencional e a abordagem proposta. Atualmente, quando alguém quer obter um determinado conteúdo na internet é necessários utilizar endereços URL[3] e URI[4], representando os nomes dos servidores que possuem as páginas desejadas. Esta é uma forma bastante prática de chegar a uma página *Web* conhecida. Porém, quando tais endereços URI/URL não são conhecidos, são necessários mecanismos externos de busca.

Atualmente, o grande referencial de buscas por contextos ou palavras-chave na internet é o *Google* [11]. Na nossa proposta, a principal função da DHT é que funcione tal como o *Google*, fornecendo referências de localização de conteúdos na rede *peer-to-peer*, sem a necessidade de utilização de URL ou URI.

Para atingir os objetivos aos quais se propõe, a infra-estrutura executa quatro tarefas consideradas básicas: Publicação de Páginas, Localização de Páginas, Replicação de Páginas e o Controle de Versões das páginas publicadas. Estas funcionalidades apresentadas pelo modelo proposto serão apresentadas a seguir.

4.2.1 Publicação de Páginas

Para criar e disponibilizar uma página na rede *peer-to-peer*, o usuário utiliza o editor HTML provido pelo próprio *Web Browser*. Após criar a página, o *Publisher* é ativado no próprio editor. Para cada documento (página) criado um número *n* de palavras-chave[1] deve ser informada pelo usuário. Estas palavras-chave são inseridas no *meta-tags* de um documento html. As palavras-chave são utilizadas pelo *Publisher* para indexar a página *Web* na DHT que mais tarde são utilizadas para procura de páginas.

Quando acionado pelo editor HTML (passo 1 da figura 4.2), o *Publisher* retira do documento HTML todas as palavras-chave inseridas como *meta-tags* (passos 2 e 3 da figura 4.2). Estas palavras-chave são gravadas no documento HTML pelo editor, como citado anteriormente. O exemplo a seguir mostra uma *meta-tag* criada com duas palavras-chave: *Soccer e Brazil*.

<META NAME="Keywords" CONTENT="*Soccer, Brazil*">

Após extrair as palavras, o *Publisher* publica as referências na DHT criando entradas do tipo <*chave*, *valor*> (passos 5 e 6 da figura 4.2). Onde a chave é uma palavra-chave da página *Web* e o valor é a localização da página na rede *peer-to-peer* (JXTA). Esta localização é informada ao *Publisher* através do próprio *peer* que está conectado à rede JXTA (passo 4 da figura 4.2)

A figura 4.2 mostra passo a passo como ocorre a publicação da página *Web*.

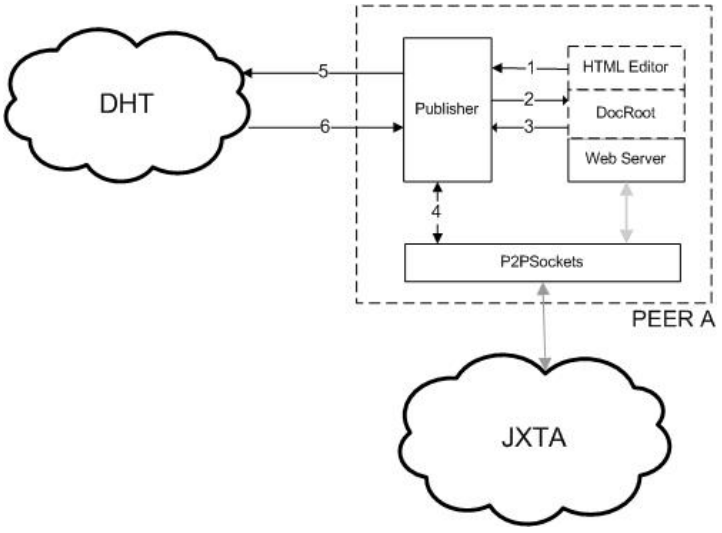


Figura 4.2: Publicação de páginas no *Web2Peer*

[1] As palavras-chaves são definidas de acordo com o assunto tratado na página *Web*.

Na figura 4.3, alguns exemplos de publicações na DHT. Destacado uma publicação realizada pelo *peer B* que publicou duas palavras-chave (*Soccer* e *Brazil*) associadas a sua página *index.html*.

	DHT entry ? <key, value>	
Peer	**Keyword (subject)**	**P2P address**
peer A	*Soccer*	*p2p://mypeeraddressA/index.html*
peer A	*Brazil*	*p2p://mypeeraddressA/index.html*
peer A	*Pelé*	*p2p://mypeeraddressA/index.html*
peer B	*Soccer*	*p2p://mypeeraddressB/index.html*
peer B	*Brazil*	*p2p://mypeeraddressB/index.html*
peer C	*Soccer*	*p2p://mypeeraddressC/index.html*
peer C	*Zidane*	*p2p://mypeeraddressC/index.html*
peer C	*World cup*	*p2p://mypeeraddressC/index.html*

Figura 4.3: Publicação de Páginas na DHT

4.2.2 Localização de Páginas

Para um outro usuário poder localizar e buscar uma página na rede *peer-to-peer*, o *Web2Peer* inicialmente acessa a DHT para pesquisar por palavras-chave referentes ao conteúdo desejado. A DHT devolverá como resultado desta pesquisa uma lista com todos os *peer* que possuem páginas referenciadas, no momento da publicação, com tais palavras. Esta lista é apresentada ao usuário no *Web2Peer Browser*.

Como cada palavra-chave está associada a um ou mais endereços *peer-to-peer* na DHT (ver a palavra-chave *Soccer* na figura 4.3) é necessário um esquema de filtragem. Por exemplo, se um *peer* fizer uma busca na DHT usando as palavras chaves *Soccer* e *Brazil* irá retornar para *Soccer* os endereços dos *peers A, B* e *C* e para *Brazil* os endereços dos *peers A* e *B.*

A regra de filtragem é a interseção desses dois resultados o que retorna apenas os endereços dos *peers A* e *B,* pois somente nesses dois *peers* os resultados correspondem às duas palavras pesquisadas. No entanto, se a procura for apenas pela palavra-chave *Soccer*, os resultados serão os endereços dos *peers A, B* e *C.*

A figura 4.4 mostra a localização e transferência de uma página *(página B)* no *Web2Peer*. Neste exemplo, inicialmente vemos o *peer A* consultando a DHT (passo 1), com determinadas palavras-chave, e recebendo uma lista informando que a página procurada está no *peerB* (passo 2). Então, o usuário clicando nesta opção faz com que o *HTTPClient,* usando o p2psockets, solicite ao *peer B* o envio da página *B,* através da rede JXTA (passos 3 e 4).

A página *B* é então transferida ao *peer* requisitante (*peerA*) e então exibida pelo seu browser (passos 5 e 6).

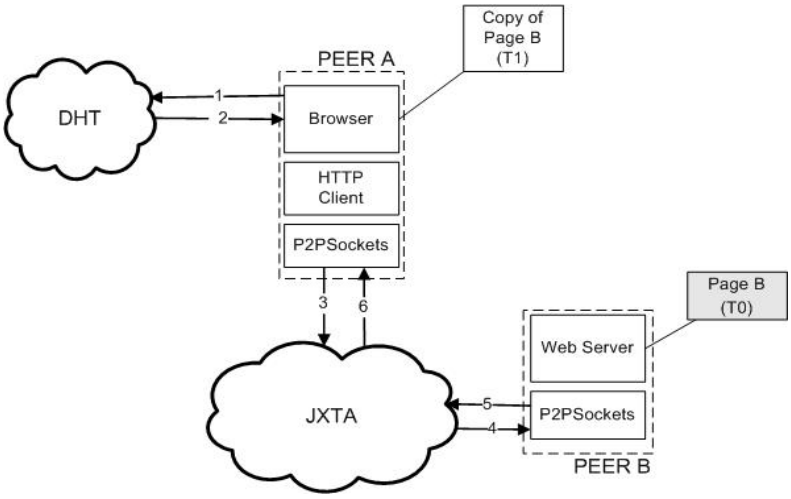


Figura 4.4: Localização de Páginas no *Web2Peer*

Ao receber a página, o *peer A* apresenta a página ao usuário na tela do *Web Browser.*

4.2.3 Replicação de Páginas

O JXTA realiza a replicação passiva dos recursos publicados na rede quando os *peers* acessam recursos de *peers* remotos. Esta replicação ocorre dinamicamente sem que haja intervenção do programador, devido à implementação do JXTA. Portanto, uma vez que uma página seja transmitida de um *peer* para outro, a replicação na rede *peer-to*-peer já acontece, porém a indexação da DHT não é atualizada automaticamente por essa replicação passiva.

Para que isso ocorra é necessário criar um mecanismo de replicação do indexador das páginas, provido pela DHT. Este mecanismo utiliza o *Publisher* para atualizar estas modificações. Cada vez que um conteúdo é acessado, uma nova publicação deve ser realizada na DHT para refletir o novo (mais um) provedor da página acessada. Por isso os processos de replicação e publicação são bastante similares.

A figura 4.5 mostra o procedimento de replicação onde o *peer C* acessa uma página chamada *Page B.* Nesta figura, vemos que após consultar a DHT (passos 1 e 2) o *peer C* recebe uma lista com duas possíveis fontes para o conteúdo procurado (*Page B*). Esta é uma página *Web* criada pelo *peer B* (*Page B copy 0*) e possui uma réplica armazenada no *peer A* (*Page B copy 1*), que foi copiada após o *Peer A* acessar a mesma página no *peer B.*

Ao decidir acessar a página no *peer A*, o HTTP Client do *peer C* solicita ao *peer A* o envio da página desejada (passos 3, 4 e 5). O *Web Server* do *peer A* envia a página solicitada (*Page B)* ao

peer C (passos 6, 7 e 8), que exibe na tela do *Web Browser* e salva uma cópia da página (*Page B copy 2*) no repositório de páginas local (*DocRoot*).

Em seguida, o mecanismo de replicação aciona o *Publisher* que gera uma nova publicação na DHT (passo 9). Esta publicação é realizada da mesma forma descrita no item *4.2.1 Publicação de Páginas*, deste capítulo.

A partir deste instante, qualquer *peer* que fizer uma busca de uma das palavras-chave indexadas à página (*Page B*), receberá da DHT os endereços dos *peers A, B* e *C* como provedores de tal página, aumentando a disponibilidade do conteúdo pelo aumento do número de cópias em fontes distintas.

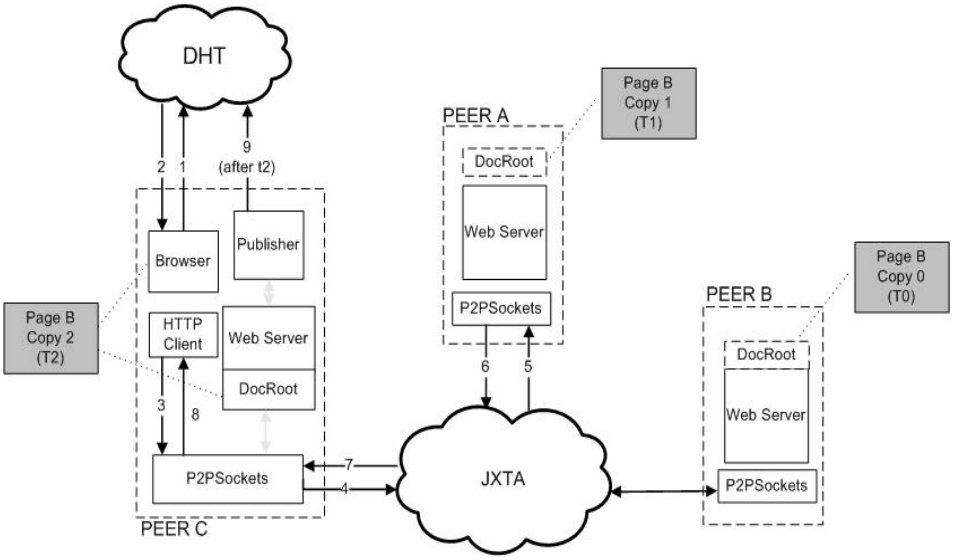


Figura 4.5: Replicação de Páginas no *Web2Peer*

4.3 Controle de Atualização de Páginas

Com o processo de replicação de páginas atuando normalmente, um problema intrínseco ao processo passa a fazer parte do sistema. Na mesma razão em que a disponibilidade de um conteúdo cresce com o aumento do número de cópias do mesmo, isso possibilita que, caso não haja nenhum tratamento para tal circunstância, ocorram possíveis alterações nos conteúdos, por alguns dos *peers* provedores, ficando o conteúdo divergente das suas cópias, sem que o usuário saiba qual é o conteúdo mais atualizado disponível na rede.

Este é um problema clássico bastante comum na área de desenvolvimento de software, onde vários programadores trabalham na criação de um mesmo sistema computacional. Nessas circunstâncias são utilizados sistemas de controle de versões, conhecidos como CVS (*control*

version systems) ou mecanismo de CSCW (*computer supported cooperative work*) [39, 40], estes sistemas tem a finalidade de gerenciar as diversas cópias existentes de um documento, ou programa em desenvolvimento, bem como manter um controle de concorrência sobre alterações realizadas.

Construir sistemas, com estes propósitos, na área de sistemas distribuídos é uma tarefa ainda mais complexa, pois a inexistência de um mecanismo centralizador aumenta a dificuldade de tratamento das inconsistências e concorrências presentes no sistema. Algumas abordagens tentam resolver estes problemas através do acordo entre os participantes do grupo, porém como a literatura clássica [41, 42] mostra não são problemas resolvíveis através de soluções triviais.

4.3.2 Abordagem Simplificada

Para evitar, ou reduzir, ao máximo o número de cópias com conteúdos divergentes, devido à possibilidade de alterações das páginas pelos *peers* auxiliares, o modelo proposto apresenta uma solução simplificada baseada na replicação ativa de documentos atualizados. A solução apresentada para este problema permite que somente o *peer* criador (dono) de uma página possa alterá-la, gerando uma atualização. Os outros *peers* , replicadores da página, devem descartar a antiga e obter essa nova atualização se quiserem continuar como *peers* provedores desta página.

Por tratar-se de um tópico de extrema relevância e complexidade, esta solução inicial tem o intuito de valorizar a utilidade da infra-estrutura proposta, porém define algumas premissas para atingir a simplificação desejada. Os mecanismos do *browser* proposto devem garantir tais premissas, que determinam:

- Somente o criador de um conteúdo pode modificá-lo.
- Caso haja diferentes versões de uma mesma página na rede, o usuário será informado de tal situação para que opte a partir de qual fonte deseja acessar uma página.
- Somente documentos editados no editor do *browser* devem ser publicados; documentos criados externamente devem ser excluídos da rede.

Para atingir tais metas o mecanismo de controle de atualizações deve integrar-se com outros mecanismos já apresentados, como o editor, *publisher*, DHT, etc. A DHT é um dos principais auxiliares neste controle, pois é baseada na subdivisão do campo valor, associado a uma chave específica de um conteúdo, que as versões são especificadas. A figura 4.6 mostra quais são os campos nos quais o valor de um par (*chave, valor*) é subdividido.

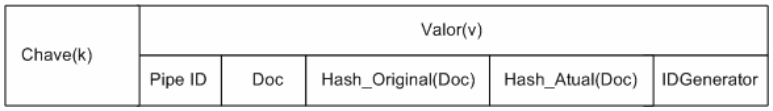


Figura 4.6: Campos do Par (*chave, valor*) armazenados na DHT

Na DHT o campo valor subdivide-se em outros campos com funções especificas de localização de documentos e controle de atualização. O campo *PiperID* identifica qual é o *pipe* de entrada de um *peer* específico na JXTA, onde uma página publicada, especificada pelo campo *Doc,* pode ser encontrada. Os campos seguintes são campos utilizados especificamente para o controle de atualização de páginas. O campo *Hash_Original(Doc)* armazena um resumo digital, obtido através de uma função *hash* aplicada ao conteúdo original da página, que identifica como a página foi concebida inicialmente (alterações realizadas em uma página *Web*, obrigatoriamente alteram o valor do *hash* desta página [43]) portanto este campo não é alterado e serve para rastrear todas páginas com a mesma origem, independente de qual versão possuem. O campo *Hash_Atual(Doc)* armazena o resumo digital de um conteúdo alterado pelo próprio autor da página e um número de versão, que é incrementado a cada nova atualização. Desta forma identifica uma alteração realizada sobre um determinado conteúdo, possibilitando mostrar ao usuário que uma nova versão está disponível. Portanto, se os campos *Hash_Original(Doc)* e *Hash_Atual(Doc)* foram iguais é porque a página *Web* nunca foi alterada (atualizada) pelo autor. E o último campo, *IDGenerator* identifica o criador da página publicada, este campo só é alterado no momento da primeira publicação da página, não sendo alterado em nenhuma outra circunstância.

4.3.3 O Processo de Atualização

Em linhas gerais, o processo de atualização de páginas é composto das seguintes ações:

- Quando um *peer* tentar editar uma página local, o sistema deve verificar se o nó é o dono do conteúdo;
- Quando um conteúdo é alterado, o conteúdo atualizado deve ser publicado na DHT (atualizando somente o campo *Hash_Atual(Doc)*) e os *peers* auxiliares no provimento deste conteúdo devem ser avisados;
- Caso alguma réplica não seja contatada, o browser, ao obter uma dessas réplicas, deve identificar o seu conteúdo como conteúdo antigo, isso é verificado na DHT;

- Ao entrar na rede *peer-to-peer* o nó deve verificar se os conteúdos, das páginas por ele providas, estão íntegros. Isto é feito através da comparação entre o resumo digital obtido no campo *Hash_Atual(Doc)* e dos resumos obtidos nos documentos disponíveis no *DocRoot*.

Para deixar o entendimento do processo mais detalhado será utilizado um exemplo mais próximo da prática. Para isso utilizaremos a rede, apresentada pela figura 4.7 (a,b), como exemplo.

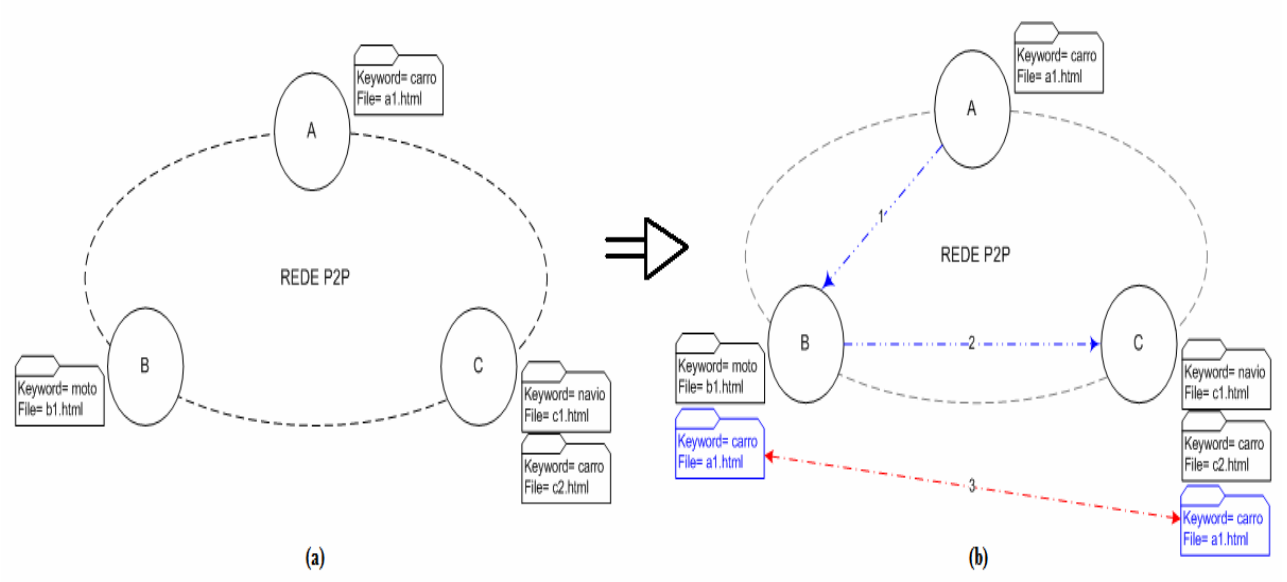


Figura 4.7: Ambiente de Replicação e Atualização de Páginas

Neste exemplo, vemos inicialmente na figura 4.7a, uma rede *peer-to-peer* que possui três nós chamados respectivamente de A, B e C, que disponibilizam quatro páginas (*a1.html, b1.html, c1.html* e *c2.html*) na rede. Cada uma das páginas foi publicada de acordo com as palavras-chave que identificam os conteúdos presentes nas páginas (*carro*, *moto* e *navio*). A figura 4.8 mostra o estado da DHT neste momento (figura 4.7a) em que a rede foi criada e os nós acrescentados, e seus respectivos arquivos publicados.

DHT - POSICAO INICIAL						
Chave (K)	Valor (V)	Valor Detalhado				
		PipeID	Doc	Hash_Original(Doc)	Hash_Atual(Doc)	IDGenerator
carro	p2p://PeerA/a1.html/123@123/v1@PeerA	PeerA	a1.html	123	123/v1	PeerA
moto	p2p://PeerB/b1.html/234@234/v1@PeerB	PeerB	b1.html	234	234/v1	PeerB
navio	p2p://PeerC/c1.html/345@345/v1@PeerC	PeerC	c1.html	345	345/v1	PeerC
carro	p2p://PeerC/c2.html/456@456/v1@PeerC	PeerC	c2.html	456	456/v1	PeerC

Figura 4.8: Estado Inicial da DHT

Em seguida, alguns acessos ocorrem nesta rede, conforme apresentado na figura 4.7b. O nó B procura por conteúdos referenciados com a palavra-chave *carro* e recebe como resposta dois nós

(nó A, página *a1.html* e nó C, página *c2.html*), e decide acessar o conteúdo do nó A. Após isso, o nó C pesquisa pela mesma palavra-chave *carro*, e recebe três possíveis provedores (nó A, página *a1.html*, nó B, página *a1.html* e o nó C, página *c2.html*), e decide acessar o conteúdo do nó B. Criando com isso uma replicação da página a1.html em dois novos nós (B e C).

Com a ocorrência destas replicações o estado da DHT, que é o indexador das páginas do sistema, também é alterado e as entradas referentes às novas localizações, nós B e C são adicionadas, conforme mostrado na figura 4.9.

DHT - REPLICACOES OCORRIDAS						
Chave (K)	Valor (V)	Valor Detalhado				
		PipeID	Doc	Hash_Original(Doc)	Hash_Atual(Doc)	IDGenerator
carro	p2p://PeerA/a1.html/123@123/v1@PeerA	PeerA	a1.html	123	123/v1	PeerA
moto	p2p://PeerB/b1.html/234@234/v1@PeerB	PeerB	b1.html	234	234/v1	PeerB
navio	p2p://PeerC/c1.html/345@345/v1@PeerC	PeerC	c1.html	345	345/v1	PeerC
carro	p2p://PeerC/c2.html/456@456/v1@PeerC	PeerC	c2.html	456	456/v1	PeerC
carro	p2p://PeerB/a1.html/123@123/v1@PeerA	**PeerB**	**a1.html**	**123**	**123/v1**	**PeerA**
carro	p2p://PeerC/a1.html/123@123/v1@PeerA	**PeerC**	**a1.html**	**123**	**123/v1**	**PeerA**

Figura 4.9: Entradas Adicionais na DHT

Quando um dos *peers* tentar alterar uma das páginas por ele provida, o processo de atualização, propriamente dito, se iniciará. Isto ocorre quando o usuário através do editor HTML embutido ao *Web2Peer* é executado e uma página local é selecionada para edição.

Quando isso ocorre, internamente, a primeira coisa que o mecanismo de atualização faz é consultar a DHT para verificar se o conteúdo, o qual foi selecionado para edição, é de propriedade[2] do *peer* em questão. A figura 4.10 mostra que para a página *a1.html*, presente nos *peers* A, B e C, somente o *peer* A pode editar tal conteúdo, pois o campo *IDGenerator* identifica o criador da página, impedindo que *peers* replicadores (nós B e C no exemplo) alterem o conteúdo das páginas.

DHT - PEERS TENTANDO EDITAR CONTEUDOS ALHEIOS						
Chave (K)	Valor (V)	Valor Detalhado				
		PipeID	Doc	Hash_Original(Doc)	Hash_Atual(Doc)	IDGenerator
carro	p2p://PeerA/a1.html/123@123/v1@PeerA	**PeerA**	a1.html	123	123/v1	**PeerA**
moto	p2p://PeerB/b1.html/234@234/v1@PeerB	**PeerB**	b1.html	234	234/v1	**PeerB**
navio	p2p://PeerC/c1.html/345@345/v1@PeerC	**PeerC**	c1.html	345	345/v1	**PeerC**
carro	p2p://PeerC/c2.html/456@456/v1@PeerC	**PeerC**	c2.html	456	456/v1	**PeerC**
carro	p2p://PeerB/a1.html/123@123/v1@PeerA	**PeerB**	a1.html	123	123/v1	**PeerA**
carro	p2p://PeerC/a1.html/123@123/v1@PeerA	**PeerC**	a1.html	123	123/v1	**PeerA**

Figura 4.10: Verificação de Autor para Edição da Página

[2] Uma página é considerada de propriedade de um determinado nó caso este seja o seu criador inicial.

Na verdade, o *peer* A possui uma senha que lhe permite alterar apenas as entradas <chave, valor> inseridas por ele na DHT.

Com isso pode-se dizer que, mesmo de forma rudimentar, há um controle de acesso para atualização de páginas. Caso o nó tenha direito de edição da página (nó A, página *a1.html*, do exemplo), o editor entra em modo de edição e permite que o usuário altere a página desejada. Após realizar todas as alterações desejadas e salvar o novo arquivo, o controle da operação passa novamente ao *Web2Peer*, que gera uma lista, chamada *backup_list*, com todos os *peers* auxiliares que possuem réplicas do conteúdo antigo. Esta lista é gerada com uma consulta na DHT pelo documento em atualização. A *backup_list* permitirá ao *peer* A avisar os *peers* auxiliares para que obtenham a nova atualização da página, para que seja reduzido o número de versões diferentes de uma página na rede.

Após isso, o *browser* altera a entrada atual da DHT que referencia este documento atualizado, com o novo resumo (*hash*) do documento, informando à rede que uma nova versão do arquivo anterior foi criada. Isso é feito substituindo-se o campo *Hash_Atual(Doc)* com o novo valor obtido através da aplicação de uma função *hash* ao novo arquivo e incrementando o número da versão. É importante ressaltar que campo *Hash_Original(Doc)* é mantido para que sirva como referência do novo conteúdo com o antigo. Assim, as novas consultas realizadas por outros nós na rede poderão identificar uma página com versões distintas, possibilitando que o usuário opte qual versão deseja acessar. O tratamento dessa informação é realizado pelo *browser*, e apresentada pela interface gráfica, quando uma pesquisa é realizada. Isto é baseado nos campos *Hash_Atual(Doc)* e *Hash_Original(Doc)* de cada página retornada na pesquisa. Na figura 4.11, onde é apresentada uma DHT com o novo valor do *Hash_Atual(Doc)* alterado e a versão incrementada, após a edição do documento.

DHT - PEER A ALTERA O CONTEUDO DA PÁGINA a1.html						
Chave (K)	Valor (V)	Valor Detalhado				
		PipeID	Doc	Hash_Original(Doc)	Hash_Atual(Doc)	IDGenerator
carro	p2p://PeerA/a1.html/123@321/v2@PeerA	**PeerA**	**a1.html**	**123**	**321/v2**	**PeerA**
moto	p2p://PeerB/b1.html/234@234/v1@PeerB	PeerB	b1.html	234	234/v1	PeerB
navio	p2p://PeerC/c1.html/345@345/v1@PeerC	PeerC	c1.html	345	345/v1	PeerC
carro	p2p://PeerC/c2.html/456@456/v1@PeerC	PeerC	c2.html	456	456/v1	PeerC
carro	p2p://PeerB/a1.html/123@123/v1@PeerA	PeerB	**a1.html**	**123**	**123/v1**	PeerA
carro	p2p://PeerC/a1.html/123@123/v1@PeerA	PeerC	**a1.html**	**123**	**123/v1**	PeerA

Figura 4.11: *Peer* A Atualiza a Página *a1.html*

A partir de então, com o documento publicado, o *Web2Peer* inicia a segunda fase da atualização. A partir da lista *backup_list* armazenada no inicio do processo, o nó tenta contatar

todos os *peers* auxiliares desta lista e envia para cada um deles uma mensagem de atualização da replicação, através da rede *peer-to-peer*, para que eles atualizem localmente a página alterada. Este processo é inspirado na replicação ativa das redes *peer-to-peer*, conforme citado no *capítulo 2, seção 2.4 Replicação em Redes Peer-to-Peer.*

Caso todos os nós sejam contatados, a referida página é atualizada em cada um deles, nos seus diretórios *DocRoot* e também nas entradas da DHT, equalizando a versão local do documento com a versão recém atualizada pelo autor no peer A, conforme mostrado na figura 4.12. Quando isso ocorre o processo é finalizado com a atualização máxima atingida (*update total)*, referente ao número de réplicas existentes no momento da atualização.

DHT - PEER INICIA REPLICACAO ATIVA DE a1.html => UPDATE TOTAL						
Chave (K)	Valor (V)	Valor Detalhado				
		PipeID	Doc	Hash_Original(Doc)	Hash_Atual(Doc)	IDGenerator
carro	p2p://PeerA/a1.html/123@321/v2@PeerA	PeerA	**a1.html**	**123**	**321/v2**	PeerA
moto	p2p://PeerB/b1.html/234@234/v1@PeerB	PeerB	b1.html	234	234/v1	PeerB
navio	p2p://PeerC/c1.html/345@345/v1@PeerC	PeerC	c1.html	345	345/v1	PeerC
carro	p2p://PeerC/c2.html/456@456/v1@PeerC	PeerC	c2.html	456	456/v1	PeerC
carro	p2p://PeerB/a1.html/123@321/v2@PeerA	PeerB	**a1.html**	**123**	**321/v2**	PeerA
carro	p2p://PeerC/a1.html/123@321/v2@PeerA	PeerC	**a1.html**	**123**	**321/v2**	PeerA

Figura 4.12: Atualização Máxima de Réplicas

No entanto, caso não seja possível contatar algum dos nós presentes na lista *backup_list* (ou outros que não estão nesta lista), seja por problemas de comunicação ou saídas e/ou entradas repentinas dos nós, estes nós permanecerão com seus conteúdos desatualizados, restando ao usuário optar qual das páginas irá acessar, quando realizar uma consulta. Este processo é chamado de atualização parcial de réplicas, e o estado da DHT, para este caso, é mostrado na figura 4.13. Vale ressaltar que na consulta o usuário pode localizar no campo *Hash_Atual(Doc)* o número de versão e saber qual a versão mais atualizada da página.

DHT - PEER INICIA REPLICACAO ATIVA DE a1.html => UPDATE PARCIAL						
Chave (K)	Valor (V)	Valor Detalhado				
		PipeID	Doc	Hash_Original(Doc)	Hash_Atual(Doc)	IDGenerator
carro	p2p://PeerA/a1.html/123@321/v2@PeerA	PeerA	**a1.html**	**123**	**321/v2**	PeerA
moto	p2p://PeerB/b1.html/234@234/v1@PeerB	PeerB	b1.html	234	234/v1	PeerB
navio	p2p://PeerC/c1.html/345@345/v1@PeerC	PeerC	c1.html	345	345/v1	PeerC
carro	p2p://PeerC/c2.html/456@456/v1@PeerC	PeerC	c2.html	456	456/v1	PeerC
carro	p2p://PeerB/a1.html/123@123/v1@PeerA	PeerB	**a1.html**	**123**	**123/v1**	PeerA
carro	p2p://PeerC/a1.html/123@321/v2@PeerA	PeerC	**a1.html**	**123**	**321/v2**	PeerA

Figura 4.13: Atualização Parcial de Réplicas

Outro ponto importante deste processo, é que todas as vezes que o *Web2Peer* conectar-se a rede *peer-to-peer*, ele verifica, baseado nos campos de resumo de páginas (*hash*) da DHT, se alguma página local foi alterada. Essa verificação de integridade dos arquivos é importante para garantir que documentos de autores externos não sejam alterados manualmente ou substituídos, através de ferramentas externas ao *browser*¸ e os conteúdos das páginas sejam atualizados de forma a atender as propriedades estipuladas pela infra-estrutura.

4.4 Conclusão do Capítulo

Neste capítulo, foram apresentadas as características funcionais da infra-estrutura proposta, bem como sua interface gráfica, módulos internos e mecanismos funcionais, que incluem os mecanismos de publicação, localização e replicação de páginas *Web*. Além disso, foi apresentado um mecanismo simples, baseado em replicação ativa, para a atualização controlada das páginas providas pela infra-estrutura proposta.

No capítulo a seguir serão apresentados os detalhes de como os principais mecanismos e módulos foram implementados.

Capítulo 5

Implementação e Avaliação dos Resultados

5.1 Introdução

Um dos pontos considerados importantes, quando iniciado o projeto e o desenvolvimento do primeiro protótipo, foi a compatibilidade do novo modelo com a forma usada atualmente para acesso à internet.

Portando, um dos requisitos estipulados foi que o *Web2Peer* mantivesse as mesmas funcionalidades atuais e acrescentasse à tecnologia tradicional as vantagens do novo modelo de acesso, conforme descrito no capítulo anterior.

Neste capítulo serão apresentados os detalhes da implementação do protótipo utilizado para a prova de conceito e avaliação da infra-estrutura proposta, bem como alguns resultados obtidos nos testes realizados com este protótipo.

5.2 O Navegador

No momento em que é iniciado o navegador executa algumas tarefas importantes. A primeira delas é conectar-se a rede JXTA (*peer-to-peer).* Durante esta conexão, o *peer* recebe um identificador único na rede (JXTA ID[22]). Para esta conexão o navegador utiliza *peer rendezvous* ou *peer relays*, dependendo do ambiente de rede no qual o nó está inserido. O navegador criará também, neste momento, um *pipe* de entrada que aguardará por requisições de páginas através do seu servidor *Web Jetty*[44], reescrito para atender estas requisições através da rede JXTA,

possibilitando a transferência de todas as páginas solicitadas pelos clientes *HTTP[45]* dos outros *peers* da rede. A figura 5.1 mostra a interface do *Web2Peer* e os seus principais componentes:

(1) **Botões de controle**: *Voltar*, *Avançar*, *Parar*, *Recarregar*, *Editor HTML*, *Página Inicial*, *Ferramentas* e *Preferências*.

(2) **Barra de endereços**.

(3) **Barra de pesquisa** de páginas, por palavras-chave.

(4) **Área de exibição de páginas *Web***.

Figura 5.1. Interface do *Web2Peer*

Assim que o *Web2Peer* é carregado, o usuário pode fazer qualquer consulta por conteúdo na rede, a partir da barra de pesquisa de conteúdos (Item 3 da figura 5.1). Quando uma ou mais palavras-chave, relacionadas ao conteúdo desejado, são inseridas neste campo o *Web browser* realiza a pesquisa na DHT para localizar em quais *peers* pode obter as páginas referentes aos assuntos pesquisados.

O navegador (*Web2Peer*) foi desenvolvido utilizando a linguagem de programação Java em virtude da sua independência de plataforma.

5.2.1 Diagrama em Blocos

O navegador criado possui módulos divididos em dois principais grupos de funcionalidades, os módulos responsáveis pela interface gráfica, chamado GUI, e os módulos responsáveis pelos mecanismos *peer-to-peer*, chamado de Engine.

Estes dois principais grupos foram também divididos em blocos menores para que o protótipo ficasse mais modularizado, facilitando assim a sua manutenção e melhoria da infra-estrutura. O diagrama dos principais módulos do protótipo é apresentado na figura 5.2.

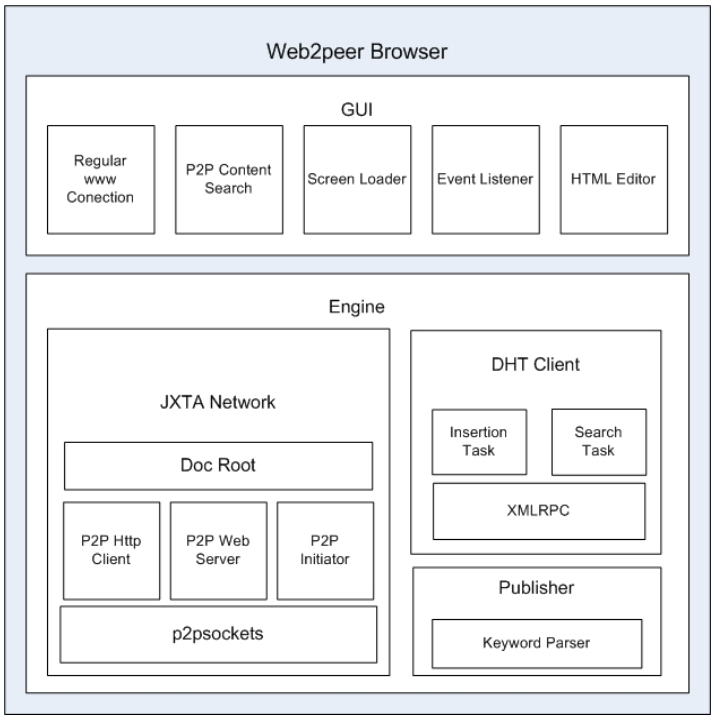


Figura 5.2: Diagrama em Blocos do *Web2Peer*

O módulo da GUI tem cinco principais funcionalidades, todas interagindo diretamente com o usuário através da interface. As funções mais primitivas deste módulo foram criadas com a utilização de uma biblioteca, de domínio público, chamada Calpa HTML[46]. Esta biblioteca fornece todas as funcionalidades para interação com a internet atual, através de métodos que permitem comunicação através do protocolo HTTP, métodos para apresentação das páginas na tela, bem como um conjunto de tratadores de eventos (*listeners*) para tratar as diversas ações do usuário sobre a tela do *Web2Peer*.

O módulo *engine* é responsável pelas principais tarefas do *Web2Peer*. Todas as funções da rede JXTA, do *Publisher* e da DHT estão neste bloco e as veremos com mais detalhes a seguir.

5.3 Cliente DHT

5.3.1 A DHT Utilizada

As três principais funcionalidades da infra-estrutura usam diretamente a DHT. Entre algumas possibilidades, optou-se por utilizar uma implementação de uma DHT previamente testada e consolidada.

A *DHT* escolhida foi o projeto *openDHT*[32], que é um serviço de acesso público executado atualmente, conforme citado anteriormente, em mais de 700 nós espalhados por laboratórios no mundo, sob a responsabilidade do *PlanetLab[35]*. Cada um desses nós utiliza o *Linux* como sistema operacional e executa a implementação *Bamboo*[47]. O *Bamboo* é uma implementação do protocolo *Pastry*, que foi apresentado em 2001, quando as primeiras arquiteturas de DHT surgiram, como já citado.

Cada nó da OpenDHT armazena em seu disco local uma porção do armazenamento total da DHT. Além disso, realizam operações de inserção, consulta e remoção de pares <*chave*, *valor*> e outras operações de roteamento de mensagens. Cada nó da *openDHT* também é chamado de *DHT-gateway*[48], pois permite que clientes que não participam da DHT também armazenem, consultem e removam pares como qualquer outro nó presente na DHT. É essa funcionalidade que permite ao *Web Browser* criar o seu serviço de indexação de páginas e suas respectivas localizações.

5.3.2 O Cliente DHT Desenvolvido

O cliente DHT utilizado pelo *Web2Peer* usa a DHT através do DHT-Gateways sem fazer parte direta da DHT. Para comunicar-se com o *DHT-Gateway*, o Cliente DHT utiliza XMLRPC[49]. O cliente DHT monta as mensagens necessárias para executar as funções de inserção, consulta e remoção de valores e as envia ao *DHT-Gateway* que se encarrega de executá-las.

Para inserir um valor na DHT, uma mensagem precisa ser criada com os seguintes campos: *key, value, put_application_name, secret e ttl.* O campo *key* representa a chave (palavra-chave) a ser inserida na DHT e *value* é o valor (endereço p2p) referente à chave inserida. Os outros são campos de controle, *put_application_name,* é o campo que identifica o método a ser executado, *secret* que é a senha utilizada em caso de necessidade de remoção do par da DHT e *ttl* que é o tempo, em segundos, que tal entrada deverá permanecer na DHT.

A figura 5.3 mostra a mensagem XML montada pelo cliente DHT para inserção de um par na OpenDHT. O retorno desta requisição de inserção é uma mensagem com três possíveis respostas: sucesso, capacidade esgotada e falha.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
   <methodName>put_application_name</methodName>
   <params>
      <param><value> key </value></param>
      <param><value> value </value></param>
      <param><value> secret_hash </value></param>
      <param><value><int> ttl </int></value></param>
   </params>
</methodCall>
```

Figura 5.3: Mensagem de inserção na DHT

A inserção do par (*chave, valor*) na DHT é sem dúvida o passo mais importante para a publicação de uma página, porém como citado no capítulo anterior, a publicação de uma página envolve ainda outras atividades executadas previamente.

A chave inserida precisa ser retirada dos *meta-tags* do arquivo HTML. O responsável por retirar as palavras-chave do arquivo fonte da página é um mecanismo interno ao *Publisher.*

Este mecanismo chamado de *KeyWord Parser* é quem faz a leitura do arquivo e retira cada uma das palavras para que o cliente DHT monte as mensagens XML enviadas ao DHT-Gateway como citado anteriormente. A figura 5.4 mostra o código do *Keyword Parser* usado para remoção das palavras-chave.

O *KeyWord Parser* é invocado pelo próprio *Publisher* que ao receber como retorno um *array* de *Strings* com todas as palavras-chave indexadas no arquivo, chama o cliente DHT para que as mensagens sejam montadas e enviadas à DHT para publicação.

```
public class KeywordParser {

	@SuppressWarnings("unused")
	private String[] getMetaKeywords(String arquivo){

		String[] keywords = null;
		String line;
		BufferedReader inReader = null;

		if(arquivo == null){
			return null;
			}

	    try {
			inReader = new BufferedReader(new FileReader(arquivo));
		    }
		    catch( FileNotFoundException e1 ) {
		        System.err.println("Error reading the file: " + arquivo);
		        return null;
		    }
		try {
		    while((line = inReader.readLine())!= null) {
		        if(!(line.trim().equals("")) && (line.substring(0,5).equals("<META"))) {
		            String tagParcial = line.substring(12,20);
		            if(tagParcial.equals("Keywords")){
		                System.out.println(line);
		                keywords = this.getTokens(line.substring(31,line.length()-2).trim(), ", ");
		                }
		            }
		        }
		    inReader.close();
		    }
		    catch (IOException e1) {
		        System.err.println("Erro lendo arquivo de entrada");
		        return null;
		    }
		return keywords;
	}

	private String[] getTokens(String text, String delimitador){

		StringTokenizer tokens = new StringTokenizer(text, delimitador);
		if (tokens.countTokens() <= 0) return null;
		String[] names = new String[tokens.countTokens()];
		for (int i = 0; i < names.length; i++)
		    names[i] = tokens.nextToken();
		return names;
	}
}
```

Figura 5.4: Mecanismo KeyWord Parser

Após a criação da página a retirada das palavras-chave, a montagem e envio das mensagens XML para a DHT são as tarefas mais importantes do processo de publicação da página. Para entender como o processo completo de publicação funciona veja a figura 5.5.

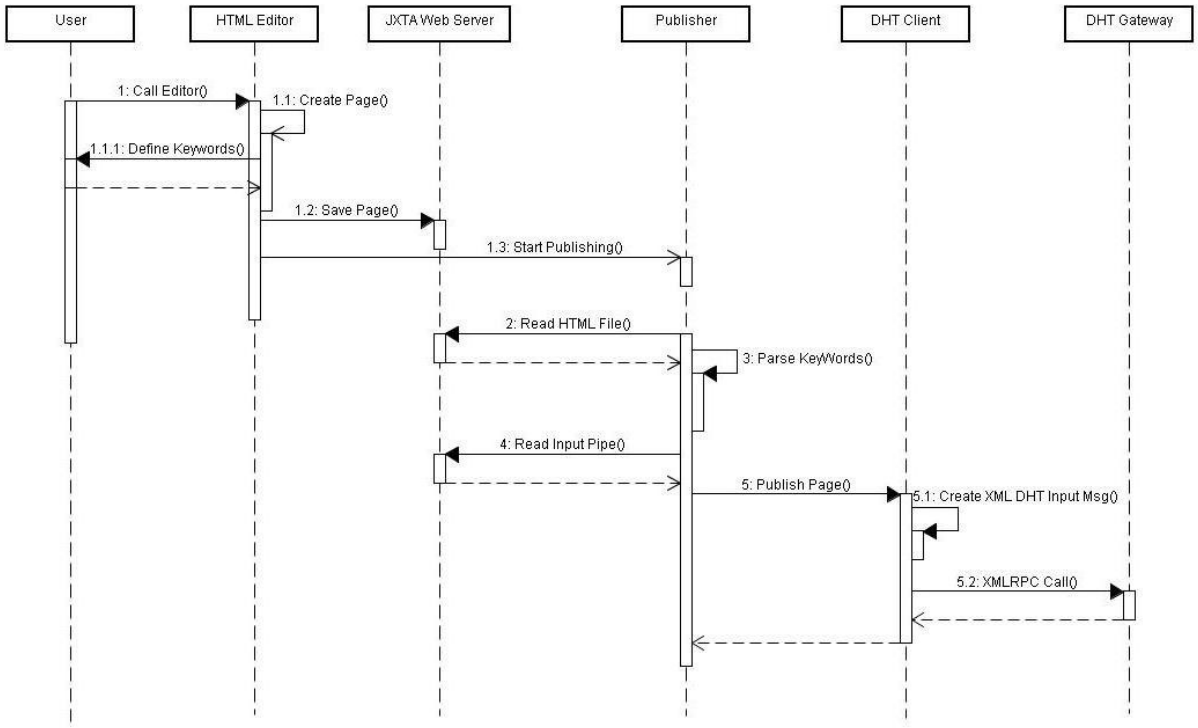


Figura 5.5: Processo Completo de Publicação de uma Página

Para consultar um valor na DHT, uma mensagem de busca precisa ser enviada ao *DHT-gateway*. Esta mensagem possui 4 campos: *value*, *get_application_name*, *maxvals* e *placemark.*

O campo *value* é a palavra-chave que se deseja consultar. Os outros três são parâmetros de controle enviados à DHT, *get_application_name* representa o nome do método a ser executado pela DHT, *maxvals* e *placemark*, que representam o número máximo de elementos que devem ser retornados por consulta e o *flag* de paginação que indica a existência de mais elementos no *array* de resultados, além daqueles determinados por *maxvals.*

O retorno desta consulta é um *array* com todos os valores encontrados para as chaves informadas na pesquisa. A figura 5.6 mostra a mensagem montada pelo cliente DHT para enviar uma consulta à DHT.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
   <methodName>get_application_name</methodName>
   <params>
      <param><value> key </value></param>
      <param><value> maxvals </value></param>
      <param><value> placemark </value></param>
   </params>
</methodCall>
```

Figura 5.6: Mensagem de Busca na DHT

Como há um número elevado de *DHT-gateways* ativos na OpenDHT, foi criado também uma ferramenta que permite ao usuário do *Web2Peer* gerenciar os seus *gateways*. Internamente o *Web2Peer* possui uma lista pré-definida de *gateways*, na qual o usuário pode selecionar o *DHT-gateway* que deseja usar. Com isso o usuário pode selecionar outro gateway da DHT em caso de falha na comunicação com o gateway definido como padrão. Além disso, a ferramenta permite atualizar esta lista, através da opção *Localizar* Gateways, conectando via HTTP e fazendo o *download* da lista atualizada de *gateways*. A figura 5.7 mostra a tela de gerenciamento dos gateways.

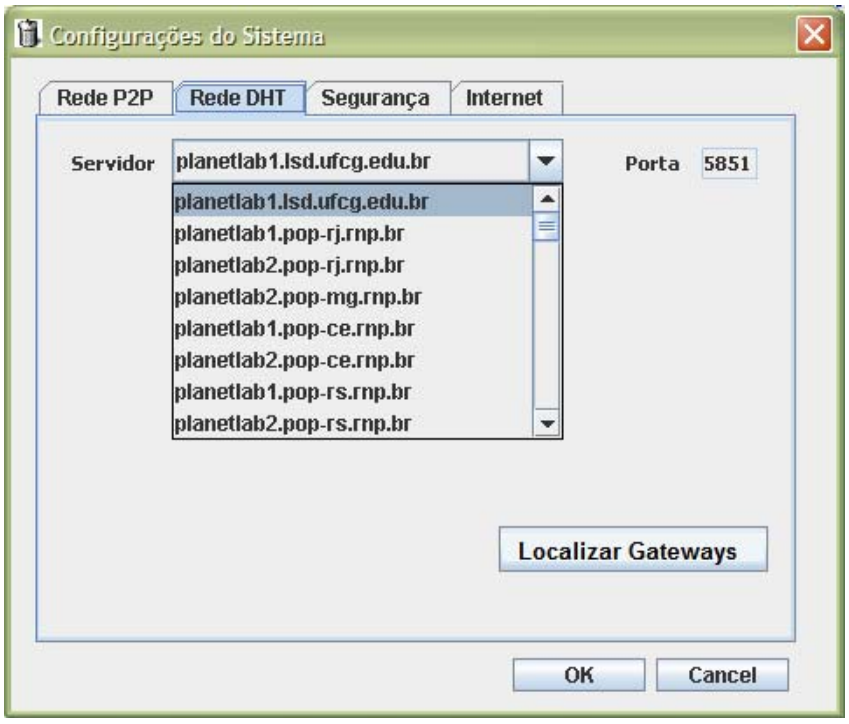


Figura 5.7: Gerenciador de Gateways do *Web2Peer*

5.3.3 Atualizações de Páginas na DHT

Para que a atualização de páginas fosse realizada, conforme descrito na *seção 4.3 Controle de Atualização de Páginas*, alguns métodos auxiliares foram necessários, integrando-se as funcionalidades básicas do *Web2Peer*. Alguns destes métodos auxiliares são:

- *checkPageConsistence()* : Verifica se o valor do resumo digital (*hash)* de cada uma das páginas presentes no diretório *DocRoot* do *Web2Peer*, corresponde às suas respectivas publicações na DHT. A resposta desta verificação é um *array* chamado *test_result*, que

contém as páginas não-íntegras. Caso este *array* esteja nulo, todas as páginas estarão consistentes.

- *getOneKey(selected_page)*: Extrai uma das palavras-chave de uma determinada página, função similar à executada pelo *Parser*. Este método auxilia a localização do *IDGenerator* referente a página que o usuário quer alterar. A primeira palavra-chave localizada é armazenada na variável *first_key_page*.
- *findValue(first_key_page, selected_page, peerID)*: Procura na DHT uma determinada página, através de uma das suas palavras-chave, que reside em um determinado nó. Ao localizar a página com tais características, o campo *valor*¸ correspondente a página é armazenado em um *array* chamado *pageValue*.
- *getIDGen(pageValue)*: Extrai o *IDGenerator* de um campo *valor*, correspondente a uma página *Web*. Este valor é usado para verificar se o nó local, que deseja editar a página, é o criador desta (*selected_page*). Caso estes campos sejam iguais o editor entra em modo de edição, permitindo a alteração da página.
- *getBackupList*(*first_key_page, selected_page*): Procura na DHT por todos os nós que possuem cópias da página que foi alterada. Isso é realizado filtrando-se o campo *Hash_Original* de uma determinada página, que é especificada pelos campos passados como parâmetro ao método em questão.
- *startLocalUpdate(key, value, pass)*: Atualiza o campo *Hash_Original* de uma determinada página, incrementando também a versão da página. Esta atualização refere-se apenas às publicações do nó criador do conteúdo.
- *startRemoteUpdate(page,key,backup_list)*: Envia para todos os nós, armazenados na *backup_list*¸ uma mensagem informando que há uma nova versão da página informada. Esta mensagem é enviada através da rede JXTA e a atualização remota dos conteúdos é realizada por um novo acesso a pàgina, através dos mecanismos básicos de acesso.

Estes métodos em conjunto com os outros mecanismos que foram, ou serão, mostrados proporcionam o funcionamento de acordo com o proposto pela infra-estrutura.

Na figura 5.8 é mostrado como as informações de versão chegam ao usuário, bem como os campos que são utilizados pelo sistema para identificar autores, provedores, versões, entre outros. Nesta figura é possível observar também que o *peerA* é o autor da página *a1.html*, que está replicada nos nós *peerB* e *peerC*. Esta constatação é possível, pois estes nós possuem os mesmo valores para o

campo *Hash_Original* da página *a1.html*. É possível observar também que o *peerA* atualizou o conteúdo da página *a1.html*, pois o campo *Hash_Atual* é diferente do *Hash_Original* e tem versão 2 assinalada para esta mesma página. Outro ponto importante a ser notado é que o *peerA* propagou a nova versão da página alterada para a rede, mas apenas o *PeerB* possui uma réplica atualizada da página. Por algum motivo, o *peerC* possui uma versão antiga, apresentada no seu campo *Hash_Atual*.

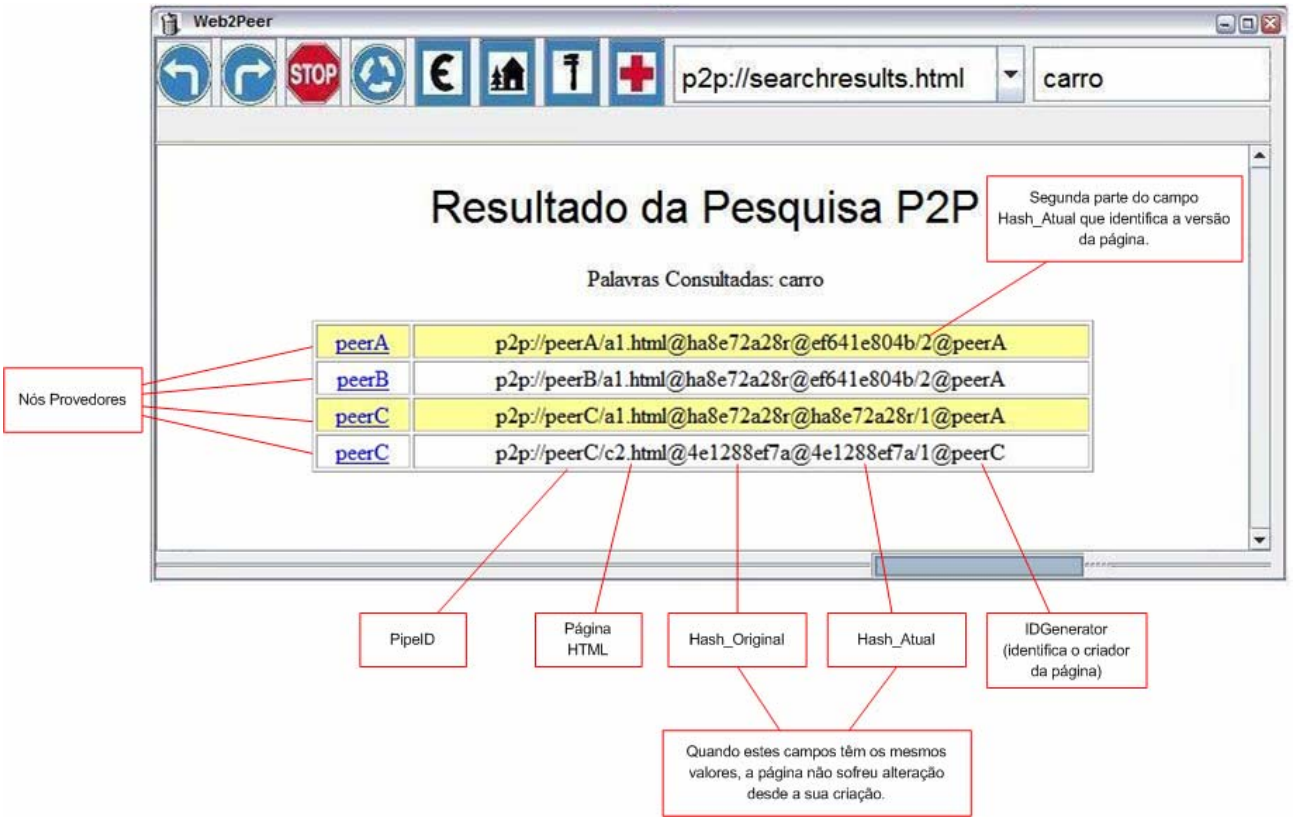


Figura 5.8: Pesquisa de Páginas e Informações de Versões

5.4 A Rede JXTA

Como citado anteriormente, a rede JXTA foi criada utilizando a API da biblioteca *p2psockets*[50]. O *p2psockets* é uma implementação da biblioteca *Sockets* do Java para trabalhar sobre a rede JXTA, facilitando o trabalho dos programadores já habituados a programação com essa biblioteca e também reduzindo o tempo necessário para portar uma aplicação para a plataforma JXTA. Muitos serviços já foram portados para esta plataforma utilizando a API p2psockets. Uma lista completa e atualizada destes projetos pode ser consultada no site oficial do projeto.

Dois desses serviços são fundamentais para a infra-estrutura apresentada, o *Web Server Jetty* e o JXTA *HTTP Client*. Com estes serviços é possível, além de criar o ambiente básico da rede *peer-to-peer*, transferir arquivos HTML pela rede JXTA. Tanto o *Web Server* quanto o cliente HTTP conectam-se à rede JXTA utilizando o mesmo método *signin* da classe *P2PNetwork* do *p2psockets*.

A figura 5.9 mostra o método utilizado para conexão na rede JXTA, onde os parâmetros *userName*, *password*, *network* e *recreate* representam respectivamente o nome do *peer*, senha para entrar na rede p2p, *peergroup* no qual o *peer* deve entrar e o último parâmetro é um parâmetro local que recria a configuração do *peer* caso não encontre o arquivo de configuração local.

```
try {
        P2PNetwork.signin(userName, password, network, recreate);

    } catch (Exception e) {

        System.out.println("Erro ao entrar na rede p2p");
        e.printStackTrace();

      }
```

Figura 5.9: Conexão do nó à rede JXTA

Como citado, o servidor conecta-se a rede JXTA assim que o navegador é iniciado, após conectar-se o servidor publica o *input pipe* que ficará aguardando as conexões solicitando páginas. Já o cliente HTTP conecta-se na rede JXTA somente quando houver a necessidade de solicitar alguma página a outro *peer* remoto. A figura 5.10 mostra parte do código do cliente HTTP JXTA, usado para transferir as páginas selecionadas pelo usuário. Na tabela 5.1 é apresentada uma lista de parâmetros utilizados tanto pelo cliente HTTP, durante uma requisição por uma página, quanto pelo *Jetty Web Server*, no momento em que o navegador é iniciado.

TABELA 5.1: Parâmetros Utilizados pelo *Web* Server e pelo Cliente http

JXTA HTTP CLIENT	JXTA WEB SERVER
Peer Name	Pipe Name
Password Local	PeerGroup
PeerGroup	Peer Name
Pipe Name + HTML file	Password Local
	Virtual Port (default=80)

```
public class JettyClient{
                :
                :
      public String query(String[] arguments){

                try {
                        P2PNetwork.signin(userName, userPass,network, true);
                }
                catch (Exception e) {
                         e.printStackTrace();
                        return(null);
                }

                Protocol jxtaHttp = new Protocol("p2p", new P2PProtocolSocketFactory(),80);
                Protocol.registerProtocol( "p2p", jxtaHttp );
                HttpClient client = new HttpClient();
                client.setConnectionTimeout(50000);

                System.out.println("Connecting to " + remotePipe + "...");
                HttpMethod method = null;
                method = new GetMethod(remotePipe);
                method.setFollowRedirects(true);
                method.setStrictMode(false);

                String responseBody = null;
                 try{
                        client.executeMethod(method);
                        responseBody = method.getResponseBodyAsString();
                 } catch (HttpException he) {
                        System.err.println("Http error connecting to "  + remotePipe );
                        System.err.println(he.getMessage());
                        return(null);
                 } catch (IOException ioe){
                        System.err.println("Unable to connect to '" + remotePipe + "'");
                        return(null);
      }

                        :
                        :
}
```

Figura 5.10: Cliente HTTP JXTA

5.5 Resultados Obtidos

5.5.1 O Ambiente de Desenvolvimento

Todos os módulos do *Web* Browser foram criados utilizando a linguagem de programação *Java 2 Platform Standard Edition Development Kit 5*.0 em conjunto com a *IDE Eclipse 3.1.1* usando o sistema operacional *Windows XP Professional*.

Além do próprio *Web Browser*, foi criado também um *peer relay/rendezvous* sobre o sistema operacional *Linux UBUNTU* versão 6.06/*kernel* 2.6. Este *peer relay* ficou conectado à rede JXTA pública, provida por cinco servidores da *Sun Microsystems*, e serviu como intermediário para *peer* de redes protegidas por NAT e *Firewall*, como a rede da universidade. Este *peer* auxiliar também ajudou a reduzir o tempo de conexão dos *peers* na rede JXTA, por estar fisicamente mais próximo, dentro da universidade e por funcionar como um *rendezvous* reduzia o tempo de busca na rede.

A experiência prática mostrou que o uso desde *peer* reduziu o tempo da conexão do *Web Server*, que acontece quando o *browser* é iniciado, da casa de uma ou duas dezenas de segundos para algo em torne de dois ou três segundos.

Outro aspecto interessante ocorrido durante a criação deste *peer*, foi observar o aumento autônomo do número de conexões entre o *peer rendezvous* local e os demais *rendezvous* da rede JXTA. Inicialmente o *peer* auxiliar local conecta-se apenas a um *rendezvous* da rede JXTA pública, mas conforme o tempo passa e mensagens de propagação são trocadas entre eles as conexões vão aumentando e o *peer* auxiliar local passa a se comunicar com outros *rendezvous* espalhados pela rede.

A figura 5.11 mostra o código do *peer relay/rendezvous* auxiliar criado para que os *peers*, usados para testar o ambiente, conectassem à rede JXTA pública.

```
import net.jxta.discovery.DiscoveryService;
import net.jxta.exception.PeerGroupException;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;
import net.jxta.rendezvous.RendezVousService;

public class MyRelay implements Runnable {

   @SuppressWarnings("unused") private PeerGroup group;
   @SuppressWarnings("unused") private RendezVousService rdvService;
   @SuppressWarnings("unused") private DiscoveryService discovery;

   public static void main(String args[]) throws Exception {

      MyRelay app = new MyRelay();
      app.startJxta();
      app.run();

   }

   @SuppressWarnings("deprecation") private void startJxta() throws Exception {

      try {

      group = PeerGroupFactory.newNetPeerGroup();
      }
      catch (PeerGroupException e) {
                        System.out.println("Erro ao instanciar o grupo");
                        e.printStackTrace();
                        System.exit(-1);
                }

      discovery = group.getDiscoveryService();
      RendezVousService rdvServicee = group.getRendezVousService();
      rdvServicee.startRendezVous();
   }

   public void run() {

      System.out.println("Peer Relay/Rendezvous Iniciado !!!\n");
      while (true) {
         try {
            Thread.sleep(60000);
         } catch (InterruptedException ie) {
            System.exit(0);
         }
      }
   }
}
```

Figura 5.11: *Peer Relay/Rendezvous* auxiliar

5.5.2 O Ambiente de Testes e Validação

O ambiente de teste do protótipo utilizou dois *peers* executados em máquinas físicas distintas, em ambiente *Windows* e um *peer relay*/rendezvous em ambiente *Linux*, conforme cenário detalhado na figura 5.12.

Durante os testes os dois *peers* estavam conectados a partir de redes remotas distintas. Um deles conectado através de uma conexão ADSL doméstica e o segundo conectado na rede *ethernet* 100Mbps do laboratório da universidade. O *peer relay* conectado diretamente à internet com endereço IP válido, aguardando por conexões nas portas 80, 22 e 9701.

Para validar o funcionamento da infra-estrutura foi criada, com o *Web2Peer*, uma página HTML simples, em um dos *peers* (rede doméstica). A esta página foram relacionadas algumas palavras-chave e em seguida foi publicada na rede *peer-to-peer*, conforme processos já discutidos anteriormente.

Em seguida o outro *peer* (rede da PUCPR) consultou por uma das palavras-chave usadas para a publicação, e a lista de *peers* provedores dos conteúdos foi recebida. Selecionando o *peer* provedor, que era único neste caso (rede doméstica), o arquivo HTML referente a página foi transferido através da rede JXTA, com a interação direta dos cliente HTTP e do *Web* Server. A página foi então exibida na tela do *Web2Peer*.

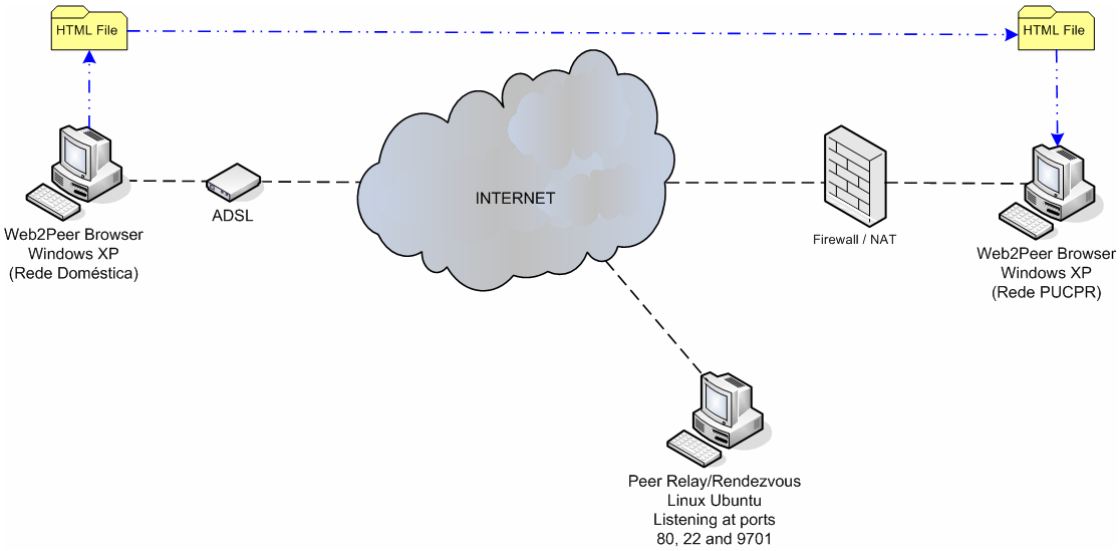


Figura 5.12: Ambiente de Testes e Validação

Após transferência do arquivo, foi realizada uma nova consulta com a mesma palavra-chave utilizada anteriormente. Porém, nesta segunda consulta, após receber a lista de *peers* provedores, constatou-se que a mesma página estava disponível agora em dois *peers* distintos, conforme esperado, pois devido ao acesso anterior o *peer* da rede da universidade (PUCPR) passou a ser um novo provedor do conteúdo acessado, comprovando o correto funcionamento dos mecanismos de replicação, transferência, consulta e publicação do *Web2Peer*.

5.5.3 Resultados e Avaliação

Foram realizadas algumas medições em relação aos tempos de pesquisa e transferência das páginas para que houvesse alguns parâmetros de comparação entre a tecnologia proposta e a atualmente utilizada, visando também demonstrar a viabilidade do uso prático desta abordagem.

Em relação ao tempo de pesquisa de páginas, os valores encontrados são próximos aos obtidos com a utilização de mecanismos de busca utilizados atualmente na internet, como o *Google*, que é atualmente o mecanismo de busca mais utilizado para pesquisa de conteúdos na rede[51]. Apesar de possuir uma arquitetura totalmente diferente da infra-estrutura proposta, por utilizar um cluster com dezenas de milhares de máquinas e mecanismos ativos de procura e indexação de páginas, ainda assim os tempos obtidos foram similares. Isto torna o *Web2Peer* uma alternativa viável sem perda em relação aos tempos usados pelos seus processos internos.

Em uma primeira análise, foram realizadas 10 consultas com diferentes números de palavras-chave (1, 2, 3, 5, e 7 palavras), onde as respostas retornaram dez resultados de cada vez, tal como o *Google* apresenta as suas respostas, o tempo gasto é mostrado na figura 5.13.

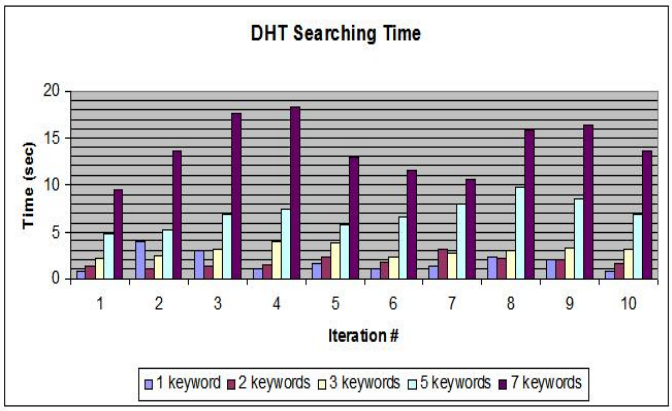


Figura 5.13: Tempo de Busca com Diferentes Números de Palavras-Chave

No gráfico da figura 5.13 é possível observar que as buscas que demandam um tempo maior de espera são buscar com um número superior de palavras-chave (5 e 7 palavras). Para as outras pesquisas os resultados não são elevados, ficando em alguns casos inferior a um segundo de espera.

Como citado anteriormente, o ambiente no qual o *Google* é executado é muito diferente em termos de hardware e software, proporcionando tal ganho. A comparação entre os sistemas não pode ser considerada justa devido à superioridade, em termos de hardware, do ambiente do Google[51]. Mas como esta é a ferramenta atualmente mais utilizada, e os resultados obtidos foram satisfatórios a comparação torna-se válida como uma referência para usuário.

Após consultar as páginas e escolher a fonte da qual quer receber o conteúdo, o peer precisa aguardar um tempo gasto pelo *peer* para transferir tal arquivo sobre a rede JXTA do *peer* provedor ao *peer* requisitante.

Para tal transferência também foi analisado o tempo decorrido entre o *click* do mouse, que seleciona o *peer* que proverá a página e o armazenamento local do arquivo. Em uma das amostras colhidas este teste foi registrado dez vezes, para avaliar se o tempo médio gasto para tal operação seria aceitável. Os resultados obtidos em uma das análises são apresentados na figura 5.14.

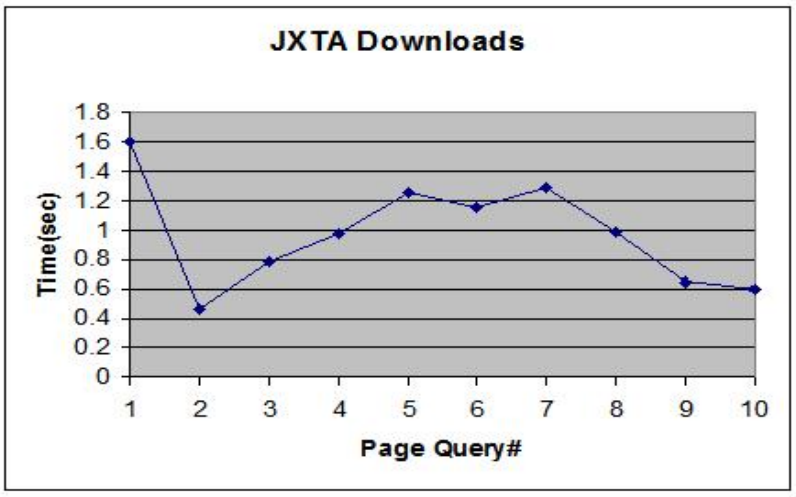


Figura 5.14: Tempo de transferência do arquivo pela rede JXTA

Também foi observado o tempo médio gasto pelo processo de publicação de uma página, tendo em vista que não só a criação de um documento pode executa esta tarefa, mas também os processos de replicação, por acesso ou atualização de páginas executam esta tarefa com relativa freqüência na infra-estrutura.

Para tal aferição foram criadas dez páginas *Web* e realizada uma publicação simultânea destas páginas para obter o tempo médio de cada uma das publicações. Este teste foi repetido por volta de dez vezes para identificar possíveis variações, porém o tempo médio de publicação manteve-se estável sem grandes variações. Os resultados destes testes são apresentados na figura 5.15.

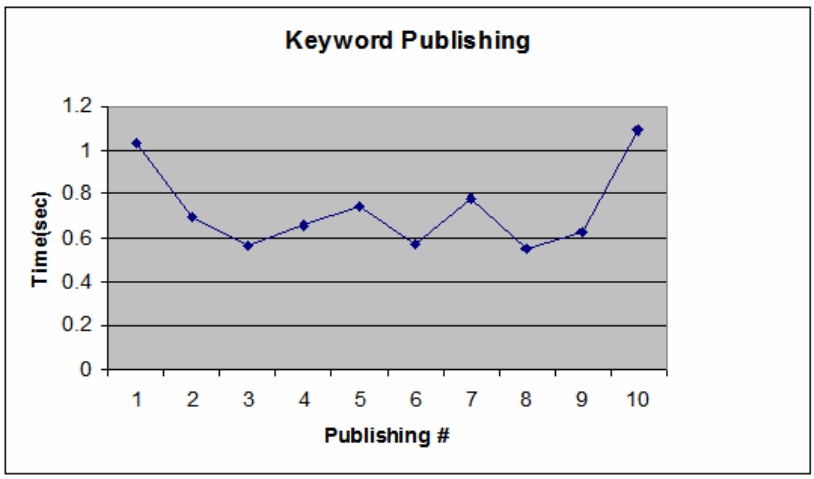


Figura 5.15: Tempo de publicação na DHT

5.6 Conclusão do Capítulo

Neste capítulo foram apresentados alguns detalhes da implementação do *Web2Peer*, sua divisão em blocos, aspectos importantes em relação à construção desses blocos e a interação entre o navegador e a tabela *hash* distribuída (DHT), utilizada como indexador de páginas.

Também foram mostrados detalhes sobre a transferência dos arquivos pela rede JXTA, procurando demonstrar a forma com a qual os objetivos e funcionalidades apresentadas pelo modelo foram atingidos.

O capítulo explica também o ambiente utilizado para avaliação do modelo, como os testes com o protótipo inicial foram realizados e alguns resultados numéricos obtidos durante os testes.

Com estes resultados fica demonstrado que a utilização prática do *Web2Peer* é viável, pois os tempos necessários por seus processos internos não diferem tanto dos tempos utilizados em mecanismos atualmente utilizados.

Capítulo 6

Conclusão e Trabalhos Futuros

Redes *peer-to-peer* foram concebidas inicialmente para compartilhamento de recursos, principalmente multimídia, tais como arquivo de músicas, vídeo, etc. Com o passar do tempo verificou-se que estas redes virtuais poderiam contribuir de uma maneira mais importante, eliminando o estigma inicial de software para contravenção e quebra de direitos autorais.

Este trabalho explora uma das tantas outras possibilidades de uso das redes *peer-to-peer* em favor da comunidade, oferecendo um serviço simplificado que permite a publicação, a busca e a replicação de páginas *Web* na Internet.

A solução proposta é original de acordo com a literatura pesquisada até o presente momento, torna a disponibilização de conteúdos na internet através das páginas *Web* mais democrática e com menos custos.O *Web2Peer* permite que qualquer usuário conectado à internet, faça de sua máquina um provedor de conteúdo *Web* com total autonomia de administração, sem depender de nenhuma entidade reguladora, ao mesmo tempo em que colabora com outros provedores, aumentando a disponibilidade dos conteúdos por eles providos.

Os principais objetivos deste trabalho foram aumentar a disponibilidade das páginas publicadas na internet (evitando pontos críticos de falha) e transformar o processo de publicação em um processo muito mais simples, democratizando ainda mais o acesso a essa tecnologia.

Esses objetivos foram atingidos com a utilização do conjunto de protocolos JXTA e um conjunto de outras tecnologias existentes, obtendo-se como resultado um *Web Browser* com uma interface amigável, que mantém a sua compatibilidade com a *Web* convencional, e adiciona os recursos necessários para a utilização de uma rede *peer-to-peer* para publicação de páginas *Web*.

As avaliações de desempenho mostraram que a solução proposta não é custosa, mesmo se comparada com a *Web* convencional, o que torna o uso do protótipo final viável.

Existem alguns pontos que precisam ser melhorados, sendo que alguns deles já estão sendo direcionados, para tornar a solução apresentada ainda mais completa. Entre estas melhorias podem ser citadas:

- Criação de mecanismos de segurança que permitam ao usuário assinar digitalmente as páginas por ele criadas e providas, fornecendo dessa maneira valores de autenticidade e integridade dos documentos publicados.
- Mecanismos criptográficos com o intuito de fornecer confidencialidade às páginas.
- Mecanismos de reputação para que os usuários da rede possam ter seus comportamentos avaliados pelos demais usuários, a fim de fornecer um respaldo ainda maior aos provedores de informação.
- Mecanismos de controle de acesso, baseados em infra-estruturas de chave pública para que conteúdos possam ser resguardados de usuários indesejados.
- Avaliação da escalabilidade da infra-estrutura em ambientes com um número elevador de peers.
- Integração da DHT aos nós presentes na rede *peer-to-peer*, distribuindo os nós da DHT em cada um dos nós da rede *peer-to-peer*.

Com o progresso obtido com este trabalho, algumas das contribuições citadas foram direcionadas e novos trabalhos procuraram contribuir para a maior abrangência da infra-estrutura aqui proposta.

Um dos trabalhos, desenvolvido por membros deste projeto (que gerou uma outra dissertação de mestrado), nesta mesma instituição, propõe a utilização de uma infra-estrutura SDKI/SPKI (*Simple Distributed Security Infrastructure/Simple Public Key Infrastructure*) sobre a infra-estrutura aqui proposta, com o objetivo de prover autenticidade e controle de acesso ás páginas providas por esta infra-estrutura.

Além disso, este trabalho acrescenta à infra-estrutura um mecanismo de reputação de chaves públicas para aumentar a credibilidade destas chaves perante os usuários do sistema.

Com estes e outros objetivos este projeto será também apresentado como um pacote de melhorias a infra-estrutura do Web2Peer e também será defendido em forma de dissertação.

Como resultados, em termos de publicações científicas, esta dissertação obteve dois artigos aceitos e publicados em congressos internacionais até o presente momento:

- ***Implementing a Peer-to-Peer Web Browser for Publishing and Searching Web Pages on Internet.*** RIBEIRO, Heverson, LUNG, Lau Cheuk, SANTIN, Altair Olivo, BRISOLA, N. L. Publicado no IEEE 21st International Conference on Advanced Information Networking and Applications, 2007, Niagara Falls, Canada. (AINA07).

- ***Web2Peer: A Peer-to-Peer Infrastructure for Publishing/Locating/Replicating Web Pages on Internet.*** RIBEIRO, Heverson, LUNG, Lau Cheuk, SANTIN, Altair Olivo, BRISOLA, N. L. Publicado no IEEE 8th International Symposium on Autonomous Decentralized Systems, 2007, Sedona, Arizona.(ISADS07).

Referências Bibliográficas

[1] "World Internet Usage Statistics News and Population Stats", www.internetworldstats.com, http://www.internetworldstats.com/stats.htm Access: [16 January 2007]

[2] L. J. Stanton, "Factors influencing the adoption of residential broadband connections to the internet," *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pp. 128-137, 2004.

[3] L. M. T. Berners-Lee, and M. McCahill, "RFC1738: Uniform Resource Locators (URL) Internet RFC," 1994.

[4] R. F. T. Berners-Lee, and L. Masinter, "RFC2396: Uniform Resource Identifiers (URI): Generic Syntax," Internet RFC," 1998.

[5] "Registro BR - Fapesp", www.registro.br, http://www.registro.br/ Access: [16 January 2007]

[6] "ICANN - Internet Corporation for Assigned Names and Numbers", www.icann.org, http://www.icann.org/ Access: [16 January 2007]

[7] D. H. R. Orfali, and J. Edwards, *Client/Server Survival Guide*. New York, NY, USA: John Wiley & Sons, Inc., 1999.

[8] P. Weygant, *Clusters for High Availability: A Primer of Hp Solutions*: Prentice Hall PTR, 2001.

[9] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys (CSUR)*, vol. 36, pp. 335-371, 2004.

[10] J. A. Mussini, "Distributed Hash Table: The State of the Art," 2006.

[11] "Google", www.google.com.br, http://www.google.com.br/ Access: [17 January 2007]

[12] D. D. a. D. O'Mahony, "Overlay Networks: A Scalable Alternative for P2P," presented at Fifth Metaheuristics International Conference, Kyoto, Japan, 2003.

[13] J. C. E. K. Lua, M. Pias, R. Sharma, and S. Lim "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," presented at IEEE Communications Survey and Tutorial, 2004.

[14] R. Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Application," presented at First International Conference on Peer-to-Peer Computing (P2P`01), Linköpings universitet, Sweden, 2001.

[15] J. Lindroos, "Peer-to-Peer Content Distribution," in *Department of Computer Science*. Turku: Åbo Akademi University, 2003.

[16] "Napster ", www.napster.com, http://www.napster.com/ Access: [18 January 2007]

[17] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, "A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems," *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II-Volume 02*, pp. 205-213, 2005.

[18] C. Hoong Ding, S. Nutanong, and R. Buyya, "P2P Networks for Content Sharing," *Arxiv preprint cs/0402018*, 2004.

[19] M. Castro, M. Costa, and A. Rowstron, "Should we build Gnutella on a structured overlay?."

[20] O. S. I. Clarke, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," presented at ICSI Workshop on Design Issues in Anonymity and Unobservability, International Computer Science Institute (ICSA), Berkeley, California, USA, 2000.

[21] L. Gong, "Project JXTA: A Technology Overview," Sun Microsystems Inc 2002.

[22] S. Microsystems, "JXTA v2.3: Java Programmer's Guide," Sun Microsystems, 2005.

[23] A. A. B. Traversat, Mohamed Abdelaziz, M. Duigou, Carl Haywood, J.-C. Hugly, and B. Y. Eric Pouyoul, "Project JXTA 2.0 Super-Peer Virtual Network." ted at IEEE Communi2003.

[24] B. Traversat, M. Abdelaziz, M. Duigou, J. C. Hugly, E. Pouyoul, and B. Yeager, "Project JXTA Virtual Network," *Sun Microsystems, Oktober*, 2002.

[25] B. J. Wilson, *JXTA*. Indianapolis: New Riders Publishing, 2002.

[26] F. a. Favarim, "Espaço de Tuplas como Suporte para Serviços em Grids Computacionais," Universidade Federal de Santa Catarina, 2005.

[27] R. M. I. Stoica, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," presented at Conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, California, United States, 2001.

[28] P. F. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," presented at ACM SIGCOMM Special Interest Group on Data Communications (SIGCOMM 01), San Diego, California, USA, 2001.

[29] P. D. A. Rowstron, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Lecture Notes in Computer Science*, vol. 2218, 2001, pp. 329–350.

[30] L. H. B. Y. Zhao, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 41-53, 2004.

[31] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: a public DHT service and its uses," *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 73-84, 2005.

[32] "OpenDHT: A Publicly Accessible DHT Service", www.opendht.org, http://www.opendht.org/ Access: [18 January 2007]

[33] R. Srinivasan, "RFC 1831," *RPC: Remote Procedure Call Protocol Specification Version*, vol. 2.

[34] X. Specification, "URL: http://www. xmlrpc. com/spec," *Zugriff am*, vol. 11, 2003.

[35] "PlanetLab: Home", l. o. www.planet, http://www.planet-lab.org/ Access: [18 January 2007]

[36] "FreeWeb Website", Freeweb, http://freeweb.sourceforge.net/ Access: [18 January 2007]

[37] J. M. M. Nakamura, K. Chiba, M. Shizuka, and Y. Miyoshi, "Design and implementation of a P2P shared Web browser using JXTA," presented at 17th International Conference on Advanced Information Networking and Applications (AINA 2003), 2003.

[38] M. J. Freedman, E. Freudenthal, and D. Mazières, "Democratizing content publication with Coral."

[39] B. G. Lee, K. H. Chang, and N. H. Narayanan, "An integrated approach to version control management in computer supported collaborative writing," *Proceedings of the 36th annual Southeast regional conference*, pp. 34-43, 1998.

[40] "Sistema de controle de versão - Wikipédia", pt.wikipedia.org, http://pt.wikipedia.org/wiki/Sistema_de_controle_de_versao, Access: [21 January 2007]

[41] M. Barborak, M. Malek, and A. Dahbura, "The Consensus Problem in Fault-Tolerant Computing," *ACM Computing Surveys*, vol. 25, 1993.

[42] M. J. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," *Proceedings of the 10th International Colloquium on Automata, Languages, and Programming*, pp. 127–140, 1983.

[43] D. R. Stinson, *Cryptography: Theory and Practice*: CRC Press, 2002.

[44] "jetty6 - Jetty WebServer", jetty.mortbay.org, http://jetty.mortbay.org/ Access: [18 January 2007]

[45] "HttpClient - HttpClient Home", jakarta.apache.org, http://jakarta.apache.org/commons/httpclient/ Access: [18 January 2007]

[46] "Calpa Java Browser Home Page", Calpa, http://htmlbrowser.sourceforge.net/ Access: [18 June 2006]

[47] S. Rhea, "The Bamboo Distributed Hash Table, 2004," *URL http://bamboo-dht. org*.

[48] B. G. S. Rhea, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: a public DHT service and its uses," presented at Proceedings of the 2005 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2005.

[49] "XML-RPC Specification", www.xmlrpc.com, http://www.xmlrpc.com/spec Access: [18 January 2007]

[50] "p2psockets: P2PSockets", p2psockets.jxta.org, http://p2psockets.jxta.org/ Access: [18 January 2007]

[51] L. A. Barroso, J. Dean, and U. Holzle, "Web search for a planet: The Google cluster architecture," *Micro, IEEE*, vol. 23, pp. 22-28, 2003.